



# MCUXpresso SDK Documentation

Release 25.12.00-pvw2



NXP  
Nov 14, 2025



# Table of contents

|          |                                                              |            |
|----------|--------------------------------------------------------------|------------|
| <b>1</b> | <b>FRDM-MCXA153</b>                                          | <b>3</b>   |
| 1.1      | Overview                                                     | 3          |
| 1.2      | Getting Started with MCUXpresso SDK Package                  | 3          |
| 1.2.1    | Getting Started with MCUXpresso SDK Package                  | 3          |
| 1.3      | Getting Started with MCUXpresso SDK GitHub                   | 55         |
| 1.3.1    | Getting Started with MCUXpresso SDK Repository               | 55         |
| 1.4      | Release Notes                                                | 68         |
| 1.4.1    | MCUXpresso SDK Release Notes                                 | 68         |
| 1.5      | ChangeLog                                                    | 72         |
| 1.5.1    | MCUXpresso SDK Changelog                                     | 72         |
| 1.6      | Driver API Reference Manual                                  | 142        |
| 1.7      | Middleware Documentation                                     | 142        |
| 1.7.1    | MCU Boot                                                     | 142        |
| 1.7.2    | FreeMASTER                                                   | 142        |
| 1.7.3    | FreeRTOS                                                     | 142        |
| <b>2</b> | <b>MCXA153</b>                                               | <b>143</b> |
| 2.1      | AOI: Crossbar AND/OR/INVERT Driver                           | 143        |
| 2.2      | CDOG                                                         | 146        |
| 2.3      | Clock Driver                                                 | 150        |
| 2.4      | CRC: Cyclic Redundancy Check Driver                          | 168        |
| 2.5      | CTIMER: Standard counter/timers                              | 170        |
| 2.6      | eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver | 180        |
| 2.7      | eDMA core Driver                                             | 211        |
| 2.8      | eDMA soc Driver                                              | 218        |
| 2.9      | EIM: error injection module                                  | 218        |
| 2.10     | EQDC: Enhanced Quadrature Encoder/Decoder                    | 218        |
| 2.11     | ERM: error recording module                                  | 236        |
| 2.12     | FGPIO Driver                                                 | 239        |
| 2.13     | FREQME: Frequency Measurement                                | 239        |
| 2.14     | GLIKEY                                                       | 239        |
| 2.15     | GLIKEY                                                       | 239        |
| 2.16     | GPIO: General-Purpose Input/Output Driver                    | 243        |
| 2.17     | GPIO Driver                                                  | 246        |
| 2.18     | I3C: I3C Driver                                              | 252        |
| 2.19     | I3C Common Driver                                            | 254        |
| 2.20     | I3C Master Driver                                            | 256        |
| 2.21     | I3C Master DMA Driver                                        | 282        |
| 2.22     | I3C Slave Driver                                             | 285        |
| 2.23     | I3C Slave DMA Driver                                         | 298        |
| 2.24     | INPUTMUX: Input Multiplexing Driver                          | 300        |
| 2.25     | Common Driver                                                | 341        |
| 2.26     | Lin_lpuart_driver                                            | 353        |
| 2.27     | LPADC: 12-bit SAR Analog-to-Digital Converter Driver         | 361        |
| 2.28     | Lpc_freqme                                                   | 381        |
| 2.29     | LPCMP: Low Power Analog Comparator Driver                    | 386        |

|          |                                                                      |            |
|----------|----------------------------------------------------------------------|------------|
| 2.30     | LPI2C: Low Power Inter-Integrated Circuit Driver                     | 396        |
| 2.31     | LPI2C Master Driver                                                  | 397        |
| 2.32     | LPI2C Master DMA Driver                                              | 411        |
| 2.33     | LPI2C Slave Driver                                                   | 414        |
| 2.34     | LPSPi: Low Power Serial Peripheral Interface                         | 424        |
| 2.35     | LPSPi Peripheral driver                                              | 424        |
| 2.36     | LPSPi eDMA Driver                                                    | 445        |
| 2.37     | LPTMR: Low-Power Timer                                               | 452        |
| 2.38     | LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver | 458        |
| 2.39     | LPUART Driver                                                        | 458        |
| 2.40     | LPUART eDMA Driver                                                   | 476        |
| 2.41     | MCX_CMC: Core Mode Controller Driver                                 | 479        |
| 2.42     | MCX_SPC: System Power Control driver                                 | 491        |
| 2.43     | MCX_VBAT: Smart Power Switch                                         | 529        |
| 2.44     | OSTIMER: OS Event Timer Driver                                       | 548        |
| 2.45     | PORT: Port Control and Interrupts                                    | 552        |
| 2.46     | PWM: Pulse Width Modulator                                           | 560        |
| 2.47     | Reset Driver                                                         | 585        |
| 2.48     | ROMAPI Driver                                                        | 588        |
| 2.49     | TRDC: Trusted Resource Domain Controller                             | 596        |
| 2.50     | Trdc_core                                                            | 618        |
| 2.51     | Trdc_soc                                                             | 632        |
| 2.52     | UTICK: MicroTick Timer Driver                                        | 632        |
| 2.53     | WAKETIMER: WAKETIMER Driver                                          | 633        |
| 2.54     | WUU: Wakeup Unit driver                                              | 636        |
| 2.55     | WWDT: Windowed Watchdog Timer Driver                                 | 641        |
| <b>3</b> | <b>Middleware</b>                                                    | <b>645</b> |
| 3.1      | Boot                                                                 | 645        |
| 3.1.1    | MCUXpresso SDK : mcuxsdk-middleware-mcuboot_opensource               | 645        |
| 3.1.2    | MCUboot                                                              | 646        |
| 3.2      | Motor Control                                                        | 647        |
| 3.2.1    | FreeMASTER                                                           | 647        |
| <b>4</b> | <b>RTOS</b>                                                          | <b>685</b> |
| 4.1      | FreeRTOS                                                             | 685        |
| 4.1.1    | FreeRTOS kernel                                                      | 685        |
| 4.1.2    | FreeRTOS drivers                                                     | 685        |
| 4.1.3    | backoffalgorithm                                                     | 685        |
| 4.1.4    | corehttp                                                             | 685        |
| 4.1.5    | corejson                                                             | 685        |
| 4.1.6    | coremqtt                                                             | 686        |
| 4.1.7    | corepkcs11                                                           | 686        |
| 4.1.8    | freertos-plus-tcp                                                    | 686        |

This documentation contains information specific to the frdm-mcxa153 board.



# Chapter 1

## FRDM-MCXA153

### 1.1 Overview

The NXP FRDM-MCXA153 is a development board for the A153 96 MHz Arm Cortex-M33 microcontroller.



MCU device and part on board is shown below:

- Device: MCXA153
- PartNumber: MCXA153VLH

### 1.2 Getting Started with MCUXpresso SDK Package

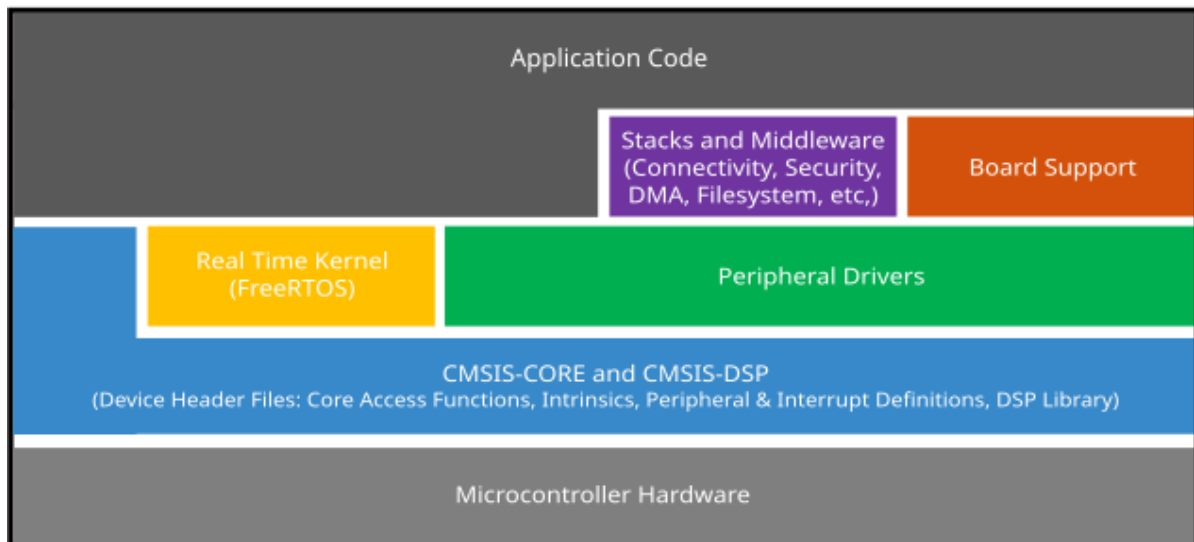
#### 1.2.1 Getting Started with MCUXpresso SDK Package

##### 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):

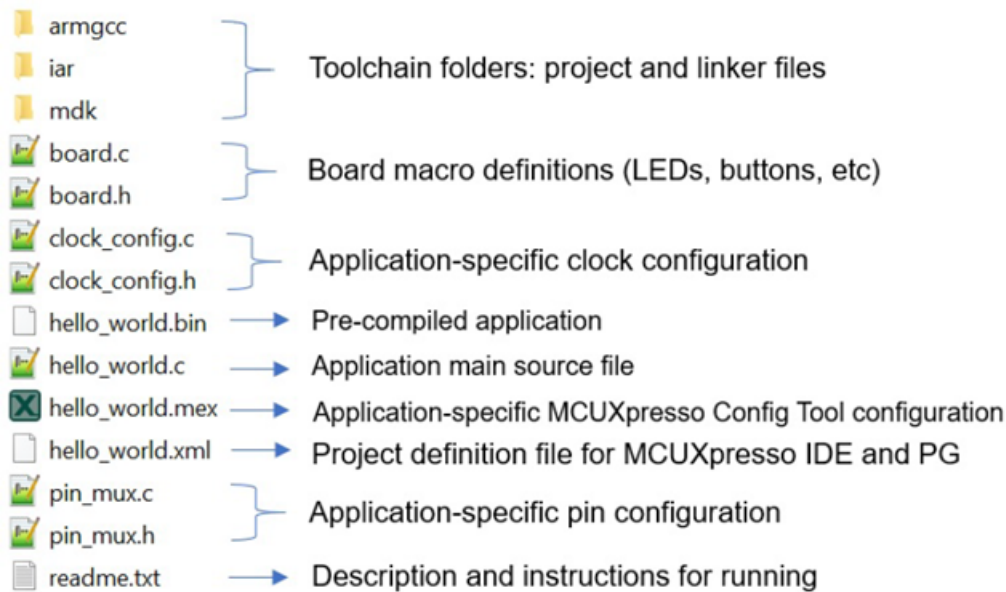
- `cmsis_driver_examples`: Simple applications intended to show how to use CMSIS drivers.
- `demo_apps`: Full-featured applications that highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- `driver_examples`: 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/<device_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.

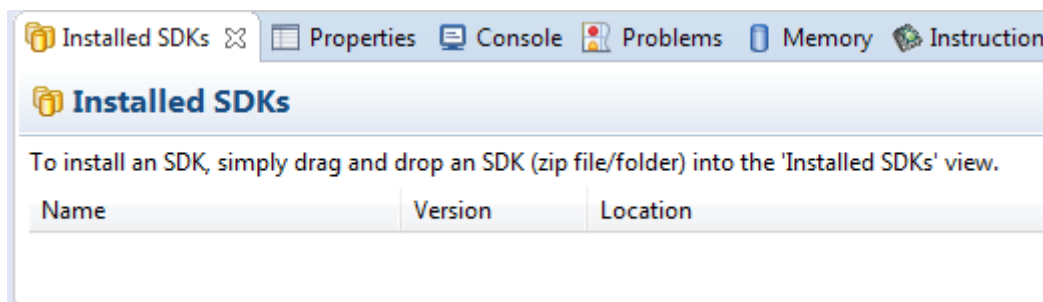
This section describes the steps required to configure MCUXpresso IDE to build, run, and debug example applications. The `hello_world` demo application targeted for the hardware platform is

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

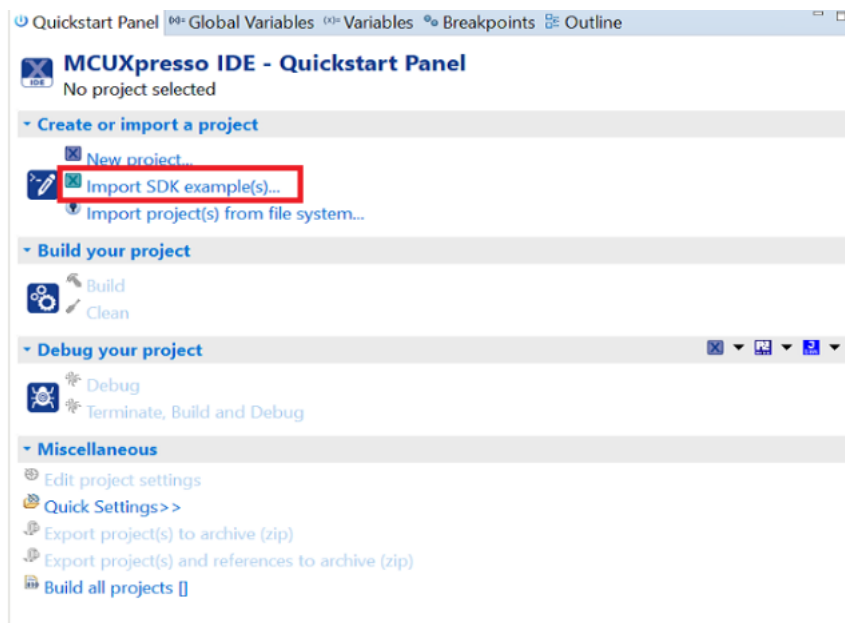
**Select the workspace location** Every time MCUXpresso IDE launches, it prompts the user to select a workspace location. MCUXpresso IDE is built on top of Eclipse which uses workspace to store information about its current configuration, and in some use cases, source files for the projects are in the 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.

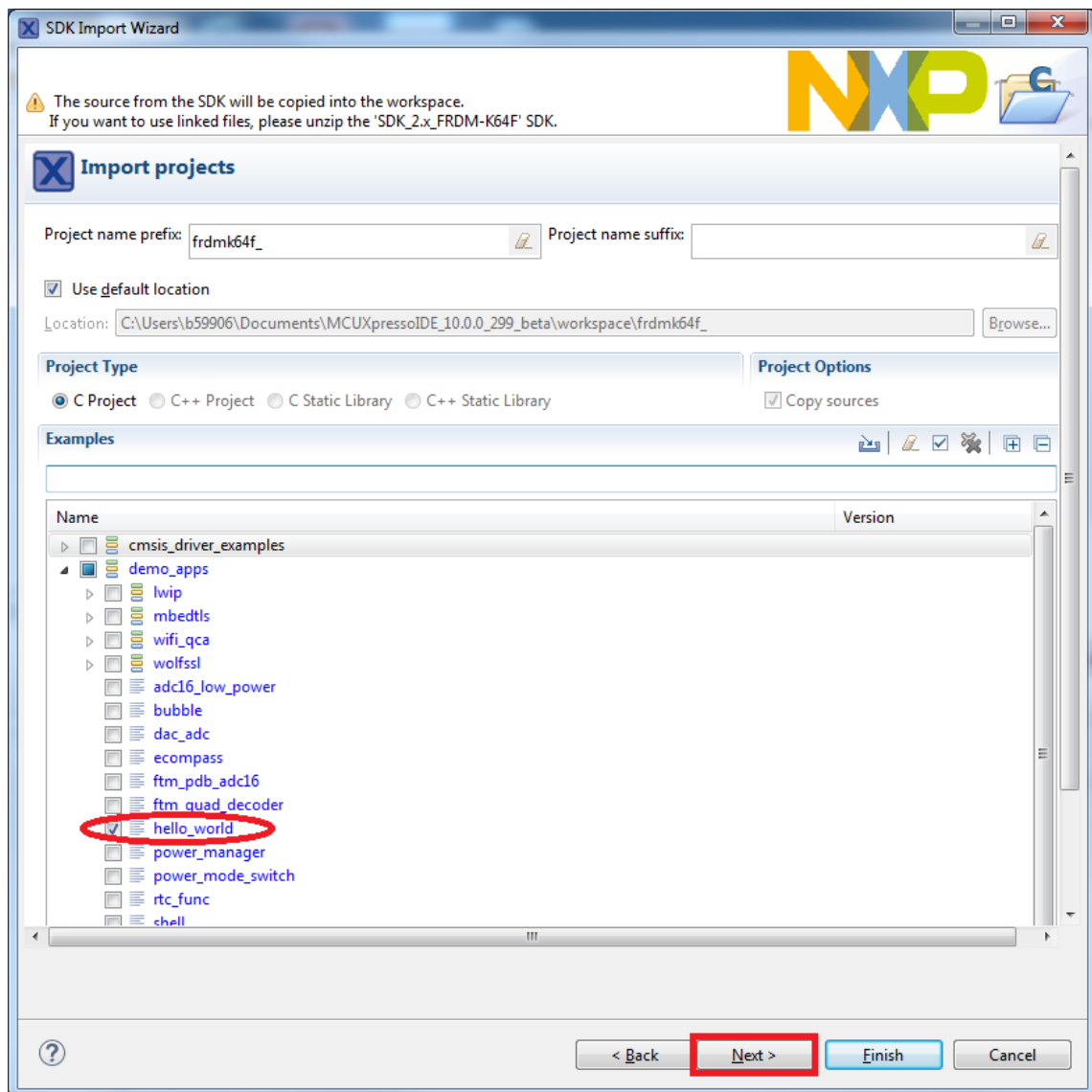
1. Drag and drop the SDK zip file into the **Installed SDKs** view to install an SDK. In the window that appears, click **OK** and wait until the import has finished.



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



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



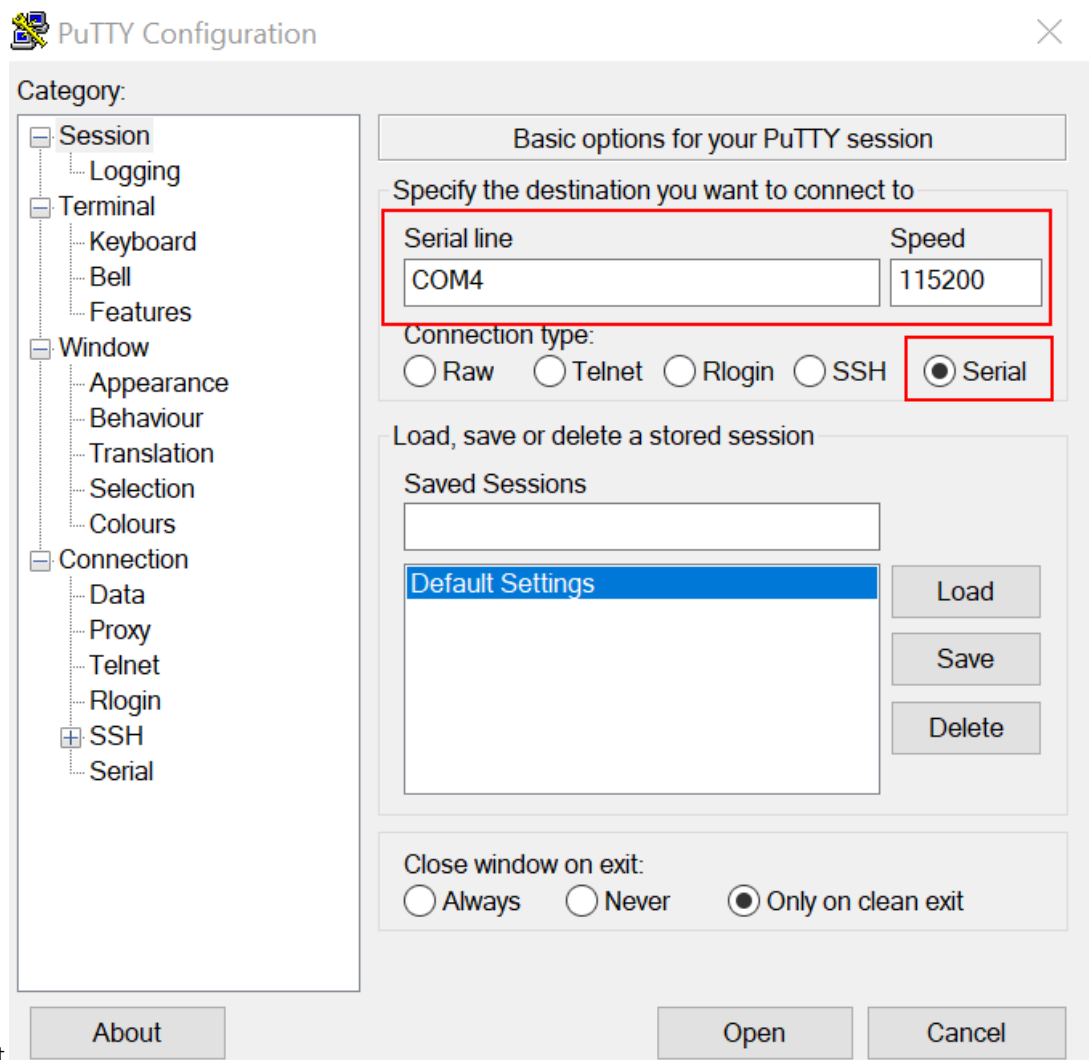
5. Ensure **Redlib: Use floating-point version of printf** is selected if the example prints floating-point numbers on the terminal for demo applications such as `adc_basic`, `adc_burst`, `adc_dma`, and `adc_interrupt`. Otherwise, it is not necessary to select this option. Then, click **Finish**.

**Run an example application** For more information on debug probe support in the MCUXpresso IDE, see [community.nxp.com](http://community.nxp.com).

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

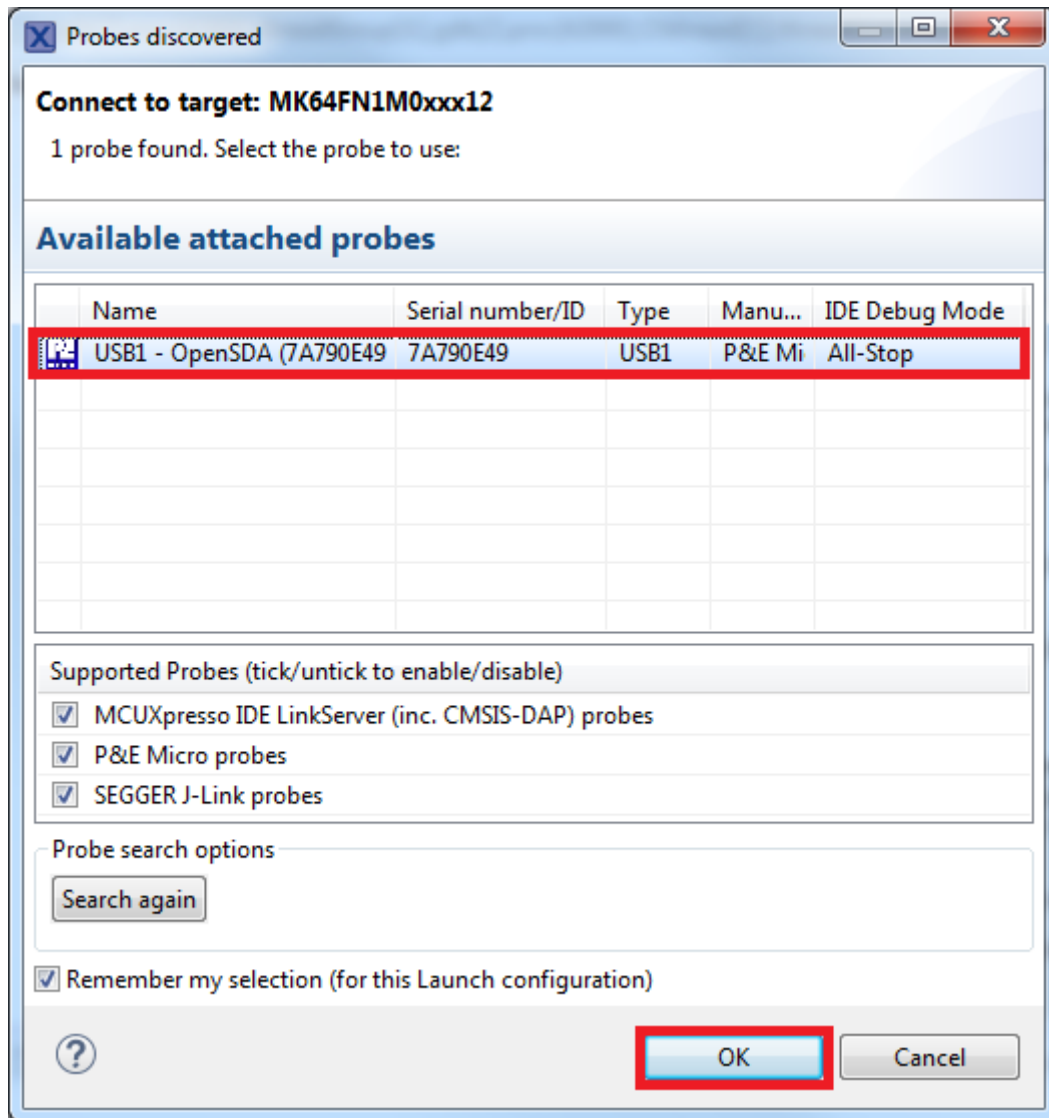
1. Ensure the host driver for the debugger firmware has been installed. See [On-board debugger](#).
2. Connect the development platform to your PC via a USB cable.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM 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.)



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

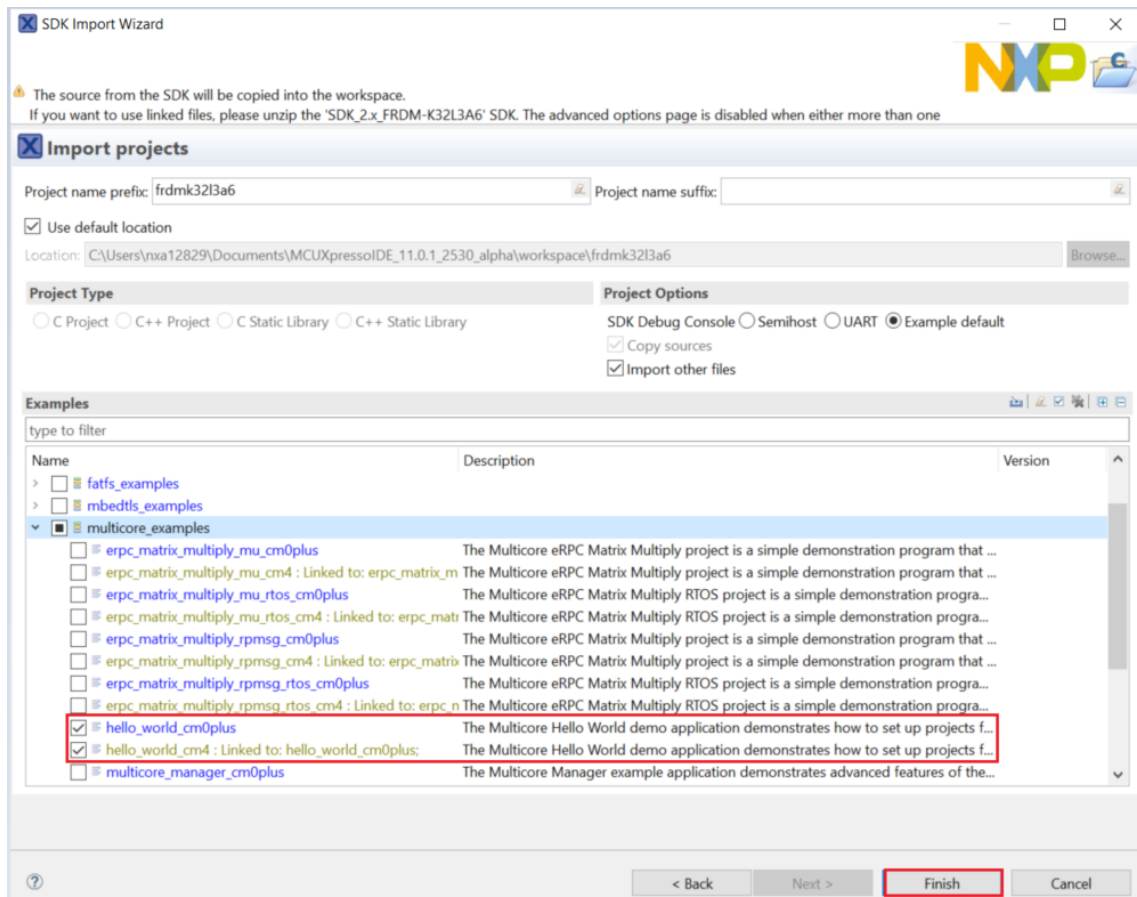


The `hello_world` application is now running and a banner is displayed on the terminal. If not, check your terminal settings and connections.

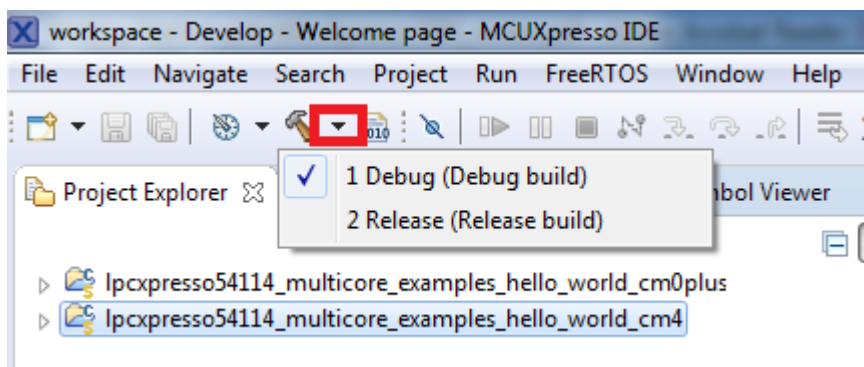


**Build a multicore example application** This section describes the steps required to configure MCUXpresso IDE to build, run, and debug multicore example applications. The following steps can be applied to any multicore example application in the MCUXpresso SDK. Here, the dual-core version of hello\_world 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**.



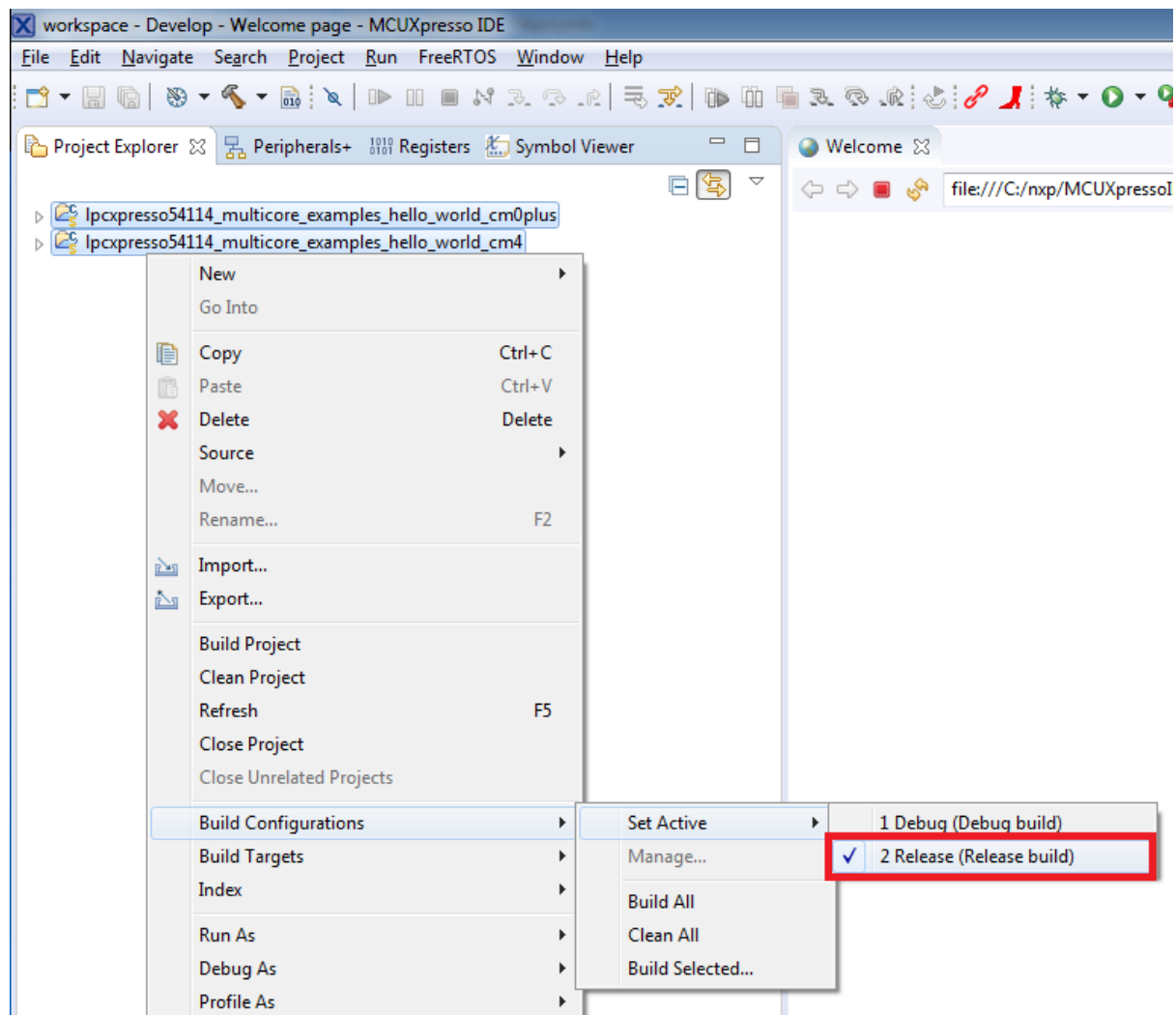
3. Now, two projects should be imported into the workspace. To start building the multicore application, highlight the `lpcxpresso54114_multicore_examples_hello_world_cm4` project (multicore master project) in the Project Explorer. Then choose the appropriate build target, **Debug** or **Release**, by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



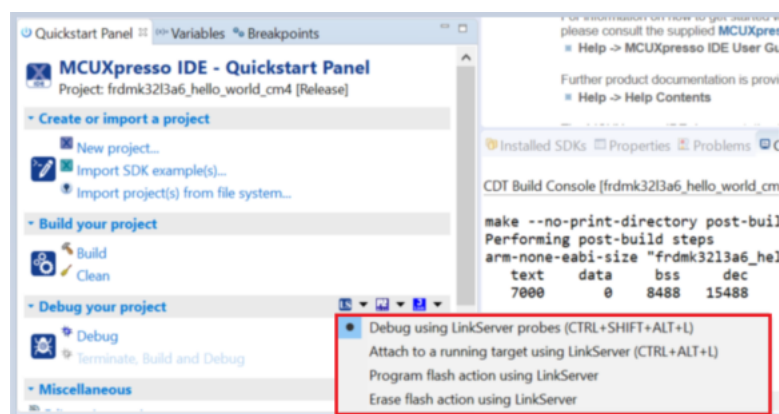
The project starts building after the build target is selected. Because of the project reference settings in multicore projects, triggering the build of the primary core application (cm4) also causes the referenced auxiliary core application (cm0plus) to build.

**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.

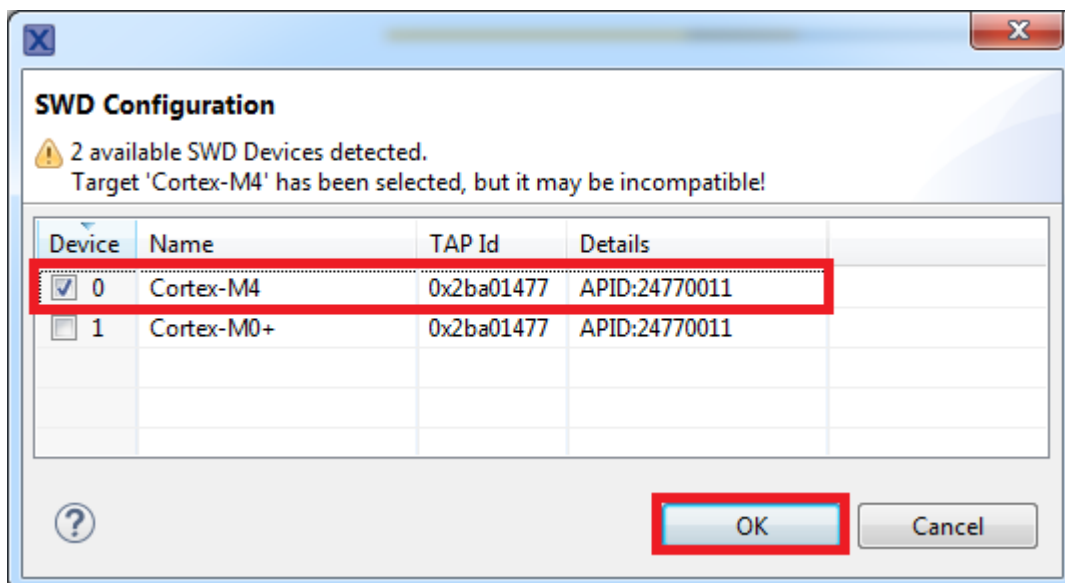
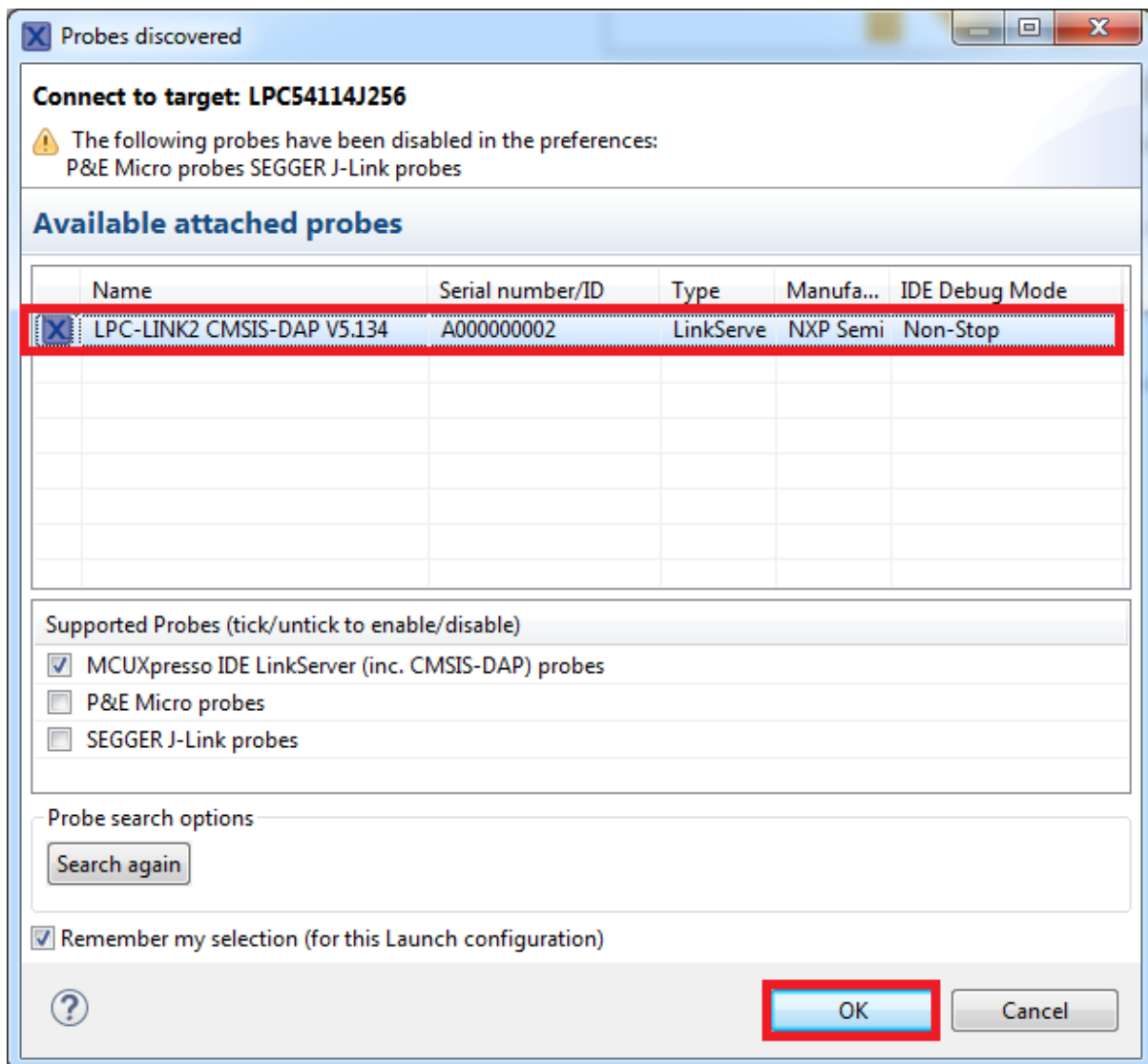


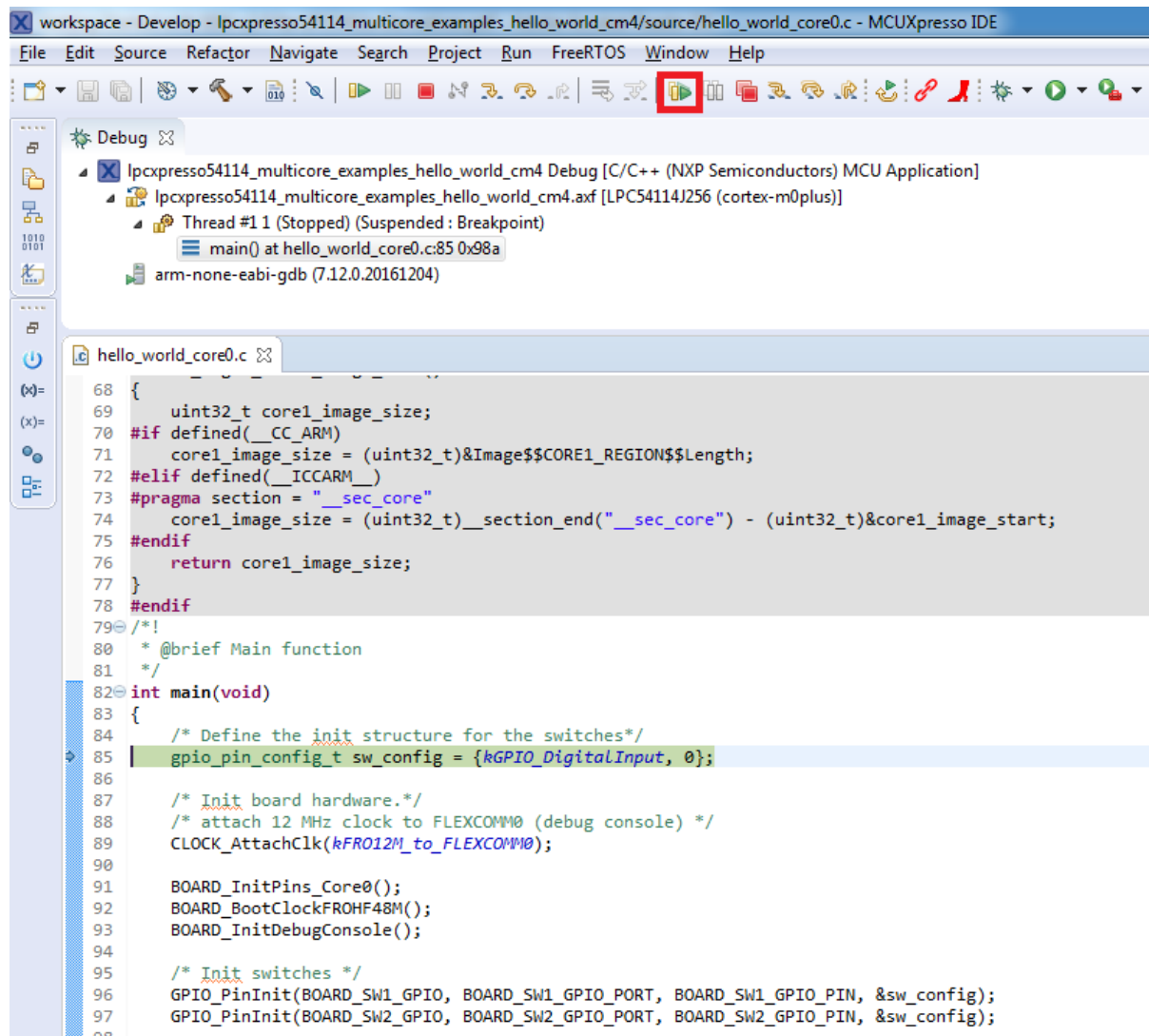


**Run a multicore example application** The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform all steps as described in **Run 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.

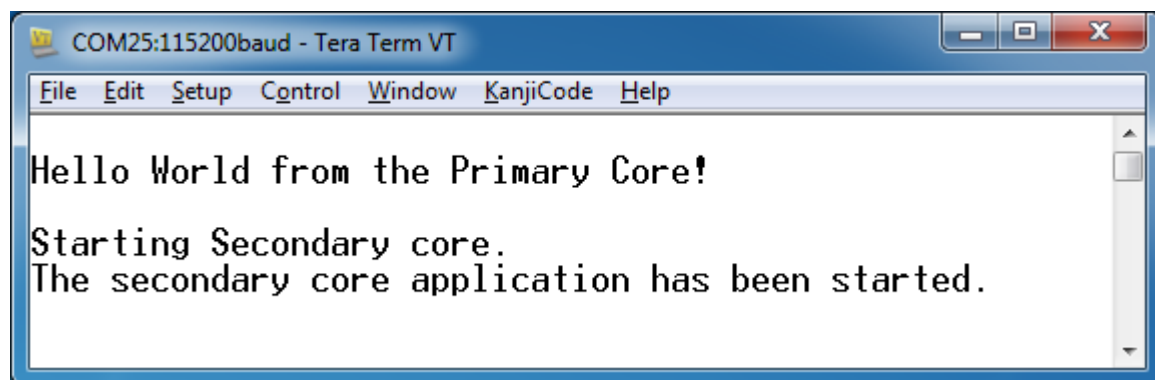






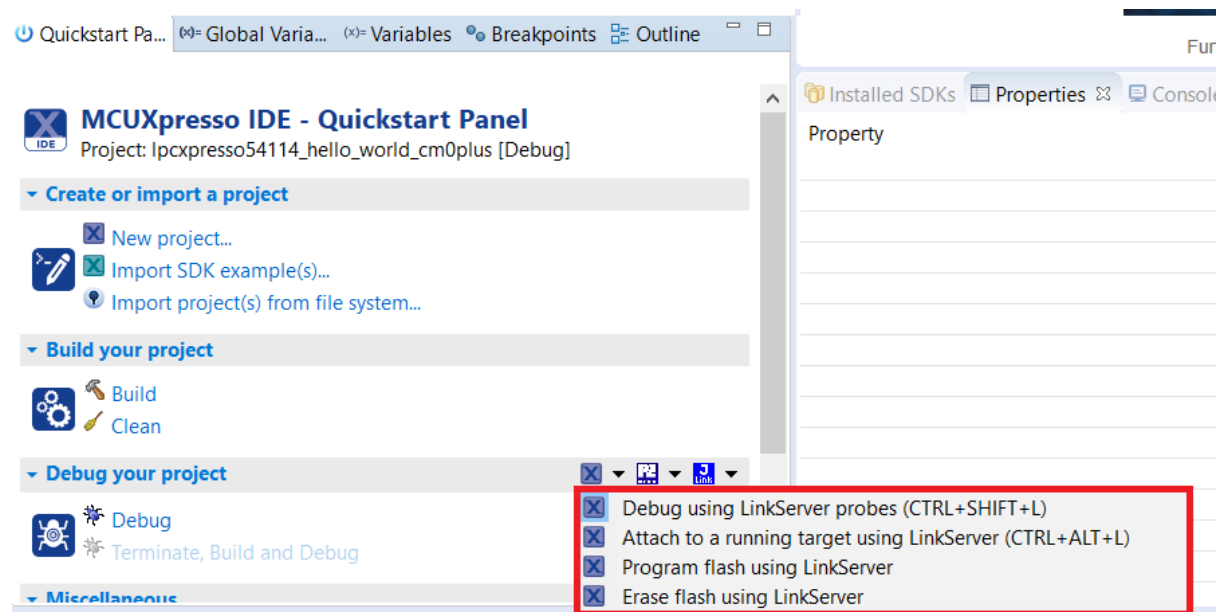


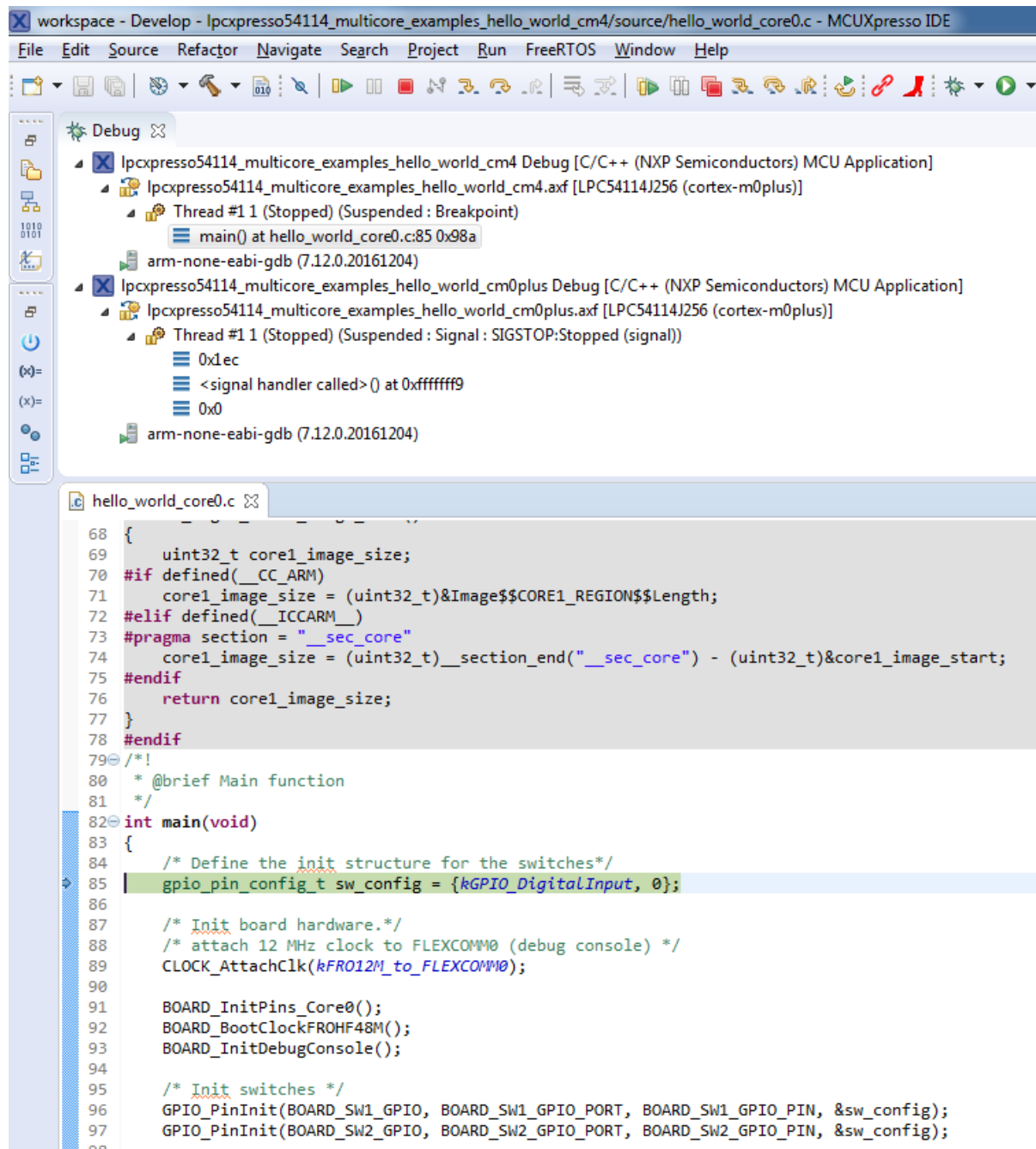
After clicking the “Resume All Debug sessions” button, the hello\_world multicore application runs and a banner is displayed on the terminal. If this is not the case, check your terminal settings and connections.



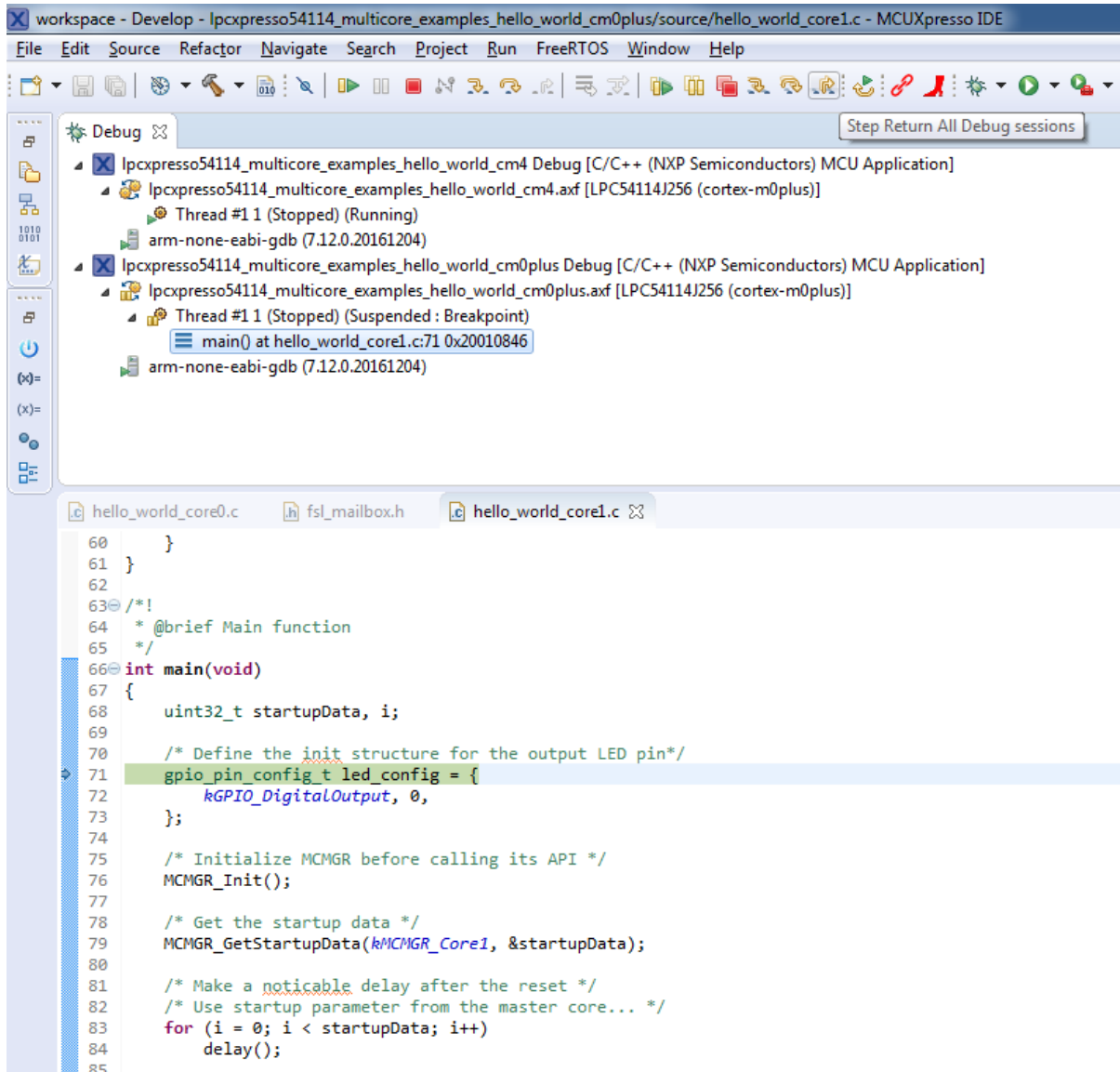
An LED controlled by the auxiliary core starts flashing, indicating that the auxiliary core has been released from the reset and running correctly. It is also possible to debug both sides of the multicore application in parallel. After creating the debug session for the primary core, perform same steps 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.

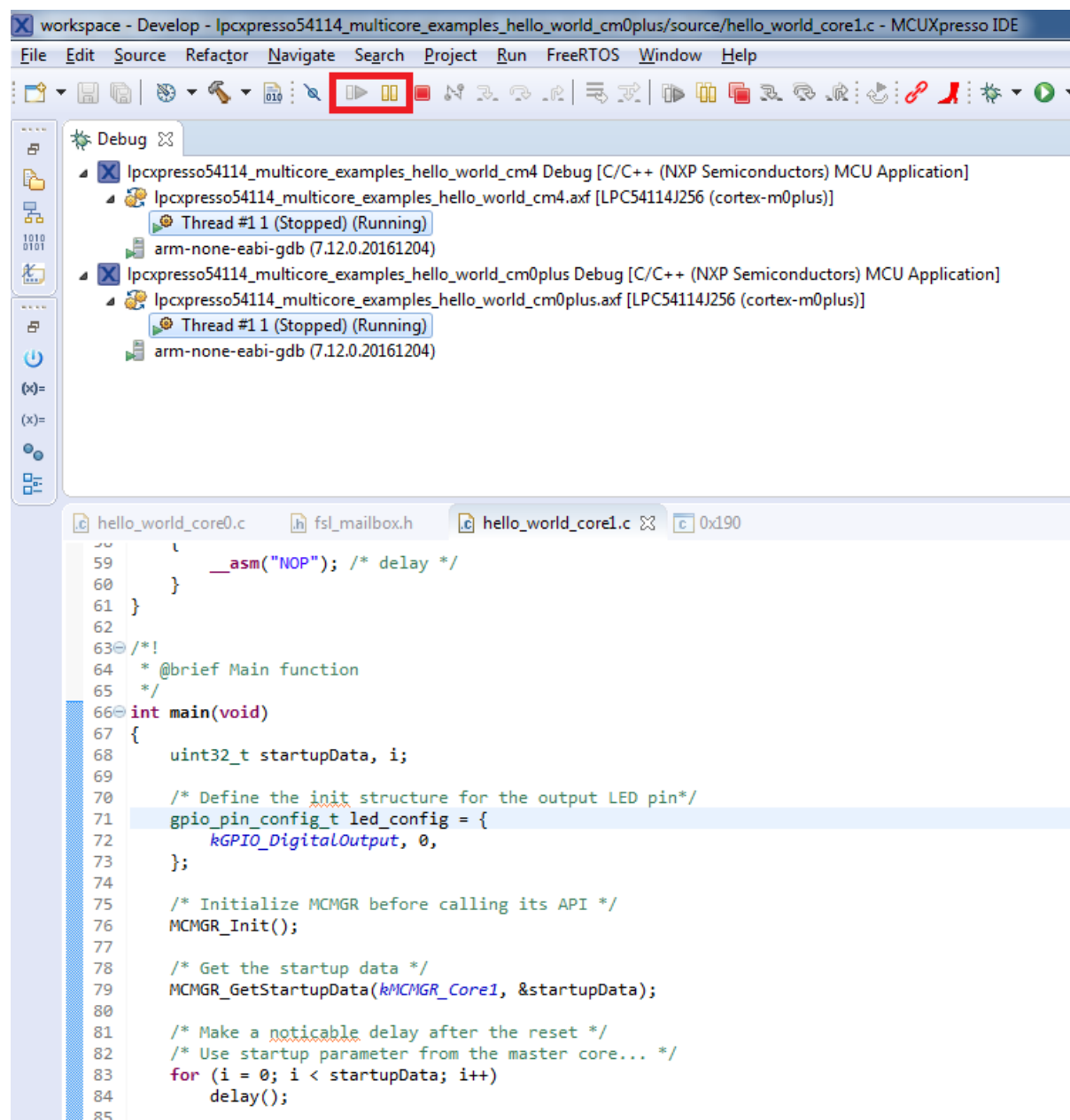


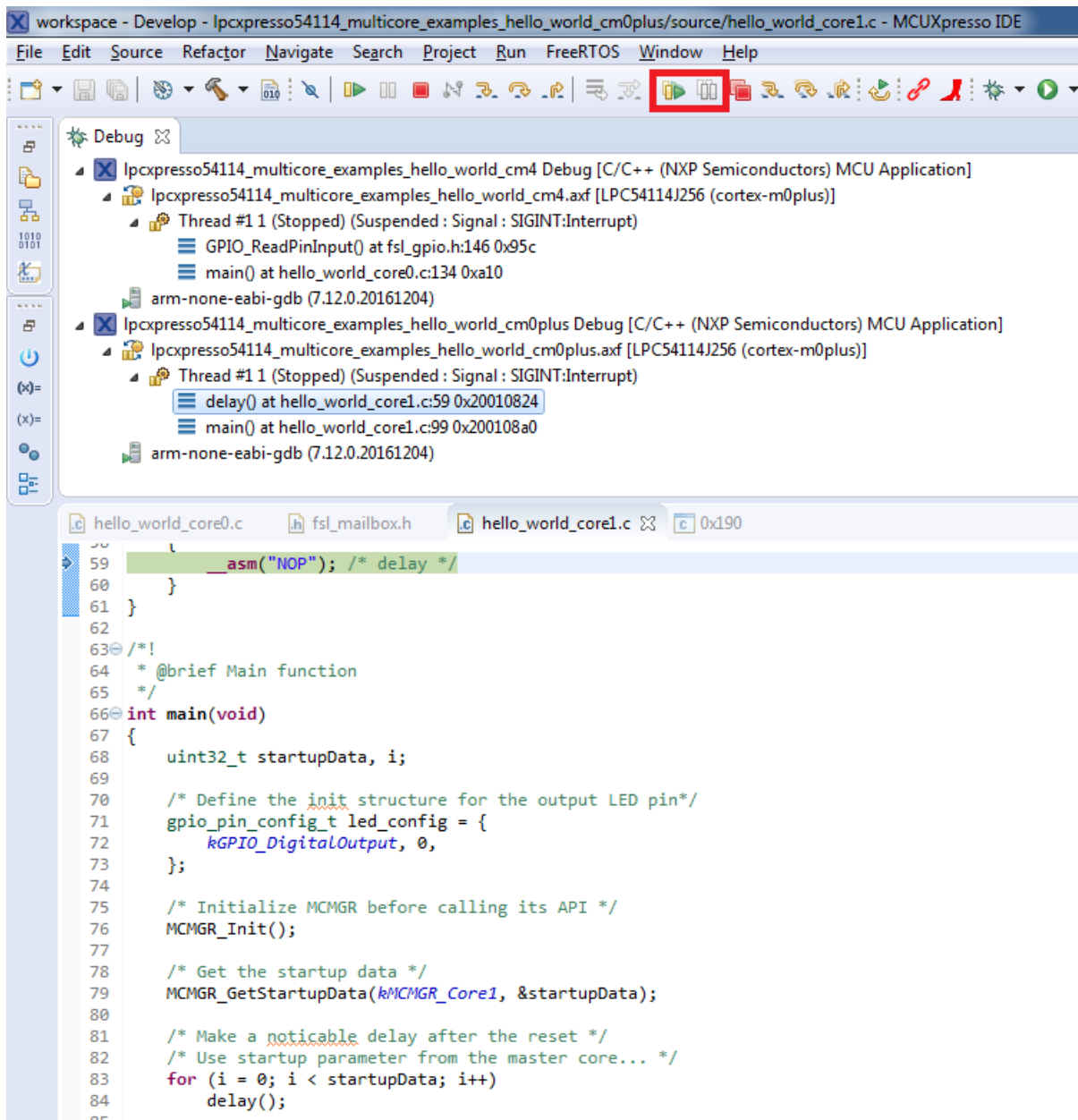


Now, the two debug sessions should be opened, and the debug controls can be used for both debug sessions depending on the debug session selection. Keep the primary core debug session selected by clicking the “Resume” button. The hello\_world multicore application then starts running. The primary core 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.

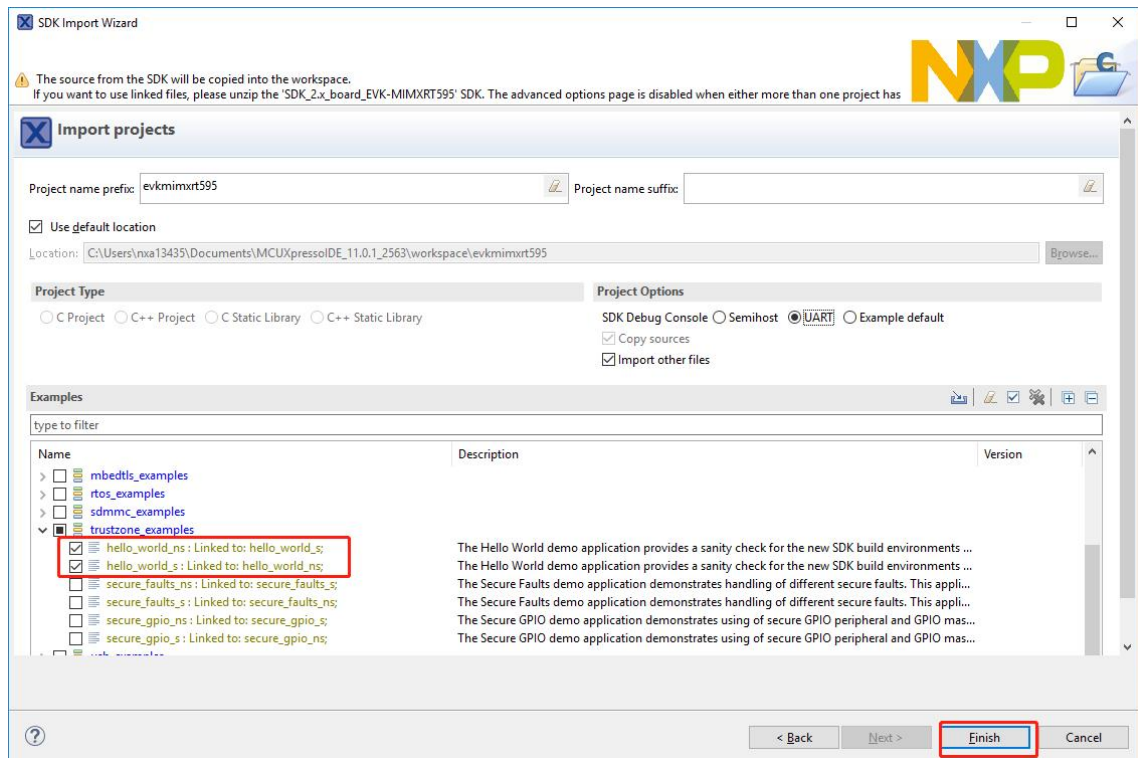




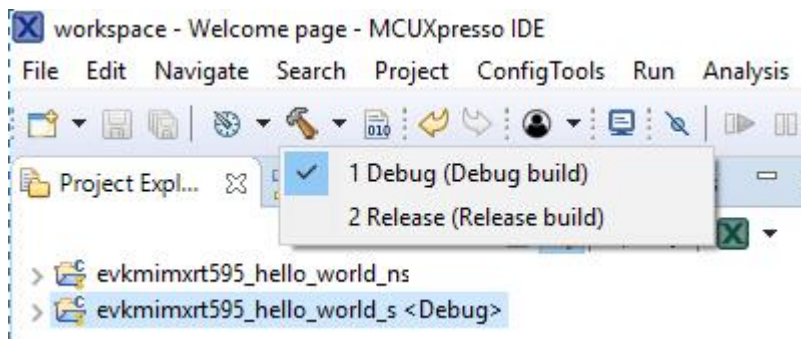
**Build a TrustZone example application** This section describes the steps required to configure MCUXpresso IDE to build, run, and debug TrustZone example applications. The TrustZone version of the `hello_world` example application targeted for the MIMXRT595-EVK hardware platform is used as an example, 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**.





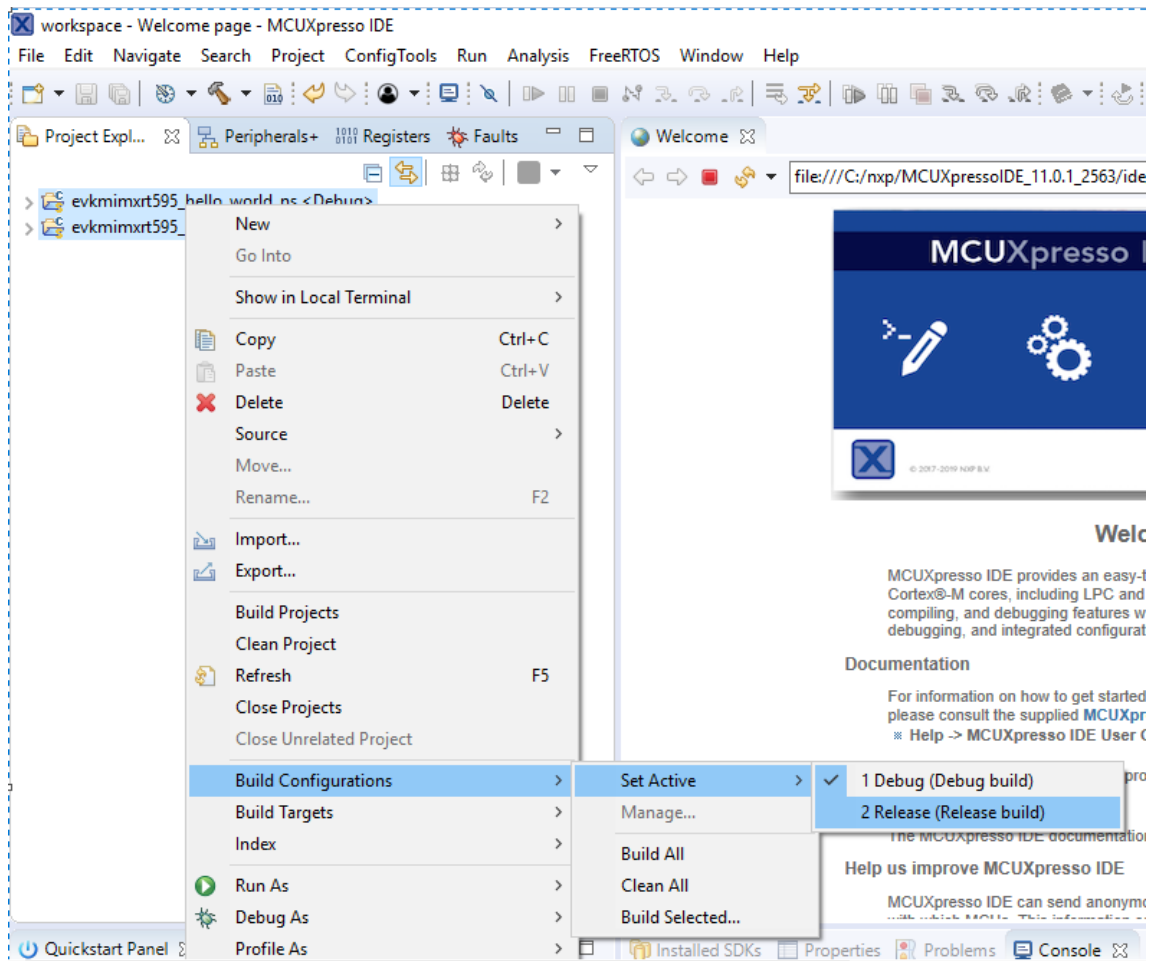
- Now, two projects should be imported into the workspace. To start building the TrustZone application, highlight the `evkmimxrt595_hello_world_s` project (TrustZone master project) in the Project Explorer. Then, choose the appropriate build target, **Debug** or **Release**, by clicking the downward 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.

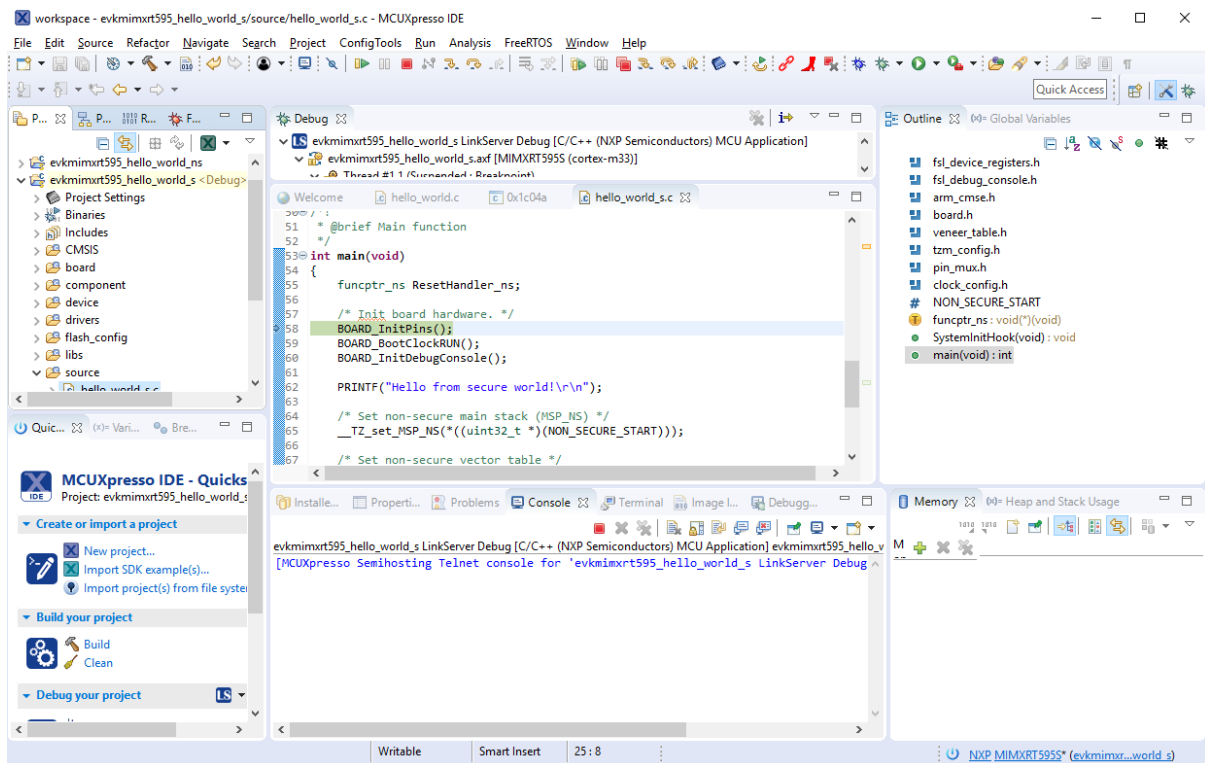
**Note:** When the **Release** build is requested, it is necessary to change the build configuration of both the secure and non-secure application projects first. To do this, select both projects in the Project Explorer view by clicking to select the first project, then using shift-click or control-click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

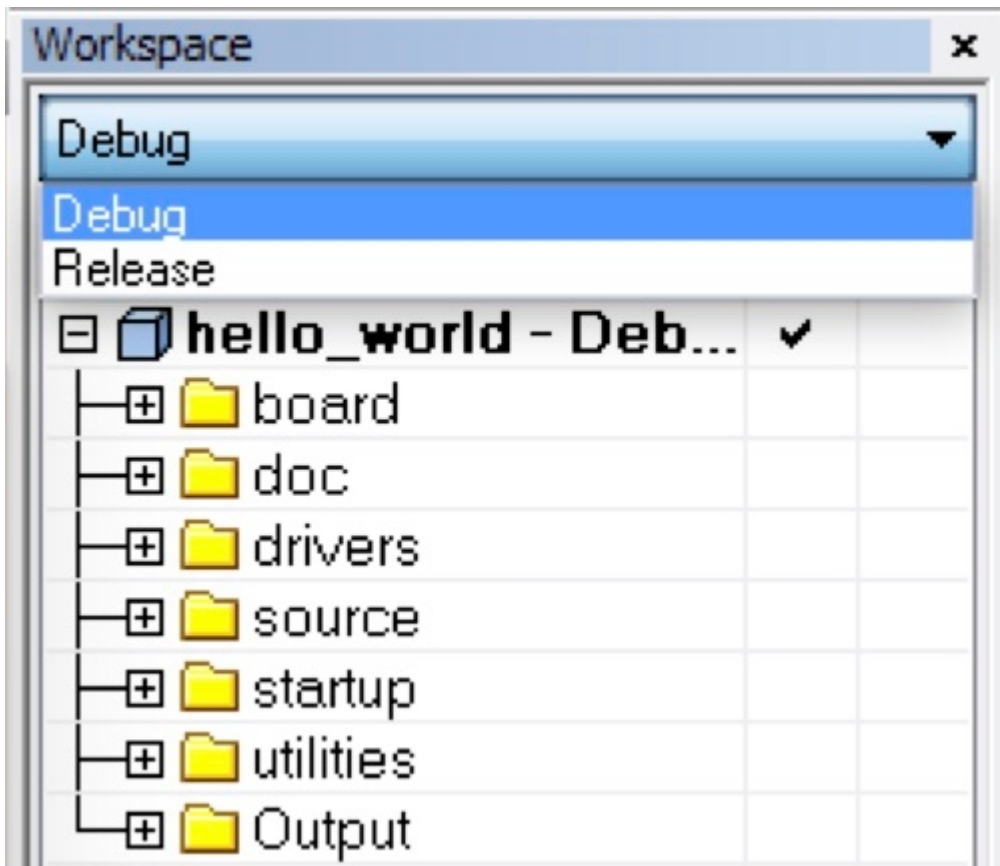
1. Open the desired demo application workspace. Most example application workspace files can be located using the following path:

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

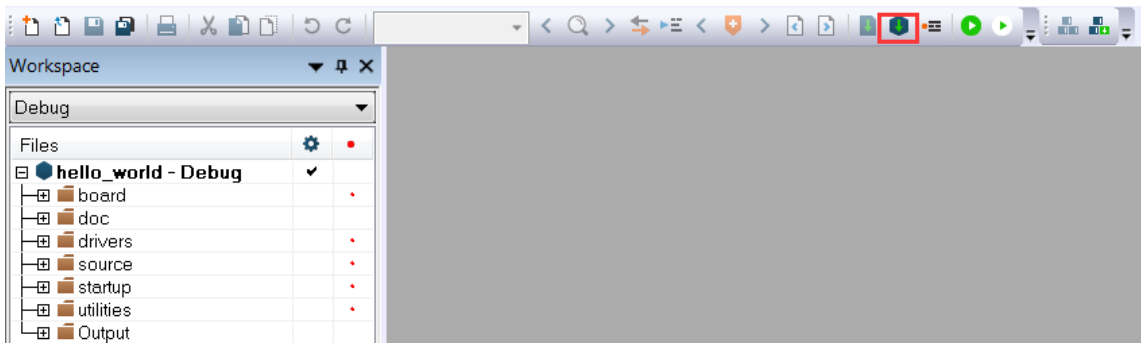
Other example applications may have additional folders in their path.

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

For this example, select **hello\_world – debug**.



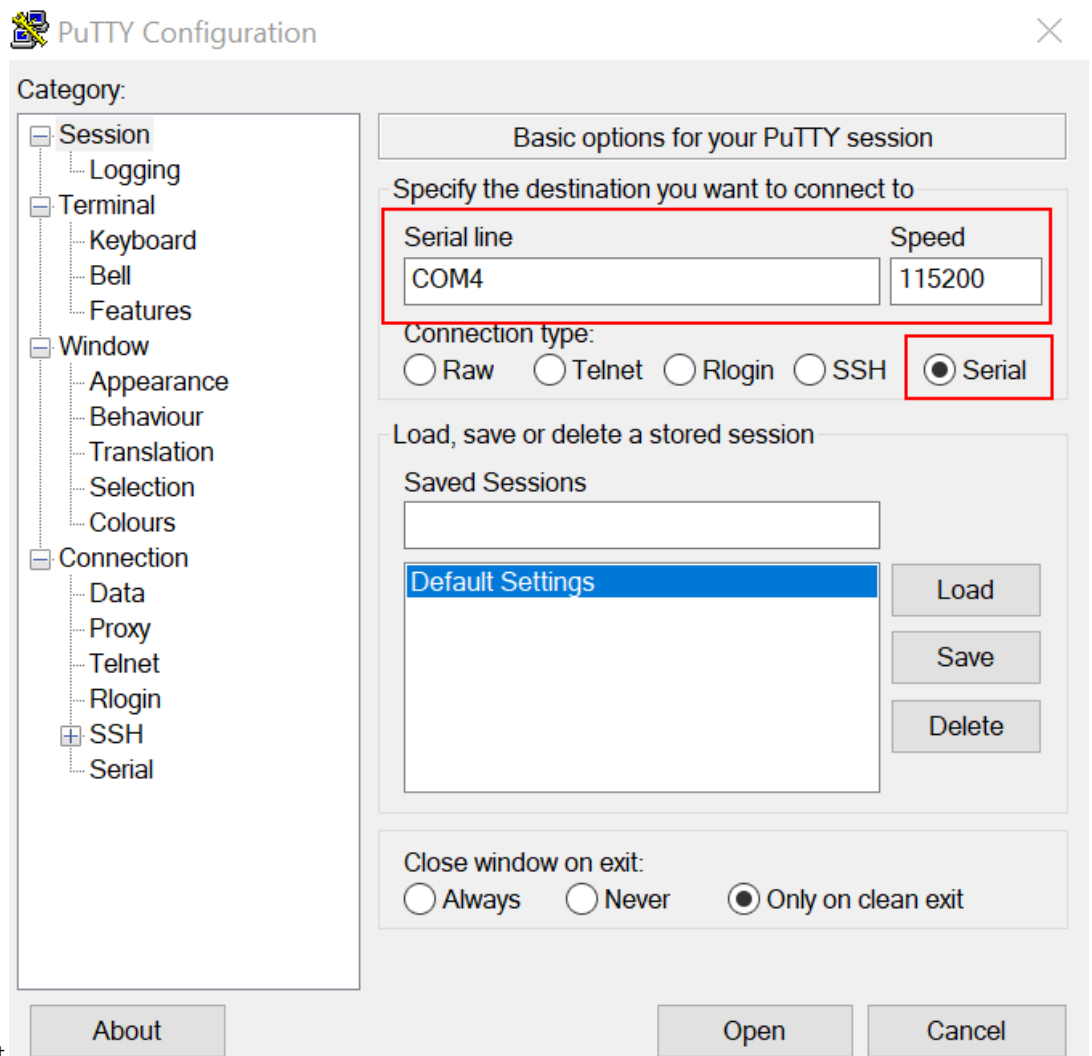
3. To build the demo application, click **Make**, highlighted in red in following figure.



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

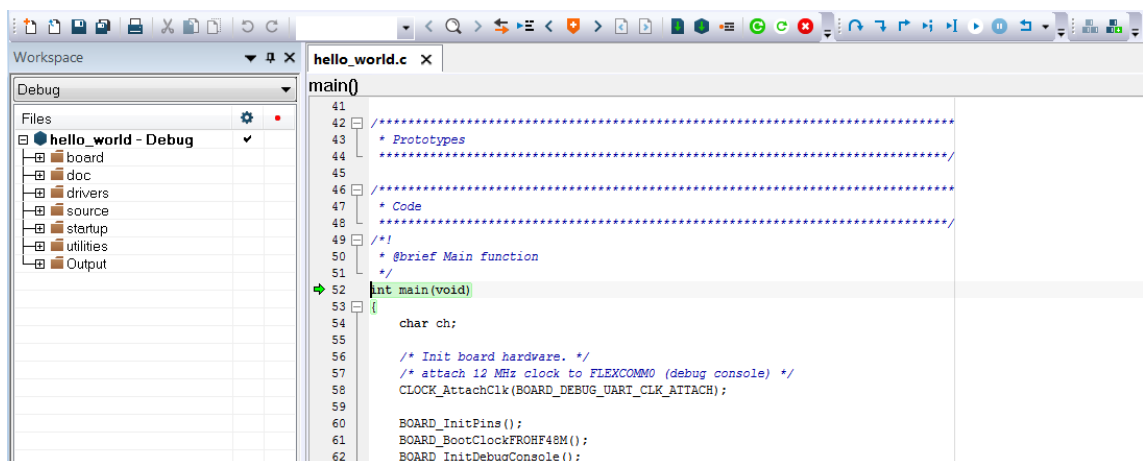


4. 1 stop bit

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.

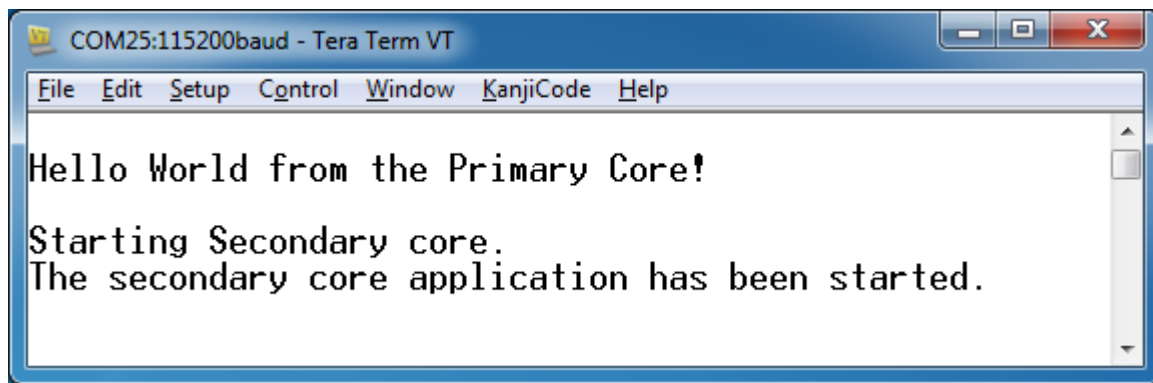
**Run a multicore example application** The primary core debugger handles flashing both primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 4 as described in **Run an 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:

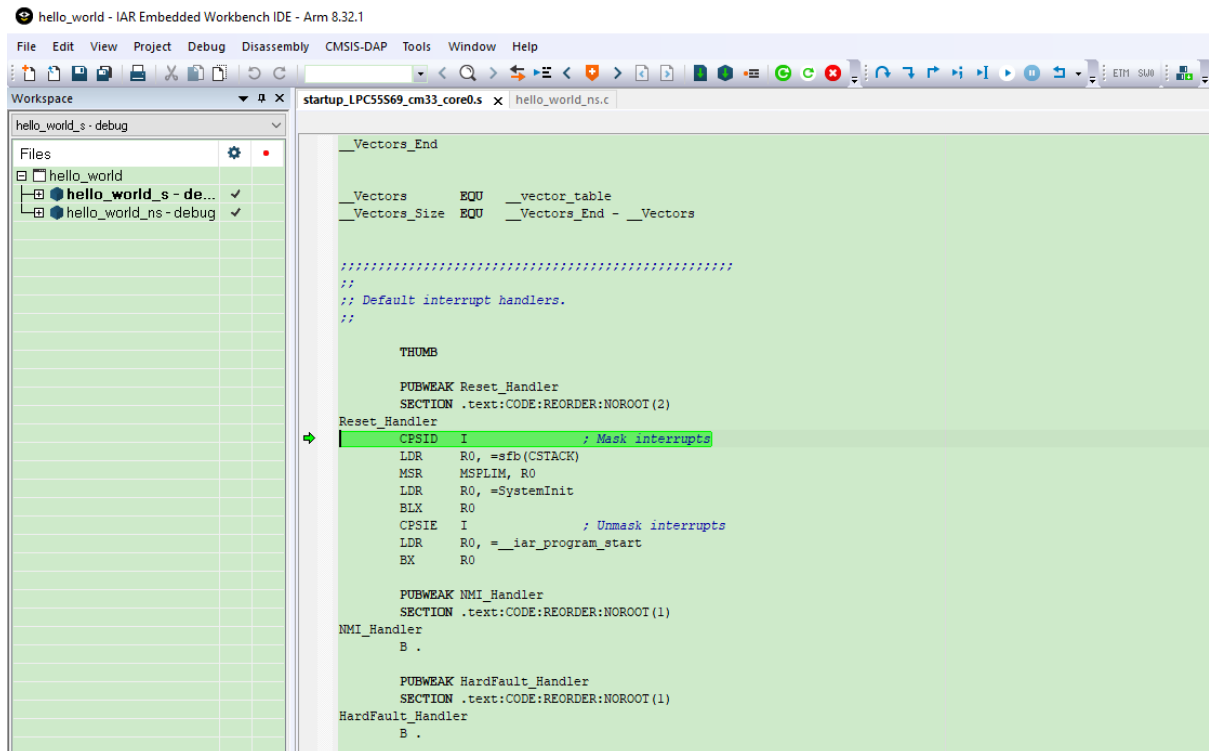
```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_ns/iar/hello_world_  
↪ns.eww
```

```
<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
```

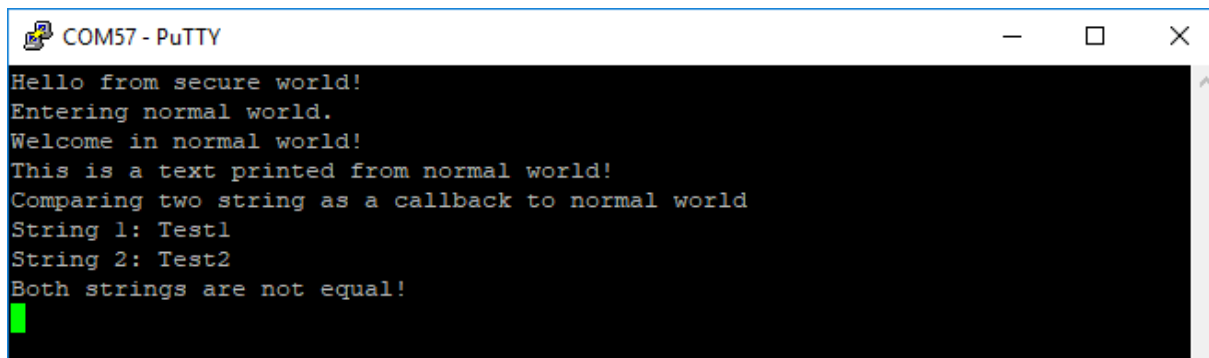
This project `hello_world.eww` contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another. Build both applications separately by clicking **Make**. It is requested to build the application for the secure project first, because 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.



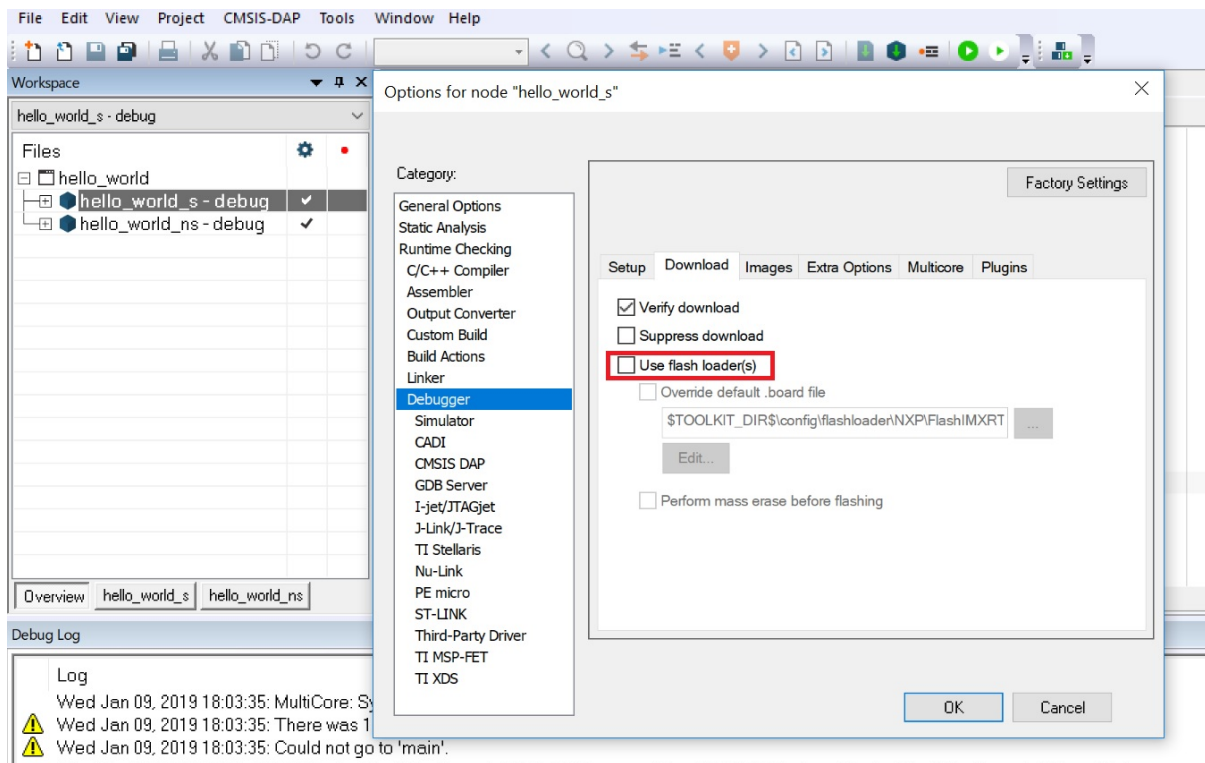
Run the code by clicking **Go** to start the application.

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.



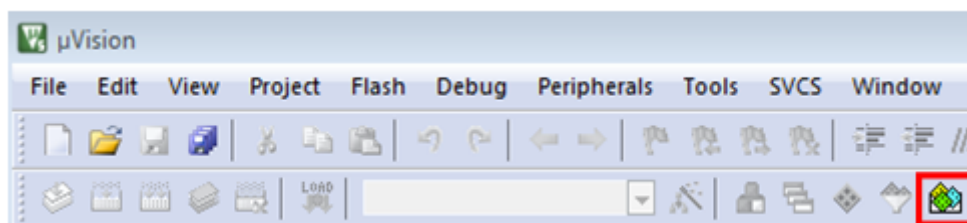


## Run a demo using Keil MDK/μVision

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:



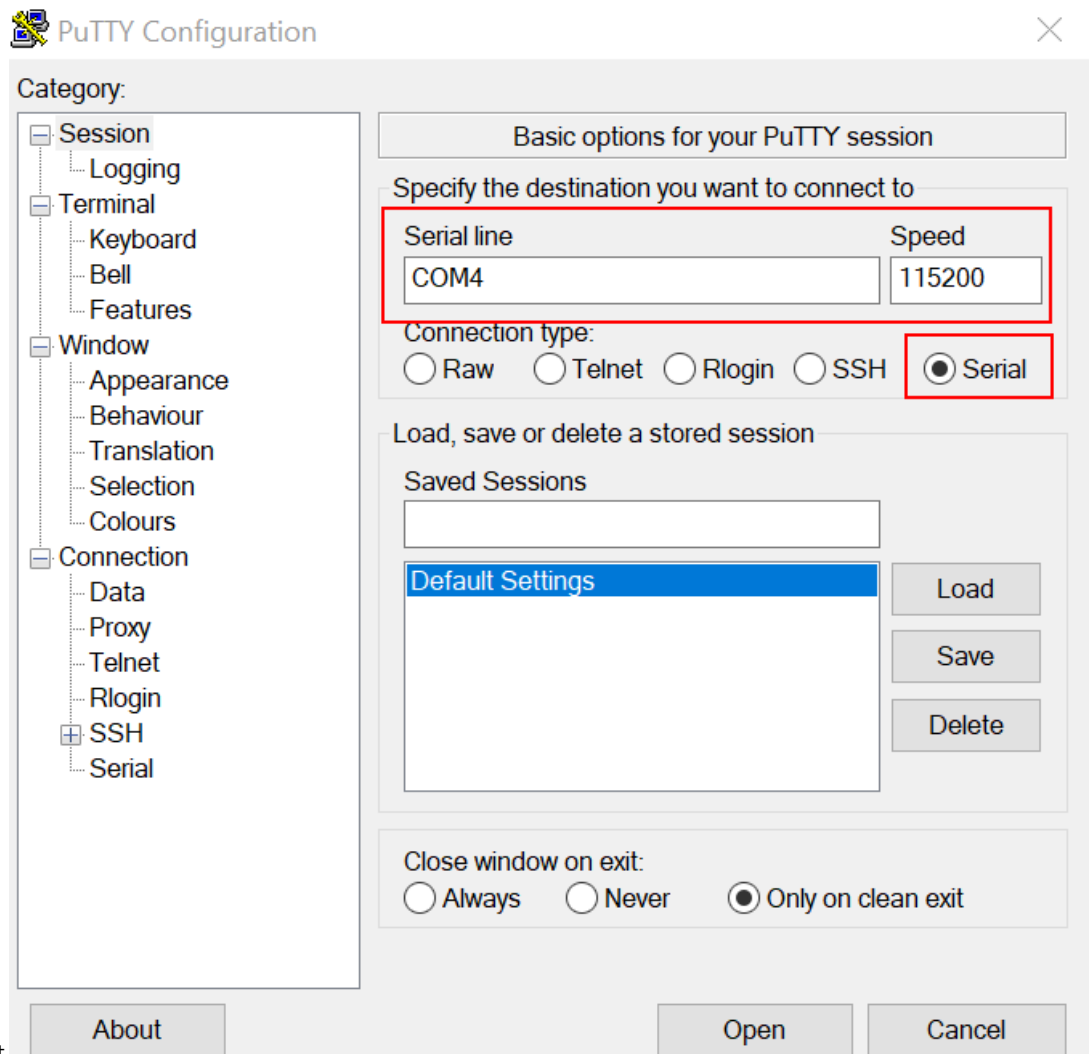
- To build the demo project, select **Rebuild**, highlighted in red.



- The build completes without errors.

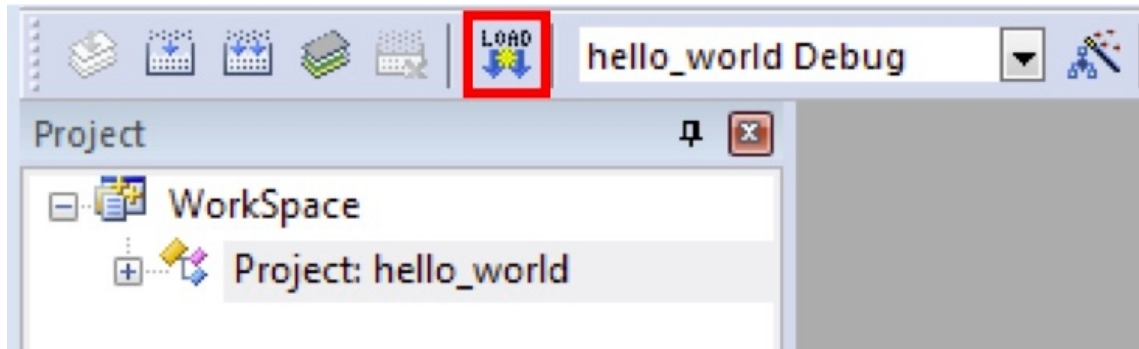
**Run an example application** To download and run the application, perform these steps:

- Ensure the host driver for the debugger firmware has been installed. See [On-board debugger](#).
- Connect the development platform to your PC via USB cable using USB connector.
- Open the terminal application on the PC, such as PuTTY or TeraTerm and connect to the debug serial port number (to determine the COM port number, see [How to determine COM port](#)). Configure the terminal with these settings:
  - 115200 or 9600 baud rate, depending on your board (reference BOARD\_DEBUG\_UART\_BAUDRATE variable in the board.h file)
  - No parity
  - 8 data bits

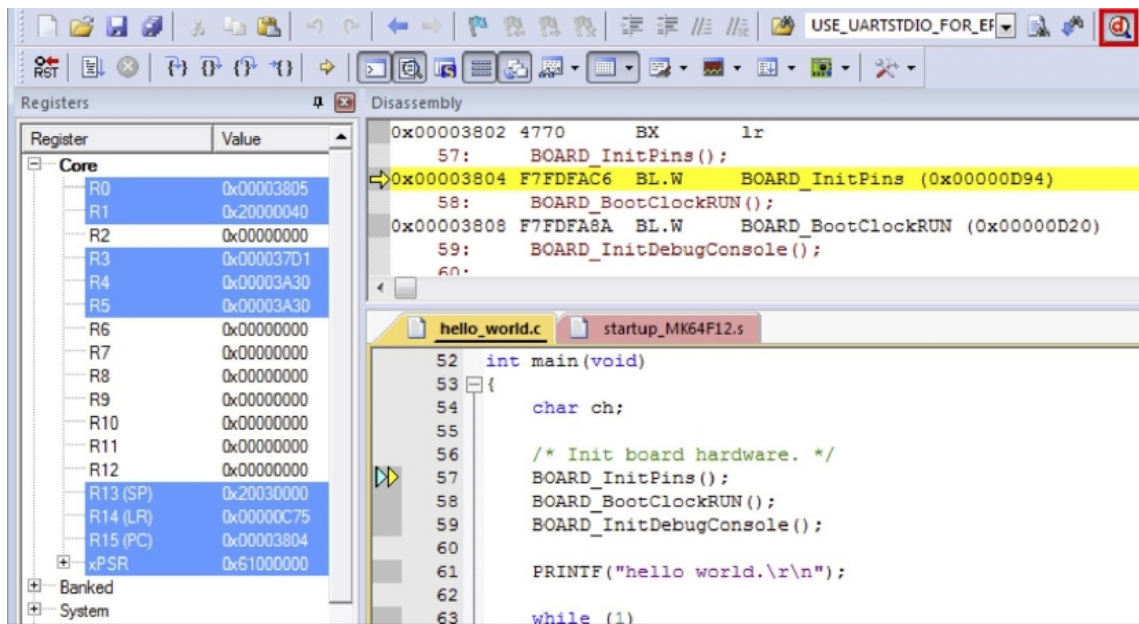


- 1 stop bit

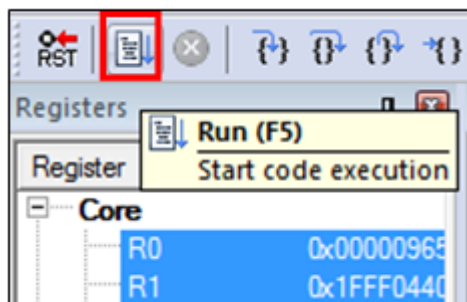
- In  $\mu$ Vision, after the application is built, click the **Download** button to download the application to the target.



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:

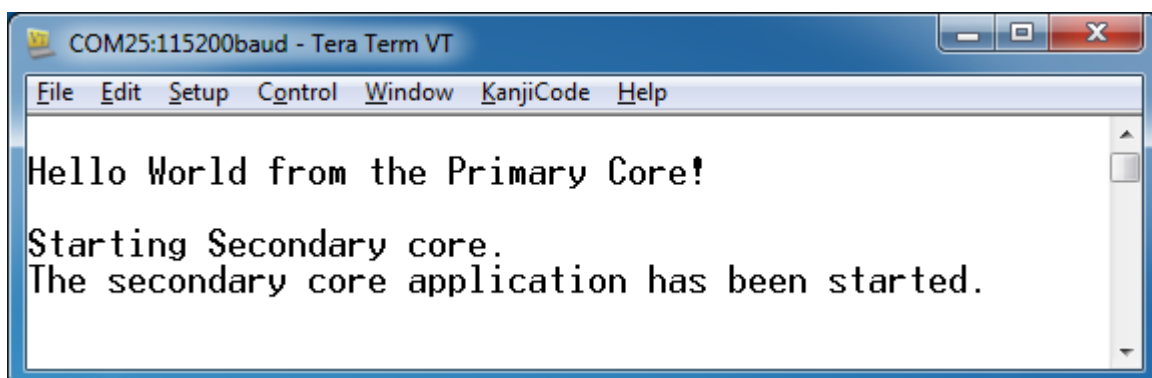
```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm0plus/mdk/hello_world_
→cm0plus.uvmpw
```

```
<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.

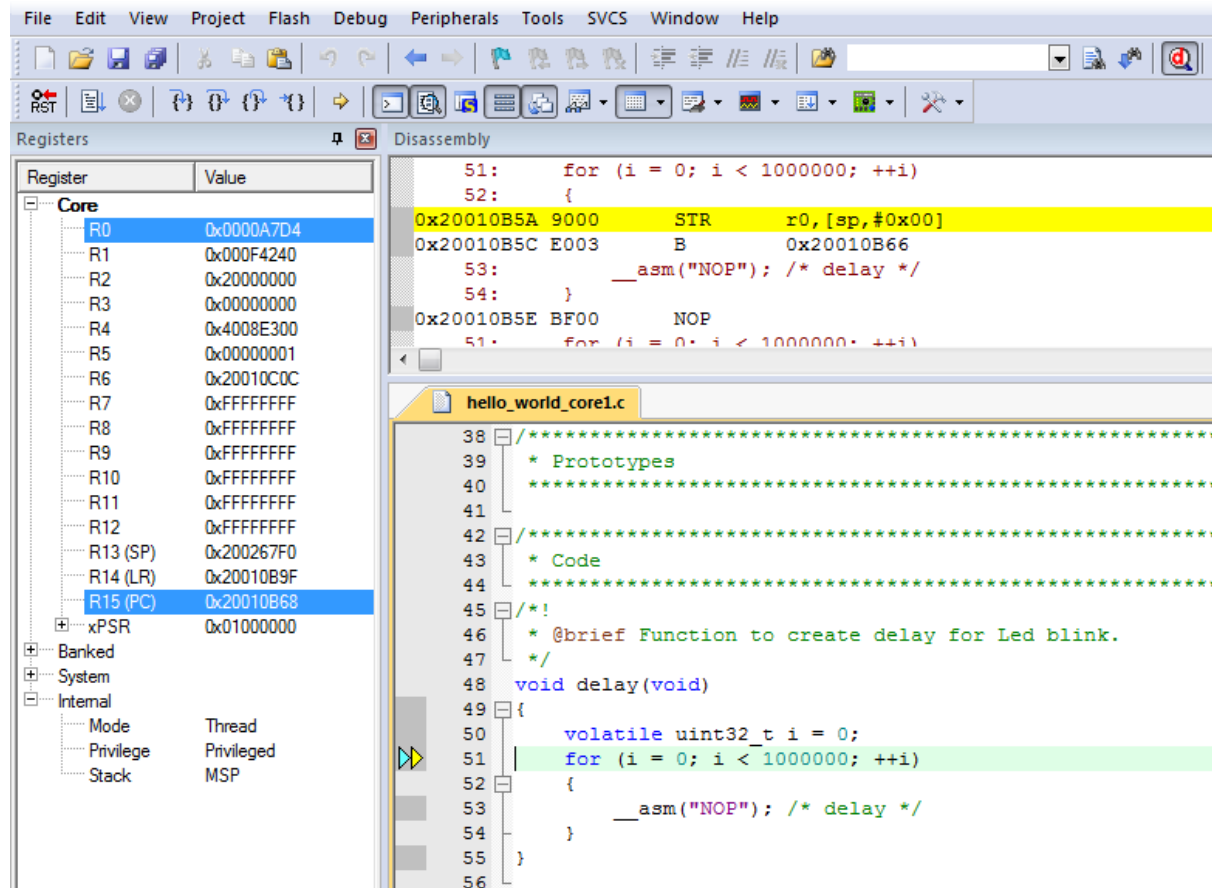
**Run a multicore example application** The primary core debugger flashes both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 5 as described in **Run an example 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  $\mu$ Vision 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](http://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/ $\mu$ Vision 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
```

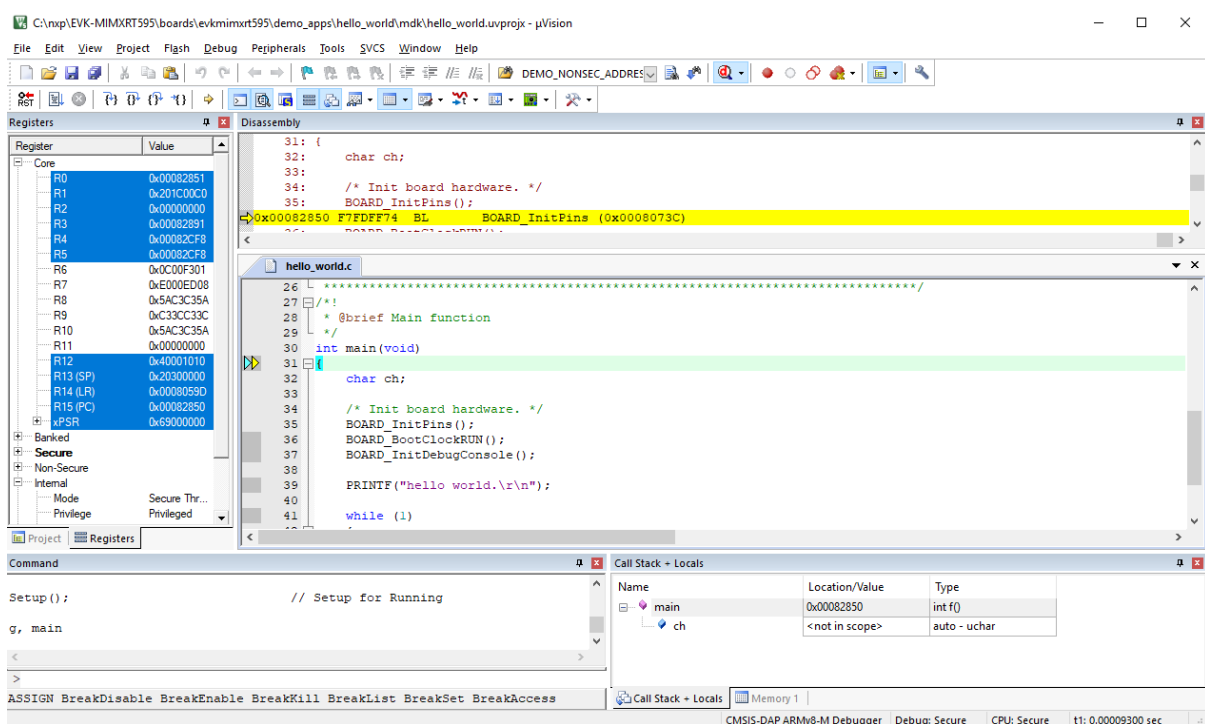
```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_s/mdk/hello_world.  
→ uvmpw
```

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.

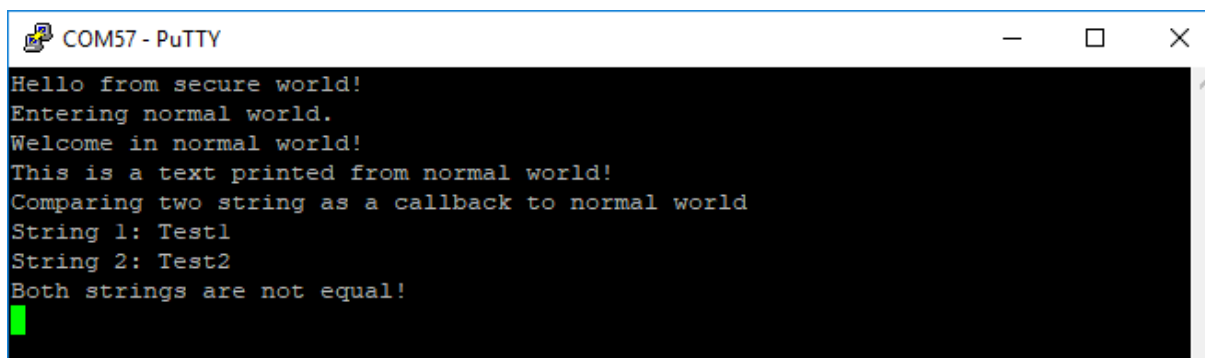
**Run a TrustZone example application** The secure project is configured to download both secure and non-secure output files so debugging can be fully managed from the secure project.

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  $\mu$ Vision. 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 Arm GCC

This section describes the steps to configure the command-line Arm GCC tools to build, run, and debug demo applications and necessary driver libraries provided in the MCUXpresso SDK. The `hello_world` demo application is targeted which is used as an example.

**Set up toolchain** This section contains the steps to install the necessary components required to build and run an MCUXpresso SDK demo application with the Arm GCC toolchain, as supported by the MCUXpresso SDK. There are many ways to use Arm GCC tools, but this example focuses on a Windows operating system environment.

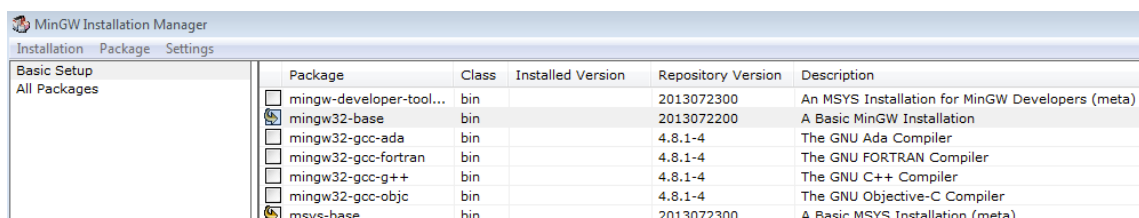
**Install GCC Arm Embedded tool chain** Download and run the installer from GNU Arm Embedded Toolchain. This is the actual toolset (in other words, compiler, linker, and so on). The GCC toolchain should correspond to the latest supported version, as described in **MCUXpresso SDK Release Notes**.

**Install MinGW (only required on Windows OS)** The Minimalist GNU for Windows (MinGW) development tools provide a set of tools that are not dependent on third-party C-Runtime DLLs (such as Cygwin). The build environment used by the MCUXpresso SDK does not use the MinGW build tools, but does leverage the base install of both MinGW and MSYS. MSYS provides a basic shell with a Unix-like interface and tools.

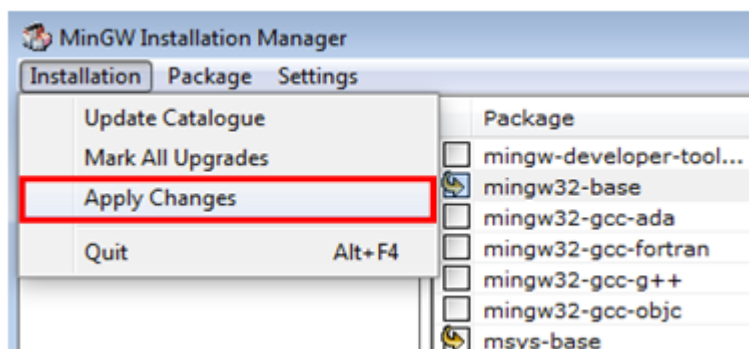
1. Download the latest MinGW mingw-get-setup installer from [MinGW](#).
2. Run the installer. The recommended installation path is `C:\MinGW`, however, you may install to any location.

**Note:** The installation path cannot contain any spaces.

3. Ensure that the **mingw32-base** and **msys-base** are selected under **Basic Setup**.



4. In the **Installation** menu, click **Apply Changes** and follow the remaining instructions to complete the installation.



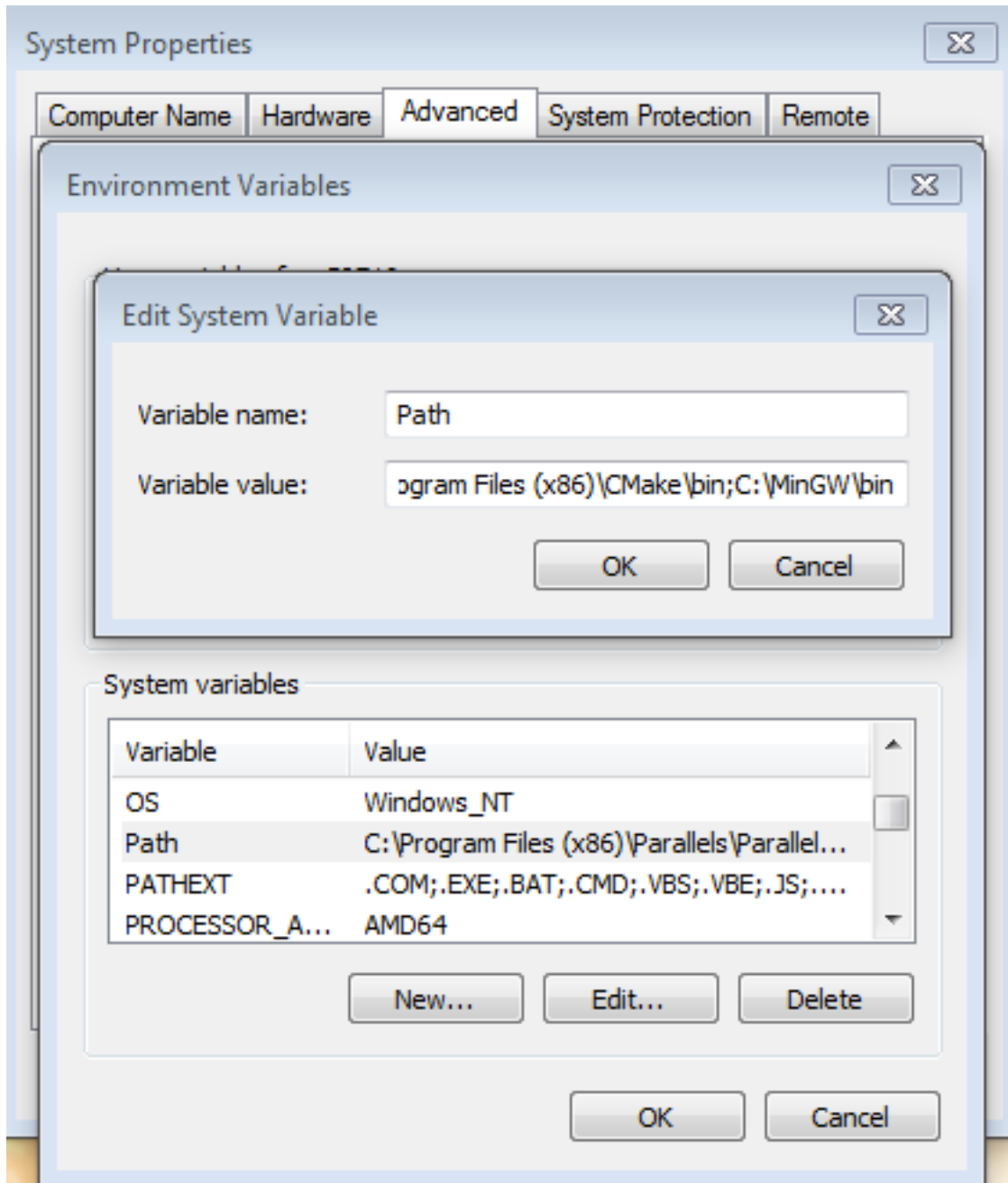
5. Add the appropriate item to the Windows operating system path environment variable. It can be found under **Control Panel->System and Security->System->Advanced System Settings** in the **Environment Variables...** section. The path is:



```
<mingw_install_dir>\bin
```

Assuming the default installation path, C:\MinGW, an example is shown below. If the path is not set correctly, the toolchain will not work.

**Note:** If you have C:\MinGW\msys\x.x\bin in your PATH variable (as required by Kinetis SDK 1.0.0), remove it to ensure that the new GCC build system works correctly.



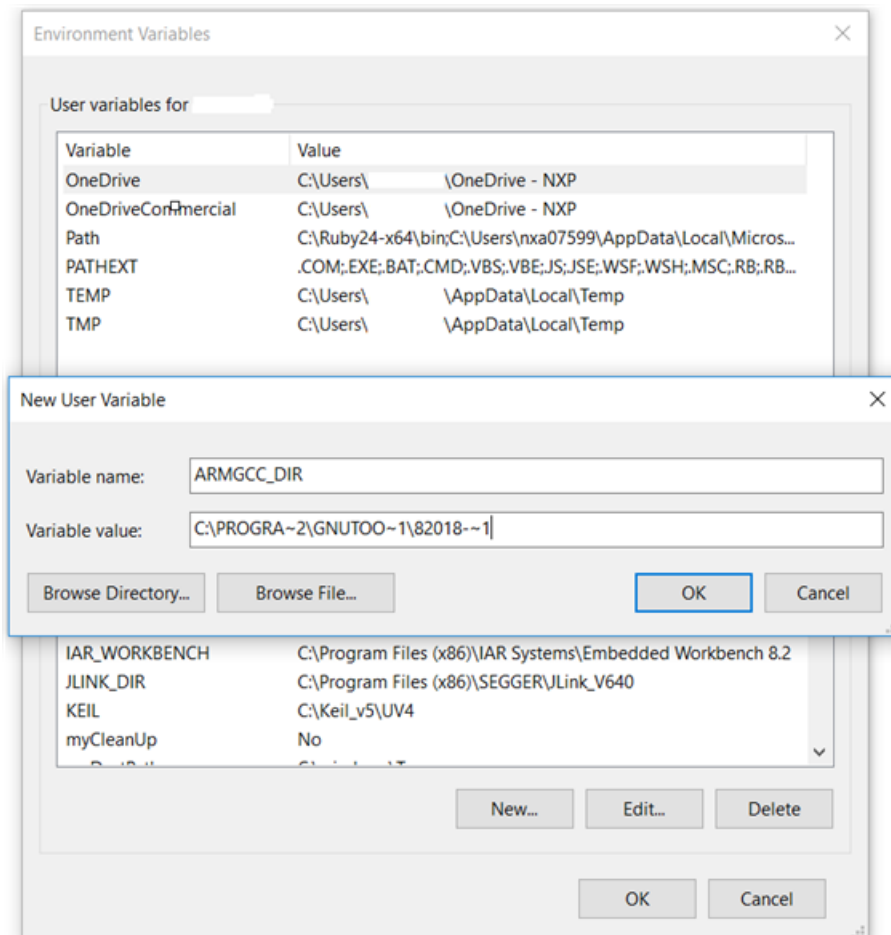
**Add a new system environment variable for ARMGCC\_DIR** Create a new *system* environment variable and name it as ARMGCC\_DIR. The value of this variable should point to the Arm GCC Embedded tool chain installation path. For this example, the path is:

```
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major
```

See the installation folder of the GNU Arm GCC Embedded tools for the exact pathname of your installation.

Short path should be used for path setting, you could convert the path to short path by running command for %I in (.) do echo %~sI

```
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>for %I in (.) do echo %~sI
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>echo C:\PROGRA~2\GNUTOO~1\82018~1
C:\PROGRA~2\GNUTOO~1\82018~1
```

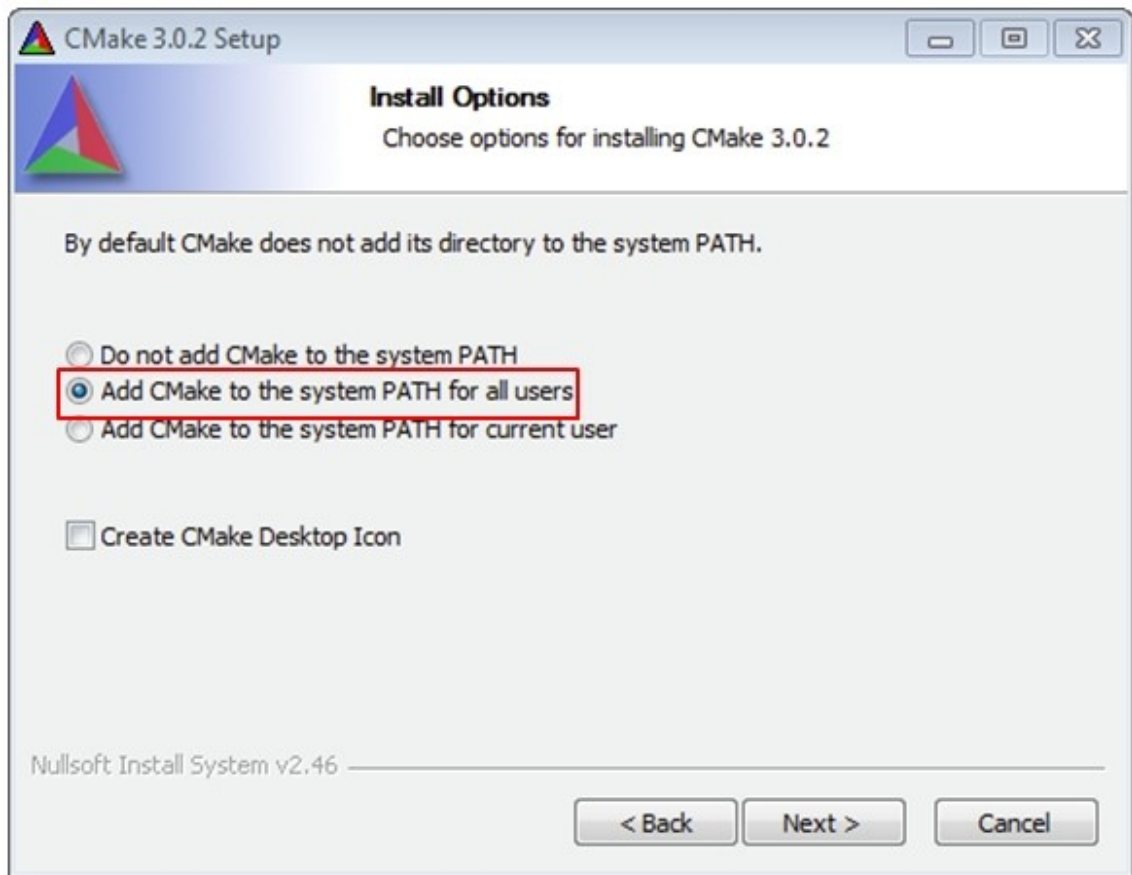


## Install CMake

### Windows OS

1. Download CMake 3.0.x from [www.cmake.org/cmake/resources/software.html](http://www.cmake.org/cmake/resources/software.html).
2. Install CMake, ensuring that the option **Add CMake to system PATH** is selected when installing. The user chooses to select whether it is installed into the PATH for all users or just the current user. In this example, it is installed for all users.





3. Follow the remaining instructions of the installer.
4. You may need to reboot your system for the PATH changes to take effect.
5. Make sure `sh.exe` is not in the Environment Variable PATH. This is a limitation of mingw32-make.

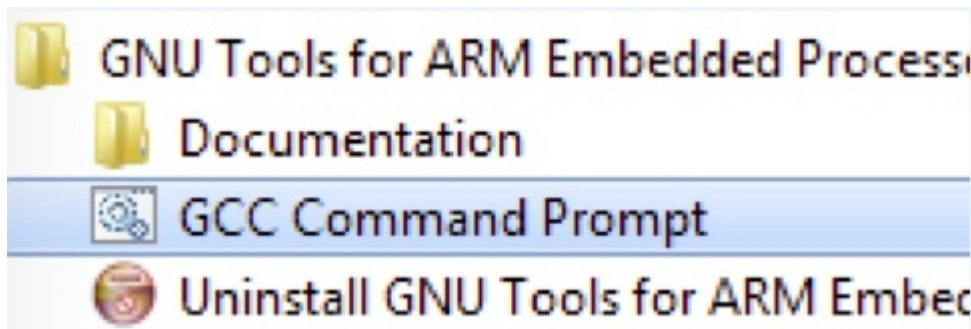
**Linux OS** It depends on the distributions of Linux Operation System. Here we use Ubuntu as an example.

Open shell and use following commands to install cmake and its version. Ensure the cmake version is above 3.0.x.

```
$ sudo apt-get install cmake
$ cmake --version
```

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

1. Open a GCC Arm Embedded tool chain command window. To launch the window, from the Windows operating system **Start** menu, go to **Programs > GNU Tools Arm Embedded <version>** and select **GCC Command Prompt**.



2. Change the directory to the example application project directory which has a path similar to the following:

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

For this example, the exact path is:

**Note:** To change directories, use the `cd` command.

3. Type **build\_debug.bat** on the command line or double click on **build\_debug.bat** file in Windows Explorer to build it. The output is as shown in following figure.

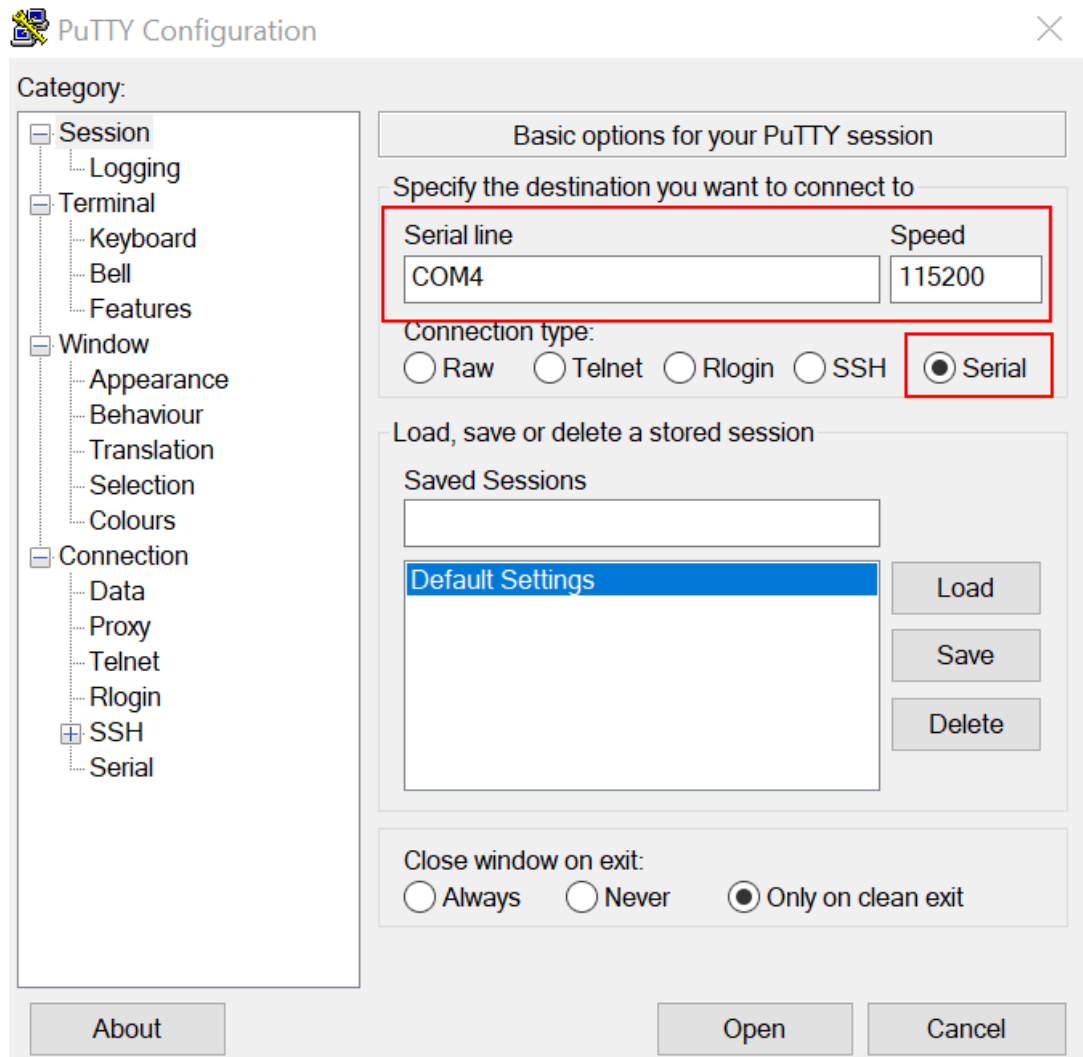
```
[ 84%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.0_FRDM-K64F
/devices/MK64F12/drivers/fsl_smc.c.obj
[ 92%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.0_FRDM-K64F
/devices/MK64F12/drivers/fsl_clock.c.obj
[100%] Linking C executable debug\hello_world.elf
[100%] Built target hello_world.elf

C:\nxp\SDK_2.0_FRDM-K64F\boards\frdmk64f\demo_apps\hello_world\armgcc>IF "" == "
" <pause >
Press any key to continue . . .
```

**Run an example application** This section describes steps to run a demo application using J-Link GDB Server application. To install J-Link host driver and update the on-board debugger firmware to Jlink firmware, see [On-board debugger](#).

After the J-Link interface is configured and connected, follow these steps to download and run the demo applications:

1. Connect the development platform to your PC via USB cable between the on-board debugger USB connector and the PC USB connector. If using a standalone J-Link debug pod, connect it to the SWD/JTAG connector of the board.
2. 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

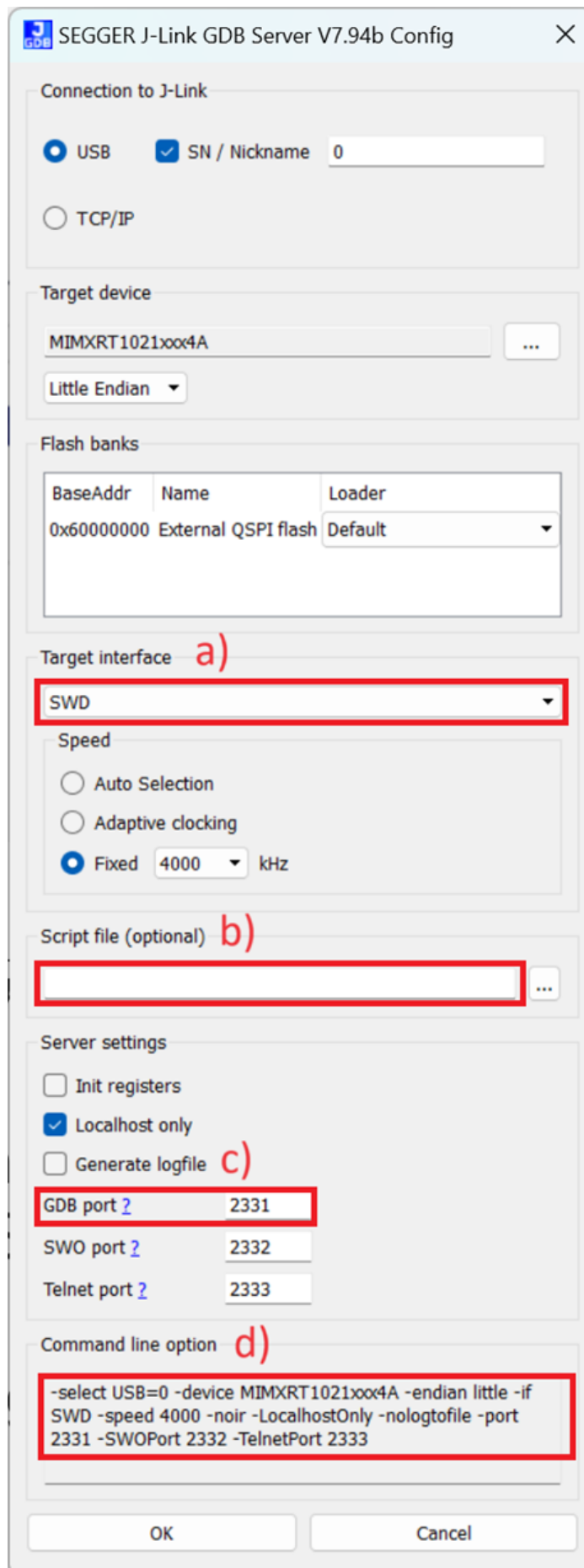


3. To launch the application, open the Windows **Start** menu and select **Programs > SEGGER > J-Link <version> J-Link GDB Server**.

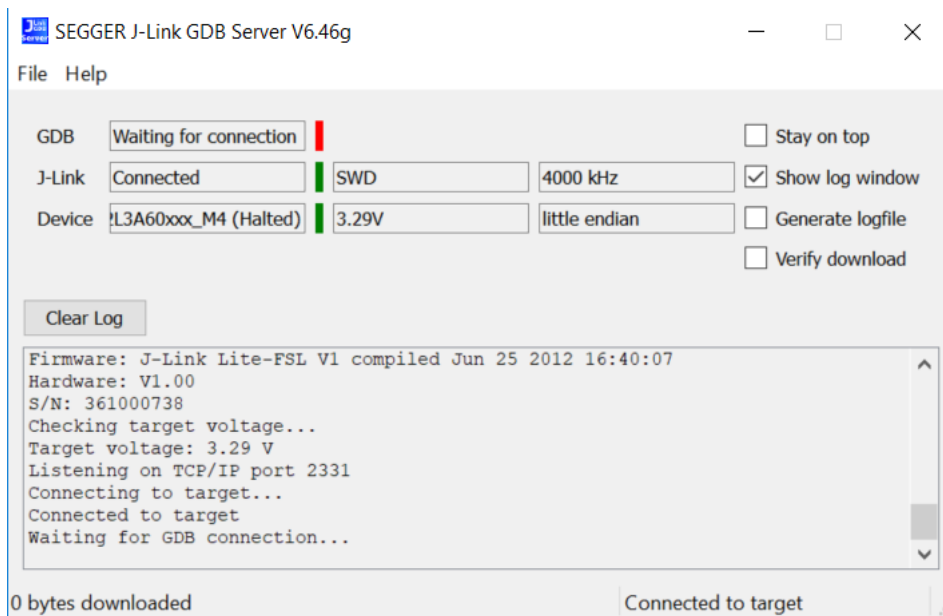
**Note:** It is assumed that the J-Link software is already installed.

The **SEGGER J-Link GDB Server Config** settings dialog appears.

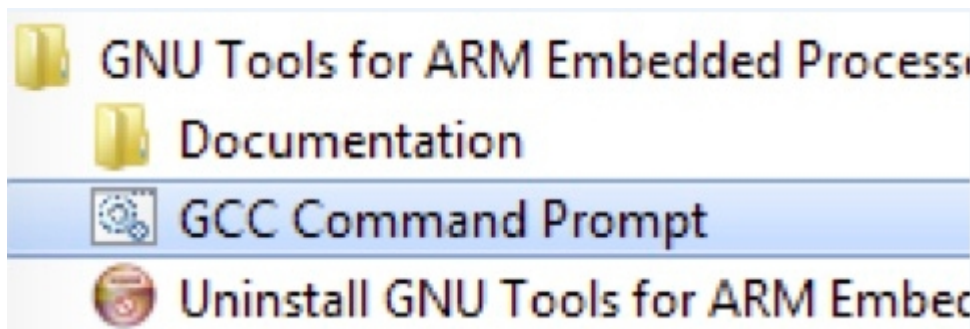
4. Make sure to check the following options.
  1. **Target interface:** The debug connection on board uses internal SWD signaling. In case of a wrong setting J-Link is unable to communicate with device under test.
  2. **Script file:** If required, a J-Link init script file can be used for board initialization. The file with the ".jlinkscript" file extension is located in the <install\_dir>/boards/<board\_name>/ directory.
  3. Under the **Server settings**, check the GDB port for connection with the gdb target remote command. For more information, see step 9.
  4. There is a command line version of J-Link GDB server "JLinkGDBServerCL.exe". Typical path is C:\Program Files\SEGGER\JLink\. To start the J-Link GDB server with the same settings as are selected in the UI, you can use these command line options.



5. After it is connected, the screen should look like this figure:



6. If not already running, open a GCC Arm Embedded tool chain command window. To launch the window, from the Windows operating system Start menu, go to **Programs - GNU Tools Arm Embedded <version>** and select **GCC Command Prompt**.



7. Change to the directory that contains the example application output. The output can be found in using one of these paths, depending on the build target selected:

```
<install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc/debug
```

```
<install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc/release
```

8. Run the `arm-none-eabi-gdb.exe <application_name>.elf` command. For this example, it is `arm-none-eabi-gdb.exe hello_world.elf`.

```

C:\Program Files (x86)\GNU Tools ARM Embedded\8 2018-q4-major>arm-none-eabi-gdb.exe C:\Users\nxa12829\Desktop\k3213\boards\frdmk3213a6\demo_apps\hello_world\cm4\armgcc\debug\hello_world_demo_cm4.elf
C:\Program Files (x86)\GNU Tools ARM Embedded\8 2018-q4-major\bin>arm-none-eabi-gdb.exe: warning: Couldn't determine a path for the index cache directory.
GNU gdb (GNU Tools for Arm Embedded Processors 8-2018-q4-major) 8.2.50.20181213-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from C:\Users\nxa12829\Desktop\k3213\boards\frdmk3213a6\demo_apps\hello_world\cm4\armgcc\debug\hello_world_demo_cm4.elf...
(gdb)

```

9. Run these commands:

1. target remote localhost:2331
2. monitor reset
3. monitor halt
4. load
5. monitor reset

10. The application is now downloaded and halted. Execute the `monitor go` command to start the demo 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.

```

COM4 - PuTTY
hello world.

```

**Build a multicore example application** This section describes the steps to build and run a dual-core application. The demo application build scripts are located in this folder:

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

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

```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm0plus/armgcc/build_debug.bat
```

```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm4/armgcc/build_debug.bat
```

Build both applications separately following steps for single core examples as described in **Build an example application**.

```

GCC Command Prompt - build_debug.bat

[ 47%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/drivers/fsl_common.c.obj
[ 52%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/drivers/fsl_msmc.c.obj
[ 56%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/utilities/debug_console/fsl_debug_console.c.obj
[ 60%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/utilities/fsl_assert.c.obj
[ 65%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/utilities/str/fsl_str.c.obj
[ 69%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/uart/lpuart_adapter.c.obj
[ 73%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/serial_manager/serial_manager.c.obj
[ 78%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/serial_manager/serial_port_uart.c.obj
[ 82%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/lists/generic_list.c.obj
[ 86%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/system_K32L3A60_cm0plus.c.obj
[ 91%] Building ASM object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/gcc/startup_K32L3A60_cm0plus.S.obj
[ 95%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/middleware/multicore/mcmgr/src/mcmgr.c.obj
[100%] Linking C executable debug\hello_world_cm0plus.elf
[100%] Built target hello_world_cm0plus.elf

c:\packages\SDK_2.6.0_FRDM-K32L3A6_RC1\boards\frdmk32l3a6\multicore_examples\hello_world\cm0plus\armgcc>IF "" == "" (pause)
Press any key to continue . . .

GCC Command Prompt - build_debug.bat

[ 50%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/drivers/fsl_lpuart.c.obj
[ 54%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/drivers/fsl_common.c.obj
[ 58%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/drivers/fsl_msmc.c.obj
[ 62%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/utilities/str/fsl_str.c.obj
[ 66%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/utilities/fsl_assert.c.obj
[ 70%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/utilities/debug_console/fsl_debug_console.c.obj
[ 75%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/uart/lpuart_adapter.c.obj
[ 79%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/serial_manager/serial_port_uart.c.obj
[ 83%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/serial_manager/serial_manager.c.obj
[ 87%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/lists/generic_list.c.obj
[ 91%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/system_K32L3A60_cm4.c.obj
[ 95%] Building ASM object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/gcc/startup_K32L3A60_cm4.S.obj
[100%] Linking C executable debug\hello_world_cm4.elf
[100%] Built target hello_world_cm4.elf

c:\packages\SDK_2.6.0_FRDM-K32L3A6_RC1\boards\frdmk32l3a6\multicore_examples\hello_world\cm4\armgcc>IF "" == "" (pause)
Press any key to continue . . .
  
```

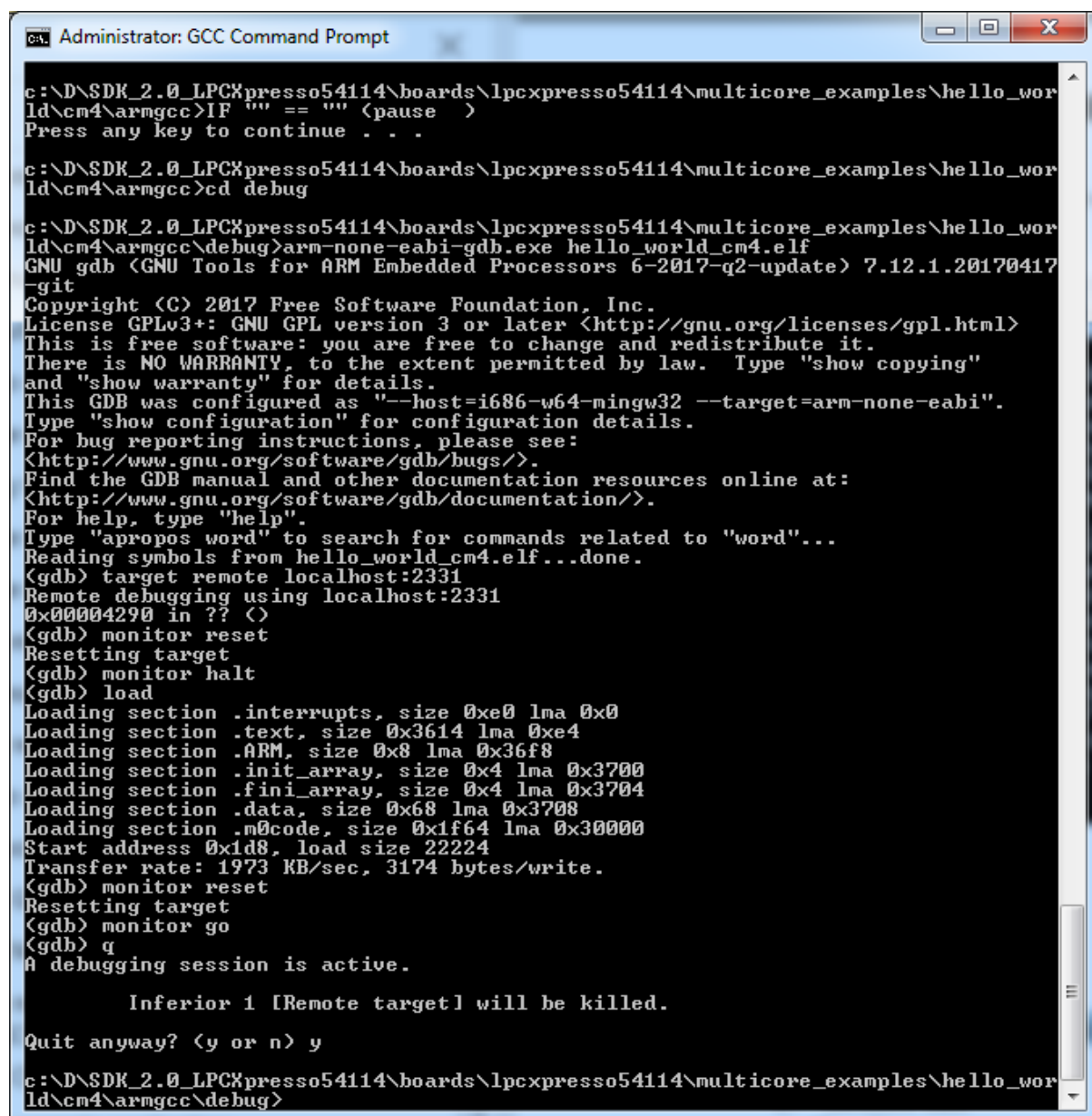
**Run a multicore example application** When running a multicore application, the same prerequisites for J-Link/J-Link OpenSDA firmware, and the serial console as for the single-core application, applies, as described in **Run an example application**.

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 steps 1 to 10, as described in **Run an example application**. These steps are common for both single-core and dual-core applications in Arm GCC.

Both the primary and the auxiliary image is loaded into the SPI flash memory. After execution of the monitor go command, the primary core application is executed. During the primary core code execution, the auxiliary core code is reallocated from the flash memory to the RAM, and 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 is not true, check your terminal settings and connections.



```

Administrator: GCC Command Prompt

c:\D\SDK_2.0_LPCXpresso54114\boards\lpcxpresso54114\multicore_examples\hello_world_cm4\armgcc>IF "" == "" <pause >
Press any key to continue . . .

c:\D\SDK_2.0_LPCXpresso54114\boards\lpcxpresso54114\multicore_examples\hello_world_cm4\armgcc>cd debug

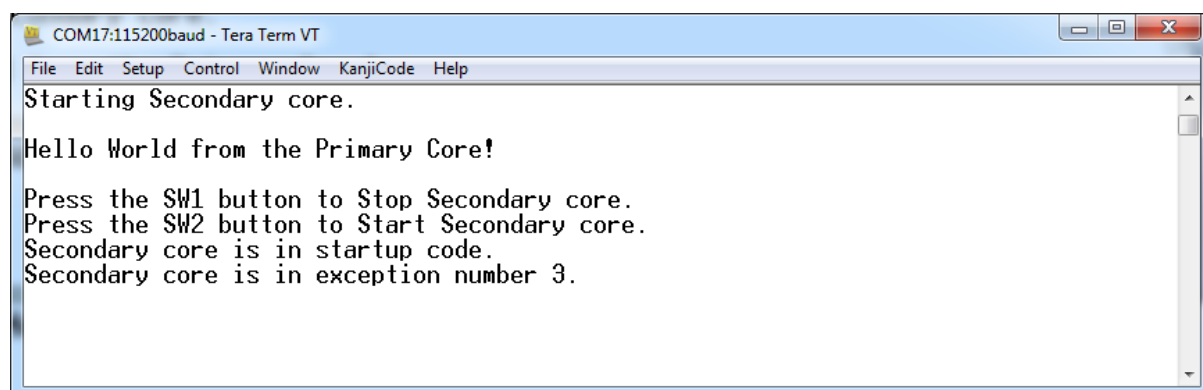
c:\D\SDK_2.0_LPCXpresso54114\boards\lpcxpresso54114\multicore_examples\hello_world_cm4\armgcc\debug>arm-none-eabi-gdb.exe hello_world_cm4.elf
GNU gdb (GNU Tools for ARM Embedded Processors 6-2017-q2-update) 7.12.1.20170417-git
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hello_world_cm4.elf...done.
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
0x00004290 in ?? ()
(gdb) monitor reset
Resetting target
(gdb) monitor halt
(gdb) load
Loading section .interrupts, size 0xe0 lma 0x0
Loading section .text, size 0x3614 lma 0xe4
Loading section .ARM, size 0x8 lma 0x36f8
Loading section .init_array, size 0x4 lma 0x3700
Loading section .fini_array, size 0x4 lma 0x3704
Loading section .data, size 0x68 lma 0x3708
Loading section .m0code, size 0x1f64 lma 0x30000
Start address 0x1d8, load size 22224
Transfer rate: 1973 KB/sec, 3174 bytes/write.
(gdb) monitor reset
Resetting target
(gdb) monitor go
(gdb) q
A debugging session is active.

    Inferior 1 [Remote target] will be killed.

Quit anyway? (y or n) y

c:\D\SDK_2.0_LPCXpresso54114\boards\lpcxpresso54114\multicore_examples\hello_world_cm4\armgcc\debug>

```



```

COM17:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help
Starting Secondary core.

Hello World from the Primary Core!

Press the SW1 button to Stop Secondary core.
Press the SW2 button to Start Secondary core.
Secondary core is in startup code.
Secondary core is in exception number 3.

```

**Build a TrustZone example application** This section describes the steps to build and run a TrustZone application. The demo application build scripts are located in this folder:



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

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

Begin with a simple TrustZone version of the Hello World application. The TrustZone Hello World GCC build scripts are located in this folder:

```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_ns/armgcc/build_  
↪debug.bat
```

```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_s/armgcc/build_  
↪debug.bat
```

Build both applications separately, following steps for single core examples as described in **Build an example application**. 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 the CMSE library is not ready.

```
C:\WINDOWS\system32\cmd.exe
[ 55%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/utilities/fsl_
l_assert.c.obj
[ 59%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/uart/usart_adapter.c
.obj
[ 62%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
flexspi.c.obj
[ 66%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
cache.c.obj
[ 70%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/serial_manager/seria
l_manager.c.obj
[ 74%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/serial_manager/seria
l_port_uart.c.obj
[ 77%] Building ASM object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/gcc/startu
p_MIMXRT595S_cm33.S.obj
[ 81%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/lists/generic_list.c
.obj
[ 85%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
usart.c.obj
[ 88%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
flexcomm.c.obj
[ 92%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
gpio.c.obj
[ 96%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
iap.c.obj
[100%] Linking C executable debug\hello_world_s.elf
[100%] Built target hello_world_s.elf

C:\npx\SDK_2.6.0_EVK-MIMXRT595\boards\evkmimxrt595\trustzone_examples\hello_world\hello_world_s\armgcc>IF "" == "" (paus
e)
Press any key to continue . . .
```

```

C:\WINDOWS\system32\cmd.exe
[ 52%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/uart/usart_adapter.c.obj
[ 56%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/utilities/flexspi_assert.c.obj
[ 60%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/flexspi/flexspi.c.obj
[ 64%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/flexspi/cache.c.obj
[ 68%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/serial_manager/serial_manager.c.obj
[ 72%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/serial_manager/serial_port_uart.c.obj
[ 76%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/flexspi/usart.c.obj
[ 80%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/lists/generic_list.c.obj
[ 84%] Building ASM object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/gcc/startup_MIMXRT595S_cm33.S.obj
[ 88%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/flexcomm/flexcomm.c.obj
[ 92%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/flexcomm/gpio.c.obj
[ 96%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/flexcomm/iap.c.obj
[100%] Linking C executable debug\hello_world_ns.elf
[100%] Built target hello_world_ns.elf

C:\npx\SDK_2.6.0_EVK-MIMXRT595\boards\evkmimxrt595\trustzone_examples\hello_world\hello_world_ns\armgcc>IF "" == "" (pause)
Press any key to continue . . .

```

**Run a TrustZone example application** When running a TrustZone application, the same prerequisites for J-Link/J-Link OpenSDA firmware, and the serial console as for the single core application, apply, as described in **Run an example application**.

To download and run the TrustZone application, perform steps 1 to 10, as described in **Run an example application**. These steps are common for both single core and TrustZone applications in Arm GCC.

Then, run these commands:

1. arm-none-eabi-gdb.exe
2. target remote localhost:2331
3. monitor reset
4. monitor halt
5. monitor exec SetFlashDLNoRMWThreshold = 0x20000
6. load <install\_dir>/boards/evkmimxrt595/trustzone\_examples/hello\_world/hello\_world\_ns/armgcc/debug/hello\_world\_ns.elf
7. load <install\_dir>/boards/evkmimxrt595/trustzone\_examples/hello\_world/hello\_world\_s/armgcc/debug/hello\_world\_s.elf
8. monitor reset

The application is now downloaded and halted. Execute the `c` command to start the demo application.

The first screenshot shows a Windows Command Prompt window titled "Command Prompt - arm-none-eabi-gdb". The user has navigated to the directory `C:\nxp\SDK_2.6.0_EVK-MIMXRT595\boards\evkmimxrt595\trustzone_examples\hello_world` and executed `arm-none-eabi-gdb`. The prompt displays the GNU GDB version (8.2.50.20181213-git) and copyright information. The user enters `target remote localhost:2331`, which results in a warning that no executable has been specified. The user then enters `load hello_world_ns/armgcc/debug/hello_world_ns.elf`, which successfully loads the ELF file. The user enters `c` to continue execution.

The second screenshot shows a PuTTY window titled "COM57 - PuTTY". The output of the program is displayed as follows:

```

Hello from secure world!
Entering normal world.
Welcome in normal world!
This is a text printed from normal world!
Comparing two string as a callback to normal world
String 1: Test1
String 2: Test2
Both strings are not equal!

```

## 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                                                                               |
|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| <b>Pins tool</b>                 | For configuration of pin routing and pin electrical properties.                                                                                                                |  |
| <b>Clock tool</b>                | For system clock configuration                                                                                                                                                 |  |
| <b>Peripherals tools</b>         | For configuration of other peripherals                                                                                                                                         |  |
| <b>TEE tool</b>                  | Configures access policies for memory area and peripherals helping to protect and isolate sensitive parts of the application.                                                  |  |
| <b>Device Configuration tool</b> | 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](http://www.nxp.com/mcuxpresso). Recommended for customers using IAR Embedded Workbench, Keil MDK  $\mu$ Vision, or Arm GCC.
- **Online version** available on [mcuxpresso.nxp.com](http://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.

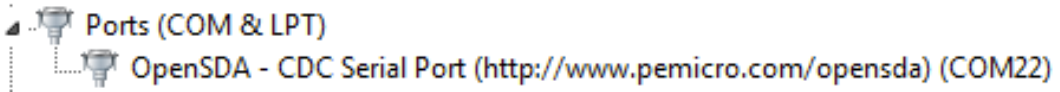
2. **Windows:** To determine the COM port open Device Manager in the Windows operating system. Click the **Start** menu and type **Device Manager** in the search bar.

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.

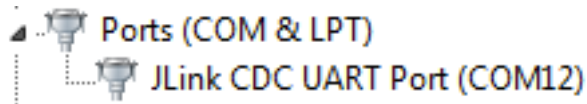
1. **CMSIS-DAP/mbd/DAPLink** interface:



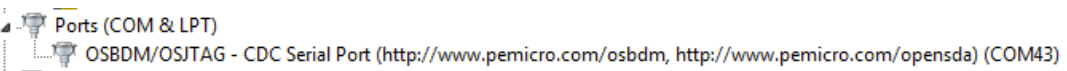
## 2. P&E Micro:



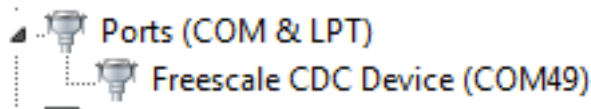
## 3. J-Link:



## 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](http://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](http://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](https://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](http://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 LPCScript. 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 LPCScript 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 [LPCScript](#).

These steps show how to update the debugger firmware on your board for Windows operating system. For Linux OS, follow the instructions described in LPCScript user guide ([LPCScript](#), select **LPCScript**, 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 LPCScript installation directory (<LPCScript install dir>).
  1. To program CMSIS-DAP debug firmware: <LPCScript install dir>/scripts/program\_CMSIS
  2. To program J-Link debug firmware: <LPCScript 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](https://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:

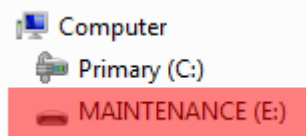
- J-Link: Download appropriate image from [www.segger.com/opensda.html](http://www.segger.com/opensda.html). Choose the appropriate J-Link binary based on the table in [Default debug interfaces](#). Any OpenSDA v1.0 interface should use the standard OpenSDA download (in other words, the one with no version). For OpenSDA 2.0 or 2.1, select the corresponding binary.
- CMSIS-DAP: CMSIS-DAP OpenSDA firmware is available at [www.nxp.com/opensda](http://www.nxp.com/opensda).



- P&E Micro: Downloading P&E Micro OpenSDA firmware images requires registration with P&E Micro ([www.pemicro.com](http://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              |
| 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-MCXA266      | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXA344      | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXA346      | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXA366      | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXC041      | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXC242      | 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-MCXW23       | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXW71       | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXW72       | CMSIS-DAP                 | MCU-Link                |
| FRDM-RW612        | 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                |

continues on next page

Table 1 – continued from previous page

| Hardware platform | Default debugger firmware | On-board debugger probe |
|-------------------|---------------------------|-------------------------|
| 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               |
| 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
}
```

## 1.3 Getting Started with MCUXpresso SDK GitHub

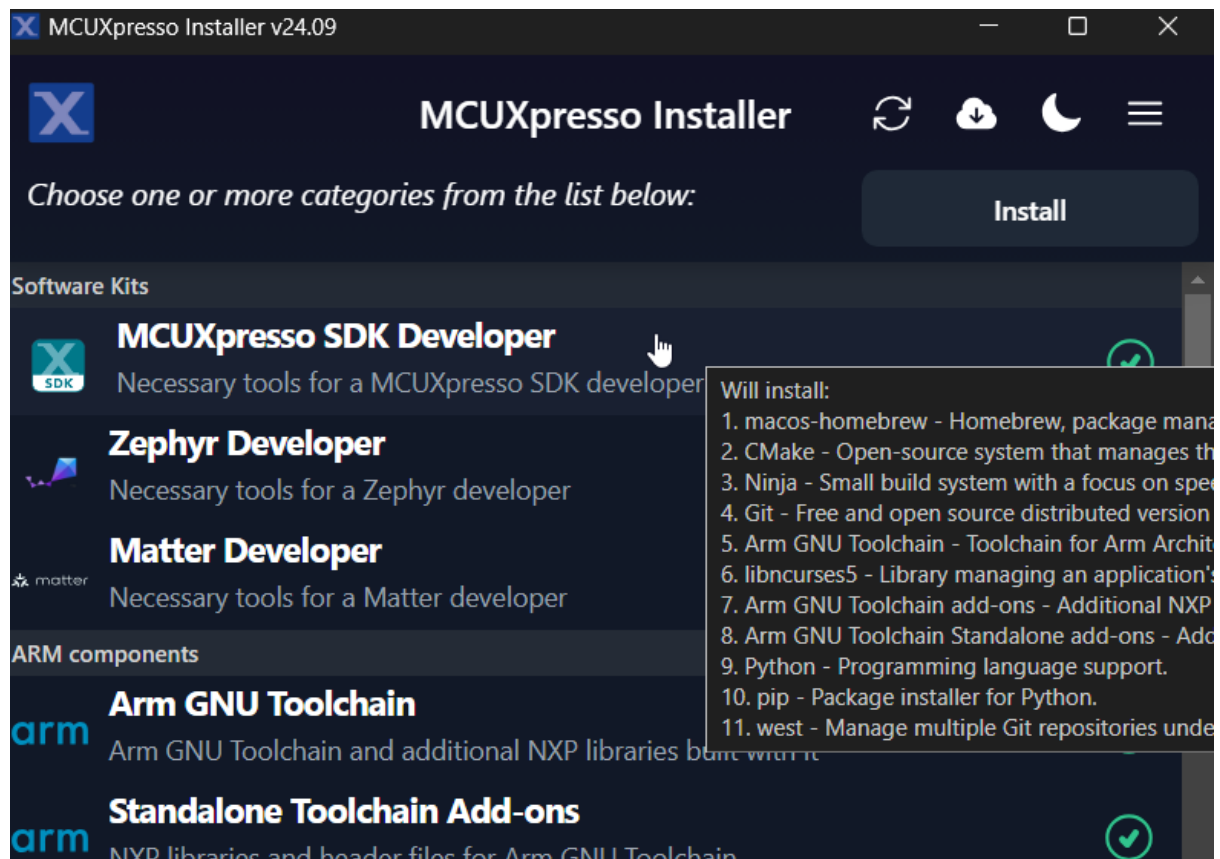
### 1.3.1 Getting Started with MCUXpresso SDK Repository

#### Installation

##### NOTE

If the installation instruction asks/selects whether to have the tool installation path added to the PATH variable, agree/select the choice. This option ensures that the tool can be used in any terminal in any path. [Verify the installation](#) after each tool installation.

**Install Prerequisites with MCUXpresso Installer** The MCUXpresso Installer offers a quick and easy way to install the basic tools needed. The MCUXpresso Installer can be obtained from <https://github.com/nxp-mcuxpresso/vscode-for-mcux/wiki/Dependency-Installation>. The MCUXpresso Installer is an automated installation process, simply select MCUXpresso SDK Developer from the menu and click install. If you prefer to install the basic tools manually, refer to the next section.



## Alternative: Manual Installation

### Basic tools

**Git** Git is a free and open source distributed version control system. Git is designed to handle everything from small to large projects with speed and efficiency. To install Git, visit the official [Git website](#). Download the appropriate version (you may use the latest one) for your operating system (Windows, macOS, Linux). Then run the installer and follow the installation instructions.

User `git --version` to check the version if you have a version installed.

Then configure your username and email using the commands:

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

**Python** Install python 3.10 or latest. Follow the [Python Download](#) guide.

Use `python --version` to check the version if you have a version installed.

**West** Please use the west version equal or greater than 1.2.0

```
# Note: you can add option '--default-timeout=1000' if you meet connection issue. Or you may set a different
↪source using option '-i'.
# for example, in China you could try: pip install -U west -i https://pypi.tuna.tsinghua.edu.cn/simple
pip install -U west
```

## Build And Configuration System

**CMake** It is strongly recommended to use CMake version equal or later than 3.30.0. You can get latest CMake distributions from [the official CMake download page](#).

For Windows, you can directly use the .msi installer like [cmake-3.31.4-windows-x86\\_64.msi](#) to install.

For Linux, CMake can be installed using the system package manager or by getting binaries from [the official CMake download page](#).

After installation, you can use `cmake --version` to check the version.

**Ninja** Please use the ninja version equal or later than 1.12.1.

By default, Windows comes with the Ninja program. If the default Ninja version is too old, you can directly download the [ninja binary](#) and register the ninja executor location path into your system path variable to work.

For Linux, you can use your [system package manager](#) or you can directly download the [ninja binary](#) to work.

After installation, you can use `ninja --version` to check the version.

**Kconfig** MCUXpresso SDK uses Kconfig python implementation. We customize it based on our needs and integrate it into our build and configuration system. The Kconfiglib sources are placed under `mcuxsdk/scripts/kconfig` folder.

Please make sure [python](#) environment is setup ready then you can use the Kconfig.

**Ruby** Our build system supports IDE project generation for iar, mdk, codewarrior and xtensa to provide OOB from build to debug. This feature is implemented with ruby. You can follow the guide ruby environment setup to setup the ruby environment. Since we provide a built-in portable ruby, it is just a simple one cmd installation.

If you only work with CLI, you can skip this step.

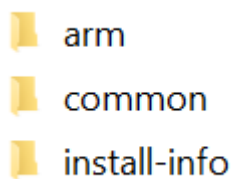
**Toolchain** MCUXpresso SDK supports all mainstream toolchains for embedded development. You can install your used or interested toolchains following the guides.

| Toolchain                    | Download and Installation Guide                                      | Note                        |
|------------------------------|----------------------------------------------------------------------|-----------------------------|
| Armgcc                       | <a href="#">Arm GNU Toolchain Install Guide</a>                      | ARMGCC is default toolchain |
| IAR                          | <a href="#">IAR Installation and Licensing quick reference guide</a> |                             |
| MDK                          | <a href="#">MDK Installation</a>                                     |                             |
| Armclang                     | <a href="#">Installing Arm Compiler for Embedded</a>                 |                             |
| Zephyr                       | <a href="#">Zephyr SDK</a>                                           |                             |
| Codewarrior                  | <a href="#">NXP CodeWarrior</a>                                      |                             |
| Xtensa                       | <a href="#">Tensilica Tools</a>                                      |                             |
| NXP S32Compiler RISC-V Zen-V | <a href="#">NXP Website</a>                                          |                             |

After you have installed the toolchains, register them in the system environment variables. This will allow the west build to recognize them:

| Toolchain                    | Environment Variable | Example                                                                                                                                                | Cmd Line Argument        |
|------------------------------|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|
| Armgcc                       | AR-MGCC_DIR          | C:\armgcc for windows/usr for Linux. Typically arm-none-eabi-* is installed under /usr/bin                                                             | – toolchain armgcc       |
| IAR                          | IAR_DIR              | C:\iar\ewarm-9.60.3 for Windows/opt/iarsystems/bxarm-9.60.3 for Linux                                                                                  | – toolchain iar          |
| MDK                          | MDK_DIR              | C:\Keil_v5 for Windows.MDK IDE is not officially supported with Linux.                                                                                 | – toolchain mdk          |
| Armclang                     | ARM-CLANG_DIR        | C:\ArmCompilerforEmbedded6.22 for Windows/opt/ArmCompilerforEmbedded6.21 for Linux                                                                     | – toolchain mdk          |
| Zephyr                       | ZEPHYR_DIR           | c:\NXP\zephyr-sdk-<version> for windows/opt/zephyr-sdk-<version> for Linux                                                                             | – toolchain zephyr       |
| CodeWarrior                  | CW_DIR               | C:\Freescall\CW MCU v11.2 for windowsCodeWarrior is not supported with Linux                                                                           | – toolchain code-warrior |
| Xtensa                       | XCC_DIR              | C:\xtensa\XtDevTools\install\tools\RI-2023.11-win32\XtensaTools for windows/opt/xtensa/XtDevTools/install/tools/RI-2023.11-Linux/XtensaTools for Linux | – toolchain xtensa       |
| NXP S32Compiler RISC-V Zen-V | RISCV-LVM_DIR        | C:\riscv-llvm-win32_b298_b298_2024.08.12 for Windows/opt/riscv-llvm-Linux-x64_b298_b298_2024.08.12 for Linux                                           | – toolchain riscv-llvm   |

- The <toolchain>\_DIR is the root installation folder, not the binary location folder. For IAR, it is directory containing following installation folders:



- MDK IDE using armclang toolchain only officially supports Windows. In Linux, please directly use armclang toolchain by setting ARMCLANG\_DIR. In Windows, since most Keil users will install MDK IDE instead of standalone armclang toolchain, the MDK\_DIR has higher priority than ARMCLANG\_DIR.
- For Xtensa toolchain, please set the XTENSA\_CORE environment variable. Here's an example list:



| Device Core      | XTENSA_CORE              |
|------------------|--------------------------|
| RT500 fusion1    | nxp_rt500_RI23_11_newlib |
| RT600 hifi4      | nxp_rt600_RI23_11_newlib |
| RT700 hifi1      | rt700_hifi1_RI23_11_nlib |
| RT700 hifi4      | t700_hifi4_RI23_11_nlib  |
| i.MX8ULP fusion1 | fusion_nxp02_dsp_prod    |

- In Windows, the short path is used in environment variables. If any toolchain is using the long path, you can open a command window from the toolchain folder and use below command to get the short path: `for %i in (.) do echo %~fsi`

**Tool installation check** Once installed, open a terminal or command prompt and type the associated command to verify the installation.

If you see the version number, you have successfully installed the tool. Else, check whether the tool's installation path is added into the PATH variable. You can add the installation path to the PATH with the commands below:

- Windows: Open command prompt or powershell, run below command to show the user PATH variable.

```
reg query HKEY_CURRENT_USER\Environment /v PATH
```

The tool installation path should be `C:\Users\xxx\AppData\Local\Programs\Git\cmd`. If the path is not seen in the output from above, append the path value to the PATH variable with the command below:

```
reg add HKEY_CURRENT_USER\Environment /v PATH /d "%PATH%;C:\Users\xxx\AppData\
↪Local\Programs\Git\cmd"
```

Then close the command prompt or powershell and verify the tool command again.

- Linux:
  1. Open the `$HOME/.bashrc` file using a text editor, such as `vim`.
  2. Go to the end of the file.
  3. Add the line which appends the tool installation path to the PATH variable and export PATH at the end of the file. For example, `export PATH="/Directory1:$PATH"`.
  4. Save and exit.
  5. Execute the script with `source .bashrc` or reboot the system to make the changes live. To verify the changes, run `echo $PATH`.
- macOS:
  1. Open the `$HOME/.bash_profile` file using a text editor, such as `nano`.
  2. Go to the end of the file.
  3. Add the line which appends the tool installation path to the PATH variable and export PATH at the end of the file. For example, `export PATH="/Directory1:$PATH"`.
  4. Save and exit.
  5. Execute the script with `source .bash_profile` or reboot the system to make the changes live. To verify the changes, run `echo $PATH`.

## Get MCUXpresso SDK Repo

**Establish SDK Workspace** To get the MCUXpresso SDK repository, use the `west` tool to clone the manifest repository and checkout all the west projects.

```
# Initialize west with the manifest repository
west init -m https://github.com/nxp-mcuxpresso/mcuxsdk-manifests/ mcuxpresso-sdk

# Update the west projects
cd mcuxpresso-sdk
west update

# Allow the usage of west extensions provided by MCUXpresso SDK
west config commands.allow_extensions true
```

**Install Python Dependency(If do tool installation manually)** To create a Python virtual environment in the west workspace core repo directory `mcuxsdk`, follow these steps:

1. Navigate to the core directory:

```
cd mcuxsdk
```

2. [Optional] Create and activate the virtual environment: If you don't want to use the python virtual environment, skip this step. **We strongly suggest you use `venv` to avoid conflicts with other projects using python.**

```
python -m venv .venv

# For Linux/macOS
source .venv/bin/activate

# For Windows
.\.venv\Scripts\activate
# If you are using powershell and see the issue that the activate script cannot be run.
# You may fix the issue by opening the powershell as administrator and run below command:
powershell Set-ExecutionPolicy RemoteSigned
# then run above activate command again.
```

Once activated, your shell will be prefixed with `(.venv)`. The virtual environment can be deactivated at any time by running `deactivate` command.

**Remember to activate the virtual environment every time you start working in this directory.** If you are using some modern shell like `zsh`, there are some powerful plugins to help you auto switch `venv` among workspaces. For example, `zsh-autoswitch-virtualenv`.

3. Install the required Python packages:

```
# Note: you can add option '--default-timeout=1000' if you meet connection issue. Or you may set a
↪ different source using option '-i'.
# for example, in China you could try: pip3 install -r mcuxsdk/scripts/requirements.txt -i https://pypi.
↪ tuna.tsinghua.edu.cn/simple
pip install -r scripts/requirements.txt
```

## Explore Contents

This section helps you build basic understanding of current fundamental project content and guides you how to build and run the provided example project in whole SDK delivery.

**Folder View** The whole MCUXpresso SDK project, after you have done the `west init` and `west update` operations follow the guideline at [Getting Started Guide](#), have below folder structure:

| Folder    | Description                                                                            |
|-----------|----------------------------------------------------------------------------------------|
| manifests | Manifest repo, contains the manifest file to initialize and update the west workspace. |
| mcuxsdk   | The MCUXpresso SDK source code, examples, middleware integration and script files.     |

All the projects record in the [Manifest repo](#) are checked out to the folder `mcuxsdk/`, the layout of `mcuxsdk` folder is shown as below:

| Folder     | Description                                                                                                                                                                                         |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| arch       | Arch related files such as ARM CMSIS core files, RISC-V files and the build files related to the architecture.                                                                                      |
| cmake      | The cmake modules, files which organize the build system.                                                                                                                                           |
| components | Software components.                                                                                                                                                                                |
| devices    | Device support package which categorized by device series. For each device, header file, feature file, startup file and linker files are provided, also device specific drivers are included.       |
| docs       | Documentation source and build configuration for this sphinx built online documentation.                                                                                                            |
| drivers    | Peripheral drivers.                                                                                                                                                                                 |
| examples   | Various demos and examples, support files on different supported boards. For each board support, there are board configuration files.                                                               |
| middleware | Middleware components integrated into SDK.                                                                                                                                                          |
| rtos       | Rtos components integrated into SDK.                                                                                                                                                                |
| scripts    | Script files for the west extension command and build system support.                                                                                                                               |
| svd        | Svd files for devices, this is optional because of large size. Customers run <code>west manifest config group.filter +optional</code> and <code>west update mcux-soc-svd</code> to get this folder. |

**Examples Project** The examples project is part of the whole SDK delivery, and locates in the folder `mcuxsdk/examples` of west workspace.

Examples files are placed in folder of `<example_category>`, these examples include (but are not limited to)

- `demo_apps`: Basic demo set to start using SDK, including `hello_world` and `led_blinky`.
- `driver_examples`: Simple applications that show how to use the 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 transfer using DMA).

Board porting layers are placed in folder of `_boards/<board_name>` which aims at providing the board specific parts for examples code mentioned above.

### Run a demo using MCUXpresso for VS Code

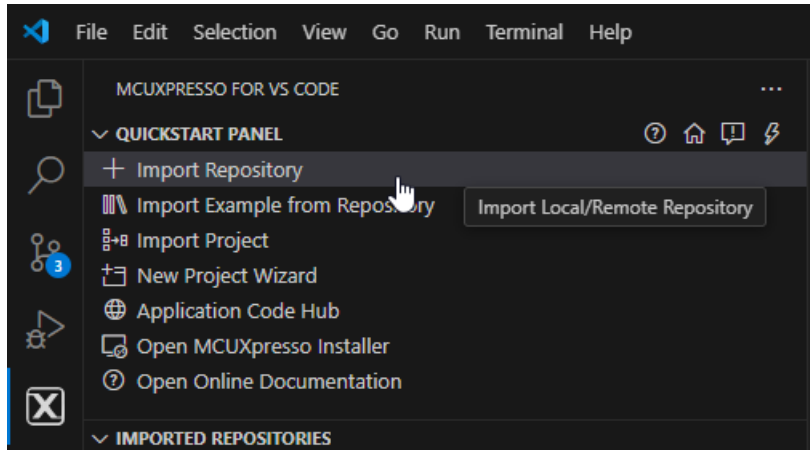
This section explains how to configure MCUXpresso for VS Code to build, run, and debug example applications. This guide uses the `hello_world` demo application as an example. However, these

steps can be applied to any example application in the MCUXpresso SDK.

**Build an example application** This section assumes that the user has already obtained the SDK as outlined in [Get MCUXpresso SDK Repo](#).

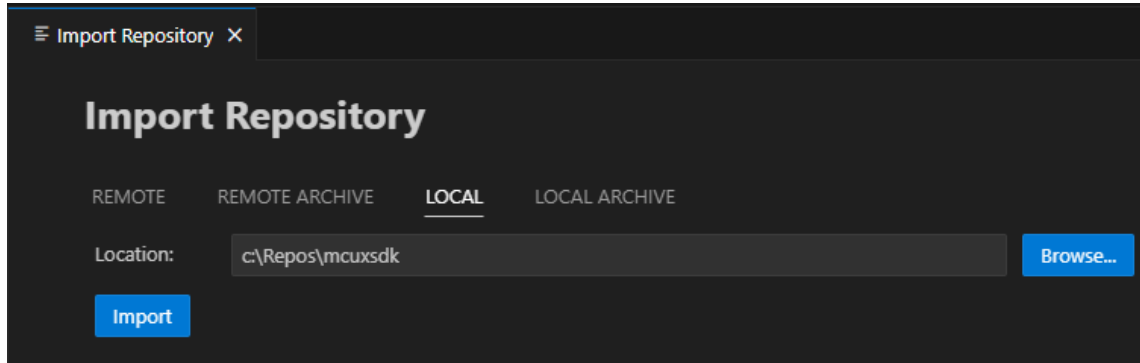
To build an example application:

1. Import the SDK into your workspace. Click **Import Repository** from the **QUICKSTART PANEL**.

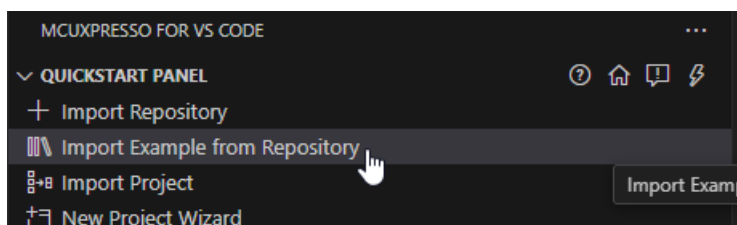


**Note:** You can import the SDK in several ways. Refer to [MCUXpresso for VS Code Wiki](#) for details.

Select **Local** if you've already obtained the SDK as seen in [Get MCUXpresso SDK Repo](#). Select your location and click **Import**.



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



In the dropdown menu, select the MCUXpresso SDK, the Arm GNU Toolchain, your board, template, and application type. Click **Import**.

**Import Example from Repository**

Repository: c:\Repos\mcuxsdk (MCUXpresso SDK Repository) | ▾

Toolchain: (Arm GNU Toolchain 13.2.rel1 (Build arm-13.7)) 13.2.1 20231009 | ▾

Board: FRDM-MCXC444 | ▾

 FRDM-MCXC444

Template: demo\_apps/hello\_world | ▾

The HelloWorld demo prints the "Hello World" string to the terminal using the SDK UART drivers and repeat what user input. The purpose of this demo is to show how to use the UART, and to provide a simple project for debugging and further development.  
Please refer to [README](#) file for more details.

App type: Freestanding application | ▾

Name: frdmxc444\_hello\_world

Location: c:\nxp\_examples [Browse...](#)

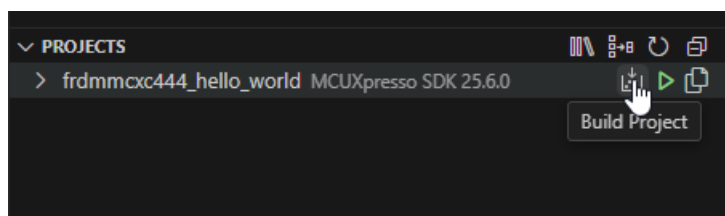
Note: Path doesn't exist. Folder(s) will be created.

☐ Open readme file after project is imported

[Import](#)

**Note:** The MCUXpresso SDK projects can be imported as **Repository applications** or **Freestanding applications**. The difference between the two is the import location. Projects imported as Repository examples will be located inside the MCUXpresso SDK, whereas Freestanding examples can be imported to a user-defined location. Select between these by designating your selection in the **App type** dropdown menu.

3. VS Code will prompt you to confirm if the imported files are trusted. Click **Yes**.
4. Navigate to the **PROJECTS** view. Find your project and click the **Build Project** icon.



The integrated terminal will open at the bottom and will display the build output.

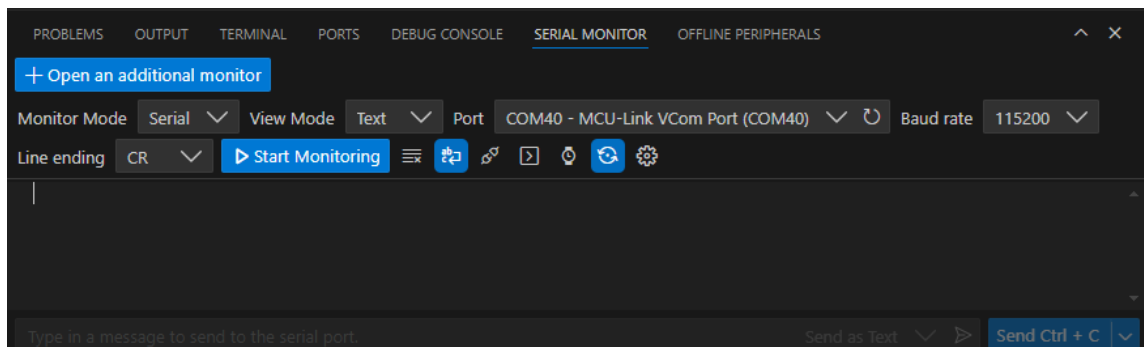
```

[17/21] Building C object CMakeFiles/hello_world.dir/C:/Repos/mcuxsdk/mcuxsdk/components/debug_console_lite/fsl_debug_console.c.obj
[18/21] Building C object CMakeFiles/hello_world.dir/C:/Repos/mcuxsdk/mcuxsdk/devices/MCXC/MCXC444/drivers/fsl_clock.c.obj
[19/21] Building C object CMakeFiles/hello_world.dir/C:/Repos/mcuxsdk/mcuxsdk/drivers/lpuart/fsl_lpuart.c.obj
[20/21] Building C object CMakeFiles/hello_world.dir/C:/Repos/mcuxsdk/mcuxsdk/drivers/uart/fsl_uart.c.obj
[21/21] Linking C executable hello_world.elf
Memory region      Used Size  Region Size  %age Used
m_interrupts:      192 B      512 B      37.50%
m_flash_config:     16 B      16 B      100.00%
m_text:            7892 B    261104 B     3.02%
m_data:            2128 B     32 KB     6.49%
build finished successfully.
Terminal will be reused by tasks, press any key to close it.

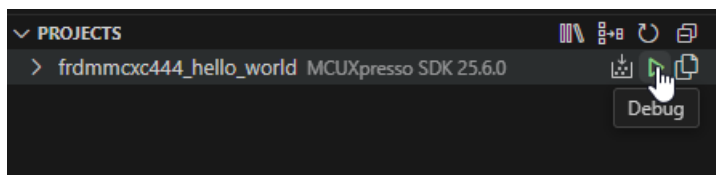
```

**Run an example application** **Note:** for full details on MCUXpresso for VS Code debug probe support, see [MCUXpresso for VS Code Wiki](#).

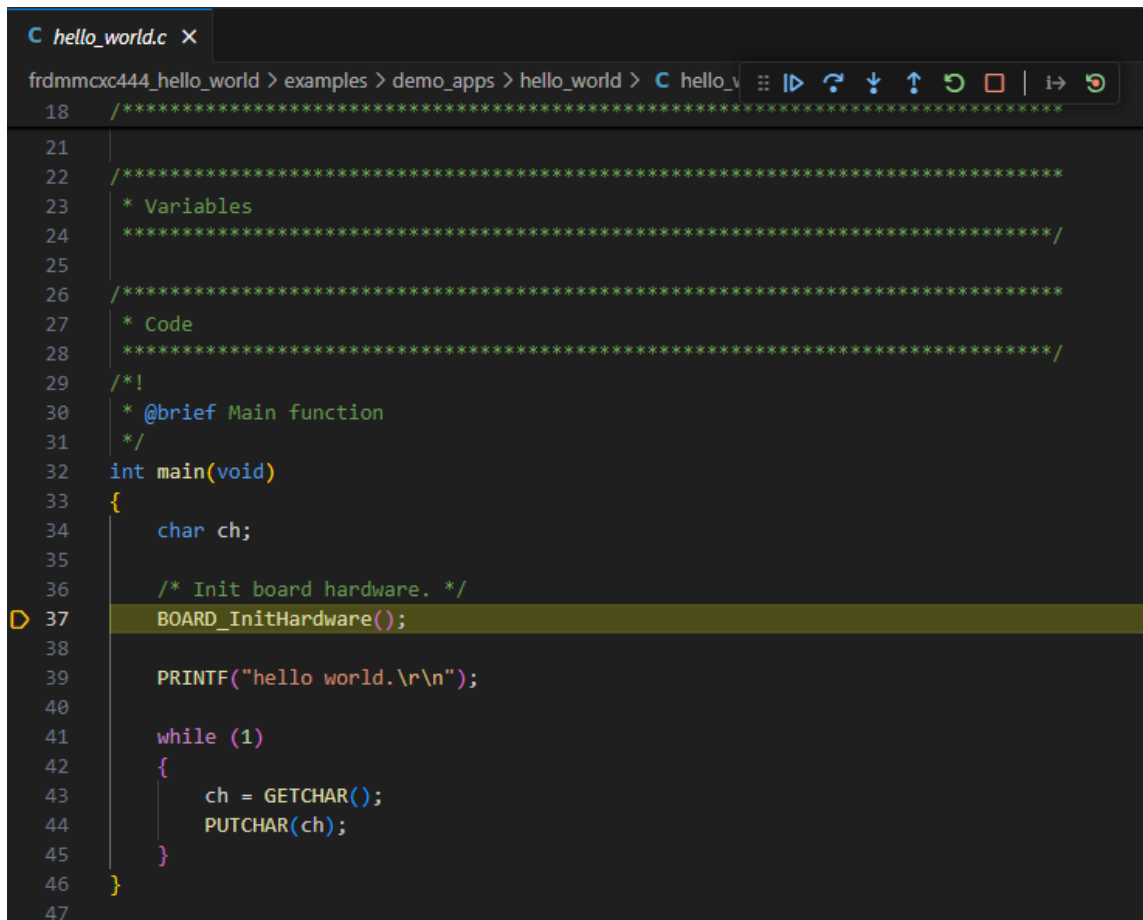
1. Open the **Serial Monitor** from the VS Code's integrated terminal. Select the VCom Port for your device and set the baud rate to 115200.



2. Navigate to the **PROJECTS** view and click the play button to initiate a debug session.



The debug session will begin. The debug controls are initially at the top.

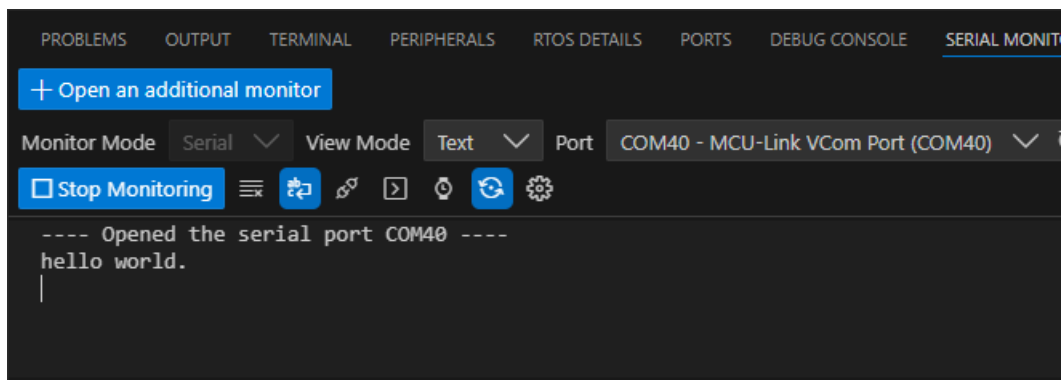


```

18  /*****
21
22  /*****
23  * Variables
24  *****/
25
26  /*****
27  * Code
28  *****/
29  /*!
30  * @brief Main function
31  */
32  int main(void)
33  {
34      char ch;
35
36      /* Init board hardware. */
37      BOARD_InitHardware();
38
39      PRINTF("hello world.\r\n");
40
41      while (1)
42      {
43          ch = GETCHAR();
44          PUTCHAR(ch);
45      }
46  }
47

```

3. Click **Continue** on the debug controls to resume execution of the code. Observe the output on the **Serial Monitor**.



```

PROBLEMS  OUTPUT  TERMINAL  PERIPHERALS  RTOS DETAILS  PORTS  DEBUG CONSOLE  SERIAL MONITOR
+ Open an additional monitor
Monitor Mode: Serial View Mode: Text Port: COM40 - MCU-Link VCom Port (COM40)
[Stop Monitoring] [Icons]
---- Opened the serial port COM40 ----
hello world.
|

```

### Running a demo using ARMGCC CLI/IAR/MDK

**Supported Boards** Use the west extension `west list_project` to understand the board support scope for a specified example. All supported build command will be listed in output:

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

```
INFO: [ 1][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↳ evk9mimx8ulp -Dcore_id=cm33]
```

```
INFO: [ 2][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↳ evkbimxrt1050]
```

```
INFO: [ 3][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
```

(continues on next page)



(continued from previous page)

```

↪ evkbnimxrt1060]
INFO: [ 4][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbnimxrt1170 -Dcore_id=cm4]
INFO: [ 5][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbnimxrt1170 -Dcore_id=cm7]
INFO: [ 6][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbnimxrt1060]
INFO: [ 7][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbnimxrt7ulp]
...

```

The supported toolchains and build targets for an example are decided by the example-self example.yml and board example.yml, please refer Example Toolchains and Targets for more details.

**Build the project** Use west build -h to see help information for west build command. Compared to zephyr's west build, MCUXpresso SDK's west build command provides following additional options for mcux examples:

- --toolchain: specify the toolchain for this build, default armgcc.
- --config: value for CMAKE\_BUILD\_TYPE. If not provided, build system will get all the example supported build targets and use the first debug target as the default one. Please refer Example Toolchains and Targets for more details about example supported build targets.

Here are some typical usages for generating a SDK example:

```

# Generate example with default settings, default used device is the mainset MK22F51212
west build -b frdmk22f examples/demo_apps/hello_world

# Just print cmake commands, do not execute it
west build -b frdmk22f examples/demo_apps/hello_world --dry-run

# Generate example with other toolchain like iar, default armgcc
west build -b frdmk22f examples/demo_apps/hello_world --toolchain iar

# Generate example with other config type
west build -b frdmk22f examples/demo_apps/hello_world --config release

# Generate example with other devices with --device
west build -b frdmk22f examples/demo_apps/hello_world --device MK22F12810 --config release

```

For multicore devices, you shall specify the corresponding core id by passing the command line argument -Dcore\_id. For example

```

west build -b evkbnimxrt1170 examples/demo_apps/hello_world --toolchain iar -Dcore_id=cm7 --config_
↪ flexspi_nor_debug

```

For shield, please use the --shield to specify the shield to run, like

```

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

```

**Sysbuild(System build)** To support multicore project building, we ported Sysbuild from Zephyr. It supports combine multiple projects for compilation. You can build all projects by adding --sysbuild for main application. For example:

```

west build -b evkbnimxrt1170 --sysbuild ./examples/multicore_examples/hello_world/primary -Dcore_
↪ id=cm7 --config flexspi_nor_debug --toolchain=armgcc -p always

```

For more details, please refer to System build.

**Config a Project** Example in MCUXpresso SDK is configured and tested with pre-defined configuration. You can follow steps blow to change the configuration.

1. Run cmake configuration

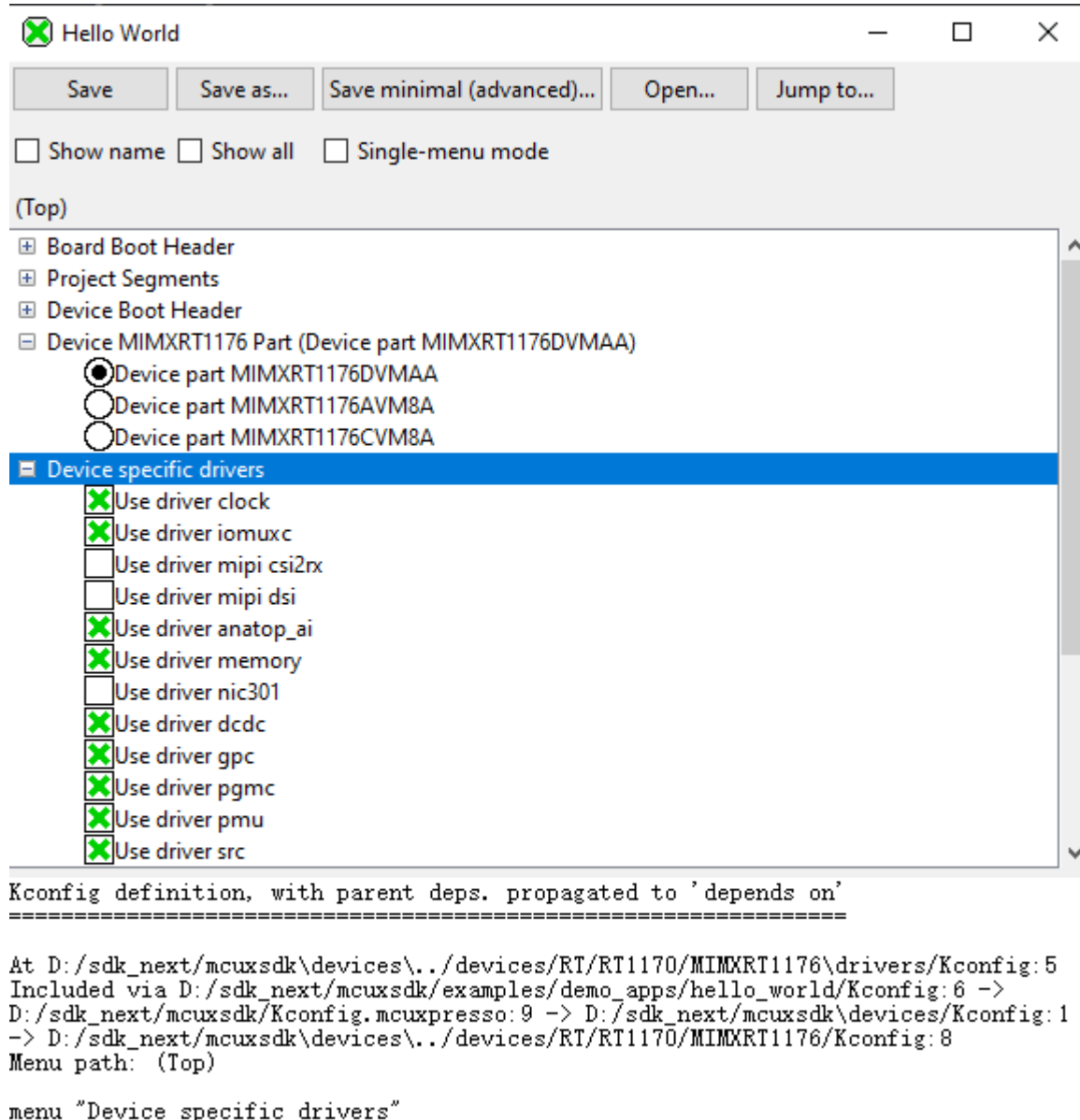
```
west build -b evkbnimxrt1170 examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

Please note the project will be built without --cmake-only parameter.

2. Run guiconfig target

```
west build -t guiconfig
```

Then you will get the Kconfig GUI launched, like



You can reconfigure the project by selecting/deselecting Kconfig options.

After saving and closing the Kconfig GUI, you can directly run `west build` to build with the new configuration.

**Flash** *Note:* Please refer Flash and Debug The Example to enable west flash/debug support.  
Flash the hello\_world example:

```
west flash -r linkserver
```

**Debug** Start a gdb interface by following command:

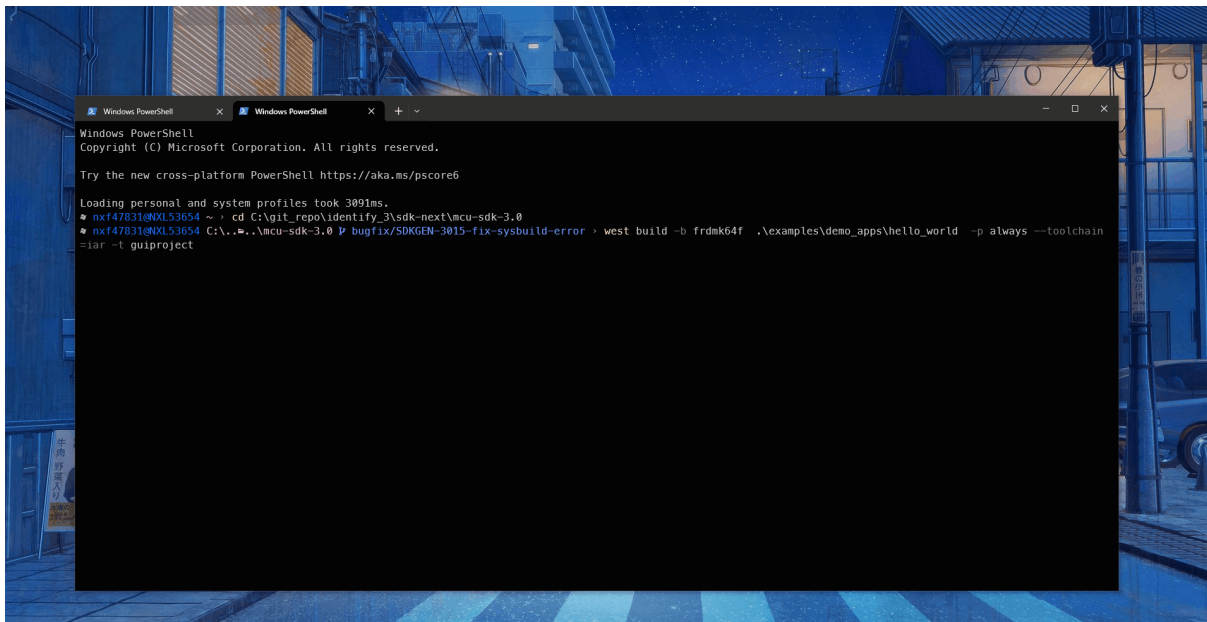
```
west debug -r linkserver
```

**Work with IDE Project** The above build functionalities are all with CLI. If you want to use the toolchain IDE to work to enjoy the better user experience especially for debugging or you are already used to develop with IDEs like IAR, MDK, Xtensa and CodeWarrior in the embedded world, you can play with our IDE project generation functionality.

This is the cmd to generate the evkbmimxrt1170 hello\_world IAR IDE project files.

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

By default, the IDE project files are generated in mcuxsdk/build/<toolchain> folder, you can open the project file with the IDE tool to work:



Note, please follow the [Installation](#) to setup the environment especially make sure that [ruby](#) has been installed.

## 1.4 Release Notes

### 1.4.1 MCUXpresso SDK Release Notes

#### Overview

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 Bluetooth-enabled MCUs. MCUXpresso SW and Tools for DSC

further extends the SDK support to current 32-bit Digital Signal Controllers. The MCUXpresso SDK includes production-grade software with integrated RTOS (optional), integrated enabling software technologies (stacks and middleware), reference software, and more.

In addition to working seamlessly with the MCUXpresso IDE, the MCUXpresso SDK also supports and provides example projects for various toolchains. The Development tools chapter in the associated Release Notes provides details about toolchain support for your board. Support for the MCUXpresso Config Tools allows easy cloning of existing SDK examples and demos, allowing users to leverage the existing software examples provided by the SDK for their own projects.

Underscoring our commitment to high quality, the MCUXpresso SDK is MISRA compliant and checked with Coverity static analysis tools. For details on MCUXpresso SDK, see [MCUXpresso-SDK: Software Development Kit for MCUXpresso](#).

## MCUXpresso SDK

As part of the MCUXpresso software and tools, MCUXpresso SDK is the evolution of Kinetis SDK, includes support for LPC, DSC, PN76, and i.MX System-on-Chip (SoC). The same drivers, APIs, and middleware are still available with support for Kinetis, LPC, DSC, and i.MX silicon. The MCUXpresso SDK adds support for the MCUXpresso IDE, an Eclipse-based toolchain that works with all MCUXpresso SDKs. Easily import your SDK into the new toolchain to access to all of the available components, examples, and demos for your target silicon. In addition to the MCUXpresso IDE, support for the MCUXpresso Config Tools allows easy cloning of existing SDK examples and demos, allowing users to leverage the existing software examples provided by the SDK for their own projects.

In order to maintain compatibility with legacy Freescale code, the filenames and source code in MCUXpresso SDK containing the legacy Freescale prefix FSL has been left as is. The FSL prefix has been redefined as the NXP Foundation Software Library.

## 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 9.60.4
- Keil MDK, version is 5.42
- MCUXpresso for VS Code v25.09
- 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.

|                                      |                     |             |             |             |             |
|--------------------------------------|---------------------|-------------|-------------|-------------|-------------|
| De-<br>vel-<br>op-<br>ment<br>boards | MCU devices         |             |             |             |             |
| <b>FRDM</b>                          | MCXA142VFM,         | MCXA142VFT, | MCXA142VLF, | MCXA142VLH, | MCXA143VFM, |
| <b>MCXA</b>                          | MCXA143VFT,         | MCXA143VLF, | MCXA143VLH, | MCXA152VFM, | MCXA152VFT, |
|                                      | MCXA152VLF,         | MCXA152VLH, | MCXA153VFM, | MCXA153VFT, | MCXA153VLF, |
|                                      | <b>MCXA153VLH</b> , | MCXA132VFM, | MCXA132VFT, | MCXA132VLF, | MCXA133VFM, |
|                                      | MCXA133VFT,         | MCXA133VLF  |             |             |             |

## 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.

**Secure Element Host Library** Secure Element Host Library

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

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

**MCU Boot** Open source MCU Bootloader.

**Motor Control Software (ACIM, BLDC, PMSM)** Motor control examples.

**FreeMASTER** FreeMASTER communication driver for 32-bit platforms.

## 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_name> |
| Utilities such as debug console                                                      | INSTALL_DIR/devices/<device_name>/utilities        |
| Device Project Template for MCUXpresso IDE NPW                                       | INSTALL_DIR/devices/<device_name>/project_template |
| 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.

### 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.

## [NPW][cpp component][device pack][link issue] MCUXpresso IDE new c++ projects had build errors

The new C++ project created with MCUXpresso IDE NPW and device cannot build pass.

## 1.5 ChangeLog

### 1.5.1 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
- 

#### AOI

###### [2.0.2]

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

###### [2.0.1]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.8, 2.2.

###### [2.0.0]

- Initial version.
-



## CACHE LPCAC

### [2.2.1]

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

### [2.2.0]

- Improvements
  - Used new add macros `FSL_FEATURE_SYSCON_HAS_LPCAC_CTRL_PARITY_MISS_EN_BIT` and `FSL_FEATURE_SYSCON_HAS_LPCAC_CTRL_PARITY_FAULT_EN_BIT` to support some platforms which `PARITY_MISS_EN` and `PARITY_FAULT_EN` bit may be reserved.

### [2.1.1]

- Bug Fixes
  - Fixed an issue of `L1CACHE_InvalidateCodeCache()` function, to clean cache the `LP-CAC_CTRL[CLR_LPCAC]` should be set not clear.

### [2.1.0]

- Improvements
  - Supported more features, such as write buffer contron, write buffer limit and so on.

### [2.0.0]

- Initial version.
- 

## CDOG

### [2.1.3]

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

### [2.1.2]

- Support multiple IRQs
- Fix default CONTROL values

### [2.1.1]

- Remove bit `CONTROL[CONTROL_CTRL]`.

### [2.1.0]

- Rename CWT to CDOG.

#### [2.0.2]

- Fix MISRA-2012 issues.

#### [2.0.1]

- Fix doxygen issues.

#### [2.0.0]

- Initial version.
- 

### CLOCK

#### [2.0.0]

- Initial version.
- 

### MCX\_CMC

#### [2.5.0]

- New features
  - Update MCX\_CMC driver to be compatible with new platforms.

#### [2.4.0]

- New features
  - Update MCX\_CMC driver to be compatible with new platforms.

#### [2.3.0]

- Improvements
  - Added macros to support some device have BSR\_SCR bit field.

#### [2.2.3]

- Improvements
  - Clear SCB\_SCR[SLEEPDEEP] bitfield after wakeup.

#### [2.2.2]

- Improvements
  - Fixed the violation of MISRA C-2012 rules.

**[2.2.1]**

- Improvements
  - Updated `_cmc_system_reset_interrupt_enable`, `_cmc_system_reset_interrupt_flag` and `_cmc_system_reset_sources` to support new added bit field.
- Bug Fixes
  - Fixed issue in `CMC_PowerOffSRAMAllMode()` and `CMC_PowerOffSRAMLowPowerOnly()` which overwrite reserved bit fields.

**[2.2.0]**

- Improvements
  - Added feature macro “`FSL_FEATURE_MCX_CMC_HAS_NO_FLASHCR_WAKE`” to support some devices where `FLASHCR[WAKE]` is reserved.

**[2.1.0]**

- Improvements
  - Added macros to support some devices(such as MCXA family) that only support one power domain.

**[2.0.0]**

- Initial version.
- 

**COMMON****[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 irq's 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 `stdint.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****[2.0.4]**

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

**[2.0.3]**

- Bug fix:
  - Fix MISRA issues.

**[2.0.2]**

- Bug fix:
  - Fix MISRA issues.

**[2.0.1]**

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

**[2.0.0]**

- Initial version.
- 

**CTIMER****[2.3.3]**

- Bug Fixes
  - Fix CERT INT30-C INT31-C issue.
  - Make API CTIMER\_SetupPwm and CTIMER\_UpdatePwmDutycycle return fail if pulse width register overflow.

**[2.3.2]**

- Bug Fixes
  - Clear unexpected DMA request generated by RESET\_PeripheralReset in API CTIMER\_Init to avoid trigger DMA by mistake.



### [2.3.1]

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

### [2.3.0]

- Improvements
  - Added the CTIMER\_SetPrescale(), CTIMER\_GetCaptureValue(), CTIMER\_EnableResetMatchChannel(), CTIMER\_EnableStopMatchChannel(), CTIMER\_EnableRisingEdgeCapture(), CTIMER\_EnableFallingEdgeCapture(), CTIMER\_SetShadowValue(), APIs Interface to reduce code complexity.

### [2.2.2]

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

### [2.2.1]

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

### [2.2.0]

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

### [2.1.0]

- Improvements
  - Added the CTIMER\_GetOutputMatchStatus() API Interface.
  - Added feature macro for FSL\_FEATURE\_CTIMER\_HAS\_NO\_CCR\_CAP2 and FSL\_FEATURE\_CTIMER\_HAS\_NO\_IR\_CR2INT.

### [2.0.3]

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

**[2.0.2]**

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

**[2.0.1]**

- Improvements
  - API Interface Change
    - \* Changed API interface by adding CTIMER\_SetupPwmPeriod API and CTIMER\_UpdatePwmPulsePeriod API, which both can set up the right PWM with high resolution.

**[2.0.0]**

- Initial version.
- 

**EDMA****[2.10.7]**

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

**[2.10.6]**

- Improvements
  - Add macro FSL\_FEATURE\_EDMA\_HAS\_EDMA\_TCD\_CLOCK\_ENABLE to enable tcd clocks in EDMA\_Init function.

**[2.10.5]**

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

**[2.10.4]**

- Improvements
  - Add new MP register macros to ensure compatibility with different devices.
  - Add macro DMA\_CHANNEL\_ARRAY\_STEPn to adapt to complex addressing of edma tcd registers.

### [2.10.3]

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

### [2.10.2]

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

### [2.10.1]

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

### [2.10.0]

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

### [2.9.2]

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

**[2.9.1]**

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

**[2.9.0]**

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

**[2.8.1]**

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

**[2.8.0]**

- Improvements
  - Added feature FSL\_FEATURE\_EDMA\_HAS\_NO\_CH\_SBR\_SEC to separate DMA without SEC bitfield.

**[2.7.1]**

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

**[2.7.0]**

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

**[2.6.0]**

- Improvements
  - Modify the type of parameter channelRequestSource from dma\_request\_source\_t to int32\_t in the EDMA\_SetChannelMux.

**[2.5.3]**

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

#### [2.5.2]

- Improvements
  - Applied ERRATA 51327.

#### [2.5.1]

- Bug Fixes
  - Fixed the EDMA\_ResetChannel function cannot reset channel DONE/ERROR status.

#### [2.5.0]

- Improvements
  - Added feature FSL\_FEATURE\_EDMA\_HAS\_NO\_SBR\_ATTR\_BIT to separate DMA without ATTR bitfield.
  - Added api EDMA\_GetChannelSystemBusInformation to gets the channel identification and attribute information on the system bus interface.
- Bug Fixes
  - Fixed the ESG bit not set in scatter gather mode issue.
  - Fixed the DREQ bit configuration missed in single transfer issue.
  - Cleared the interrupt status before invoke callback to avoid miss interrupt issue.
  - Removed disableRequestAfterMajorLoopComplete from edma\_transfer\_config\_t structure as driver will handle it.
  - Fixed the channel mux configuration not compatible issue.
  - Fixed the out of bound access in function EDMA\_DriverIRQHandler.

#### [2.4.4]

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

#### [2.4.3]

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

#### [2.4.2]

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

### [2.4.1]

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

### EIM

#### [2.0.4]

- Improvements
  - Change `eim_memory_channel_t` to `uint32_t`.
  - Support channel number 16-23.

#### [2.0.3]

- Bug Fixes
  - Fixed `s_eimResets` variable declaration issue.

#### [2.0.2]

- Improvements
  - Reset the peripheral during initialization, otherwise some platforms may not work.
- Bug Fixes
  - Fixed incorrect register access in `EIM_GetDataBitMask()`.

#### [2.0.1]

- Improvements
  - Update driver to support fewer channel.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.3.

#### [2.0.0]

- Initial version.
- 

### EQDC

#### [2.3.1]

- Bug Fix
- Fixed `CTRL2[CMODE]` field overwritten in API `EQDC_Init`.

#### [2.3.0]

- Improvements
- Add feature macro to support platforms which do not have compare interrupt.

**[2.2.3]**

- Bug Fix
- Clear Revolution Counter Register(REV) in init function to prevent its value not equal to zero after reset.

**[2.2.2]**

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

**[2.2.1]**

- Bug Fix
- Fixed violations of the MISRA C-2012 rules 20.9.

**[2.2.0]**

- New features
- Supported the feature that the position counter to be initialized by Index Event Edge Mark.

**[2.1.0]**

- Bug Fix
- Fixed typo in interrupt enumeration values.
  - Improvements
- Supported Count Direct Change interrupt.
- Removed unused parameter in user configuration.
- Supported ERRATA\_051383 check, the CTRL[DMAEN] can't be cleared.

**[2.0.1]**

- Bug Fix
- Fixed violations of the MISRA C-2012 rules 10.3, 10.6, 10.8, 14.4, 16.4.

**[2.0.0]**

- Initial version.
- 

**ERM****[2.0.4]**

- Improvements
  - Change erm\_memory\_channel\_t to uint32\_t.
  - Support channel number 16-23.

### [2.0.3]

- Bug Fixes
  - Fixed s\_ermResets variable declaration issue.

### [2.0.2]

- Improvements
  - Reset the peripheral during initialization, otherwise some platforms may not work.
- Bug Fixes
  - Only keep bit 0-3 of ERM\_GetInterruptStatus() as it may get other channel results in higher bits.

### [2.0.1]

- Improvements
  - Update driver to support fewer channel.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.3.

### [2.0.0]

- Initial version.
- 

## FREQME

### [2.1.2]

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

### [2.1.1]

- Fixed MISRA issues.

### [2.1.0]

- Updated register name.

### [2.0.0]

- Initial version.
- 

## GLIKEY

### [2.0.1]

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

**[2.0.0]**

- Initial version.
- 

**GPIO****[2.8.2]**

- 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\_GetPinsDMAResourceFlags 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 privilege status.

#### [2.5.3]

- Bug Fixes
  - Correct the feature macro typo: FSL\_FEATURE\_GPIO\_HAS\_NO\_INDEP\_OUTPUT\_CONTORL.

#### [2.5.2]

- Improvements
  - Improved GPIO\_PortSet/GPIO\_PortClear/GPIO\_PortToggle functions to support devices without Set/Clear/Toggle registers.

#### [2.5.1]

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

#### [2.4.1]

- Improvements
  - Improved GPIO\_CheckAttributeBytes() function to support 8 bits width GACR register.

#### [2.4.0]

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

### [2.3.2]

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

### [2.3.1]

- Improvements
  - Removed deprecated APIs.

### [2.3.0]

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

### [2.2.1]

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

### [2.1.1]

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

### [2.1.0]

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

## I3C

### [2.14.3]

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

**[2.14.2]**

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

**[2.14.1]**

- Improvements
  - Split the function I3C\_MasterTransferBlocking to meet the HIS-CCM requirement.

**[2.14.0]**

- Improvements
  - Added the choice to set fast start header with push-pull speed when all targets addresses have MSB 0 instead of forcing to set it.
  - Deleted duplicated busy check in I3C\_MasterStart function.

**[2.13.1]**

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

**[2.13.0]**

- New features
  - Added the hot-join support for I3C bus initialization API.
- Bug Fixes
  - Set read termination with START at the same time in case unknown issue.
  - Set MCTRL[TYPE] as 0 for DDR force exit.
- Improvements
  - Added the API to reset device count assigned by ENTDAAs.
  - Provided the method to set global macro I3C\_MAX\_DEVCNT to determine how many device addresses ENTDAAs can allocate at one time.
  - Initialized target management static array based on instance number for the case that multiple instances are used at the same time.

**[2.12.0]**

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

**[2.11.1]**

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

**[2.11.0]**

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

**[2.10.8]**

- Improvements
  - Support more instances.

**[2.10.7]**

- Improvements
  - Fixed the potential compile warning.

**[2.10.6]**

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

**[2.10.5]**

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

**[2.10.4]**

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



### [2.10.3]

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

### [2.10.2]

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

### [2.10.1]

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

### [2.10.0]

- New features
  - Added I3C extended IBI data support.

### [2.9.0]

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

### [2.8.2]

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

### [2.8.1]

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

### [2.8.0]

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

### [2.7.1]

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

**[2.7.0]**

- Fixed the CCM problem in file fsl\_i3c.c.
- Fixed the FSL\_FEATURE\_I3C\_HAS\_NO\_SCONFIG\_IDRAND usage issue in I3C\_GetDefaultConfig and I3C\_Init.

**[2.6.0]**

- Fixed the FSL\_FEATURE\_I3C\_HAS\_NO\_SCONFIG\_IDRAND usage issue in fsl\_i3c.h.
- Changed some static functions in fsl\_i3c.c as non-static and define the functions in fsl\_i3c.h to make I3C DMA driver reuse:
  - I3C\_GetIBIType
  - I3C\_GetIBIAddress
  - I3C\_SlaveCheckAndClearError
- Changed the handle pointer parameter in IRQ related functions to void \* type to make it reuse in I3C DMA driver.
- Added new API I3C\_SlaveRequestIBIWithSingleData for slave to request single data byte, this API could be used regardless slave is working in non-blocking interrupt or non-blocking dma.
- Added new API I3C\_MasterGetDeviceListAfterDAA for master application to get the device information list built up in DAA process.

**[2.5.4]**

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

**[2.5.3]**

- Updated driver for new feature FSL\_FEATURE\_I3C\_HAS\_NO\_SCONFIG\_BAMATCH and FSL\_FEATURE\_I3C\_HAS\_NO\_SCONFIG\_IDRAND.

**[2.5.2]**

- Updated driver for new feature FSL\_FEATURE\_I3C\_HAS\_NO\_MERRWARN\_TERM.
- Fixed the issue that call to I3C\_MasterTransferBlocking API did not generate STOP signal when NAK status was returned.

**[2.5.1]**

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

### [2.5.0]

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

### [2.4.0]

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

### [2.3.2]

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

### [2.3.1]

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

### [2.3.0]

- Improvements
  - Added I3C common driver APIs to initialize I3C with both master and slave configuration.
  - Updated I3C master transfer callback to function set structure to include callback invoke for IBI event and slave2master event.
  - Updated I3C master non-blocking transfer model and always enable the interrupts to be able to re-act to the slave start event and handle slave IBI.

**[2.2.0]**

- Bug Fixes
  - Fixed the issue that I3C transfer size limit to 255 bytes.

**[2.1.2]**

- Bug Fixes
  - Reset default hkeep value to kI3C\_MasterHighKeeperNone in I3C\_MasterGetDefaultConfig

**[2.1.1]**

- Bug Fixes
  - Fixed incorrect FIFO reset operation in I3C Master Transfer APIs.
  - Fixed i3c slave IRQ handler issue, slave transmit could be underrun because tx FIFO is not filled in time right after start flag detected.

**[2.1.0]**

- Added definitions and APIs for I3C slave functionality, updated previous I3C APIs to support I3C functionality.

**[2.0.0]**

- Initial version.
- 

**I3C\_EDMA****[2.2.12]**

- Bug Fixes
  - Fixed the issue that EDMA transfer configuration use wrong parameter.

**[2.2.11]**

- Improvements
  - Fixed Coverity CERT-C violations.

**[2.2.10]**

- Bug Fixes
  - Fixed the issue that slave start event is cleared when it has not been handled.
- Added
  - Supported I3C HDR-DDR transfer with EDMA.
- Changed
  - Used linked EDMA to transfer all I3C subaddress and data without handling of intermediate states, simplifying code logic.

- Prepare DMA before I3C START to ensure there's no time delay between START and transmitting data.
- Added the MCTRLDONE flag check after START and STOP request to ensure all states are handled properly.

#### [2.2.9]

- Bug Fixes
  - Fixed MISRA issue rule 11.3.
  - Added the master control done flag waiting code after STOP in case the bus is not idle when transfer function finishes.

#### [2.2.8]

- Improvements
  - Removed I3C IRQ handler calling in the EDMA callback. Previously driver doesn't use the END byte which can trigger the STOP interrupt for controller sending and receiving, now let I3C event handler deal with all I3C events.
- Bug Fixes
  - Fixed the bug that the END type Tx register is not used when command length or data length is one byte.

#### [2.2.7]

- Bug Fixes
  - Fixed MISRA issue rule 11.6.

#### [2.2.6]

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

#### [2.2.5]

- Improvements
  - Added the workaround for RT1180 I3C EDMA issue ERR052086.

#### [2.2.4]

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

#### [2.2.3]

- Bug Fixes
  - Fixed issue that slave polulates the last byte when Tx FIFO may be full.

**[2.2.2]**

- Bug Fixes
  - Fixed I3C MISRA issue rule 10.4, 11.3.

**[2.2.1]**

- Bug Fixes
  - Fixed the issue that I3C slave send the last byte data without using the END type register.
- Improvements
  - There's no need to reserve two bytes FIFO for DMA transfer which is for IP issue workaround.

**[2.2.0]**

- Improvements
  - Deleted legacy IBI data request code.

**[2.1.0]**

- Bug Fixes
  - Fixed MISRA issue rule 8.4, 8.6, 11.8.

**[2.0.1]**

- Bug Fixes
  - Fixed MISRA issue rule 9.1.

**[2.0.0]**

- Initial version.
- 

**INPUTMUX****[2.0.10]**

- Bug Fixes
  - Fixed CERT-C violations.

**[2.0.9]**

- Improvements
  - Use INPUTMUX\_CLOCKS to initialize the inputmux module clock to adapt to multiple inputmux instances.
  - Modify the API base type from INPUTMUX\_Type to void.

#### [2.0.8]

- Improvements
  - Updated a feature macro usage for function INPUTMUX\_EnableSignal.

#### [2.0.7]

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

#### [2.0.6]

- Bug Fixes
  - Fixed the documentation wrong in API INPUTMUX\_AttachSignal.

#### [2.0.5]

- Bug Fixes
  - Fixed build error because some devices has no sct.

#### [2.0.4]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rule 10.4, 12.2 in INPUTMUX\_EnableSignal() function.

#### [2.0.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.4, 10.7, 12.2.

#### [2.0.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.4, 12.2.

#### [2.0.1]

- Support channel mux setting in INPUTMUX\_EnableSignal().

#### [2.0.0]

- Initial version.
-

## LPADC

### [2.9.4]

- Improvements
  - Update LPADC\_GetDefaultConfig, change default conversionAverageMode value to: kLPADC\_ConversionAverage128 for 3 bit width. kLPADC\_ConversionAverage1024 for 4 bit width.

### [2.9.3]

- Improvements
  - Add timeout for while loop code.

### [2.9.2]

- Improvements
  - Fixed CERT-C issues.

### [2.9.1]

- Bug Fixes
  - Fixed incorrect channel B FIFO selection logic.

### [2.9.0]

- Bug Fixes
  - Add code to handle the case where GCC[GAIN\_CAL] is a signed number.
  - Split LPADC\_FinishAutoCalibration function into two functions.
  - Improved LPADC driver.

### [2.8.4]

- Bug Fixes
  - Remove function 'LPADC\_SetOffsetValue' assert statement, this statement may cause runtime errors in existing code.

### [2.8.3]

- Bug Fixes
  - Fixed SDK lpadc driver examples compile issue, move condition 'commandId < ADC\_CV\_COUNT' to a more appropriate location.

### [2.8.2]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rule 18.1, 10.3, 10.1 and 10.4.



#### [2.8.1]

- Bug Fixes
  - Fixed LPADC sample mode enum name mistake.

#### [2.8.0]

- Improvements
  - Release peripheral from reset if necessary in init function.
- Bug Fixes
  - Fixed function LPADC\_GetConvResult() issue.
  - Fixed function LPADC\_SetConvCommandConfig() bugs.

#### [2.7.2]

- Improvements
  - Use feature macros instead of header file macros.
- Bug Fixes
  - Fixed the violations of MISRA C-2012 rule 10.1, 10.3, 10.4 and 14.3.

#### [2.7.1]

- Improvements
  - Corrected descriptions of several functions.
  - Improved function LPADC\_GetOffsetValue and LPADC\_SetOffsetValue.
  - Revert changes of feature macros for lpadc.
  - Use feature macros instead of header file macros.
- Bug Fixes
  - Fixed the violations of MISRA C-2012 rule 10.8.
  - Fixed the violations of MISRA C-2012 rule 10.1, 10.3, 10.4 and 14.3.

#### [2.7.0]

- Improvements
  - Added supports of CFG2 register.
  - Removed some useless macros.

#### [2.6.2]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules.
  - Fixed LPADC driver code compile error issue.

### [2.6.1]

- Improvements
  - Updated the use of macros in the driver code.

### [2.6.0]

- Improvements
  - Added the API LPADC\_SetOffset12BitValue() to configure 12bit ADC conversion offset trim value manually.
  - Added the API LPADC\_SetOffset16BitValue() to configure 16bit ADC conversion offset trim value manually.
  - Added API to set offset calibration mode.
  - Added configuration of alternate channel.
  - Updated auto calibration API and added calibration value conversion API.
- New feature
  - Added API LPADC\_EnableHardwareTriggerCommandSelection() to enable trigger commands controlled by ADC\_ETC.
  - Updated LPADC\_DoAutoCalibration() to allow doing something else before the ADC initialization to be totally complete. Enhance initialization duration time of the ADC.
  - Added two new APIs to get/set calibration value.

### [2.5.2]

- Improvements
  - Added while loop, LPADC\_GetConvResult() will return only when the FIFO will not be empty.

### [2.5.1]

- Bug Fixes
  - Fixed some typos in Lpadc driver comments.

### [2.5.0]

- Improvements
  - Added missing items to enable trigger interrupts.

### [2.4.0]

- New features
  - Added APIs to get/clear trigger status flags.

### [2.3.0]

- Improvements
  - Removed LPADC\_MeasureTemperature() function for the LPADC supports different temperature sensor calculation equations.

### [2.2.1]

- Improvements
  - Optimized LPADC\_MeasureTemperature() function to support the specific series with flash solidified calibration value.
  - Clean doxygen warnings.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.3, rule 10.8 and rule 17.7.

### [2.2.0]

- New Feature
  - Added API LPADC\_MeasureTemperature() to get correct temperature from the internal sensor.
- Improvements
  - Separated lpadc\_conversion\_resolution\_mode\_t with related feature macro.
- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules:
    - \* Rule 10.3, 10.4, 10.6, 10.7 and 17.7.

### [2.1.1]

- Improvements
  - Updated the gain calibration formula.
  - Used feature to segregate the new item kLPADC\_TriggerPriorityPreemptSubsequently.

### [2.1.0]

- New Features
  - Added the API LPADC\_SetOffsetValue() to support configure offset trim value manually.
  - Added the API LPADC\_DoOffsetCalibration() to do offset calibration independently.
- Improvements
  - Improved the usage of macros and removed invalid macros.

### [2.0.2]

- Improvements
  - Added support for platforms with 2 FIFOs and different calibration measures.

### [2.0.1]

- Bug Fixes
  - Ensured the API LPADC\_SetConvCommandConfig configure related registers correctly.

**[2.0.0]**

- Initial version.
- 

**LPCMP****[2.3.2]**

- Improvements
  - Fixed LPCMP CERT-C issues.

**[2.3.1]**

- Improvements
  - Update LPCMP driver to be compatible with platforms that do not support LPCMP nano power mode selection.

**[2.3.0]**

- New Feature
  - Added some new features for platforms which support
    - \* Plus input source selection.
    - \* Minus input source selection.
    - \* CMP to DAC link.
- Improvements
  - Removed some new features for platforms which doesn't support
    - \* Functional clock source selection.
    - \* DAC high power mode selection.
    - \* Round Robin clock source selection.
    - \* Round Robin trigger source selection.
    - \* Round Robin channel sample numbers setting.
    - \* Round Robin channel sample time threshold setting.
    - \* Round Robin internal trigger configuration.

**[2.2.0]**

- Improvements
  - Change `FSL_FEATURE_LPCMP_HAS_NO_CCR0_CMP_STOP_EN` to `FSL_FEATURE_LPCMP_HAS_CCR0_CMP_STOP_EN`.

**[2.1.3]**

- New Feature
  - Added new macro to handle the case where some instances do not have the CCR0 CMP\_STOP\_EN bit field.

### [2.1.2]

- New Feature
  - Add macros to be compatible with some platforms that do not have the CCR0 CMP\_STOP\_EN bitfield.

### [2.1.1]

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

### [2.1.0]

- New Features:
  - Supported round robin mode and window mode feature.

### [2.0.3]

- Bug Fixes:
  - Fixed the violation of MISRA-2012 rule 17.7.

### [2.0.2]

- Bug Fixes:
  - The current API LPCMP\_ClearStatusFlags has to check w1c bits.

### [2.0.1]

- Added control macro to enable/disable the CLOCK code in current driver.

### [2.0.0]

- Initial version.
- 

## LPI2C

### [2.6.2]

- Improvements
  - Added timeout for while loop in LPI2C\_TransferStateMachineSendCommand().

### [2.6.1]

- Bug Fixes
  - Fixed coverity issues.

**[2.6.0]**

- New Feature
  - Added common IRQ handler entry LPI2C\_DriverIRQHandler.

**[2.5.7]**

- Improvements
  - Added support for separated IRQ handlers.

**[2.5.6]**

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

**[2.5.5]**

- Bug Fixes
  - Fixed LPI2C\_SlaveInit() - allow to disable SDA/SCL glitch filter.

**[2.5.4]**

- Bug Fixes
  - Fixed LPI2C\_MasterTransferBlocking() - the return value was sometime affected by call of LPI2C\_MasterStop().

**[2.5.3]**

- Improvements
  - Added handler for LPI2C7 and LPI2C8.

**[2.5.2]**

- Bug Fixes
  - Fixed ERR051119 to ignore the nak flag when IGNACK=1 in LPI2C\_MasterCheckAndClearError.

**[2.5.1]**

- Bug Fixes
  - Added bus stop incase of bus stall in LPI2C\_MasterTransferBlocking.
- Improvements
  - Release peripheral from reset if necessary in init function.

**[2.5.0]**

- New Features
  - Added new function LPI2C\_SlaveEnableAckStall to enable or disable ACKSTALL.

#### [2.4.1]

- Improvements
  - Before master transfer with transactional APIs, enable master function while disable slave function and vise versa for slave transfer to avoid the one affecting the other.

#### [2.4.0]

- Improvements
  - Split some functions, fixed CCM problem in file fsl\_lpi2c.c.
- Bug Fixes
  - Fixed bug in LPI2C\_MasterInit that the MCFGR2's value set in LPI2C\_MasterSetBaudRate may be overwritten by mistake.

#### [2.3.2]

- Improvements
  - Initialized the EDMA configuration structure in the LPI2C EDMA driver.

#### [2.3.1]

- Improvements
  - Updated LPI2C\_GetCyclesForWidth to add the parameter of minimum cycle, because for master SDA/SCL filter, master bus idle/pin low timeout and slave SDA/SCL filter configuration, 0 means disabling the feature and cannot be used.
- Bug Fixes
  - Fixed bug in LPI2C\_SlaveTransferHandleIRQ that when restart detect event happens the transfer structure should not be cleared.
  - Fixed bug in LPI2C\_RunTransferStateMachine, that when only slave address is transferred or there is still data remaining in tx FIFO the last byte's nack cannot be ignored.
  - Fixed bug in slave filter doze enable, that when FILTDZ is set it means disable rather than enable.
  - Fixed bug in the usage of LPI2C\_GetCyclesForWidth. First its return value cannot be used directly to configure the slave FILTSDA, FILTSCL, DATAVD or CLKHOLD, because the real cycle width for them should be FILTSDA+3, FILTSCL+3, FILTSCL+DATAVD+3 and CLKHOLD+3. Second when cycle period is not affected by the prescaler value, prescaler value should be passed as 0 rather than 1.
  - Fixed wrong default setting for LPI2C slave. If enabling the slave tx SCL stall, then the default clock hold time should be set to 250ns according to I2C spec for 100kHz standard mode baudrate.
  - Fixed bug that before pushing command to the tx FIFO the FIFO occupation should be checked first in case FIFO overflow.

#### [2.3.0]

- New Features
  - Supported reading more than 256 bytes of data in one transfer as master.
  - Added API LPI2C\_GetInstance.
- Bug Fixes

- Fixed bug in LPI2C\_MasterTransferAbortEDMA, LPI2C\_MasterTransferAbort and LPI2C\_MasterTransferHandleIRQ that before sending stop signal whether master is active and whether stop signal has been sent should be checked, to make sure no FIFO error or bus error will be caused.
- Fixed bug in LPI2C master EDMA transactional layer that the bus error cannot be caught and returned by user callback, by monitoring bus error events in interrupt handler.
- Fixed bug in LPI2C\_GetCyclesForWidth that the parameter used to calculate clock cycle should be  $2^{\text{prescaler}}$  rather than prescaler.
- Fixed bug in LPI2C\_MasterInit that timeout value should be configured after baudrate, since the timeout calculation needs prescaler as parameter which is changed during baudrate configuration.
- Fixed bug in LPI2C\_MasterTransferHandleIRQ and LPI2C\_RunTransferStateMachine that when master writes with no stop signal, need to first make sure no data remains in the tx FIFO before finishes the transfer.

#### [2.2.0]

- Bug Fixes
  - Fixed issue that the SCL high time, start hold time and stop setup time do not meet I2C specification, by changing the configuration of data valid delay, setup hold delay, clock high and low parameters.
  - MISRA C-2012 issue fixed.
    - \* Fixed rule 8.4, 13.5, 17.7, 20.8.

#### [2.1.12]

- Bug Fixes
  - Fixed MISRA advisory 15.5 issues.

#### [2.1.11]

- Bug Fixes
  - Fixed the bug that, during master non-blocking transfer, after the last byte is sent/received, the kLPI2C\_MasterNackDetectFlag is expected, so master should not check and clear kLPI2C\_MasterNackDetectFlag when remainingBytes is zero, in case FIFO is emptied when stop command has not been sent yet.
  - Fixed the bug that, during non-blocking transfer slave may nack master while master is busy filling tx FIFO, and NDF may not be handled properly.

#### [2.1.10]

- Bug Fixes
  - MISRA C-2012 issue fixed.
    - \* Fixed rule 10.3, 14.4, 15.5.
  - Fixed unaligned access issue in LPI2C\_RunTransferStateMachine.
  - Fixed uninitialized variable issue in LPI2C\_MasterTransferHandleIRQ.



- Used linked TCD to disable tx and enable rx in read operation to fix the issue that for platform sharing the same DMA request with tx and rx, during LPI2C read operation if interrupt with higher priority happened exactly after command was sent and before tx disabled, potentially both tx and rx could trigger dma and cause trouble.
- Fixed MISRA issues.
  - \* Fixed rules 10.1, 10.3, 10.4, 11.6, 11.9, 14.4, 17.7.
- Fixed the waitTimes variable not re-assignment issue for each byte read.
- New Features
  - Added the IRQHandler for LPI2C5 and LPI2C6 instances.
- Improvements
  - Updated the LPI2C\_WAIT\_TIMEOUT macro to unified name I2C\_RETRY\_TIMES.

#### [2.1.9]

- Bug Fixes
  - Fixed Coverity issue of unchecked return value in I2C\_RTOS\_Transfer.
  - Fixed Coverity issue of operands did not affect the result in LPI2C\_SlaveReceive and LPI2C\_SlaveSend.
  - Removed STOP signal wait when NAK detected.
  - Cleared slave repeat start flag before transmission started in LPI2C\_SlaveSend/LPI2C\_SlaveReceive. The issue was that LPI2C\_SlaveSend/LPI2C\_SlaveReceive did not handle with the reserved repeat start flag. This caused the next slave to send a break, and the master was always in the receive data status, but could not receive data.

#### [2.1.8]

- Bug Fixes
  - Fixed the transfer issue with LPI2C\_MasterTransferNonBlocking, kLPI2C\_TransferNoStopFlag, with the wait transfer done through callback in a way of not doing a blocking transfer.
  - Fixed the issue that STOP signal did not appear in the bus when NAK event occurred.

#### [2.1.7]

- Bug Fixes
  - Cleared the stopflag before transmission started in LPI2C\_SlaveSend/LPI2C\_SlaveReceive. The issue was that LPI2C\_SlaveSend/LPI2C\_SlaveReceive did not handle with the reserved stop flag and caused the next slave to send a break, and the master always stayed in the receive data status but could not receive data.

#### [2.1.6]

- Bug Fixes
  - Fixed driver MISRA build error and C++ build error in LPI2C\_MasterSend and LPI2C\_SlaveSend.
  - Reset FIFO in LPI2C Master Transfer functions to avoid any byte still remaining in FIFO during last transfer.

- Fixed the issue that LPI2C\_MasterStop did not return the correct NAK status in the bus for second transfer to the non-existing slave address.

#### [2.1.5]

- Bug Fixes
  - Extended the Driver IRQ handler to support LPI2C4.
  - Changed to use ARRAY\_SIZE(kLpi2cBases) instead of FEATURE COUNT to decide the array size for handle pointer array.

#### [2.1.4]

- Bug Fixes
  - Fixed the LPI2C\_MasterTransferEDMA receive issue when LPI2C shared same request source with TX/RX DMA request. Previously, the API used scatter-gather method, which handled the command transfer first, then the linked TCD which was pre-set with the receive data transfer. The issue was that the TX DMA request and the RX DMA request were both enabled, so when the DMA finished the first command TCD transfer and handled the receive data TCD, the TX DMA request still happened due to empty TX FIFO. The result was that the RX DMA transfer would start without waiting on the expected RX DMA request.
  - Fixed the issue by enabling IntMajor interrupt for the command TCD and checking if there was a linked TCD to disable the TX DMA request in LPI2C\_MasterEDMACallback API.

#### [2.1.3]

- Improvements
  - Added LPI2C\_WATI\_TIMEOUT macro to allow the user to specify the timeout times for waiting flags in functional API and blocking transfer API.
  - Added LPI2C\_MasterTransferBlocking API.

#### [2.1.2]

- Bug Fixes
  - In LPI2C\_SlaveTransferHandleIRQ, reset the slave status to idle when stop flag was detected.

#### [2.1.1]

- Bug Fixes
  - Disabled the auto-stop feature in eDMA driver. Previously, the auto-stop feature was enabled at transfer when transferring with stop flag. Since transfer was without stop flag and the auto-stop feature was enabled, when starting a new transfer with stop flag, the stop flag would be sent before the new transfer started, causing unsuccessful sending of the start flag, so the transfer could not start.
  - Changed default slave configuration with address stall false.

#### [2.1.0]

- Improvements
  - API name changed:
    - \* LPI2C\_MasterTransferCreateHandle -> LPI2C\_MasterCreateHandle.
    - \* LPI2C\_MasterTransferGetCount -> LPI2C\_MasterGetTransferCount.
    - \* LPI2C\_MasterTransferAbort -> LPI2C\_MasterAbortTransfer.
    - \* LPI2C\_MasterTransferHandleIRQ -> LPI2C\_MasterHandleInterrupt.
    - \* LPI2C\_SlaveTransferCreateHandle -> LPI2C\_SlaveCreateHandle.
    - \* LPI2C\_SlaveTransferGetCount -> LPI2C\_SlaveGetTransferCount.
    - \* LPI2C\_SlaveTransferAbort -> LPI2C\_SlaveAbortTransfer.
    - \* LPI2C\_SlaveTransferHandleIRQ -> LPI2C\_SlaveHandleInterrupt.

#### [2.0.0]

- Initial version.
- 

### LPI2C\_EDMA

#### [2.4.5]

- Improvements
  - Added condition to IRQ handler to check whether the interrupt is enabled - kLPI2C\_MasterTxReadyFlag.

#### [2.4.4]

- Improvements
  - Added support for 2KB data transfer

#### [2.4.3]

- Improvements
  - Added support for separated IRQ handlers.

#### [2.4.2]

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

#### [2.4.1]

- Refer LPI2C driver change log 2.0.0 to 2.4.1
-

## LPSPi

### [2.7.3]

- Improvements
  - Added timeout for while loop in LPSPi\_MasterTransferWriteAllTxData().
  - Make SPI\_RETRY\_TIMES configurable by CONFIG\_SPI\_RETRY\_TIMES.

### [2.7.2]

- Bug Fixes
  - Fixed coverity issues.

### [2.7.1]

- Bug Fixes
  - Workaround for errata ERR050607
  - Workaround for errata ERR010655

### [2.7.0]

- New Feature
  - Added common IRQ handler entry LPSPi\_DriverIRQHandler.

### [2.6.10]

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

### [2.6.9]

- Bug Fixes
  - Fixed reading of TCR register
  - Workaround for errata ERR050606

### [2.6.8]

- Bug Fixes
  - Fixed build error when SPI\_RETRY\_TIMES is defined to non-zero value.

### [2.6.7]

- Bug Fixes
  - Fixed the txData from void \* to const void \* in transmit API \_lpspi\_master\_handle and \_lpspi\_slave\_handle.

#### [2.6.6]

- Bug Fixes
  - Added LPSPI register init in LPSPI\_MasterInit incase of LPSPI register exist.

#### [2.6.5]

- Improvements
  - Introduced FSL\_FEATURE\_LPSPI\_HAS\_NO\_PCSCFG and FSL\_FEATURE\_LPSPI\_HAS\_NO\_MULTI\_WIDTH for conditional compile.
  - Release peripheral from reset if necessary in init function.

#### [2.6.4]

- Bug Fixes
  - Added LPSPI6\_DriverIRQHandler for LPSPI6 instance.

#### [2.6.3]

- Hot Fixes
  - Added macro switch in function LPSPI\_Enable about ERRATA051472.

#### [2.6.2]

- Bug Fixes
  - Disabled lpspi before LPSPI\_MasterSetBaudRate incase of LPSPI opened.

#### [2.6.1]

- Bug Fixes
  - Fixed return value while calling LPSPI\_WaitTxFifoEmpty in function LPSPI\_MasterTransferNonBlocking.

#### [2.6.0]

- Feature
  - Added the new feature of multi-IO SPI .

#### [2.5.3]

- Bug Fixes
  - Fixed 3-wire txmask of handle vaule reentrant issue.

#### [2.5.2]

- Bug Fixes
  - Workaround for errata ERR051588 by clearing FIFO after transmit underrun occurs.

**[2.5.1]**

- Bug Fixes
  - Workaround for errata ERR050456 by resetting the entire module using LPSPI<sub>CR</sub>[RST] bit.

**[2.5.0]**

- Bug Fixes
  - Workaround for errata ERR011097 to wait the TX FIFO to go empty when writing TCR register and TCR[TXMSK] value is 1.
  - Added API LPSPI\_WaitTxFifoEmpty for wait the txfifo to go empty.

**[2.4.7]**

- Bug Fixes
  - Fixed bug that the SR[REF] would assert if software disabled or enabled the LPSPI module in LPSPI\_Enable.

**[2.4.6]**

- Improvements
  - Moved the configuration of registers for the 3-wire lpspi mode to the LPSPI\_MasterInit and LPSPI\_SlaveInit function.

**[2.4.5]**

- Improvements
  - Improved LPSPI\_MasterTransferBlocking send performance when frame size is 1-byte.

**[2.4.4]**

- Bug Fixes
  - Fixed LPSPI\_MasterGetDefaultConfig incorrect default inter-transfer delay calculation.

**[2.4.3]**

- Bug Fixes
  - Fixed bug that the ISR response speed is too slow on some platforms, resulting in the first transmission of overflow, Set proper RX watermarks to reduce the ISR response times.

**[2.4.2]**

- Bug Fixes
  - Fixed bug that LPSPI\_MasterTransferBlocking will modify the parameter txbuff and rxbuff pointer.

#### [2.4.1]

- Bug Fixes
  - Fixed bug that LPSPI\_SlaveTransferNonBlocking can't detect RX error.

#### [2.4.0]

- Improvements
  - Split some functions, fixed CCM problem in file fsl\_lpspi.c.

#### [2.3.1]

- Improvements
  - Initialized the EDMA configuration structure in the LPSPI EDMA driver.
- Bug Fixes
  - Fixed bug that function LPSPI\_MasterTransferBlocking should return after the transfer complete flag is set to make sure the PCS is re-asserted.

#### [2.3.0]

- New Features
  - Supported the master configuration of sampling the input data using a delayed clock to improve slave setup time.

#### [2.2.1]

- Bug Fixes
  - Fixed bug in LPSPI\_SetPCSContinuous when disabling PCS continuous mode.

#### [2.2.0]

- Bug Fixes
  - Fixed bug in 3-wire polling and interrupt transfer that the received data is not correct and the PCS continuous mode is not working.

#### [2.1.0]

- Improvements
  - Improved LPSPI\_SlaveTransferHandleIRQ to fill up TX FIFO instead of write one data to TX register which improves the slave transmit performance.
  - Added new functional APIs LPSPI\_SelectTransferPCS and LPSPI\_SetPCSContinuous to support changing PCS selection and PCS continuous mode.
- Bug Fixes
  - Fixed bug in non-blocking and EDMA transfer APIs that kStatus\_InvalidArgument is returned if user configures 3-wire mode and full-duplex transfer at the same time, but transfer state is already set to kLPSPI\_Busy by mistake causing following transfer can not start.
  - Fixed bug when LPSPI slave using EDMA way to transfer, tx should be masked when tx data is null, otherwise in 3-wire mode which tx/rx use the same pin, the received data will be interfered.

**[2.0.5]**

- Improvements
  - Added timeout mechanism when waiting certain states in transfer driver.
- Bug Fixes
  - Fixed the bug that LPSPI can not transfer large data using EDMA.
  - Fixed MISRA 17.7 issues.
  - Fixed variable overflow issue introduced by MISRA fix.
  - Fixed issue that rxFifoMaxBytes should be calculated according to transfer width rather than FIFO width.
  - Fixed issue that completion flag was not cleared after transfer completed.

**[2.0.4]**

- Bug Fixes
  - Fixed in LPSPI\_MasterTransferBlocking that master rxfifo may overflow in stall condition.
  - Eliminated IAR Pa082 warnings.
  - Fixed MISRA issues.
    - \* Fixed rules 10.1, 10.3, 10.4, 10.6, 11.9, 14.2, 14.4, 15.7, 17.7.

**[2.0.3]**

- Bug Fixes
  - Removed LPSPI\_Reset from LPSPI\_MasterInit and LPSPI\_SlaveInit, because this API may glitch the slave select line. If needed, call this function manually.

**[2.0.2]**

- New Features
  - Added dummy data set up API to allow users to configure the dummy data to be transferred.
  - Enabled the 3-wire mode, SIN and SOUT pins can be configured as input/output pin.

**[2.0.1]**

- Bug Fixes
  - Fixed the bug that the clock source should be divided by the PRESCALE setting in LPSPI\_MasterSetDelayTimes function.
  - Fixed the bug that LPSPI\_MasterTransferBlocking function would hang in some corner cases.
- Optimization
  - Added #ifndef/#endif to allow user to change the default TX value at compile time.



## [2.0.0]

- Initial version.
- 

## LPSPI\_EDMA

### [2.4.9]

- Improvements
  - Removed unused code from LPSPI\_SeparateEdmaReadData().

### [2.4.8]

- Improvements
  - Added timeout for while loop in EDMA\_LpspiMasterCallback() and EDMA\_LpspiSlaveCallback().

### [2.4.7]

- Bug Fixes
  - Add macro LPSPI\_ALIGN\_TCD\_SIZE\_MASK to align an address to edma\_tcd\_t size.

### [2.4.6]

- Improvements
  - Increased transmit FIFO watermark to ensure whole transmit FIFO will be used during data transfer.

### [2.4.5]

- Bug Fixes
  - Fixed reading of TCR register
  - Workaround for errata ERR050606

### [2.4.4]

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

### [2.4.3]

- Improvements
  - Supported 32K bytes transmit in DMA, improve the max datasize in LPSPI\_MasterTransferEDMALite.

### [2.4.2]

- Improvements
  - Added callback status in EDMA\_LpspiMasterCallback and EDMA\_LpspiSlaveCallback to check transferDone.

**[2.4.1]**

- Improvements
  - Add the TXMSK wait after TCR setting.

**[2.4.0]**

- Improvements
    - Separated LPSPI\_MasterTransferEDMA functions to LP-SPI\_MasterTransferPrepareEDMA and LPSPI\_MasterTransferEDMALite to optimize the process of transfer.
- 

**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

### [2.10.0]

- 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 controlled 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 lpuart 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 accurate baudrate setting. Changed `osr` from `uint32_t` to `uint8_t` since `osr`'s biggest 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
-



## OSTIMER

### [2.2.6]

- Improvements
  - Drop the check of MATCH\_WR\_RDY and ostimer counter value in OSTIMER\_SetMatchRawValue. In most applications, they are useless and may bring at least 7 OSTimer ticks latency, which is unacceptable when OSTimer is working under slow peripheral clock.
  - Optimize software gray code to binary conversion: replaced loop-based implementation with branchless bitwise operations.

### [2.2.5]

- Improvements
  - Support binary encoded ostimer.

### [2.2.4]

- Bug Fixes
  - Fixed CERT INT31-C violations.

### [2.2.3]

- Improvements
  - Disable and clear pending interrupts before disabling the OSTIMER clock to avoid interrupts being executed when the clock is already disabled.

### [2.2.2]

- Improvements
  - Support devices with different OSTIMER instance name.

### [2.2.1]

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

### [2.2.0]

- Improvements
  - Move the PMC operation out of the OSTIMER driver to board specific files.
  - Added low level APIs to control OSTIMER MATCH and interrupt.

### [2.1.2]

- Bug Fixes
  - Fixed MISRA-2012 rule 10.8.

**[2.1.1]**

- Bug Fixes
  - removes the suffix ‘n’ for some register names and bit fields’ names
- Improvements
  - Added HW CODE GRAY feature supported by CODE GRAY in SYSCTRL register group.

**[2.1.0]**

- Bug Fixes
  - Added a workaround to fix the issue that no interrupt was reported when user set smaller period.
  - Fixed violation of MISRA C-2012 rule 10.3 and 11.9.
- Improvements
  - Added return value for the two APIs to set match value.
    - \* OSTIMER\_SetMatchRawValue
    - \* OSTIMER\_SetMatchValue

**[2.0.3]**

- Bug Fixes
  - Fixed violation of MISRA C-2012 rule 10.3, 14.4, 17.7.

**[2.0.2]**

- Improvements
  - Added support for OSTIMER0

**[2.0.1]**

- Improvements
  - Removed the software reset function out of the initialization API.
  - Enabled interrupt directly instead of enabling deep sleep interrupt. Users need to enable the deep sleep interrupt in application code if needed.

**[2.0.0]**

- Initial version.
- 

**PORT****[2.5.1]**

- Bug Fixes
  - Fix CERT INT31-C issues.

#### [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.

## PWM

### [2.9.1]

- Improvements
  - Add new API `PWM_SetupFaultsExt` and `PWM_SetupFaultInputFilterExt` to support Flex-PWM which has more than one fault input channels.
  - Support fault 4-7 interrupt and its flag.
- Bug Fixes
  - Fixed violations of the CERT INT31-C.

### [2.9.0]

- Improvements
  - Support PWMX channel output for edge aligned PWM.
  - Forbid submodule 0 counter initialize with master sync and master reload mode.
  - Clarify `kPWM_BusClock` meaning.
  - Verify `pulseCnt` within 65535 when update period register.

### [2.8.4]

- Improvements
  - Support workaround for ERR051989. This function helps realize no phase delay between submodule 0 and other submodule.

### [2.8.3]

- Bug Fixes
  - Fixed MISRA C-2012 Rule 15.7

### [2.8.2]

- Bug Fixes
  - Fixed warning conversion from 'int' to 'uint16\_t' on API `PWM_Init`.
  - Fixed warning unused variable 'reg' on API `PWM_SetPwmForceOutputToZero`.

### [2.8.1]

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

### [2.8.0]

- Improvements
  - Added API `PWM_UpdatePwmPeriodAndDutycycle` to update the PWM signal's period and dutycycle for a PWM submodule.

- Added API PWM\_SetPeriodRegister and PWM\_SetDutycycleRegister to merge duplicate code in API PWM\_SetupPwm, PWM\_UpdatePwmDutycycleHighAccuracy and PWM\_UpdatePwmPeriodAndDutycycle

#### [2.7.1]

- Improvements
  - Supported UPDATE\_MASK bit in MASK register.

#### [2.7.0]

- Improvements
  - Supported platforms which don't have Capture feature with channel A and B.
  - Supported platforms which don't have Submodule 3.
  - Added assert function in API PWM\_SetPhaseDelay to prevent wrong argument.

#### [2.6.1]

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

#### [2.6.0]

- Improvements
  - Added API PWM\_SetPhaseDelay to set the phase delay from the master sync signal of submodule 0.
  - Added API PWM\_SetFilterSampleCount to set number of consecutive samples that must agree prior to the input filter.
  - Added API PWM\_SetFilterSamplePeriod to set the sampling period of the fault pin input filter.

#### [2.5.1]

- Bug Fixes
  - Fixed MISRA C-2012 rules: 10.1, 10.3, 10.4, 10.6 and 10.8.
  - Fixed the issue that PWM\_UpdatePwmDutycycle() can't update duty cycle status value correct.

#### [2.5.0]

- Improvements
  - Added API PWM\_SetOutputToIdle to set pwm channel output to idle.
  - Added API PWM\_GetPwmChannelState to get the pwm channel output duty cycle value.
  - Added API PWM\_SetPwmForceOutputToZero to set the pwm channel output to zero logic.
  - Added API PWM\_SetChannelOutput to set the pwm channel output state.
  - Added API PWM\_SetClockMode to set the value of the clock prescaler.

- Added API PWM\_SetupPwmPhaseShift to set PWM which a special phase shift and 50% duty cycle.
- Added API PWM\_SetVALxValue/PWM\_GetVALxValue to set/get PWM VALs registers values directly.

#### [2.4.0]

- Improvements
  - Supported the PWM which can't work in wait mode.

#### [2.3.0]

- Improvements
  - Add PWM output enable&disbale API for SDK.
- Bug Fixes
  - Fixed changing channel B configuration when parameter is kPWM\_PWMX and PWMX configuration is not supported yet.

#### [2.2.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rules: 10.3, 10.4.
- Bug Fixes
  - Fixed the issue that PWM drivers computed VAL1 improperly.
- Improvements
  - Updated calculation accuracy of reloadValue in dutyCycleToReloadValue function.

#### [2.2.0]

- Improvements
  - Added new enumeration and two APIs to support enabling and disabling one or more PWM output triggers.
  - Added a new function to make the most of 16-bit resolution PWM.
  - Added one API to support updating fault status of PWM output.
  - Added one API to support PWM DMA write request.
  - Added three APIs to support PWM DMA capture read request.
  - Added one API to support get default fault config of PWM.
  - Added one API to support setting PWM fault disable mapping.

#### [2.1.0]

- Improvements
  - Moved the configuration of fault input filter into a new API to avoid be initialized multiple times.
- Bug Fixes
  - MISRA C-2012 issue fixed.

- \* Fix rules, containing: rule-10.2, rule-10.3, rule-10.4, rule-10.7, rule-10.8, rule-14.4, rule-16.4.

#### [2.0.1]

- Bug Fixes
  - Fixed the issue that PWM submodule may be initialized twice in function PWM\_SetupPwm().

#### [2.0.0]

- Initial version.
- 

### RESET

#### [2.4.0]

- Improvements
  - Add RESET\_ReleasePeripheralReset API.

#### [2.0.0]

- Initial version.
- 

### ROMAPI

#### [2.0.1]

- Add ROMAPI\_BASE feature to support MCXA276.

#### [2.0.0]

- Initial version.
- 

### MCX\_SPC

#### [2.10.0]

- New Features
  - Add feature macro to be compatible with some platforms where SPC does not support:
    1. Has no SC register SPC\_LP\_REQ bitfield.
    2. Has no SC register SPC\_LP\_MODE bitfield.
    3. Has no PD\_STATUS register.
    4. Has no SRAMCTL register.

**[2.9.1]**

- Improvements
  - There is no need to ensure that the bandgap is enabled before setting the active mode core LDO regulator drive strength and voltage level, because the hardware will automatically turn on the bandgap function if it is required by the hardware when configuring the LDO.

**[2.9.0]**

- New Features
  - Add feature macro to be compatible with some platforms where SPC does not support overdrive voltage.

**[2.8.1]**

- Improvements
  - Fixed `cert_int31_c_violation` of `SPC_GetVoltageDetectStatusFlag()`, `SPC_GetExternalDomainsStatus()` and `SPC_SetDCDCBurstConfig`.

**[2.8.0]**

- New Features
  - Updated `MCX_SPC` driver for compatibility with SoCs without `PD_STATUS[PWR_REQ_STATUS]`.

**[2.7.0]**

- New Features
  - Added new function to unretain RAM array.

**[2.6.0]**

- Bug Fixes
  - The enumeration `kSPC_DeepPowerDownWithSysClockOff` should be `0x8U`.
- New Features
  - Added functions to get the last low-power mode that the power domain requested.

**[2.5.0]**

- Improvements
  - Updated `SPC_SetLowPowerModeCoreLDORegulatorConfig()` with adding check that `LP_CFG[CORELDO_VDD_LVL]` only be updated when CORELDO is in normal driver strength in active mode.
  - Updated `SPC_SetLowPowerModeRegulatorsConfig()` with adding check the if DCDC voltage level set as retention mode, CORELDO low voltage detection must be disabled.
  - Updated `spc_analog_module_control` with adding Opamp3 support.
- New Features
  - Added functions to mask/unmask voltage detections in active mode.



#### [2.4.2]

- Improvements
  - Fixed the violation of MISRA C-2012 rules.

#### [2.4.1]

- Improvements
  - Fixed the violation of MISRA C-2012 rules.

#### [2.4.0]

- Improvements
  - Refined APIs to set regulators, the input parameters will be check firstly before setting register.
  - Improved APIs' document.
  - Removed software check for DCDC's settings since there is not hardware restrictions for DCDC.
  - Added new APIs to check if glitch detector is enabled in active/low power mode.
  - Added new APIs for DCDC burst feature, make it more flexible to use.
  - Added new API to enable/disable DCDC bleed resistor.
  - Set functions of VD\_SYS\_CFG[LVSEL] configuration as deprecated, since this bit filed is reserved for all devices.
  - Updated details of spc\_core\_ldo\_voltage\_level\_t to align with description of latest RM, added comments to reminder users that the retention voltage is reserved in some devices.

#### [2.3.0]

- New Features
  - Updated glitch detect API to support devices which do not have aGDET

#### [2.2.1]

- Bug Fixes
  - Fixed an issue of SPC\_SetActiveModeRegulatorsConfig() which will cause LVD in case of setting DCDC and CORE LDO into higher voltage level.

#### [2.2.0]

- New Features
  - Added some macros to support some devices(such as MCXA family) do not equipped DCDC, SYS\_LDO and so on.
  - Added new function SPC\_EnableSRAMLdOLowPowerModeIREF(), SPC\_TrimSRAMLdoRefVoltage(), SPC\_EnableSRAMLdo() to support some devices(such as MCXA family) that support control of SRAM\_LDO.
  - Fixed violation of MISRA C-2012 rule 17.7.
  - Fixed an issue in SPC\_SetLowPowerModeRegulatorsConfig() function that set ACTIVE\_CFG register by mistake.

**[2.1.0]**

- Improvements
  - Added new functions to set regulators' voltage level and drive strength individually.
  - Updated SPC\_SetActiveModeRegulatorsConfig() and SPC\_SetLowPowerModeRegulatorsConfig() based on new added functions.

**[2.0.1]**

- Bug Fixes
  - Fixed a bug that external burst not working properly.
  - Fixed a bug that is SPC has VDD\_DS the voltage is not configured correctly.

**[2.0.0]**

- Initial version.
- 

**TRDC****[2.2.2]**

- Improvements
  - Update APIs to check whether the memory access configuration can be updated.
  - Update APIs to mask the MRC address since only high 18 bits are valid.

**[2.2.1]**

- Bug Fixes:
  - Fix MISRA violations.

**[2.2.0]**

- New Features:
  - Supported SoCs that do not have all TRDC modules.

**[2.1.0]**

- Bug Fixes:
  - Fix MISRA violations.
  - Fixed wrong operation of domain mask in TRDC\_MbcNseClearAll and TRDC\_MrcDomainNseClear.

**[2.0.0]**

- Initial version.
-

## UTICK

### [2.0.5]

- Improvements
  - Improved for SOC RW610.

### [2.0.4]

- Bug Fixes
  - Fixed compile fail issue of no-supporting PD configuration in utick driver.

### [2.0.3]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rules: 8.4, 14.4, 17.7

### [2.0.2]

- Added new feature definition macro to enable/disable power control in drivers for some devices have no power control function.

### [2.0.1]

- Added control macro to enable/disable the CLOCK code in current driver.

### [2.0.0]

- Initial version.
- 

## MCX\_VBAT

### [2.5.0]

- New features
  - Update MCX\_VBAT driver to be compatible with new platforms.

### [2.4.0]

- New features
  - Update MCX\_VBAT driver to be compatible with new platforms.

### [2.3.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 low impact rules.

**[2.3.0]**

- Improvements
  - OSCCTLA[FINE\_AMP\_GAIN] is reserved, added new macro to support this change.
  - Defined VBAT\_LDORAMC\_RET in case of this macro is not defined in header file.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.8.

**[2.2.0]**

- Improvements
  - Added macros to support some devices(such as MCXA family) that do not support OSC control, LDO control, bandgap timer, and tamper.

**[2.1.0]**

- New features
  - Added functions to support tamper and clock monitor.

**[2.0.0]**

- Initial version.
- 

**WAKETIMER****[2.0.1]**

- Bug Fix
  - MISRA C-2012 issue fixed: rule 8.2, 8.8, 10.1, 10.4.

**[2.0.0]**

- Initial version.
- 

**WUU****[2.4.1]**

- Improvements
  - The semantics of kWUU\_FilterActiveDSPD and kWUU\_ExternalPinActiveDSPD are not clear enough, because on some platforms it's not 'DSPD' (deep sleep, power down) but PDDPD (power down, deep power down). We deprecate them and will use the more generic kWUU\_FilterActiveLowLeakage and kWUU\_ExternalPinActiveLowLeakage in the future.

#### [2.4.0]

- New Features
  - Added `WUU_ClearExternalWakeupPinsConfig()` to clear settings of PDC and PE register.

#### [2.3.0]

- New Features
  - Added `WUU_ClearInternalWakeUpModulesConfig()` to clear settings of DM and ME register.

#### [2.2.1]

- Bug Fixes
  - Fixed `WUU_SetPinFilterConfig()` unable to set edge detection of pin filter config.
  - Fixed wrong macro used in `WUU_GetPinFilterFlag()` function.

#### [2.2.0]

- New Features
  - Added the `WUU_GetExternalWakeupPinFlag()` and `WUU_ClearExternalWakeupPinFlag()` function .

#### [2.1.0]

- New Features
  - Added the `WUU_GetModuleInterruptFlag()` function to support the devices that equipped MF register.

#### [2.0.0]

- Initial version.
- 

## WWDT

#### [2.1.10]

- Bug Fixes
  - Chek `WWDT_RSTS` instead of `FSL_FEATURE_WWDT_HAS_NO_RESET` to determine whether the peripheral can be reset.

#### [2.1.9]

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

**[2.1.8]**

- Improvements
  - Updated the “WWDT\_Init” API to add wait operation. Which can avoid the TV value read by CPU still be 0xFF (reset value) after WWDT\_Init function returns.

**[2.1.7]**

- Bug Fixes
  - Fixed the issue that the watchdog reset event affected the system from PMC.
  - Fixed the issue of setting watchdog WDPROTECT field without considering the backwards compatibility.
  - Fixed the issue of clearing bit fields by mistake in the function of WWDT\_ClearStatusFlags.

**[2.1.5]**

- Bug Fixes
  - deprecated a unusable API in WWWDWT driver.
    - \* WWDT\_Disable

**[2.1.4]**

- Bug Fixes
  - Fixed violation of the MISRA C-2012 rules Rule 10.1, 10.3, 10.4 and 11.9.
  - Fixed the issue of the inseparable process interrupted by other interrupt source.
    - \* WWDT\_Init

**[2.1.3]**

- Bug Fixes
  - Fixed legacy issue when initializing the MOD register.

**[2.1.2]**

- Improvements
  - Updated the “WWDT\_ClearStatusFlags” API and “WWDT\_GetStatusFlags” API to match QN9090. WDTOF is not set in case of WD reset. Get info from PMC instead.

**[2.1.1]**

- New Features
  - Added new feature definition macro for devices which have no LCOK control bit in MOD register.
  - Implemented delay/retry in WWDT driver.

### [2.1.0]

- Improvements
  - Added new parameter in configuration when initializing WWDT module. This parameter, which must be set, allows the user to deliver the WWDT clock frequency.

### [2.0.0]

- Initial version.
- 

## 1.6 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.

[MCXA153](#)

## 1.7 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.

### 1.7.1 MCU Boot

[mcuboot\\_opensource](#)

### 1.7.2 FreeMASTER

[freemaster](#)

### 1.7.3 FreeRTOS

[FreeRTOS](#)

# Chapter 2

## MCXA153

### 2.1 AOI: Crossbar AND/OR/INVERT Driver

`void AOI_Init(AOI_Type *base)`

Initializes an AOI instance for operation.

This function un-gates the AOI clock.

#### Parameters

- `base` – AOI peripheral address.

`void AOI_Deinit(AOI_Type *base)`

Deinitializes an AOI instance for operation.

This function shutdowns AOI module.

#### Parameters

- `base` – AOI peripheral address.

`void AOI_GetEventLogicConfig(AOI_Type *base, aoi_event_t event, aoi_event_config_t *config)`

Gets the Boolean evaluation associated.

This function returns the Boolean evaluation associated.

Example:

```
aoi_event_config_t demoEventLogicStruct;

AOI_GetEventLogicConfig(AOI, kAOI_Event0, &demoEventLogicStruct);
```

#### Parameters

- `base` – AOI peripheral address.
- `event` – Index of the event which will be set of type `aoi_event_t`.
- `config` – Selected input configuration .

`void AOI_SetEventLogicConfig(AOI_Type *base, aoi_event_t event, const aoi_event_config_t *eventConfig)`

Configures an AOI event.

This function configures an AOI event according to the `aoiEventConfig` structure. This function configures all inputs (A, B, C, and D) of all product terms (0, 1, 2, and 3) of a desired event.

Example:



```
aoi_event_config_t demoEventLogicStruct;

demoEventLogicStruct.PT0AC = kAOI_InvInputSignal;
demoEventLogicStruct.PT0BC = kAOI_InputSignal;
demoEventLogicStruct.PT0CC = kAOI_LogicOne;
demoEventLogicStruct.PT0DC = kAOI_LogicOne;

demoEventLogicStruct.PT1AC = kAOI_LogicZero;
demoEventLogicStruct.PT1BC = kAOI_LogicOne;
demoEventLogicStruct.PT1CC = kAOI_LogicOne;
demoEventLogicStruct.PT1DC = kAOI_LogicOne;

demoEventLogicStruct.PT2AC = kAOI_LogicZero;
demoEventLogicStruct.PT2BC = kAOI_LogicOne;
demoEventLogicStruct.PT2CC = kAOI_LogicOne;
demoEventLogicStruct.PT2DC = kAOI_LogicOne;

demoEventLogicStruct.PT3AC = kAOI_LogicZero;
demoEventLogicStruct.PT3BC = kAOI_LogicOne;
demoEventLogicStruct.PT3CC = kAOI_LogicOne;
demoEventLogicStruct.PT3DC = kAOI_LogicOne;

AOI_SetEventLogicConfig(AOI, kAOI_Event0, demoEventLogicStruct);
```

### Parameters

- base – AOI peripheral address.
- event – Event which will be configured of type `aoi_event_t`.
- eventConfig – Pointer to type `aoi_event_config_t` structure. The user is responsible for filling out the members of this structure and passing the pointer to this function.

FSL\_AOI\_DRIVER\_VERSION

Version 2.0.2.

enum \_aoi\_input\_config

AOI input configurations.

The selection item represents the Boolean evaluations.

*Values:*

enumerator kAOI\_LogicZero

Forces the input to logical zero.

enumerator kAOI\_InputSignal

Passes the input signal.

enumerator kAOI\_InvInputSignal

Inverts the input signal.

enumerator kAOI\_LogicOne

Forces the input to logical one.

enum \_aoi\_event

AOI event indexes, where an event is the collection of the four product terms (0, 1, 2, and 3) and the four signal inputs (A, B, C, and D).

*Values:*

enumerator kAOI\_Event0

Event 0 index

enumerator kAOI\_Event1

Event 1 index

enumerator kAOI\_Event2

Event 2 index

enumerator kAOI\_Event3

Event 3 index

typedef enum *\_aoi\_input\_config* aoi\_input\_config\_t

AOI input configurations.

The selection item represents the Boolean evaluations.

typedef enum *\_aoi\_event* aoi\_event\_t

AOI event indexes, where an event is the collection of the four product terms (0, 1, 2, and 3) and the four signal inputs (A, B, C, and D).

typedef struct *\_aoi\_event\_config* aoi\_event\_config\_t

AOI event configuration structure.

Defines structure *\_aoi\_event\_config* and use the *AOI\_SetEventLogicConfig()* function to make whole event configuration.

AOI

AOI peripheral address

struct *\_aoi\_event\_config*

*#include <fsl\_aoi.h>* AOI event configuration structure.

Defines structure *\_aoi\_event\_config* and use the *AOI\_SetEventLogicConfig()* function to make whole event configuration.

## Public Members

*aoi\_input\_config\_t* PT0AC

Product term 0 input A

*aoi\_input\_config\_t* PT0BC

Product term 0 input B

*aoi\_input\_config\_t* PT0CC

Product term 0 input C

*aoi\_input\_config\_t* PT0DC

Product term 0 input D

*aoi\_input\_config\_t* PT1AC

Product term 1 input A

*aoi\_input\_config\_t* PT1BC

Product term 1 input B

*aoi\_input\_config\_t* PT1CC

Product term 1 input C

*aoi\_input\_config\_t* PT1DC

Product term 1 input D

*aoi\_input\_config\_t* PT2AC

Product term 2 input A

*aoi\_input\_config\_t* PT2BC  
Product term 2 input B

*aoi\_input\_config\_t* PT2CC  
Product term 2 input C

*aoi\_input\_config\_t* PT2DC  
Product term 2 input D

*aoi\_input\_config\_t* PT3AC  
Product term 3 input A

*aoi\_input\_config\_t* PT3BC  
Product term 3 input B

*aoi\_input\_config\_t* PT3CC  
Product term 3 input C

*aoi\_input\_config\_t* PT3DC  
Product term 3 input D

## 2.2 CDOG

*status\_t* CDOG\_Init(CDOG\_Type \*base, *cdog\_config\_t* \*conf)  
Initialize CDOG.

This function initializes CDOG block and setting.

### Parameters

- base – CDOG peripheral base address
- conf – CDOG configuration structure

### Returns

Status of the init operation

void CDOG\_Deinit(CDOG\_Type \*base)  
Deinitialize CDOG.

This function deinitializes CDOG secure counter.

### Parameters

- base – CDOG peripheral base address

void CDOG\_GetDefaultConfig(*cdog\_config\_t* \*conf)  
Sets the default configuration of CDOG.

This function initialize CDOG config structure to default values.

### Parameters

- conf – CDOG configuration structure

void CDOG\_Stop(CDOG\_Type \*base, uint32\_t stop)  
Stops secure counter and instruction timer.

This function stops instruction timer and secure counter. This also change state of CDOG to IDLE.

### Parameters

- base – CDOG peripheral base address

- stop – expected value which will be compared with value of secure counter

void CDOG\_Start(CDOG\_Type \*base, uint32\_t reload, uint32\_t start)

Sets secure counter and instruction timer values.

This function sets value in RELOAD and START registers for instruction timer and secure counter

#### Parameters

- base – CDOG peripheral base address
- reload – reload value
- start – start value

void CDOG\_Check(CDOG\_Type \*base, uint32\_t check)

Checks secure counter.

This function compares stop value in handler with secure counter value by writting to RELOAD refister.

#### Parameters

- base – CDOG peripheral base address
- check – expected (stop) value

void CDOG\_Set(CDOG\_Type \*base, uint32\_t stop, uint32\_t reload, uint32\_t start)

Sets secure counter and instruction timer values.

This function sets value in STOP, RELOAD and START registers for instruction timer and secure counter.

#### Parameters

- base – CDOG peripheral base address
- stop – expected value which will be compared with value of secure counter
- reload – reload value for instruction timer
- start – start value for secure timer

void CDOG\_Add(CDOG\_Type \*base, uint32\_t add)

Add value to secure counter.

This function add specified value to secure counter.

#### Parameters

- base – CDOG peripheral base address.
- add – Value to be added.

void CDOG\_Add1(CDOG\_Type \*base)

Add 1 to secure counter.

This function add 1 to secure counter.

#### Parameters

- base – CDOG peripheral base address.

void CDOG\_Add16(CDOG\_Type \*base)

Add 16 to secure counter.

This function add 16 to secure counter.

#### Parameters

- base – CDOG peripheral base address.

void CDOG\_Add256(CDOG\_Type \*base)

Add 256 to secure counter.

This function add 256 to secure counter.

**Parameters**

- base – CDOG peripheral base address.

void CDOG\_Sub(CDOG\_Type \*base, uint32\_t sub)

brief Subtract value to secure counter

This function subtract specified value to secure counter.

param base CDOG peripheral base address. param sub Value to be subtracted.

void CDOG\_Sub1(CDOG\_Type \*base)

Subtract 1 from secure counter.

This function subtract specified 1 from secure counter.

**Parameters**

- base – CDOG peripheral base address.

void CDOG\_Sub16(CDOG\_Type \*base)

Subtract 16 from secure counter.

This function subtract specified 16 from secure counter.

**Parameters**

- base – CDOG peripheral base address.

void CDOG\_Sub256(CDOG\_Type \*base)

Subtract 256 from secure counter.

This function subtract specified 256 from secure counter.

**Parameters**

- base – CDOG peripheral base address.

void CDOG\_WritePersistent(CDOG\_Type \*base, uint32\_t value)

Set the CDOG persistent word.

**Parameters**

- base – CDOG peripheral base address.
- value – The value to be written.

uint32\_t CDOG\_ReadPersistent(CDOG\_Type \*base)

Get the CDOG persistent word.

**Parameters**

- base – CDOG peripheral base address.

**Returns**

The persistent word.

FSL\_CDOG\_DRIVER\_VERSION

Defines CDOG driver version 2.1.3.

Change log:

- Version 2.1.3
  - Re-design multiple instance IRQs and Clocks
  - Add fix for RESTART command errata

- Version 2.1.2
  - Support multiple IRQs
  - Fix default CONTROL values
- Version 2.1.1
  - Remove bit CONTROL[CONTROL\_CTRL]
- Version 2.1.0
  - Rename CWT to CDOG
- Version 2.0.2
  - Fix MISRA-2012 issues
- Version 2.0.1
  - Fix doxygen issues
- Version 2.0.0
  - initial version

enum \_\_cdog\_debug\_Action\_ctrl\_enum

*Values:*

enumerator kCDOG\_DebugHaltCtrl\_Run

enumerator kCDOG\_DebugHaltCtrl\_Pause

enum \_\_cdog\_irq\_pause\_ctrl\_enum

*Values:*

enumerator kCDOG\_IrqPauseCtrl\_Run

enumerator kCDOG\_IrqPauseCtrl\_Pause

enum \_\_cdog\_fault\_ctrl\_enum

*Values:*

enumerator kCDOG\_FaultCtrl\_EnableReset

enumerator kCDOG\_FaultCtrl\_EnableInterrupt

enumerator kCDOG\_FaultCtrl\_NoAction

enum \_\_code\_lock\_ctrl\_enum

*Values:*

enumerator kCDOG\_LockCtrl\_Lock

enumerator kCDOG\_LockCtrl\_Unlock

typedef uint32\_t secure\_counter\_t

SC\_ADD(add)

SC\_ADD1

SC\_ADD16

SC\_ADD256

SC\_SUB(sub)

SC\_SUB1

SC\_SUB16

SC\_SUB256

SC\_CHECK(val)

```
struct cdog_config_t
    #include <fsl_cdog.h>
```

## 2.3 Clock Driver

enum \_clock\_ip\_name

Clock gate name used for CLOCK\_EnableClock/CLOCK\_DisableClock.

*Values:*

enumerator kCLOCK\_GateINPUTMUX0

Clock gate name: INPUTMUX0

enumerator kCLOCK\_InputMux

Clock gate name: INPUTMUX0

enumerator kCLOCK\_GateI3C0

Clock gate name: I3C0

enumerator kCLOCK\_GateCTIMER0

Clock gate name: CTIMER0

enumerator kCLOCK\_GateCTIMER1

Clock gate name: CTIMER1

enumerator kCLOCK\_GateCTIMER2

Clock gate name: CTIMER2

enumerator kCLOCK\_GateFREQME

Clock gate name: FREQME

enumerator kCLOCK\_GateUTICK0

Clock gate name: UTICK0

enumerator kCLOCK\_GateWWDT0

Clock gate name: WWDT0

enumerator kCLOCK\_GateDMA

Clock gate name: DMA

enumerator kCLOCK\_GateAOI0

Clock gate name: AOI0

enumerator kCLOCK\_GateCRC

Clock gate name: CRC

enumerator kCLOCK\_Crc0

Clock gate name: CRC

enumerator kCLOCK\_GateEIM

Clock gate name: EIM

enumerator kCLOCK\_GateERM

Clock gate name: ERM

enumerator kCLOCK\_GateLPI2C0  
Clock gate name: LPI2C0

enumerator kCLOCK\_GateLPSPI0  
Clock gate name: LPSPI0

enumerator kCLOCK\_GateLPSPI1  
Clock gate name: LPSPI1

enumerator kCLOCK\_GateLPUART0  
Clock gate name: LPUART0

enumerator kCLOCK\_GateLPUART1  
Clock gate name: LPUART1

enumerator kCLOCK\_GateLPUART2  
Clock gate name: LPUART2

enumerator kCLOCK\_GateUSB0  
Clock gate name: USB0

enumerator kCLOCK\_GateQDC0  
Clock gate name: QDC0

enumerator kCLOCK\_GateFLEXPWM0  
Clock gate name: FLEXPWM0

enumerator kCLOCK\_GateOSTIMER0  
Clock gate name: OSTIMER0

enumerator kCLOCK\_GateADC0  
Clock gate name: ADC0

enumerator kCLOCK\_GateCMP0  
Clock gate name: CMP0

enumerator kCLOCK\_GateCMP1  
Clock gate name: CMP1

enumerator kCLOCK\_GatePORT0  
Clock gate name: PORT0

enumerator kCLOCK\_GatePORT1  
Clock gate name: PORT1

enumerator kCLOCK\_GatePORT2  
Clock gate name: PORT2

enumerator kCLOCK\_GatePORT3  
Clock gate name: PORT3

enumerator kCLOCK\_GateATX0  
Clock gate name: ATX0

enumerator kCLOCK\_GateMTR  
Clock gate name: MTR

enumerator kCLOCK\_GateTCU  
Clock gate name: TCU

enumerator kCLOCK\_GateEZRAMC\_RAM\_A  
Clock gate name: EZRAMC\_RAM\_A



enumerator kCLOCK\_GateGPIO0

Clock gate name: GPIO0

enumerator kCLOCK\_GateGPIO1

Clock gate name: GPIO1

enumerator kCLOCK\_GateGPIO2

Clock gate name: GPIO2

enumerator kCLOCK\_GateGPIO3

Clock gate name: GPIO3

enumerator kCLOCK\_GateROMCP

Clock gate name: ROMCP

enumerator kCLOCK\_GatePWMSM0

Clock gate name: FlexPWM SM0

enumerator kCLOCK\_GatePWMSM1

Clock gate name: FlexPWM SM1

enumerator kCLOCK\_GatePWMSM2

Clock gate name: FlexPWM SM2

enumerator kCLOCK\_GateNotAvail

Clock gate name: None

enum \_clock\_name

Clock name used to get clock frequency.

*Values:*

enumerator kCLOCK\_MainClk

MAIN\_CLK

enumerator kCLOCK\_CoreSysClk

Core/system clock(CPU\_CLK)

enumerator kCLOCK\_SYSTEM\_CLK

AHB clock

enumerator kCLOCK\_BusClk

Bus clock (AHB clock)

enumerator kCLOCK\_ExtClk

External Clock

enumerator kCLOCK\_FroHf

FRO192

enumerator kCLOCK\_FroHfDiv

Divided by FRO192

enumerator kCLOCK\_Clk48M

CLK48M

enumerator kCLOCK\_Fro12M

FRO12M

enumerator kCLOCK\_Clk1M

CLK1M

enumerator kCLOCK\_Fro16K  
FRO16K

enumerator kCLOCK\_Clk16K0  
CLK16K[0]

enumerator kCLOCK\_Clk16K1  
CLK16K[1]

enumerator kCLOCK\_SLOW\_CLK  
SYSTEM\_CLK divided by 4

enum \_clock\_select\_name

Clock name used to get clock frequency.

*Values:*

enumerator kCLOCK\_SelI3C0\_FCLK  
I3C0\_FCLK clock selection

enumerator kCLOCK\_SelCTIMER0  
CTIMER0 clock selection

enumerator kCLOCK\_SelCTIMER1  
CTIMER1 clock selection

enumerator kCLOCK\_SelCTIMER2  
CTIMER2 clock selection

enumerator kCLOCK\_SelLPI2C0  
LPI2C0 clock selection

enumerator kCLOCK\_SelLPSPi0  
LPSPi0 clock selection

enumerator kCLOCK\_SelLPSPi1  
LPSPi1 clock selection

enumerator kCLOCK\_SelLPUART0  
LPUART0 clock selection

enumerator kCLOCK\_SelLPUART1  
LPUART1 clock selection

enumerator kCLOCK\_SelLPUART2  
LPUART2 clock selection

enumerator kCLOCK\_SelUSB0  
USB0 clock selection

enumerator kCLOCK\_SelLPTMR0  
LPTMR0 clock selection

enumerator kCLOCK\_SelOSTIMER0  
OSTIMER0 clock selection

enumerator kCLOCK\_SelADC0  
ADC0 clock selection

enumerator kCLOCK\_SelCMP0\_RR  
CMP0\_RR clock selection

enumerator kCLOCK\_SelCMP1\_RR

CMP1\_RR clock selection

enumerator kCLOCK\_SelTRACE

TRACE clock selection

enumerator kCLOCK\_SelCLKOUT

CLKOUT clock selection

enumerator kCLOCK\_SelSCGSCS

SCG SCS clock selection

enumerator kCLOCK\_SelMax

MAX clock selection

enum \_clock\_attach\_id

The enumerator of clock attach Id.

*Values:*

enumerator kCLK\_IN\_to\_MAIN\_CLK

Attach clk\_in to MAIN\_CLK.

enumerator kFRO12M\_to\_MAIN\_CLK

Attach FRO\_12M to MAIN\_CLK.

enumerator kFRO\_HF\_to\_MAIN\_CLK

Attach FRO\_HF to MAIN\_CLK.

enumerator kCLK\_16K\_to\_MAIN\_CLK

Attach CLK\_16K[1] to MAIN\_CLK.

enumerator kNONE\_to\_MAIN\_CLK

Attach NONE to MAIN\_CLK.

enumerator kFRO12M\_to\_I3C0FCLK

Attach FRO12M to I3C0FCLK.

enumerator kFRO\_HF\_DIV\_to\_I3C0FCLK

Attach FRO\_HF\_DIV to I3C0FCLK.

enumerator kCLK\_IN\_to\_I3C0FCLK

Attach CLK\_IN to I3C0FCLK.

enumerator kCLK\_1M\_to\_I3C0FCLK

Attach CLK\_1M to I3C0FCLK.

enumerator kNONE\_to\_I3C0FCLK

Attach NONE to I3C0FCLK.

enumerator kFRO12M\_to\_CTIMER0

Attach FRO12M to CTIMER0.

enumerator kFRO\_HF\_to\_CTIMER0

Attach FRO\_HF to CTIMER0.

enumerator kCLK\_IN\_to\_CTIMER0

Attach CLK\_IN to CTIMER0.

enumerator kCLK\_16K\_to\_CTIMER0

Attach CLK\_16K to CTIMER0.

enumerator kCLK\_1M\_to\_CTIMER0

Attach CLK\_1M to CTIMER0.

enumerator kNONE\_to\_CTIMER0

Attach NONE to CTIMER0.

enumerator kFRO12M\_to\_CTIMER1

Attach FRO12M to CTIMER1.

enumerator kFRO\_HF\_to\_CTIMER1

Attach FRO\_HF to CTIMER1.

enumerator kCLK\_IN\_to\_CTIMER1

Attach CLK\_IN to CTIMER1.

enumerator kCLK\_16K\_to\_CTIMER1

Attach CLK\_16K to CTIMER1.

enumerator kCLK\_1M\_to\_CTIMER1

Attach CLK\_1M to CTIMER1.

enumerator kNONE\_to\_CTIMER1

Attach NONE to CTIMER1.

enumerator kFRO12M\_to\_CTIMER2

Attach FRO12M to CTIMER2.

enumerator kFRO\_HF\_to\_CTIMER2

Attach FRO\_HF to CTIMER2.

enumerator kCLK\_IN\_to\_CTIMER2

Attach CLK\_IN to CTIMER2.

enumerator kCLK\_16K\_to\_CTIMER2

Attach CLK\_16K to CTIMER2.

enumerator kCLK\_1M\_to\_CTIMER2

Attach CLK\_1M to CTIMER2.

enumerator kNONE\_to\_CTIMER2

Attach NONE to CTIMER2.

enumerator kFRO12M\_to\_LPI2C0

Attach FRO12M to LPI2C0.

enumerator kFRO\_HF\_DIV\_to\_LPI2C0

Attach FRO\_HF\_DIV to LPI2C0.

enumerator kCLK\_IN\_to\_LPI2C0

Attach CLK\_IN to LPI2C0.

enumerator kCLK\_1M\_to\_LPI2C0

Attach CLK\_1M to LPI2C0.

enumerator kNONE\_to\_LPI2C0

Attach NONE to LPI2C0.

enumerator kFRO12M\_to\_LPSPI0

Attach FRO12M to LPSPI0.

enumerator kFRO\_HF\_DIV\_to\_LPSPI0

Attach FRO\_HF\_DIV to LPSPI0.

enumerator kCLK\_IN\_to\_LPSPI0  
Attach CLK\_IN to LPSPI0.

enumerator kCLK\_1M\_to\_LPSPI0  
Attach CLK\_1M to LPSPI0.

enumerator kNONE\_to\_LPSPI0  
Attach NONE to LPSPI0.

enumerator kFRO12M\_to\_LPSPI1  
Attach FRO12M to LPSPI1.

enumerator kFRO\_HF\_DIV\_to\_LPSPI1  
Attach FRO\_HF\_DIV to LPSPI1.

enumerator kCLK\_IN\_to\_LPSPI1  
Attach CLK\_IN to LPSPI1.

enumerator kCLK\_1M\_to\_LPSPI1  
Attach CLK\_1M to LPSPI1.

enumerator kNONE\_to\_LPSPI1  
Attach NONE to LPSPI1.

enumerator kFRO12M\_to\_LPUART0  
Attach FRO12M to LPUART0.

enumerator kFRO\_HF\_DIV\_to\_LPUART0  
Attach FRO\_HF\_DIV to LPUART0.

enumerator kCLK\_IN\_to\_LPUART0  
Attach CLK\_IN to LPUART0.

enumerator kCLK\_16K\_to\_LPUART0  
Attach CLK\_16K to LPUART0.

enumerator kCLK\_1M\_to\_LPUART0  
Attach CLK\_1M to LPUART0.

enumerator kNONE\_to\_LPUART0  
Attach NONE to LPUART0.

enumerator kFRO12M\_to\_LPUART1  
Attach FRO12M to LPUART1.

enumerator kFRO\_HF\_DIV\_to\_LPUART1  
Attach FRO\_HF\_DIV to LPUART1.

enumerator kCLK\_IN\_to\_LPUART1  
Attach CLK\_IN to LPUART1.

enumerator kCLK\_16K\_to\_LPUART1  
Attach CLK\_16K to LPUART1.

enumerator kCLK\_1M\_to\_LPUART1  
Attach CLK\_1M to LPUART1.

enumerator kNONE\_to\_LPUART1  
Attach NONE to LPUART1.

enumerator kFRO12M\_to\_LPUART2  
Attach FRO12M to LPUART2.

enumerator kFRO\_HF\_DIV\_to\_LPUART2

Attach FRO\_HF\_DIV to LPUART2.

enumerator kCLK\_IN\_to\_LPUART2

Attach CLK\_IN to LPUART2.

enumerator kCLK\_16K\_to\_LPUART2

Attach CLK\_16K to LPUART2.

enumerator kCLK\_1M\_to\_LPUART2

Attach CLK\_1M to LPUART2.

enumerator kNONE\_to\_LPUART2

Attach NONE to LPUART2.

enumerator kCLK\_48M\_to\_USB0

Attach FRO12M to USB0.

enumerator kCLK\_IN\_to\_USB0

Attach CLK\_IN to USB0.

enumerator kNONE\_to\_USB0

Attach NONE to USB0.

enumerator kFRO12M\_to\_LPTMR0

Attach FRO12M to LPTMR0.

enumerator kFRO\_HF\_DIV\_to\_LPTMR0

Attach FRO\_HF\_DIV to LPTMR0.

enumerator kCLK\_IN\_to\_LPTMR0

Attach CLK\_IN to LPTMR0.

enumerator kCLK\_1M\_to\_LPTMR0

Attach CLK\_1M to LPTMR0.

enumerator kNONE\_to\_LPTMR0

Attach NONE to LPTMR0.

enumerator kCLK\_16K\_to\_OSTIMER

Attach FRO16K to OSTIMER0.

enumerator kCLK\_1M\_to\_OSTIMER

Attach CLK\_1M to OSTIMER0.

enumerator kNONE\_to\_OSTIMER

Attach NONE to OSTIMER0.

enumerator kFRO12M\_to\_ADC0

Attach FRO12M to ADC0.

enumerator kFRO\_HF\_to\_ADC0

Attach FRO\_HF to ADC0.

enumerator kCLK\_IN\_to\_ADC0

Attach CLK\_IN to ADC0.

enumerator kCLK\_1M\_to\_ADC0

Attach CLK\_1M to ADC0.

enumerator kNONE\_to\_ADC0

Attach NONE to ADC0.

enumerator kFRO12M\_to\_CMP0  
Attach FRO12M to CMP0.

enumerator kFRO\_HF\_DIV\_to\_CMP0  
Attach FRO\_HF\_DIV to CMP0.

enumerator kCLK\_IN\_to\_CMP0  
Attach CLK\_IN to CMP0.

enumerator kCLK\_1M\_to\_CMP0  
Attach CLK\_1M to CMP0.

enumerator kNONE\_to\_CMP0  
Attach NONE to CMP0.

enumerator kFRO12M\_to\_CMP1  
Attach FRO12M to CMP1.

enumerator kFRO\_HF\_DIV\_to\_CMP1  
Attach FRO\_HF\_DIV to CMP1.

enumerator kCLK\_IN\_to\_CMP1  
Attach CLK\_IN to CMP1.

enumerator kCLK\_1M\_to\_CMP1  
Attach CLK\_1M to CMP1.

enumerator kNONE\_to\_CMP1  
Attach NONE to CMP1.

enumerator kCPU\_CLK\_to\_TRACE  
Attach CPU\_CLK to TRACE.

enumerator kCLK\_1M\_to\_TRACE  
Attach CLK\_1M to TRACE.

enumerator kCLK\_16K\_to\_TRACE  
Attach CLK\_16K to TRACE.

enumerator kNONE\_to\_TRACE  
Attach NONE to TRACE.

enumerator kFRO12M\_to\_CLKOUT  
Attach FRO12M to CLKOUT.

enumerator kFRO\_HF\_DIV\_to\_CLKOUT  
Attach FRO\_HF\_DIV to CLKOUT.

enumerator kCLK\_IN\_to\_CLKOUT  
Attach CLK\_IN to CLKOUT.

enumerator kCLK\_16K\_to\_CLKOUT  
Attach CLK\_16K to CLKOUT.

enumerator kSLOW\_CLK\_to\_CLKOUT  
Attach SLOW\_CLK to CLKOUT.

enumerator kNONE\_to\_CLKOUT  
Attach NONE to CLKOUT.

enumerator kNONE\_to\_NONE  
Attach NONE to NONE.

enum \_clock\_div\_name

Clock dividers.

*Values:*

enumerator kCLOCK\_DivI3C0\_FCLK  
I3C0\_FCLK clock divider

enumerator kCLOCK\_DivCTIMER0  
CTIMER0 clock divider

enumerator kCLOCK\_DivCTIMER1  
CTIMER1 clock divider

enumerator kCLOCK\_DivCTIMER2  
CTIMER2 clock divider

enumerator kCLOCK\_DivWWDT0  
WWDT0 clock divider

enumerator kCLOCK\_DivLPI2C0  
LPI2C0 clock divider

enumerator kCLOCK\_DivLPSPI0  
LPSPI0 clock divider

enumerator kCLOCK\_DivLPSPI1  
LPSPI1 clock divider

enumerator kCLOCK\_DivLPUART0  
LPUART0 clock divider

enumerator kCLOCK\_DivLPUART1  
LPUART1 clock divider

enumerator kCLOCK\_DivLPUART2  
LPUART2 clock divider

enumerator kCLOCK\_DivLPTMR0  
LPTMR0 clock divider

enumerator kCLOCK\_DivADC0  
ADC0 clock divider

enumerator kCLOCK\_DivCMP0\_FUNC  
CMP0\_FUNC clock divider

enumerator kCLOCK\_DivCMP0\_RR  
CMP0\_RR clock divider

enumerator kCLOCK\_DivCMP1\_FUNC  
CMP1\_FUNC clock divider

enumerator kCLOCK\_DivCMP1\_RR  
CMP1\_RR clock divider

enumerator kCLOCK\_DivTRACE  
TRACE clock divider

enumerator kCLOCK\_DivCLKOUT  
CLKOUT clock divider



enumerator kCLOCK\_DivFRO\_HF\_DIV  
FRO\_HF\_DIV clock divider

enumerator kCLOCK\_DivSLOWCLK  
SLOWCLK clock divider

enumerator kCLOCK\_DivAHBCLK  
System clock divider

enumerator kCLOCK\_DivMax  
MAX clock divider

enum \_firc\_trim\_mode  
firc trim mode.

*Values:*

enumerator kSCG\_FircTrimNonUpdate  
Trim enable but not enable trim value update. In this mode, the trim value is fixed to the initialized value which is defined by trimCoar and trimFine in configure structure firc\_trim\_config\_t.

enumerator kSCG\_FircTrimUpdate  
Trim enable and trim value update enable. In this mode, the trim value is auto update.

enum \_firc\_trim\_src  
firc trim source.

*Values:*

enumerator kSCG\_FircTrimSrcUsb0  
USB0 start of frame (1kHz).

enumerator kSCG\_FircTrimSrcSysOsc  
System OSC.

enum \_sirc\_trim\_mode  
sirc trim mode.

*Values:*

enumerator kSCG\_SircTrimNonUpdate  
Trim enable but not enable trim value update. In this mode, the trim value is fixed to the initialized value which is defined by trimCoar and trimFine in configure structure sirc\_trim\_config\_t.

enumerator kSCG\_SircTrimUpdate  
Trim enable and trim value update enable. In this mode, the trim value is auto update.

enum \_sirc\_trim\_src  
sirc trim source.

*Values:*

enumerator kNoTrimSrc  
No external trim source.

enumerator kSCG\_SircTrimSrcSysOsc  
System OSC.

enum \_scg\_sosc\_monitor\_mode  
SCG system OSC monitor mode.

*Values:*

enumerator kSCG\_SysOscMonitorDisable  
Monitor disabled.

enumerator kSCG\_SysOscMonitorInt  
Interrupt when the SOSC error is detected.

enumerator kSCG\_SysOscMonitorReset  
Reset when the SOSC error is detected.

enum \_clke\_16k  
firc trim source.  
*Values:*

enumerator kCLKE\_16K\_SYSTEM  
To VSYS domain.

enumerator kCLKE\_16K\_COREMAIN  
To VDD\_CORE domain.

typedef enum \_clock\_ip\_name clock\_ip\_name\_t  
Clock gate name used for CLOCK\_EnableClock/CLOCK\_DisableClock.

typedef enum \_clock\_name clock\_name\_t  
Clock name used to get clock frequency.

typedef enum \_clock\_select\_name clock\_select\_name\_t  
Clock name used to get clock frequency.

typedef enum \_clock\_attach\_id clock\_attach\_id\_t  
The enumerator of clock attach Id.

typedef enum \_clock\_div\_name clock\_div\_name\_t  
Clock dividers.

typedef enum \_firc\_trim\_mode firc\_trim\_mode\_t  
firc trim mode.

typedef enum \_firc\_trim\_src firc\_trim\_src\_t  
firc trim source.

typedef struct \_firc\_trim\_config firc\_trim\_config\_t  
firc trim configuration.

typedef enum \_sirc\_trim\_mode sirc\_trim\_mode\_t  
sirc trim mode.

typedef enum \_sirc\_trim\_src sirc\_trim\_src\_t  
sirc trim source.

typedef struct \_sirc\_trim\_config sirc\_trim\_config\_t  
sirc trim configuration.

typedef enum \_scg\_sosc\_monitor\_mode scg\_sosc\_monitor\_mode\_t  
SCG system OSC monitor mode.

typedef enum \_clke\_16k clke\_16k\_t  
firc trim source.

static inline void CLOCK\_EnableClock(*clock\_ip\_name\_t* clk)  
Enable the clock for specific IP.

### Parameters

- `clk` – : Clock to be enabled.

**Returns**

Nothing

static inline void CLOCK\_DisableClock(*clock\_ip\_name\_t* clk)

Disable the clock for specific IP.

**Parameters**

- `clk` – : Clock to be Disabled.

**Returns**

Nothing

void CLOCK\_AttachClk(*clock\_attach\_id\_t* connection)

Configure the clock selection muxes.

**Parameters**

- `connection` – : Clock to be configured.

**Returns**

Nothing

*clock\_attach\_id\_t* CLOCK\_GetClockAttachId(*clock\_attach\_id\_t* connection)

Get the actual clock attach id. This fuction uses the offset in input attach id, then it reads the actual source value in the register and combine the offset to obtain an actual attach id.

**Parameters**

- `connection` – : Clock attach id to get.

**Returns**

Clock source value.

void CLOCK\_SetClockSelect(*clock\_select\_name\_t* sel\_name, uint32\_t value)

Set the clock select value. This fuction set the peripheral clock select value.

**Parameters**

- `sel_name` – : Clock select.
- `value` – : value to be set.

uint32\_t CLOCK\_GetClockSelect(*clock\_select\_name\_t* sel\_name)

Get the clock select value. This fuction get the peripheral clock select value.

**Parameters**

- `sel_name` – : Clock select.

**Returns**

Clock source value.

void CLOCK\_SetClockDiv(*clock\_div\_name\_t* div\_name, uint32\_t value)

Setup peripheral clock dividers.

**Parameters**

- `div_name` – : Clock divider name
- `value` – : Value to be divided

**Returns**

Nothing

uint32\_t CLOCK\_GetClockDiv(*clock\_div\_name\_t* div\_name)

Get peripheral clock dividers.

**Parameters**

- div\_name – : Clock divider name

**Returns**

peripheral clock dividers

void CLOCK\_HaltClockDiv(*clock\_div\_name\_t* div\_name)

Halt peripheral clock dividers.

**Parameters**

- div\_name – : Clock divider name

**Returns**

Nothing

status\_t CLOCK\_SetupFROHFClocking(uint32\_t iFreq)

Initialize the FROHF to given frequency (48,64,96,192). This function turns on FIRC and select the given frequency as the source of fro\_hf.

**Parameters**

- iFreq – : Desired frequency.

**Returns**

returns success or fail status.

status\_t CLOCK\_SetupFRO12MClocking(void)

Initialize the FRO12M. This function turns on FRO12M.

**Returns**

returns success or fail status.

status\_t CLOCK\_SetupFRO16KClocking(uint8\_t clk\_16k\_enable\_mask)

Initialize the FRO16K. This function turns on FRO16K.

**Parameters**

- clk\_16k\_enable\_mask – 0-3 0b00: disable both clk\_16k0 and clk\_16k1 0b01: only enable clk\_16k0 0b10: only enable clk\_16k1 0b11: enable both clk\_16k0 and clk\_16k1

**Returns**

returns success or fail status.

status\_t CLOCK\_SetupExtClocking(uint32\_t iFreq)

Initialize the external osc clock to given frequency.

**Parameters**

- iFreq – : Desired frequency (must be equal to exact rate in Hz)

**Returns**

returns success or fail status.

status\_t CLOCK\_SetupExtRefClocking(uint32\_t iFreq)

Initialize the external reference clock to given frequency.

**Parameters**

- iFreq – : Desired frequency (must be equal to exact rate in Hz)

**Returns**

returns success or fail status.

uint32\_t CLOCK\_GetFreq(*clock\_name\_t* clockName)

Return Frequency of selected clock.

**Returns**

Frequency of selected clock

uint32\_t CLOCK\_GetCoreSysClkFreq(void)

Return Frequency of core.

**Returns**

Frequency of the core

uint32\_t CLOCK\_GetI3CFClkFreq(void)

Return Frequency of I3C FCLK.

**Returns**

Frequency of I3C FCLK.

uint32\_t CLOCK\_GetCTimerClkFreq(uint32\_t id)

Return Frequency of CTimer functional Clock.

**Returns**

Frequency of CTimer functional Clock

uint32\_t CLOCK\_GetLpi2cClkFreq(void)

Return Frequency of LPI2C0 functional Clock.

**Returns**

Frequency of LPI2C0 functional Clock

uint32\_t CLOCK\_GetLpspiClkFreq(uint32\_t id)

Return Frequency of LPSPI functional Clock.

**Returns**

Frequency of LPSPI functional Clock

uint32\_t CLOCK\_GetLpuartClkFreq(uint32\_t id)

Return Frequency of LPUART functional Clock.

**Returns**

Frequency of LPUART functional Clock

uint32\_t CLOCK\_GetLptmrClkFreq(void)

Return Frequency of LPTMR functional Clock.

**Returns**

Frequency of LPTMR functional Clock

uint32\_t CLOCK\_GetOstimerClkFreq(void)

Return Frequency of OSTIMER.

**Returns**

Frequency of OSTIMER Clock

uint32\_t CLOCK\_GetAdcClkFreq(void)

Return Frequency of Adc Clock.

**Returns**

Frequency of Adc.

uint32\_t CLOCK\_GetCmpFClkFreq(uint32\_t id)

Return Frequency of CMP Function Clock.

**Returns**

Frequency of CMP Function.

uint32\_t CLOCK\_GetCmpRRClkFreq(uint32\_t id)

Return Frequency of CMP Round Robin Clock.

**Returns**

Frequency of CMP Round Robin.

uint32\_t CLOCK\_GetTraceClkFreq(void)

Return Frequency of Trace Clock.

**Returns**

Frequency of Trace.

uint32\_t CLOCK\_GetClkoutClkFreq(void)

Return Frequency of CLKOUT Clock.

**Returns**

Frequency of CLKOUT.

uint32\_t CLOCK\_GetWwdtClkFreq(void)

brief Return Frequency of WWDT Clock return Frequency of WWDT.

status\_t CLOCK\_FROHFTrimConfig(*firc\_trim\_config\_t* config)

Setup FROHF trim.

**Parameters**

- config – : FROHF trim value

**Returns**

returns success or fail status.

status\_t CLOCK\_FRO12MTrimConfig(*sirc\_trim\_config\_t* config)

Setup FRO 12M trim.

**Parameters**

- config – : FRO 12M trim value

**Returns**

returns success or fail status.

void CLOCK\_SetSysOscMonitorMode(*scg\_sosc\_monitor\_mode\_t* mode)

Sets the system OSC monitor mode.

This function sets the system OSC monitor mode. The mode can be disabled, it can generate an interrupt when the error is disabled, or reset when the error is detected.

**Parameters**

- mode – Monitor mode to set.

bool CLOCK\_EnableUsbfsClock(void)

brief Enable USB FS clock. Enable USB Full Speed clock.

FSL\_CLOCK\_DRIVER\_VERSION

CLOCK driver version 2.0.0.

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.

---

SDK\_DEVICE\_MAXIMUM\_CPU\_CLOCK\_FREQUENCY

CLK\_GATE\_REG\_OFFSET(value)

Clock gate name used for CLOCK\_EnableClock/CLOCK\_DisableClock.

CLK\_GATE\_BIT\_SHIFT(value)

REG\_PWM0SUBCTL

AOI\_CLOCKS

Clock ip name array for AOI.

CRC\_CLOCKS

Clock ip name array for CRC.

CTIMER\_CLOCKS

Clock ip name array for CTIMER.

DMA\_CLOCKS

Clock ip name array for DMA.

EDMA\_CLOCKS

Clock gate name array for EDMA.

ERM\_CLOCKS

Clock ip name array for ERM.

EIM\_CLOCKS

Clock ip name array for EIM.

FREQME\_CLOCKS

Clock ip name array for FREQME.

GPIO\_CLOCKS

Clock ip name array for GPIO.

I3C\_CLOCKS

Clock ip name array for I3C.

INPUTMUX\_CLOCKS

Clock ip name array for INPUTMUX.

LPCMP\_CLOCKS

Clock ip name array for GPIO.

LPADC\_CLOCKS

Clock ip name array for LPADC.

LPUART\_CLOCKS

Clock ip name array for LPUART.

LPI2C\_CLOCKS

Clock ip name array for LPI2C.

LPSPI\_CLOCKS

Clock ip name array for LSPI.

MTR\_CLOCKS

Clock ip name array for MTR.

OSTIMER\_CLOCKS

Clock ip name array for OSTIMER.

PWM\_CLOCKS

Clock ip name array for PWM.

QDC\_CLOCKS

Clock ip name array for QDC.

UTICK\_CLOCKS

Clock ip name array for UTICK.

WWDT\_CLOCKS

Clock ip name array for WWDT.

BUS\_CLK

Peripherals clock source definition.

CLK\_ATTACH\_REG\_OFFSET(value)

Clock Mux Switches The encoding is as follows each connection identified is 32bits wide while 24bits are valuable starting from LSB upwards.

[4 bits for choice, 0 means invalid choice] [8 bits mux ID]\*

CLK\_ATTACH\_CLK\_SEL(value)

CLK\_ATTACH\_MUX(reg, sel)

*firc\_trim\_mode\_t* trimMode

Trim mode.

*firc\_trim\_src\_t* trimSrc

Trim source.

uint16\_t trimDiv

Divider of SOSC.

uint8\_t trimCoar

Trim coarse value; Irrelevant if trimMode is kSCG\_TrimUpdate.

uint8\_t trimFine

Trim fine value; Irrelevant if trimMode is kSCG\_TrimUpdate.

*sirc\_trim\_mode\_t* trimMode

Trim mode.

*sirc\_trim\_src\_t* trimSrc

Trim source.

uint16\_t trimDiv

Divider of SOSC.

uint8\_t cltrim

Trim coarse value; Irrelevant if trimMode is kSCG\_TrimUpdate.

uint8\_t ccotrim

Trim fine value; Irrelevant if trimMode is kSCG\_TrimUpdate.

struct \_firc\_trim\_config

#include <fsl\_clock.h> firc trim configuration.

struct \_sirc\_trim\_config

#include <fsl\_clock.h> sirc trim configuration.



## 2.4 CRC: Cyclic Redundancy Check Driver

FSL\_CRC\_DRIVER\_VERSION

CRC driver version. Version 2.0.4.

Current version: 2.0.4

Change log:

- 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.

static inline 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 default.

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 CTIMER: Standard counter/timers

void CTIMER\_Init(CTIMER\_Type \*base, const *ctimer\_config\_t* \*config)

Ungates the clock and configures the peripheral for basic operation.

---

**Note:** This API should be called at the beginning of the application before using the driver.

---

**Parameters**

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

void CTIMER\_Deinit(CTIMER\_Type \*base)

Gates the timer clock.

**Parameters**

- base – Ctimer peripheral base address

`void CTIMER_GetDefaultConfig(ctimer_config_t *config)`

Fills in the timers configuration structure with the default settings.

The default values are:

```
config->mode = kCTIMER_TimerMode;
config->input = kCTIMER_Capture_0;
config->prescale = 0;
```

### Parameters

- config – Pointer to the user configuration structure.

`status_t CTIMER_SetupPwmPeriod(CTIMER_Type *base, const ctimer_match_t pwmPeriodChannel, ctimer_match_t matchChannel, uint32_t pwmPeriod, uint32_t pulsePeriod, bool enableInt)`

Configures the PWM signal parameters.

Enables PWM mode on the match channel passed in and will then setup the match value and other match parameters to generate a PWM signal. This function can manually assign the specified channel to set the PWM cycle.

---

**Note:** When setting PWM output from multiple output pins, all should use the same PWM period

---

### Parameters

- base – Ctimer peripheral base address
- pwmPeriodChannel – Specify the channel to control the PWM period
- matchChannel – Match pin to be used to output the PWM signal
- pwmPeriod – PWM period match value
- pulsePeriod – Pulse width match value
- enableInt – Enable interrupt when the timer value reaches the match value of the PWM pulse, if it is 0 then no interrupt will be generated.

### Returns

kStatus\_Success on success kStatus\_Fail If matchChannel is equal to pwmPeriodChannel; this channel is reserved to set the PWM cycle If PWM pulse width register value is larger than 0xFFFFFFFF.

`status_t CTIMER_SetupPwm(CTIMER_Type *base, const ctimer_match_t pwmPeriodChannel, ctimer_match_t matchChannel, uint8_t dutyCyclePercent, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz, bool enableInt)`

Configures the PWM signal parameters.

Enables PWM mode on the match channel passed in and will then setup the match value and other match parameters to generate a PWM signal. This function can manually assign the specified channel to set the PWM cycle.

---

**Note:** When setting PWM output from multiple output pins, all should use the same PWM frequency. Please use CTIMER\_SetupPwmPeriod to set up the PWM with high resolution.

---

### Parameters

- base – Ctimer peripheral base address
- pwmPeriodChannel – Specify the channel to control the PWM period

- `matchChannel` – Match pin to be used to output the PWM signal
- `dutyCyclePercent` – PWM pulse width; the value should be between 0 to 100
- `pwmFreq_Hz` – PWM signal frequency in Hz
- `srcClock_Hz` – Timer counter clock in Hz
- `enableInt` – Enable interrupt when the timer value reaches the match value of the PWM pulse, if it is 0 then no interrupt will be generated.

```
static inline void CTIMER_UpdatePwmPulsePeriod(CTIMER_Type *base, ctimer_match_t
                                              matchChannel, uint32_t pulsePeriod)
```

Updates the pulse period of an active PWM signal.

#### Parameters

- `base` – Ctimer peripheral base address
- `matchChannel` – Match pin to be used to output the PWM signal
- `pulsePeriod` – New PWM pulse width match value

```
status_t CTIMER_UpdatePwmDutycycle(CTIMER_Type *base, const ctimer_match_t
                                   pwmPeriodChannel, ctimer_match_t matchChannel,
                                   uint8_t dutyCyclePercent)
```

Updates the duty cycle of an active PWM signal.

---

**Note:** Please use `CTIMER_SetupPwmPeriod` to update the PWM with high resolution. This function can manually assign the specified channel to set the PWM cycle.

---

#### Parameters

- `base` – Ctimer peripheral base address
- `pwmPeriodChannel` – Specify the channel to control the PWM period
- `matchChannel` – Match pin to be used to output the PWM signal
- `dutyCyclePercent` – New PWM pulse width; the value should be between 0 to 100

#### Returns

`kStatus_Success` on success `kStatus_Fail` If PWM pulse width register value is larger than `0xFFFFFFFF`.

```
static inline void CTIMER_EnableInterrupts(CTIMER_Type *base, uint32_t mask)
```

Enables the selected Timer interrupts.

#### Parameters

- `base` – Ctimer peripheral base address
- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `ctimer_interrupt_enable_t`

```
static inline void CTIMER_DisableInterrupts(CTIMER_Type *base, uint32_t mask)
```

Disables the selected Timer interrupts.

#### Parameters

- `base` – Ctimer peripheral base address
- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `ctimer_interrupt_enable_t`

static inline uint32\_t CTIMER\_GetEnabledInterrupts(CTIMER\_Type \*base)

Gets the enabled Timer interrupts.

#### Parameters

- base – Ctimer peripheral base address

#### Returns

The enabled interrupts. This is the logical OR of members of the enumeration `ctimer_interrupt_enable_t`

static inline uint32\_t CTIMER\_GetStatusFlags(CTIMER\_Type \*base)

Gets the Timer status flags.

#### Parameters

- base – Ctimer peripheral base address

#### Returns

The status flags. This is the logical OR of members of the enumeration `ctimer_status_flags_t`

static inline void CTIMER\_ClearStatusFlags(CTIMER\_Type \*base, uint32\_t mask)

Clears the Timer status flags.

#### Parameters

- base – Ctimer peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration `ctimer_status_flags_t`

static inline void CTIMER\_StartTimer(CTIMER\_Type \*base)

Starts the Timer counter.

#### Parameters

- base – Ctimer peripheral base address

static inline void CTIMER\_StopTimer(CTIMER\_Type \*base)

Stops the Timer counter.

#### Parameters

- base – Ctimer peripheral base address

FSL\_CTIMER\_DRIVER\_VERSION

Version 2.3.3

enum \_\_ctimer\_capture\_channel

List of Timer capture channels.

*Values:*

enumerator kCTIMER\_Capture\_0

Timer capture channel 0

enumerator kCTIMER\_Capture\_1

Timer capture channel 1

enumerator kCTIMER\_Capture\_3

Timer capture channel 3

enum \_\_ctimer\_capture\_edge

List of capture edge options.

*Values:*

enumerator kCTIMER\_Capture\_RiseEdge

Capture on rising edge

enumerator kCTIMER\_Capture\_FallEdge

Capture on falling edge

enumerator kCTIMER\_Capture\_BothEdge

Capture on rising and falling edge

enum \_ctimer\_match

List of Timer match registers.

*Values:*

enumerator kCTIMER\_Match\_0

Timer match register 0

enumerator kCTIMER\_Match\_1

Timer match register 1

enumerator kCTIMER\_Match\_2

Timer match register 2

enumerator kCTIMER\_Match\_3

Timer match register 3

enum \_ctimer\_external\_match

List of external match.

*Values:*

enumerator kCTIMER\_External\_Match\_0

External match 0

enumerator kCTIMER\_External\_Match\_1

External match 1

enumerator kCTIMER\_External\_Match\_2

External match 2

enumerator kCTIMER\_External\_Match\_3

External match 3

enum \_ctimer\_match\_output\_control

List of output control options.

*Values:*

enumerator kCTIMER\_Output\_NoAction

No action is taken

enumerator kCTIMER\_Output\_Clear

Clear the EM bit/output to 0

enumerator kCTIMER\_Output\_Set

Set the EM bit/output to 1

enumerator kCTIMER\_Output\_Toggle

Toggle the EM bit/output

enum \_ctimer\_timer\_mode

List of Timer modes.

*Values:*

enumerator kCTIMER\_TimerMode

enumerator kCTIMER\_IncreaseOnRiseEdge

enumerator kCTIMER\_IncreaseOnFallEdge

enumerator kCTIMER\_IncreaseOnBothEdge

enum \_ctimer\_interrupt\_enable

List of Timer interrupts.

*Values:*

enumerator kCTIMER\_Match0InterruptEnable

Match 0 interrupt

enumerator kCTIMER\_Match1InterruptEnable

Match 1 interrupt

enumerator kCTIMER\_Match2InterruptEnable

Match 2 interrupt

enumerator kCTIMER\_Match3InterruptEnable

Match 3 interrupt

enum \_ctimer\_status\_flags

List of Timer flags.

*Values:*

enumerator kCTIMER\_Match0Flag

Match 0 interrupt flag

enumerator kCTIMER\_Match1Flag

Match 1 interrupt flag

enumerator kCTIMER\_Match2Flag

Match 2 interrupt flag

enumerator kCTIMER\_Match3Flag

Match 3 interrupt flag

enum ctimer\_callback\_type\_t

Callback type when registering for a callback. When registering a callback an array of function pointers is passed the size could be 1 or 8, the callback type will tell that.

*Values:*

enumerator kCTIMER\_SingleCallback

Single Callback type where there is only one callback for the timer. based on the status flags different channels needs to be handled differently

enumerator kCTIMER\_MultipleCallback

Multiple Callback type where there can be 8 valid callbacks, one per channel. for both match/capture

typedef enum \_ctimer\_capture\_channel ctimer\_capture\_channel\_t

List of Timer capture channels.

typedef enum \_ctimer\_capture\_edge ctimer\_capture\_edge\_t

List of capture edge options.

typedef enum \_ctimer\_match ctimer\_match\_t

List of Timer match registers.



```
typedef enum _ctimer_external_match ctimer_external_match_t
```

List of external match.

```
typedef enum _ctimer_match_output_control ctimer_match_output_control_t
```

List of output control options.

```
typedef enum _ctimer_timer_mode ctimer_timer_mode_t
```

List of Timer modes.

```
typedef enum _ctimer_interrupt_enable ctimer_interrupt_enable_t
```

List of Timer interrupts.

```
typedef enum _ctimer_status_flags ctimer_status_flags_t
```

List of Timer flags.

```
typedef void (*ctimer_callback_t)(uint32_t flags)
```

```
typedef struct _ctimer_match_config ctimer_match_config_t
```

Match configuration.

This structure holds the configuration settings for each match register.

```
typedef struct _ctimer_config ctimer_config_t
```

Timer configuration structure.

This structure holds the configuration settings for the Timer peripheral. To initialize this structure to reasonable defaults, call the `CTIMER_GetDefaultConfig()` function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

```
void CTIMER_SetupMatch(CTIMER_Type *base, ctimer_match_t matchChannel, const  
                      ctimer_match_config_t *config)
```

Setup the match register.

User configuration is used to setup the match value and action to be taken when a match occurs.

#### Parameters

- `base` – Ctimer peripheral base address
- `matchChannel` – Match register to configure
- `config` – Pointer to the match configuration structure

```
uint32_t CTIMER_GetOutputMatchStatus(CTIMER_Type *base, uint32_t matchChannel)
```

Get the status of output match.

This function gets the status of output MAT, whether or not this output is connected to a pin. This status is driven to the MAT pins if the match function is selected via IOCON. 0 = LOW. 1 = HIGH.

#### Parameters

- `base` – Ctimer peripheral base address
- `matchChannel` – External match channel, user can obtain the status of multiple match channels at the same time by using the logic of “|” enumeration `ctimer_external_match_t`

#### Returns

The mask of external match channel status flags. Users need to use the `_ctimer_external_match` type to decode the return variables.

```
void CTIMER_SetupCapture(CTIMER_Type *base, ctimer_capture_channel_t capture,
                        ctimer_capture_edge_t edge, bool enableInt)
```

Setup the capture.

#### Parameters

- base – Ctimer peripheral base address
- capture – Capture channel to configure
- edge – Edge on the channel that will trigger a capture
- enableInt – Flag to enable channel interrupts, if enabled then the registered call back is called upon capture

```
static inline uint32_t CTIMER_GetTimerCountValue(CTIMER_Type *base)
```

Get the timer count value from TC register.

#### Parameters

- base – Ctimer peripheral base address.

#### Returns

return the timer count value.

```
void CTIMER_RegisterCallBack(CTIMER_Type *base, ctimer_callback_t *cb_func,
                            ctimer_callback_type_t cb_type)
```

Register callback.

This function configures CTimer Callback in following modes:

- Single Callback: cb\_func should be pointer to callback function pointer For example: `ctimer_callback_t ctimer_callback = pwm_match_callback; CTIMER_RegisterCallBack(CTIMER, &ctimer_callback, kCTIMER_SingleCallback);`
- Multiple Callback: cb\_func should be pointer to array of callback function pointers Each element corresponds to Interrupt Flag in IR register. For example: `ctimer_callback_t ctimer_callback_table[] = { ctimer_match0_callback, NULL, NULL, ctimer_match3_callback, NULL, NULL, NULL, NULL}; CTIMER_RegisterCallBack(CTIMER, &ctimer_callback_table[0], kCTIMER_MultipleCallback);`

#### Parameters

- base – Ctimer peripheral base address
- cb\_func – Pointer to callback function pointer
- cb\_type – callback function type, singular or multiple

```
static inline void CTIMER_Reset(CTIMER_Type *base)
```

Reset the counter.

The timer counter and prescale counter are reset on the next positive edge of the APB clock.

#### Parameters

- base – Ctimer peripheral base address

```
static inline void CTIMER_SetPrescale(CTIMER_Type *base, uint32_t prescale)
```

Setup the timer prescale value.

Specifies the maximum value for the Prescale Counter.

#### Parameters

- base – Ctimer peripheral base address
- prescale – Prescale value

```
static inline uint32_t CTIMER_GetCaptureValue(CTIMER_Type *base, ctimer_capture_channel_t
                                              capture)
```

Get capture channel value.

Get the counter/timer value on the corresponding capture channel.

**Parameters**

- base – Ctimer peripheral base address
- capture – Select capture channel

**Returns**

The timer count capture value.

```
static inline void CTIMER_EnableResetMatchChannel(CTIMER_Type *base, ctimer_match_t
                                                  match, bool enable)
```

Enable reset match channel.

Set the specified match channel reset operation.

**Parameters**

- base – Ctimer peripheral base address
- match – match channel used
- enable – Enable match channel reset operation.

```
static inline void CTIMER_EnableStopMatchChannel(CTIMER_Type *base, ctimer_match_t
                                                  match, bool enable)
```

Enable stop match channel.

Set the specified match channel stop operation.

**Parameters**

- base – Ctimer peripheral base address.
- match – match channel used.
- enable – Enable match channel stop operation.

```
static inline void CTIMER_EnableMatchChannelReload(CTIMER_Type *base, ctimer_match_t
                                                  match, bool enable)
```

Enable reload channel falling edge.

Enable the specified match channel reload match shadow value.

**Parameters**

- base – Ctimer peripheral base address.
- match – match channel used.
- enable – Enable .

```
static inline void CTIMER_EnableRisingEdgeCapture(CTIMER_Type *base,
                                                  ctimer_capture_channel_t capture, bool
                                                  enable)
```

Enable capture channel rising edge.

Sets the specified capture channel for rising edge capture.

**Parameters**

- base – Ctimer peripheral base address.
- capture – capture channel used.
- enable – Enable rising edge capture.

```
static inline void CTIMER_EnableFallingEdgeCapture(CTIMER_Type *base,
                                                    ctimer_capture_channel_t capture, bool
                                                    enable)
```

Enable capture channel falling edge.

Sets the specified capture channel for falling edge capture.

#### Parameters

- base – Ctimer peripheral base address.
- capture – capture channel used.
- enable – Enable falling edge capture.

```
static inline void CTIMER_SetShadowValue(CTIMER_Type *base, ctimer_match_t match,
                                          uint32_t matchvalue)
```

Set the specified match shadow channel.

#### Parameters

- base – Ctimer peripheral base address.
- match – match channel used.
- matchvalue – Reload the value of the corresponding match register.

```
struct _ctimer_match_config
```

*#include <fsl\_ctimer.h>* Match configuration.

This structure holds the configuration settings for each match register.

#### Public Members

```
uint32_t matchValue
```

This is stored in the match register

```
bool enableCounterReset
```

true: Match will reset the counter false: Match will not reset the counter

```
bool enableCounterStop
```

true: Match will stop the counter false: Match will not stop the counter

```
ctimer_match_output_control_t outControl
```

Action to be taken on a match on the EM bit/output

```
bool outPinInitState
```

Initial value of the EM bit/output

```
bool enableInterrupt
```

true: Generate interrupt upon match false: Do not generate interrupt on match

```
struct _ctimer_config
```

*#include <fsl\_ctimer.h>* Timer configuration structure.

This structure holds the configuration settings for the Timer peripheral. To initialize this structure to reasonable defaults, call the CTIMER\_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

### Public Members

*ctimer\_timer\_mode\_t* mode

Timer mode

*ctimer\_capture\_channel\_t* input

Input channel to increment the timer; used only in timer modes that rely on this input signal to increment TC

uint32\_t prescale

Prescale value

## 2.6 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

void EDMA\_Init(*EDMA\_Type* \*base, const *edma\_config\_t* \*config)

Initializes the eDMA peripheral.

This function ungates the eDMA clock and configures the eDMA peripheral according to the configuration structure. All emda enabled request will be cleared in this function.

---

**Note:** This function enables the minor loop map feature.

---

### Parameters

- base – eDMA peripheral base address.
- config – A pointer to the configuration structure, see “edma\_config\_t”.

void EDMA\_Deinit(*EDMA\_Type* \*base)

Deinitializes the eDMA peripheral.

This function gates the eDMA clock.

### Parameters

- base – eDMA peripheral base address.

void EDMA\_InstallTCD(*EDMA\_Type* \*base, uint32\_t channel, *edma\_tcd\_t* \*tcd)

Push content of TCD structure into hardware TCD register.

### Parameters

- base – EDMA peripheral base address.
- channel – EDMA channel number.
- tcd – Point to TCD structure.

void EDMA\_GetDefaultConfig(*edma\_config\_t* \*config)

Gets the eDMA default configuration structure.

This function sets the configuration structure to default values. The default configuration is set to the following values.

```
config.enableContinuousLinkMode = false;
config.enableHaltOnError = true;
config.enableRoundRobinArbitration = false;
config.enableDebugMode = false;
```

### Parameters

- config – A pointer to the eDMA configuration structure.

```
void EDMA_InitChannel(EDMA_Type *base, uint32_t channel, edma_channel_config_t
                    *channelConfig)
```

EDMA Channel initialization.

#### Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- channelConfig – pointer to user's eDMA4 channel config structure, see *edma\_channel\_config\_t* for detail.

```
static inline void EDMA_SetChannelMemoryAttribute(EDMA_Type *base, uint32_t channel,
                                                edma_channel_memory_attribute_t
                                                writeAttribute,
                                                edma_channel_memory_attribute_t
                                                readAttribute)
```

Set channel memory attribute.

#### Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- writeAttribute – Attributes associated with a write transaction.
- readAttribute – Attributes associated with a read transaction.

```
static inline void EDMA_SetChannelSignExtension(EDMA_Type *base, uint32_t channel, uint8_t
                                                position)
```

Set channel sign extension.

#### Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- position – A non-zero value specifying the sign extend bit position. If 0, sign extension is disabled.

```
static inline void EDMA_SetChannelSwapSize(EDMA_Type *base, uint32_t channel,
                                           edma_channel_swap_size_t swapSize)
```

Set channel swap size.

#### Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- swapSize – Swap occurs with respect to the specified transfer size. If 0, swap is disabled.

```
static inline void EDMA_SetChannelAccessType(EDMA_Type *base, uint32_t channel,
                                           edma_channel_access_type_t
                                           channelAccessType)
```

Set channel access type.

#### Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.

- `channelAccessType` – eDMA4's transactions type on the system bus when the channel is active.

```
static inline void EDMA_SetChannelMux(EDMA_Type *base, uint32_t channel, uint32_t  
                                     channelRequestSource)
```

Set channel request source.

#### Parameters

- `base` – eDMA peripheral base address.
- `channel` – eDMA channel number.
- `channelRequestSource` – eDMA hardware service request source for the channel. User need to use the `dma_request_source_t` type as the input parameter. Note that devices may use other enum type to express dma request source and User can fined it in SOC header or `fsl_edma_soc.h`.

```
static inline uint32_t EDMA_GetChannelSystemBusInformation(EDMA_Type *base, uint32_t  
                                                         channel)
```

Gets the channel identification and attribute information on the system bus interface.

#### Parameters

- `base` – eDMA peripheral base address.
- `channel` – eDMA channel number.

#### Returns

The mask of the channel system bus information. Users need to use the `_edma_channel_sys_bus_info` type to decode the return variables.

```
static inline void EDMA_EnableChannelMasterIDReplication(EDMA_Type *base, uint32_t  
                                                         channel, bool enable)
```

Set channel master ID replication.

#### Parameters

- `base` – eDMA peripheral base address.
- `channel` – eDMA channel number.
- `enable` – true is enable, false is disable.

```
static inline void EDMA_SetChannelProtectionLevel(EDMA_Type *base, uint32_t channel,  
                                                  edma_channel_protection_level_t level)
```

Set channel security level.

#### Parameters

- `base` – eDMA peripheral base address.
- `channel` – eDMA channel number.
- `level` – security level.

```
void EDMA_ResetChannel(EDMA_Type *base, uint32_t channel)
```

Sets all TCD registers to default values.

This function sets TCD registers for this channel to default values.

---

**Note:** This function must not be called while the channel transfer is ongoing or it causes unpredictable results.

---

---

**Note:** This function enables the auto stop request feature.

---

**Parameters**

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

```
void EDMA_SetTransferConfig(EDMA_Type *base, uint32_t channel, const
                           edma_transfer_config_t *config, edma_tcd_t *nextTcd)
```

Configures the eDMA transfer attribute.

This function configures the transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the TCD address. Example:

```
edma_transfer_t config;
edma_tcd_t tcd;
config.srcAddr = ..;
config.destAddr = ..;
...
EDMA_SetTransferConfig(DMA0, channel, &config, &stcd);
```

---

**Note:** If nextTcd is not NULL, it means scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the eDMA\_ResetChannel.

---

**Parameters**

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- config – Pointer to eDMA transfer configuration structure.
- nextTcd – Point to TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

```
void EDMA_SetMinorOffsetConfig(EDMA_Type *base, uint32_t channel, const
                              edma_minor_offset_config_t *config)
```

Configures the eDMA minor offset feature.

The minor offset means that the signed-extended value is added to the source address or destination address after each minor loop.

**Parameters**

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- config – A pointer to the minor offset configuration structure.

```
void EDMA_SetChannelPreemptionConfig(EDMA_Type *base, uint32_t channel, const
                                     edma_channel_preemption_config_t *config)
```

Configures the eDMA channel preemption feature.

This function configures the channel preemption attribute and the priority of the channel.

**Parameters**

- base – eDMA peripheral base address.
- channel – eDMA channel number
- config – A pointer to the channel preemption configuration structure.



```
void EDMA_SetChannelLink(EDMA_Type *base, uint32_t channel, edma_channel_link_type_t  
                        type, uint32_t linkedChannel)
```

Sets the channel link for the eDMA transfer.

This function configures either the minor link or the major link mode. The minor link means that the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

---

**Note:** Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

---

#### Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- type – A channel link type, which can be one of the following:
  - kEDMA\_LinkNone
  - kEDMA\_MinorLink
  - kEDMA\_MajorLink
- linkedChannel – The linked channel number.

```
void EDMA_SetBandWidth(EDMA_Type *base, uint32_t channel, edma_bandwidth_t  
                      bandWidth)
```

Sets the bandwidth for the eDMA transfer.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

#### Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- bandWidth – A bandwidth setting, which can be one of the following:
  - kEDMABandwidthStallNone
  - kEDMABandwidthStall4Cycle
  - kEDMABandwidthStall8Cycle

```
void EDMA_SetModulo(EDMA_Type *base, uint32_t channel, edma_modulo_t srcModulo,  
                  edma_modulo_t destModulo)
```

Sets the source modulo and the destination modulo for the eDMA transfer.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

#### Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- srcModulo – A source modulo value.
- destModulo – A destination modulo value.

static inline void EDMA\_\_EnableAsyncRequest(*EDMA\_Type* \*base, uint32\_t channel, bool enable)  
Enables an async request for the eDMA transfer.

**Parameters**

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- enable – The command to enable (true) or disable (false).

static inline void EDMA\_\_EnableAutoStopRequest(*EDMA\_Type* \*base, uint32\_t channel, bool enable)

Enables an auto stop request for the eDMA transfer.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

**Parameters**

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- enable – The command to enable (true) or disable (false).

void EDMA\_\_EnableChannelInterrupts(*EDMA\_Type* \*base, uint32\_t channel, uint32\_t mask)  
Enables the interrupt source for the eDMA transfer.

**Parameters**

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

void EDMA\_\_DisableChannelInterrupts(*EDMA\_Type* \*base, uint32\_t channel, uint32\_t mask)  
Disables the interrupt source for the eDMA transfer.

**Parameters**

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of the interrupt source to be set. Use the defined `edma_interrupt_enable_t` type.

void EDMA\_\_SetMajorOffsetConfig(*EDMA\_Type* \*base, uint32\_t channel, int32\_t sourceOffset, int32\_t destOffset)

Configures the eDMA channel TCD major offset feature.

Adjustment value added to the source address at the completion of the major iteration count

**Parameters**

- base – eDMA peripheral base address.
- channel – edma channel number.
- sourceOffset – source address offset will be applied to source address after major loop done.
- destOffset – destination address offset will be applied to source address after major loop done.

```
void EDMA__ConfigChannelSoftwareTCD(edma_tcd_t *tcd, const edma_transfer_config_t
                                     *transfer)
```

Sets TCD fields according to the user's channel transfer configuration structure, *edma\_transfer\_config\_t*.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API *EDMA\_ConfigChannelSoftwareTCDExt*

Application should be careful about the TCD pool buffer storage class,

- For the platform has cache, the software TCD should be put in non cache section
- The TCD pool buffer should have a consistent storage class.

---

**Note:** This function enables the auto stop request feature.

---

#### Parameters

- *tcd* – Pointer to the TCD structure.
- *transfer* – channel transfer configuration pointer.

```
void EDMA__TcdReset(edma_tcd_t *tcd)
```

Sets all fields to default values for the TCD structure.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API *EDMA\_TcdResetExt*

This function sets all fields for this TCD structure to default value.

---

**Note:** This function enables the auto stop request feature.

---

#### Parameters

- *tcd* – Pointer to the TCD structure.

```
void EDMA__TcdSetTransferConfig(edma_tcd_t *tcd, const edma_transfer_config_t *config,
                               edma_tcd_t *nextTcd)
```

Configures the eDMA TCD transfer attribute.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API *EDMA\_TcdSetTransferConfigExt*

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The TCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

```
edma_transfer_t config = {
...
}
edma_tcd_t tcd __aligned(32);
edma_tcd_t nextTcd __aligned(32);
EDMA__TcdSetTransferConfig(&tcd, &config, &nextTcd);
```

---

**Note:** TCD address should be 32 bytes aligned or it causes an eDMA error.

---

---

**Note:** If the nextTcd is not NULL, the scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the EDMA\_TcdReset.

---

### Parameters

- tcd – Pointer to the TCD structure.
- config – Pointer to eDMA transfer configuration structure.
- nextTcd – Pointer to the next TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

```
void EDMA__TcdSetMinorOffsetConfig(edma_tcd_t *tcd, const edma_minor_offset_config_t
                                   *config)
```

Configures the eDMA TCD minor offset feature.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA\_TcdSetMinorOffsetConfigExt

A minor offset is a signed-extended value added to the source address or a destination address after each minor loop.

### Parameters

- tcd – A point to the TCD structure.
- config – A pointer to the minor offset configuration structure.

```
void EDMA__TcdSetChannelLink(edma_tcd_t *tcd, edma_channel_link_type_t type, uint32_t
                             linkedChannel)
```

Sets the channel link for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA\_TcdSetChannelLinkExt

This function configures either a minor link or a major link. The minor link means the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

---

**Note:** Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

---

### Parameters

- tcd – Point to the TCD structure.
- type – Channel link type, it can be one of:
  - kEDMA\_LinkNone
  - kEDMA\_MinorLink
  - kEDMA\_MajorLink
- linkedChannel – The linked channel number.

```
static inline void EDMA__TcdSetBandWidth(edma_tcd_t *tcd, edma_bandwidth_t bandWidth)
```

Sets the bandwidth for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA\_TcdSetBandWidthExt

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after

the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

#### Parameters

- `tcd` – A pointer to the TCD structure.
- `bandWidth` – A bandwidth setting, which can be one of the following:
  - `kEDMABandwidthStallNone`
  - `kEDMABandwidthStall4Cycle`
  - `kEDMABandwidthStall8Cycle`

```
void EDMA__TcdSetModulo(edma_tcd_t *tcd, edma_modulo_t srcModulo, edma_modulo_t destModulo)
```

Sets the source modulo and the destination modulo for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA__TcdSetModuloExt`

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

#### Parameters

- `tcd` – A pointer to the TCD structure.
- `srcModulo` – A source modulo value.
- `destModulo` – A destination modulo value.

```
static inline void EDMA__TcdEnableAutoStopRequest(edma_tcd_t *tcd, bool enable)
```

Sets the auto stop request for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA__TcdEnableAutoStopRequestExt`

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

#### Parameters

- `tcd` – A pointer to the TCD structure.
- `enable` – The command to enable (true) or disable (false).

```
void EDMA__TcdEnableInterrupts(edma_tcd_t *tcd, uint32_t mask)
```

Enables the interrupt source for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA__TcdEnableInterruptsExt`

#### Parameters

- `tcd` – Point to the TCD structure.
- `mask` – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

```
void EDMA__TcdDisableInterrupts(edma_tcd_t *tcd, uint32_t mask)
```

Disables the interrupt source for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA__TcdDisableInterruptsExt`

#### Parameters

- `tcd` – Point to the TCD structure.

- `mask` – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

`void EDMA__TcdSetMajorOffsetConfig(edma_tcd_t *tcd, int32_t sourceOffset, int32_t destOffset)`

Configures the eDMA TCD major offset feature.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA__TcdSetMajorOffsetConfigExt`

Adjustment value added to the source address at the completion of the major iteration count

#### Parameters

- `tcd` – A point to the TCD structure.
- `sourceOffset` – source address offset 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__ConfigChannelSoftwareTCDExt(EDMA_Type *base, edma_tcd_t *tcd, const edma_transfer_config_t *transfer)`

Sets TCD fields according to the user's channel transfer configuration structure, `edma_transfer_config_t`.

Application should be careful about the TCD pool buffer storage class,

- For the platform has cache, the software TCD should be put in non cache section
- The TCD pool buffer should have a consistent storage class.

---

**Note:** This function enables the auto stop request feature.

---

#### Parameters

- `base` – eDMA peripheral base address.
- `tcd` – Pointer to the TCD structure.
- `transfer` – channel transfer configuration pointer.

`void EDMA__TcdResetExt(EDMA_Type *base, edma_tcd_t *tcd)`

Sets all fields to default values for the TCD structure.

This function sets all fields for this TCD structure to default value.

---

**Note:** This function enables the auto stop request feature.

---

#### Parameters

- `base` – eDMA peripheral base address.
- `tcd` – Pointer to the TCD structure.

`void EDMA__TcdSetTransferConfigExt(EDMA_Type *base, edma_tcd_t *tcd, const edma_transfer_config_t *config, edma_tcd_t *nextTcd)`

Configures the eDMA TCD transfer attribute.

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The TCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

```
edma_transfer_t config = {  
    ...  
}  
edma_tcd_t tcd __aligned(32);  
edma_tcd_t nextTcd __aligned(32);  
EDMA_TcdSetTransferConfig(&tcd, &config, &nextTcd);
```

---

**Note:** TCD address should be 32 bytes aligned or it causes an eDMA error.

---

---

**Note:** If the nextTcd is not NULL, the scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the EDMA\_TcdReset.

---

### Parameters

- base – eDMA peripheral base address.
- tcd – Pointer to the TCD structure.
- config – Pointer to eDMA transfer configuration structure.
- nextTcd – Pointer to the next TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

```
void EDMA_TcdSetMinorOffsetConfigExt(EDMA_Type *base, edma_tcd_t *tcd, const  
                                     edma_minor_offset_config_t *config)
```

Configures the eDMA TCD minor offset feature.

A minor offset is a signed-extended value added to the source address or a destination address after each minor loop.

### Parameters

- base – eDMA peripheral base address.
- tcd – A point to the TCD structure.
- config – A pointer to the minor offset configuration structure.

```
void EDMA_TcdSetChannelLinkExt(EDMA_Type *base, edma_tcd_t *tcd,  
                              edma_channel_link_type_t type, uint32_t linkedChannel)
```

Sets the channel link for the eDMA TCD.

This function configures either a minor link or a major link. The minor link means the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

---

**Note:** Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

---

### Parameters

- base – eDMA peripheral base address.
- tcd – Point to the TCD structure.
- type – Channel link type, it can be one of:
  - kEDMA\_LinkNone
  - kEDMA\_MinorLink

- kEDMA\_MajorLink
- linkedChannel – The linked channel number.

```
static inline void EDMA__TcdSetBandWidthExt(EDMA_Type *base, edma_tcd_t *tcd,  
                                             edma_bandwidth_t bandWidth)
```

Sets the bandwidth for the eDMA TCD.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

#### Parameters

- base – eDMA peripheral base address.
- tcd – A pointer to the TCD structure.
- bandWidth – A bandwidth setting, which can be one of the following:
  - kEDMABandwidthStallNone
  - kEDMABandwidthStall4Cycle
  - kEDMABandwidthStall8Cycle

```
void EDMA__TcdSetModuloExt(EDMA_Type *base, edma_tcd_t *tcd, edma_modulo_t srcModulo,  
                           edma_modulo_t destModulo)
```

Sets the source modulo and the destination modulo for the eDMA TCD.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

#### Parameters

- base – eDMA peripheral base address.
- tcd – A pointer to the TCD structure.
- srcModulo – A source modulo value.
- destModulo – A destination modulo value.

```
static inline void EDMA__TcdEnableAutoStopRequestExt(EDMA_Type *base, edma_tcd_t *tcd,  
                                                      bool enable)
```

Sets the auto stop request for the eDMA TCD.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

#### Parameters

- base – eDMA peripheral base address.
- tcd – A pointer to the TCD structure.
- enable – The command to enable (true) or disable (false).

```
void EDMA__TcdEnableInterruptsExt(EDMA_Type *base, edma_tcd_t *tcd, uint32_t mask)
```

Enables the interrupt source for the eDMA TCD.

#### Parameters

- base – eDMA peripheral base address.
- tcd – Point to the TCD structure.
- mask – The mask of interrupt source to be set. Users need to use the defined *edma\_interrupt\_enable\_t* type.



```
void EDMA__TcdDisableInterruptsExt(EDMA_Type *base, edma_tcd_t *tcd, uint32_t mask)
```

Disables the interrupt source for the eDMA TCD.

#### Parameters

- base – eDMA peripheral base address.
- tcd – Point to the TCD structure.
- mask – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

```
void EDMA__TcdSetMajorOffsetConfigExt(EDMA_Type *base, edma_tcd_t *tcd, int32_t  
sourceOffset, int32_t destOffset)
```

Configures the eDMA TCD major offset feature.

Adjustment value added to the source address at the completion of the major iteration count

#### Parameters

- base – eDMA peripheral base address.
- tcd – A point to the TCD structure.
- sourceOffset – source address offset will be applied to source address after major loop done.
- destOffset – destination address offset will be applied to source address after major loop done.

```
static inline void EDMA__EnableChannelRequest(EDMA_Type *base, uint32_t channel)
```

Enables the eDMA hardware channel request.

This function enables the hardware channel request.

#### Parameters

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

```
static inline void EDMA__DisableChannelRequest(EDMA_Type *base, uint32_t channel)
```

Disables the eDMA hardware channel request.

This function disables the hardware channel request.

#### Parameters

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

```
static inline void EDMA__TriggerChannelStart(EDMA_Type *base, uint32_t channel)
```

Starts the eDMA transfer by using the software trigger.

This function starts a minor loop transfer.

#### Parameters

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

```
uint32_t EDMA__GetRemainingMajorLoopCount(EDMA_Type *base, uint32_t channel)
```

Gets the remaining major loop count from the eDMA current channel TCD.

This function checks the TCD (Task Control Descriptor) status for a specified eDMA channel and returns the number of major loop count that has not finished.

**Note:** 1. This function can only be used to get unfinished major loop count of transfer without the next TCD, or it might be inaccuracy.

- a. The unfinished/remaining transfer bytes cannot be obtained directly from registers while the channel is running. Because to calculate the remaining bytes, the initial NBYTES configured in DMA\_TCDn\_NBYTES\_MLNO register is needed while the eDMA IP does not support getting it while a channel is active. In another word, the NBYTES value reading is always the actual (decrementing) NBYTES value the dma\_engine is working with while a channel is running. Consequently, to get the remaining transfer bytes, a software-saved initial value of NBYTES (for example copied before enabling the channel) is needed. The formula to calculate it is shown below:  $\text{RemainingBytes} = \text{RemainingMajorLoopCount} * \text{NBYTES}(\text{initially configured})$

---

### Parameters

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

### Returns

Major loop count which has not been transferred yet for the current TCD.

```
static inline uint32_t EDMA_GetErrorStatusFlags(EDMA_Type *base)
```

Gets the eDMA channel error status flags.

### Parameters

- base – eDMA peripheral base address.

### Returns

The mask of error status flags. Users need to use the `_edma_error_status_flags` type to decode the return variables.

```
uint32_t EDMA_GetChannelStatusFlags(EDMA_Type *base, uint32_t channel)
```

Gets the eDMA channel status flags.

### Parameters

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

### Returns

The mask of channel status flags. Users need to use the `_edma_channel_status_flags` type to decode the return variables.

```
void EDMA_ClearChannelStatusFlags(EDMA_Type *base, uint32_t channel, uint32_t mask)
```

Clears the eDMA channel status flags.

### Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of channel status to be cleared. Users need to use the defined `_edma_channel_status_flags` type.

```
status_t EDMA_CreateHandle(edma_handle_t *handle, EDMA_Type *base, uint32_t channel)
```

Creates the eDMA handle.

This function is called if using the transactional API for eDMA. This function initializes the internal state of the eDMA handle.

### Parameters

- `handle` – eDMA handle pointer. The eDMA handle stores callback function and parameters.
- `base` – eDMA peripheral base address.
- `channel` – eDMA channel number.

#### Return values

- `kStatus_Success` –
- `kStatus_InvalidArgument` –

`void EDMA_InstallTCDMemory(edma_handle_t *handle, edma_tcd_t *tcdPool, uint32_t tcdSize)`

Installs the TCDs memory pool into the eDMA handle.

This function is called after the `EDMA_CreateHandle` to use scatter/gather feature. This function shall only be used while users need to use scatter gather mode. Scatter gather mode enables EDMA to load a new transfer control block (tcd) in hardware, and automatically reconfigure that DMA channel for a new transfer. Users need to prepare tcd memory and also configure tcds using interface `EDMA_SubmitTransfer`.

#### Parameters

- `handle` – eDMA handle pointer.
- `tcdPool` – A memory pool to store TCDs. It must be 32 bytes aligned.
- `tcdSize` – The number of TCD slots.

`void EDMA_SetCallback(edma_handle_t *handle, edma_callback callback, void *userData)`

Installs a callback function for the eDMA transfer.

This callback is called in the eDMA IRQ handler. Use the callback to do something after the current major loop transfer completes. This function will be called every time one tcd finished transfer.

#### Parameters

- `handle` – eDMA handle pointer.
- `callback` – eDMA callback function pointer.
- `userData` – A parameter for the callback function.

`void EDMA_PrepareTransferConfig(edma_transfer_config_t *config, void *srcAddr, uint32_t srcWidth, int16_t srcOffset, void *destAddr, uint32_t destWidth, int16_t destOffset, uint32_t bytesEachRequest, uint32_t transferBytes)`

Prepares the eDMA transfer structure configurations.

This function prepares the transfer configuration structure according to the user input.

---

**Note:** The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE). User can check if 128 bytes support is available for specific instance by `FSL_FEATURE_EDMA_INSTANCE_SUPPORT_128_BYTES_TRANSFERn`.

---

#### Parameters

- `config` – The user configuration structure of type `edma_transfer_t`.
- `srcAddr` – eDMA transfer source address.
- `srcWidth` – eDMA transfer source address width(bytes).
- `srcOffset` – source address offset.

- `destAddr` – eDMA transfer destination address.
- `destWidth` – eDMA transfer destination address width(bytes).
- `destOffset` – destination address offset.
- `bytesEachRequest` – eDMA transfer bytes per channel request.
- `transferBytes` – eDMA transfer bytes to be transferred.

```
void EDMA__PrepareTransfer(edma_transfer_config_t *config, void *srcAddr, uint32_t srcWidth,  
                          void *destAddr, uint32_t destWidth, uint32_t bytesEachRequest,  
                          uint32_t transferBytes, edma_transfer_type_t type)
```

Prepares the eDMA transfer structure.

This function prepares the transfer configuration structure according to the user input.

---

**Note:** The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

---

#### Parameters

- `config` – The user configuration structure of type `edma_transfer_t`.
- `srcAddr` – eDMA transfer source address.
- `srcWidth` – eDMA transfer source address width(bytes).
- `destAddr` – eDMA transfer destination address.
- `destWidth` – eDMA transfer destination address width(bytes).
- `bytesEachRequest` – eDMA transfer bytes per channel request.
- `transferBytes` – eDMA transfer bytes to be transferred.
- `type` – eDMA transfer type.

```
void EDMA__PrepareTransferTCD(edma_handle_t *handle, edma_tcd_t *tcd, void *srcAddr,  
                             uint32_t srcWidth, int16_t srcOffset, void *destAddr, uint32_t  
                             destWidth, int16_t destOffset, uint32_t bytesEachRequest,  
                             uint32_t transferBytes, edma_tcd_t *nextTcd)
```

Prepares the eDMA transfer content descriptor.

This function prepares the transfer content descriptor structure according to the user input.

---

**Note:** The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

---

#### Parameters

- `handle` – eDMA handle pointer.
- `tcd` – Pointer to eDMA transfer content descriptor structure.
- `srcAddr` – eDMA transfer source address.
- `srcWidth` – eDMA transfer source address width(bytes).
- `srcOffset` – source address offset.
- `destAddr` – eDMA transfer destination address.
- `destWidth` – eDMA transfer destination address width(bytes).

- `destOffset` – destination address offset.
- `bytesEachRequest` – eDMA transfer bytes per channel request.
- `transferBytes` – eDMA transfer bytes to be transferred.
- `nextTcd` – eDMA transfer linked TCD address.

`status_t` EDMA\_SubmitTransferTCD(*edma\_handle\_t* \*handle, *edma\_tcd\_t* \*tcd)

Submits the eDMA transfer content descriptor.

This function submits the eDMA transfer request according to the transfer content descriptor. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function EDMA\_InstallTCDMemory before.

Typical user case:

a. submit single transfer

```
edma_tcd_t tcd;
EDMA_PrepareTransferTCD(handle, tcd, ...)
EDMA_SubmitTransferTCD(handle, tcd)
EDMA_StartTransfer(handle)
```

b. submit static link transfer,

```
edma_tcd_t tcd[2];
EDMA_PrepareTransferTCD(handle, &tcd[0], ...)
EDMA_PrepareTransferTCD(handle, &tcd[1], ...)
EDMA_SubmitTransferTCD(handle, &tcd[0])
EDMA_StartTransfer(handle)
```

c. submit dynamic link transfer

```
edma_tcd_t tcdpool[2];
EDMA_InstallTCDMemory(&g_DMA_Handle, tcdpool, 2);
edma_tcd_t tcd;
EDMA_PrepareTransferTCD(handle, tcd, ...)
EDMA_SubmitTransferTCD(handle, tcd)
EDMA_PrepareTransferTCD(handle, tcd, ...)
EDMA_SubmitTransferTCD(handle, tcd)
EDMA_StartTransfer(handle)
```

d. submit loop transfer

```
edma_tcd_t tcd[2];
EDMA_PrepareTransferTCD(handle, &tcd[0], ..., &tcd[1])
EDMA_PrepareTransferTCD(handle, &tcd[1], ..., &tcd[0])
EDMA_SubmitTransferTCD(handle, &tcd[0])
EDMA_StartTransfer(handle)
```

### Parameters

- `handle` – eDMA handle pointer.
- `tcd` – Pointer to eDMA transfer content descriptor structure.

### Return values

- `kStatus_EDMA_Success` – It means submit transfer request succeed.
- `kStatus_EDMA_QueueFull` – It means TCD queue is full. Submit transfer request is not allowed.
- `kStatus_EDMA_Busy` – It means the given channel is busy, need to submit request later.

*status\_t* EDMA\_SubmitTransfer(*edma\_handle\_t* \*handle, const *edma\_transfer\_config\_t* \*config)

Submits the eDMA transfer request.

This function submits the eDMA transfer request according to the transfer configuration structure. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function EDMA\_InstallTCDMemory before.

#### Parameters

- handle – eDMA handle pointer.
- config – Pointer to eDMA transfer configuration structure.

#### Return values

- kStatus\_EDMA\_Success – It means submit transfer request succeed.
- kStatus\_EDMA\_QueueFull – It means TCD queue is full. Submit transfer request is not allowed.
- kStatus\_EDMA\_Busy – It means the given channel is busy, need to submit request later.

*status\_t* EDMA\_SubmitLoopTransfer(*edma\_handle\_t* \*handle, *edma\_transfer\_config\_t* \*transfer, *uint32\_t* transferLoopCount)

Submits the eDMA scatter gather transfer configurations.

The function is target for submit loop transfer request, the ring transfer request means that the transfer request TAIL is link to HEAD, such as, A->B->C->D->A, or A->A

To use the ring transfer feature, the application should allocate several transfer object, such as

```
edma_channel_transfer_config_t transfer[2];  
EDMA_TransferSubmitLoopTransfer(psHandle, &transfer, 2U);
```

Then eDMA driver will link transfer[0] and transfer[1] to each other

---

**Note:** Application should check the return value of this function to avoid transfer request submit failed

---

#### Parameters

- handle – eDMA handle pointer
- transfer – pointer to user's eDMA channel configure structure, see *edma\_channel\_transfer\_config\_t* for detail
- transferLoopCount – the count of the transfer ring, if loop count is 1, that means that the one will link to itself.

#### Return values

- kStatus\_Success – It means submit transfer request succeed
- kStatus\_EDMA\_Busy – channel is in busy status
- kStatus\_InvalidArgument – Invalid Argument

void EDMA\_StartTransfer(*edma\_handle\_t* \*handle)

eDMA starts transfer.

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

#### Parameters

- handle – eDMA handle pointer.

void EDMA\_StopTransfer(*edma\_handle\_t* \*handle)

eDMA stops transfer.

This function disables the channel request to pause the transfer. Users can call EDMA\_StartTransfer() again to resume the transfer.

#### Parameters

- handle – eDMA handle pointer.

void EDMA\_AbortTransfer(*edma\_handle\_t* \*handle)

eDMA aborts transfer.

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

#### Parameters

- handle – DMA handle pointer.

static inline uint32\_t EDMA\_GetUnusedTCDNumber(*edma\_handle\_t* \*handle)

Get unused TCD slot number.

This function gets current tcd index which is run. If the TCD pool pointer is NULL, it will return 0.

#### Parameters

- handle – DMA handle pointer.

#### Returns

The unused tcd slot number.

static inline uint32\_t EDMA\_GetNextTCDAddress(*edma\_handle\_t* \*handle)

Get the next tcd address.

This function gets the next tcd address. If this is last TCD, return 0.

#### Parameters

- handle – DMA handle pointer.

#### Returns

The next TCD address.

void EDMA\_HandleIRQ(*edma\_handle\_t* \*handle)

eDMA IRQ handler for the current major loop transfer completion.

This function clears the channel major interrupt flag and calls the callback function if it is not NULL.

Note: For the case using TCD queue, when the major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Assertion of an optional interrupt request also occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor (if scatter/gather is enabled).

For instance, when the time interrupt of TCD[0] happens, the TCD[1] has already been loaded into the eDMA engine. As sga and sga\_index are calculated based on the DLAST\_SGA bitfield lies in the TCD\_CSR register, the sga\_index in this case should be 2 (DLAST\_SGA of TCD[1] stores the address of TCD[2]). Thus, the “tcdUsed” updated should be (tcdUsed - 2U) which indicates the number of TCDs can be loaded in the memory pool (because TCD[0] and TCD[1] have been loaded into the eDMA engine at this point already.).

For the last two continuous ISRs in a scatter/gather process, they both load the last TCD (The last ISR does not load a new TCD) from the memory pool to the eDMA engine when major

loop completes. Therefore, ensure that the header and tcdUsed updated are identical for them. tcdUsed are both 0 in this case as no TCD to be loaded.

See the “eDMA basic data flow” in the eDMA Functional description section of the Reference Manual for further details.

### Parameters

- handle – eDMA handle pointer.

FSL\_EDMA\_DRIVER\_VERSION

eDMA driver version

Version 2.10.7.

\_edma\_transfer\_status eDMA transfer status

*Values:*

enumerator kStatus\_EDMA\_QueueFull

TCD queue is full.

enumerator kStatus\_EDMA\_Busy

Channel is busy and can't handle the transfer request.

enum \_edma\_transfer\_size

eDMA transfer configuration

*Values:*

enumerator kEDMA\_TransferSize1Bytes

Source/Destination data transfer size is 1 byte every time

enumerator kEDMA\_TransferSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kEDMA\_TransferSize4Bytes

Source/Destination data transfer size is 4 bytes every time

enumerator kEDMA\_TransferSize8Bytes

Source/Destination data transfer size is 8 bytes every time

enumerator kEDMA\_TransferSize16Bytes

Source/Destination data transfer size is 16 bytes every time

enumerator kEDMA\_TransferSize32Bytes

Source/Destination data transfer size is 32 bytes every time

enumerator kEDMA\_TransferSize64Bytes

Source/Destination data transfer size is 64 bytes every time

enumerator kEDMA\_TransferSize128Bytes

Source/Destination data transfer size is 128 bytes every time

enum \_edma\_modulo

eDMA modulo configuration

*Values:*

enumerator kEDMA\_ModuloDisable

Disable modulo

enumerator kEDMA\_Modulo2bytes

Circular buffer size is 2 bytes.



enumerator kEDMA\_Modulo4bytes  
Circular buffer size is 4 bytes.

enumerator kEDMA\_Modulo8bytes  
Circular buffer size is 8 bytes.

enumerator kEDMA\_Modulo16bytes  
Circular buffer size is 16 bytes.

enumerator kEDMA\_Modulo32bytes  
Circular buffer size is 32 bytes.

enumerator kEDMA\_Modulo64bytes  
Circular buffer size is 64 bytes.

enumerator kEDMA\_Modulo128bytes  
Circular buffer size is 128 bytes.

enumerator kEDMA\_Modulo256bytes  
Circular buffer size is 256 bytes.

enumerator kEDMA\_Modulo512bytes  
Circular buffer size is 512 bytes.

enumerator kEDMA\_Modulo1Kbytes  
Circular buffer size is 1 K bytes.

enumerator kEDMA\_Modulo2Kbytes  
Circular buffer size is 2 K bytes.

enumerator kEDMA\_Modulo4Kbytes  
Circular buffer size is 4 K bytes.

enumerator kEDMA\_Modulo8Kbytes  
Circular buffer size is 8 K bytes.

enumerator kEDMA\_Modulo16Kbytes  
Circular buffer size is 16 K bytes.

enumerator kEDMA\_Modulo32Kbytes  
Circular buffer size is 32 K bytes.

enumerator kEDMA\_Modulo64Kbytes  
Circular buffer size is 64 K bytes.

enumerator kEDMA\_Modulo128Kbytes  
Circular buffer size is 128 K bytes.

enumerator kEDMA\_Modulo256Kbytes  
Circular buffer size is 256 K bytes.

enumerator kEDMA\_Modulo512Kbytes  
Circular buffer size is 512 K bytes.

enumerator kEDMA\_Modulo1Mbytes  
Circular buffer size is 1 M bytes.

enumerator kEDMA\_Modulo2Mbytes  
Circular buffer size is 2 M bytes.

enumerator kEDMA\_Modulo4Mbytes  
Circular buffer size is 4 M bytes.

enumerator kEDMA\_Modulo8Mbytes

Circular buffer size is 8 M bytes.

enumerator kEDMA\_Modulo16Mbytes

Circular buffer size is 16 M bytes.

enumerator kEDMA\_Modulo32Mbytes

Circular buffer size is 32 M bytes.

enumerator kEDMA\_Modulo64Mbytes

Circular buffer size is 64 M bytes.

enumerator kEDMA\_Modulo128Mbytes

Circular buffer size is 128 M bytes.

enumerator kEDMA\_Modulo256Mbytes

Circular buffer size is 256 M bytes.

enumerator kEDMA\_Modulo512Mbytes

Circular buffer size is 512 M bytes.

enumerator kEDMA\_Modulo1Gbytes

Circular buffer size is 1 G bytes.

enumerator kEDMA\_Modulo2Gbytes

Circular buffer size is 2 G bytes.

enum \_edma\_bandwidth

Bandwidth control.

*Values:*

enumerator kEDMA\_BandwidthStallNone

No eDMA engine stalls.

enumerator kEDMA\_BandwidthStall4Cycle

eDMA engine stalls for 4 cycles after each read/write.

enumerator kEDMA\_BandwidthStall8Cycle

eDMA engine stalls for 8 cycles after each read/write.

enum \_edma\_channel\_link\_type

Channel link type.

*Values:*

enumerator kEDMA\_LinkNone

No channel link

enumerator kEDMA\_MinorLink

Channel link after each minor loop

enumerator kEDMA\_MajorLink

Channel link while major loop count exhausted

\_edma\_channel\_status\_flags eDMA channel status flags.

*Values:*

enumerator kEDMA\_DoneFlag

DONE flag, set while transfer finished, CITER value exhausted

enumerator kEDMA\_ErrorFlag

eDMA error flag, an error occurred in a transfer

enumerator kEDMA\_InterruptFlag

eDMA interrupt flag, set while an interrupt occurred of this channel

\_edma\_error\_status\_flags eDMA channel error status flags.

*Values:*

enumerator kEDMA\_DestinationBusErrorFlag

Bus error on destination address

enumerator kEDMA\_SourceBusErrorFlag

Bus error on the source address

enumerator kEDMA\_ScatterGatherErrorFlag

Error on the Scatter/Gather address, not 32byte aligned.

enumerator kEDMA\_NbytesErrorFlag

NBYTES/CITER configuration error

enumerator kEDMA\_DestinationOffsetErrorFlag

Destination offset not aligned with destination size

enumerator kEDMA\_DestinationAddressErrorFlag

Destination address not aligned with destination size

enumerator kEDMA\_SourceOffsetErrorFlag

Source offset not aligned with source size

enumerator kEDMA\_SourceAddressErrorFlag

Source address not aligned with source size

enumerator kEDMA\_ErrorChannelFlag

Error channel number of the cancelled channel number

enumerator kEDMA\_TransferCanceledFlag

Transfer cancelled

enumerator kEDMA\_ValidFlag

No error occurred, this bit is 0. Otherwise, it is 1.

\_edma\_interrupt\_enable eDMA interrupt source

*Values:*

enumerator kEDMA\_ErrorInterruptEnable

Enable interrupt while channel error occurs.

enumerator kEDMA\_MajorInterruptEnable

Enable interrupt while major count exhausted.

enumerator kEDMA\_HalfInterruptEnable

Enable interrupt while major count to half value.

enum \_edma\_transfer\_type

eDMA transfer type

*Values:*

enumerator kEDMA\_MemoryToMemory

Transfer from memory to memory

enumerator kEDMA\_PeripheralToMemory

Transfer from peripheral to memory

enumerator kEDMA\_MemoryToPeripheral

Transfer from memory to peripheral

enumerator kEDMA\_PeripheralToPeripheral

Transfer from Peripheral to peripheral

enum edma\_channel\_memory\_attribute

eDMA channel memory attribute

*Values:*

enumerator kEDMA\_ChannelNoWriteNoReadNoCacheNoBuffer

No write allocate, no read allocate, non-cacheable, non-bufferable.

enumerator kEDMA\_ChannelNoWriteNoReadNoCacheBufferable

No write allocate, no read allocate, non-cacheable, bufferable.

enumerator kEDMA\_ChannelNoWriteNoReadCacheableNoBuffer

No write allocate, no read allocate, cacheable, non-bufferable.

enumerator kEDMA\_ChannelNoWriteNoReadCacheableBufferable

No write allocate, no read allocate, cacheable, bufferable.

enumerator kEDMA\_ChannelNoWriteReadNoCacheNoBuffer

No write allocate, read allocate, non-cacheable, non-bufferable.

enumerator kEDMA\_ChannelNoWriteReadNoCacheBufferable

No write allocate, read allocate, non-cacheable, bufferable.

enumerator kEDMA\_ChannelNoWriteReadCacheableNoBuffer

No write allocate, read allocate, cacheable, non-bufferable.

enumerator kEDMA\_ChannelNoWriteReadCacheableBufferable

No write allocate, read allocate, cacheable, bufferable.

enumerator kEDMA\_ChannelWriteNoReadNoCacheNoBuffer

write allocate, no read allocate, non-cacheable, non-bufferable.

enumerator kEDMA\_ChannelWriteNoReadNoCacheBufferable

write allocate, no read allocate, non-cacheable, bufferable.

enumerator kEDMA\_ChannelWriteNoReadCacheableNoBuffer

write allocate, no read allocate, cacheable, non-bufferable.

enumerator kEDMA\_ChannelWriteNoReadCacheableBufferable

write allocate, no read allocate, cacheable, bufferable.

enumerator kEDMA\_ChannelWriteReadNoCacheNoBuffer

write allocate, read allocate, non-cacheable, non-bufferable.

enumerator kEDMA\_ChannelWriteReadNoCacheBufferable

write allocate, read allocate, non-cacheable, bufferable.

enumerator kEDMA\_ChannelWriteReadCacheableNoBuffer

write allocate, read allocate, cacheable, non-bufferable.

enumerator kEDMA\_ChannelWriteReadCacheableBufferable  
write allocate, read allocate, cacheable, bufferable.

enum \_edma\_channel\_swap\_size  
eDMA4 channel swap size

*Values:*

enumerator kEDMA\_ChannelSwapDisabled  
Swap is disabled.

enumerator kEDMA\_ChannelReadWith8bitSwap  
Swap occurs with respect to the read 8bit.

enumerator kEDMA\_ChannelReadWith16bitSwap  
Swap occurs with respect to the read 16bit.

enumerator kEDMA\_ChannelReadWith32bitSwap  
Swap occurs with respect to the read 32bit.

enumerator kEDMA\_ChannelWriteWith8bitSwap  
Swap occurs with respect to the write 8bit.

enumerator kEDMA\_ChannelWriteWith16bitSwap  
Swap occurs with respect to the write 16bit.

enumerator kEDMA\_ChannelWriteWith32bitSwap  
Swap occurs with respect to the write 32bit.

eDMA channel system bus information, \_edma\_channel\_sys\_bus\_info

*Values:*

enumerator kEDMA\_PrivilegedAccessLevel  
Privileged Access Level for DMA transfers. 0b - User protection level; 1b - Privileged protection level.

enumerator kEDMA\_MasterId  
DMA's master ID when channel is active and master ID replication is enabled.

enum \_edma\_channel\_access\_type  
eDMA4 channel access type

*Values:*

enumerator kEDMA\_ChannelDataAccess  
Data access for eDMA4 transfers.

enumerator kEDMA\_ChannelInstructionAccess  
Instruction access for eDMA4 transfers.

enum \_edma\_channel\_protection\_level  
eDMA4 channel protection level

*Values:*

enumerator kEDMA\_ChannelProtectionLevelUser  
user protection level for eDMA transfers.

enumerator kEDMA\_ChannelProtectionLevelPrivileged  
Privileged protection level eDMA transfers.

```
typedef enum _edma_transfer_size edma_transfer_size_t
    eDMA transfer configuration

typedef enum _edma_modulo edma_modulo_t
    eDMA modulo configuration

typedef enum _edma_bandwidth edma_bandwidth_t
    Bandwidth control.

typedef enum _edma_channel_link_type edma_channel_link_type_t
    Channel link type.

typedef enum _edma_transfer_type edma_transfer_type_t
    eDMA transfer type

typedef struct _edma_channel_Preemption_config edma_channel_Preemption_config_t
    eDMA channel priority configuration

typedef struct _edma_minor_offset_config edma_minor_offset_config_t
    eDMA minor offset configuration

typedef enum edma_channel_memory_attribute edma_channel_memory_attribute_t
    eDMA channel memory attribute

typedef enum _edma_channel_swap_size edma_channel_swap_size_t
    eDMA4 channel swap size

typedef enum _edma_channel_access_type edma_channel_access_type_t
    eDMA4 channel access type

typedef enum _edma_channel_protection_level edma_channel_protection_level_t
    eDMA4 channel protection level

typedef struct _edma_channel_config edma_channel_config_t
    eDMA4 channel configuration

typedef edma_core_tcd_t edma_tcd_t
    eDMA TCD.
```

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

```
typedef struct _edma_transfer_config edma_transfer_config_t
    edma4 channel transfer configuration
```

The transfer configuration structure support full feature configuration of the transfer control descriptor.

1.To perform a simple transfer, below members should be initialized at least .srcAddr - source address .dstAddr - destination address .srcWidthOfEachTransfer - data width of source address .dstWidthOfEachTransfer - data width of destination address, normally it should be as same as srcWidthOfEachTransfer .bytesEachRequest - bytes to be transferred in each DMA request .totalBytes - total bytes to be transferred .srcOffsetOfEachTransfer - offset value in bytes unit to be applied to source address as each source read is completed .dstOffsetOfEachTransfer - offset value in bytes unit to be applied to destination address as each destination write is completed enablchannelRequest - channel request can be enabled together with transfer configure submission

2.The transfer configuration structure also support advance feature: Programmable source/destination address range(MODULO) Programmable minor loop offset Programmable major loop offset Programmable channel chain feature Programmable channel transfer control descriptor link feature

---

**Note:** User should pay attention to the transfer size alignment limitation

- a. the bytesEachRequest should align with the srcWidthOfEachTransfer and the dstWidthOfEachTransfer that is to say bytesEachRequest % srcWidthOfEachTransfer should be 0
  - b. the srcOffsetOfEachTransfer and dstOffsetOfEachTransfer must be aligne with transfer width
  - c. the totalBytes should align with the bytesEachRequest
  - d. the srcAddr should align with the srcWidthOfEachTransfer
  - e. the dstAddr should align with the dstWidthOfEachTransfer
  - f. the srcAddr should align with srcAddrModulo if modulo feature is enabled
  - g. the dstAddr should align with dstAddrModulo if modulo feature is enabled If anyone of above condition can not be satisfied, the edma4 interfaces will generate assert error.
- 

typedef struct *\_edma\_config* edma\_config\_t  
eDMA global configuration structure.

typedef void (\*edma\_callback)(struct *\_edma\_handle* \*handle, void \*userData, bool transferDone, uint32\_t tcDs)

Define callback function for eDMA.

This callback function is called in the EDMA interrupt handle. In normal mode, run into callback function means the transfer users need is done. In scatter gather mode, run into callback function means a transfer control block (tcd) is finished. Not all transfer finished, users can get the finished tcd numbers using interface EDMA\_GetUnusedTCDNumber.

**Param handle**

EDMA handle pointer, users shall not touch the values inside.

**Param userData**

The callback user parameter pointer. Users can use this parameter to involve things users need to change in EDMA callback function.

**Param transferDone**

If the current loaded transfer done. In normal mode it means if all transfer done. In scatter gather mode, this parameter shows is the current transfer block in EDMA register is done. As the load of core is different, it will be different if the new tcd loaded into EDMA registers while this callback called. If true, it always means new tcd still not loaded into registers, while false means new tcd already loaded into registers.

**Param tcDs**

How many tcDs are done from the last callback. This parameter only used in scatter gather mode. It tells user how many tcDs are finished between the last callback and this.

typedef struct *\_edma\_handle* edma\_handle\_t  
eDMA transfer handle structure

FSL\_EDMA\_DRIVER\_EDMA4  
eDMA driver name

EDMA\_ALLOCATE\_TCD(name, number)  
Macro used for allocate edma TCD.

DMA\_DCHPRI\_INDEX(channel)  
Compute the offset unit from DCHPRI3.

```
struct _edma_channel_Preemption_config
#include <fsl_edma.h> eDMA channel priority configuration
```

### Public Members

```
bool enableChannelPreemption
    If true: a channel can be suspended by other channel with higher priority
bool enablePreemptAbility
    If true: a channel can suspend other channel with low priority
uint8_t channelPriority
    Channel priority
```

```
struct _edma_minor_offset_config
#include <fsl_edma.h> eDMA minor offset configuration
```

### Public Members

```
bool enableSrcMinorOffset
    Enable(true) or Disable(false) source minor loop offset.
bool enableDestMinorOffset
    Enable(true) or Disable(false) destination minor loop offset.
uint32_t minorOffset
    Offset for a minor loop mapping.
```

```
struct _edma_channel_config
#include <fsl_edma.h> eDMA4 channel configuration
```

### Public Members

```
edma_channel_Preemption_config_t channelPreemptionConfig
    channel preemption configuration
edma_channel_memory_attribute_t channelReadMemoryAttribute
    channel memory read attribute configuration
edma_channel_memory_attribute_t channelWriteMemoryAttribute
    channel memory write attribute configuration
edma_channel_swap_size_t channelSwapSize
    channel swap size configuration
edma_channel_access_type_t channelAccessType
    channel access type configuration
uint8_t channelDataSignExtensionBitPosition
    channel data sign extension bit position configuration
uint32_t channelRequestSource
    hardware service request source for the channel
bool enableMasterIDReplication
    enable master ID replication
edma_channel_protection_level_t protectionLevel
    protection level
```



struct `_edma_transfer_config`

*#include <fsl\_edma.h>* edma4 channel transfer configuration

The transfer configuration structure support full feature configuration of the transfer control descriptor.

1.To perform a simple transfer, below members should be initialized at least .srcAddr - source address .dstAddr - destination address .srcWidthOfEachTransfer - data width of source address .dstWidthOfEachTransfer - data width of destination address, normally it should be as same as srcWidthOfEachTransfer .bytesEachRequest - bytes to be transferred in each DMA request .totalBytes - total bytes to be transferred .srcOffsetOfEachTransfer - offset value in bytes unit to be applied to source address as each source read is completed .dstOffsetOfEachTransfer - offset value in bytes unit to be applied to destination address as each destination write is completed enableChannelRequest - channel request can be enabled together with transfer configure submission

2.The transfer configuration structure also support advance feature: Programmable source/destination address range(MODULO) Programmable minor loop offset Programmable major loop offset Programmable channel chain feature Programmable channel transfer control descriptor link feature

---

**Note:** User should pay attention to the transfer size alignment limitation

- a. the bytesEachRequest should align with the srcWidthOfEachTransfer and the dstWidthOfEachTransfer that is to say bytesEachRequest % srcWidthOfEachTransfer should be 0
  - b. the srcOffsetOfEachTransfer and dstOffsetOfEachTransfer must be aligne with transfer width
  - c. the totalBytes should align with the bytesEachRequest
  - d. the srcAddr should align with the srcWidthOfEachTransfer
  - e. the dstAddr should align with the dstWidthOfEachTransfer
  - f. the srcAddr should align with srcAddrModulo if modulo feature is enabled
  - g. the dstAddr should align with dstAddrModulo if modulo feature is enabled If anyone of above condition can not be satisfied, the edma4 interfaces will generate assert error.
- 

## Public Members

uint32\_t srcAddr

Source data address.

uint32\_t destAddr

Destination data address.

*edma\_transfer\_size\_t* srcTransferSize

Source data transfer size.

*edma\_transfer\_size\_t* destTransferSize

Destination data transfer size.

int16\_t srcOffset

Sign-extended offset value in byte unit applied to the current source address to form the next-state value as each source read is completed

`int16_t destOffset`

Sign-extended offset value in byte unit applied to the current destination address to form the next-state value as each destination write is completed.

`uint32_t minorLoopBytes`

bytes in each minor loop or each request range:  $1 - (2^{30} - 1)$  when minor loop mapping is enabled range:  $1 - (2^{10} - 1)$  when minor loop mapping is enabled and source or dest minor loop offset is enabled range:  $1 - (2^{32} - 1)$  when minor loop mapping is disabled

`uint32_t majorLoopCounts`

minor loop counts in each major loop, should be 1 at least for each transfer range:  $(0 - (2^{15} - 1))$  when minor loop channel link is disabled range:  $(0 - (2^9 - 1))$  when minor loop channel link is enabled total bytes in a transfer = `minorLoopCountsEachMajorLoop * bytesEachMinorLoop`

`uint16_t enabledInterruptMask`

channel interrupt to enable, can be OR'ed value of `_edma_interrupt_enable`

`edma_modulo_t srcAddrModulo`

source circular data queue range

`int32_t srcMajorLoopOffset`

source major loop offset

`edma_modulo_t dstAddrModulo`

destination circular data queue range

`int32_t dstMajorLoopOffset`

destination major loop offset

`bool enableSrcMinorLoopOffset`

enable source minor loop offset

`bool enableDstMinorLoopOffset`

enable dest minor loop offset

`int32_t minorLoopOffset`

burst offset, the offset will be applied after minor loop update

`bool enableChannelMajorLoopLink`

channel link when major loop complete

`uint32_t majorLoopLinkChannel`

major loop link channel number

`bool enableChannelMinorLoopLink`

channel link when minor loop complete

`uint32_t minorLoopLinkChannel`

minor loop link channel number

`edma_tcd_t *linkTCD`

pointer to the link transfer control descriptor

`struct __edma_config`

`#include <fsl_edma.h>` eDMA global configuration structure.

## Public Members

bool enableMasterIdReplication

Enable (true) master ID replication. If Master ID replication is disabled, the privileged protection level (supervisor mode) for eDMA4 transfers is used.

bool enableGlobalChannelLink

Enable(true) channel linking is available and controlled by each channel's link settings.

bool enableHaltOnError

Enable (true) transfer halt on error. Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

bool enableDebugMode

Enable(true) eDMA4 debug mode. When in debug mode, the eDMA4 stalls the start of a new channel. Executing channels are allowed to complete.

bool enableRoundRobinArbitration

Enable(true) channel linking is available and controlled by each channel's link settings.

*edma\_channel\_config\_t* \*channelConfig[1]

channel preemption configuration

struct *\_edma\_handle*

*#include <fsl\_edma.h>* eDMA transfer handle structure

### Public Members

*edma\_callback* callback

Callback function for major count exhausted.

void \*userData

Callback function parameter.

*EDMA\_ChannelType* \*channelBase

eDMA peripheral channel base address.

*EDMA\_Type* \*base

eDMA peripheral base address

*EDMA\_TCDType* \*tcdBase

eDMA peripheral tcd base address.

*edma\_tcd\_t* \*tcdPool

Pointer to memory stored TCDs.

uint32\_t channel

eDMA channel number.

volatile int8\_t header

The first TCD index. Should point to the next TCD to be loaded into the eDMA engine.

volatile int8\_t tail

The last TCD index. Should point to the next TCD to be stored into the memory pool.

volatile int8\_t tcdUsed

The number of used TCD slots. Should reflect the number of TCDs can be used/loaded in the memory.

volatile int8\_t tcdSize

The total number of TCD slots in the queue.

## 2.7 eDMA core Driver

```
enum _edma_tcd_type
    eDMA tcd flag type
    Values:
    enumerator kEDMA_EDMA4Flag
        Data access for eDMA4 transfers.
    enumerator kEDMA_EDMA5Flag
        Instruction access for eDMA4 transfers.
typedef struct _edma_core_mp edma_core_mp_t
    edma core channel struture definition
typedef struct _edma_core_channel edma_core_channel_t
    edma core channel struture definition
typedef enum _edma_tcd_type edma_tcd_type_t
    eDMA tcd flag type
typedef struct _edma5_core_tcd edma5_core_tcd_t
    edma5 core TCD struture definition
typedef struct _edma4_core_tcd edma4_core_tcd_t
    edma4 core TCD struture definition
typedef struct _edma_core_tcd edma_core_tcd_t
    edma core TCD struture definition
typedef edma_core_channel_t EDMA_ChannelType
    EDMA typedef.
typedef edma_core_tcd_t EDMA_TCDType
typedef void EDMA_Type
DMA_CORE_MP_CSR_EDBG_MASK
DMA_CORE_MP_CSR_ERCA_MASK
DMA_CORE_MP_CSR_HAE_MASK
DMA_CORE_MP_CSR_HALT_MASK
DMA_CORE_MP_CSR_GCLC_MASK
DMA_CORE_MP_CSR_GMRC_MASK
DMA_CORE_MP_CSR_EDBG(x)
DMA_CORE_MP_CSR_ERCA(x)
DMA_CORE_MP_CSR_HAE(x)
DMA_CORE_MP_CSR_HALT(x)
DMA_CORE_MP_CSR_GCLC(x)
DMA_CORE_MP_CSR_GMRC(x)
DMA_CSR_INTMAJOR_MASK
```

DMA\_CSR\_INTHALF\_MASK  
DMA\_CSR\_DREQ\_MASK  
DMA\_CSR\_ESG\_MASK  
DMA\_CSR\_BWC\_MASK  
DMA\_CSR\_BWC(x)  
DMA\_CSR\_START\_MASK  
DMA\_CITER\_ELINKNO\_CITER\_MASK  
DMA\_BITER\_ELINKNO\_BITER\_MASK  
DMA\_CITER\_ELINKNO\_CITER\_SHIFT  
DMA\_CITER\_ELINKYES\_CITER\_MASK  
DMA\_CITER\_ELINKYES\_CITER\_SHIFT  
DMA\_ATTR\_SMOD\_MASK  
DMA\_ATTR\_DMOD\_MASK  
DMA\_CITER\_ELINKNO\_ELINK\_MASK  
DMA\_CSR\_MAJORELINK\_MASK  
DMA\_BITER\_ELINKYES\_ELINK\_MASK  
DMA\_CITER\_ELINKYES\_ELINK\_MASK  
DMA\_CSR\_MAJORLINKCH\_MASK  
DMA\_BITER\_ELINKYES\_LINKCH\_MASK  
DMA\_CITER\_ELINKYES\_LINKCH\_MASK  
DMA\_NBYTES\_MLOFFYES\_MLOFF\_MASK  
DMA\_NBYTES\_MLOFFYES\_DMLOE\_MASK  
DMA\_NBYTES\_MLOFFYES\_SMLOE\_MASK  
DMA\_NBYTES\_MLOFFNO\_NBYTES\_MASK  
DMA\_ATTR\_DMOD(x)  
DMA\_ATTR\_SMOD(x)  
DMA\_BITER\_ELINKYES\_LINKCH(x)  
DMA\_CITER\_ELINKYES\_LINKCH(x)  
DMA\_NBYTES\_MLOFFYES\_MLOFF(x)  
DMA\_NBYTES\_MLOFFYES\_DMLOE(x)  
DMA\_NBYTES\_MLOFFYES\_SMLOE(x)  
DMA\_NBYTES\_MLOFFNO\_NBYTES(x)  
DMA\_NBYTES\_MLOFFYES\_NBYTES(x)

DMA\_ATTR\_DSIZE(x)  
DMA\_ATTR\_SSIZE(x)  
DMA\_CSR\_DREQ(x)  
DMA\_CSR\_MAJORLINKCH(x)  
DMA\_CH\_MATTR\_WCACHE(x)  
DMA\_CH\_MATTR\_RCACHE(x)  
DMA\_CH\_CSR\_SIGNEXT\_MASK  
DMA\_CH\_CSR\_SIGNEXT\_SHIFT  
DMA\_CH\_CSR\_SWAP\_MASK  
DMA\_CH\_CSR\_SWAP\_SHIFT  
DMA\_CH\_SBR\_INSTR\_MASK  
DMA\_CH\_SBR\_INSTR\_SHIFT  
DMA\_CH\_SBR\_EMI\_MASK  
DMA\_CH\_SBR\_EMI\_SHIFT  
DMA\_CH\_MUX\_SOURCE(x)  
DMA\_ERR\_DBE\_FLAG  
    **DMA error flag.**  
DMA\_ERR\_SBE\_FLAG  
DMA\_ERR\_SGE\_FLAG  
DMA\_ERR\_NCE\_FLAG  
DMA\_ERR\_DOE\_FLAG  
DMA\_ERR\_DAE\_FLAG  
DMA\_ERR\_SOE\_FLAG  
DMA\_ERR\_SAE\_FLAG  
DMA\_ERR\_ERRCHAN\_FLAG  
DMA\_ERR\_ECX\_FLAG  
DMA\_ERR\_FLAG  
DMA\_CLEAR\_DONE\_STATUS(base, channel)  
    **get/clear DONE bit**  
DMA\_GET\_DONE\_STATUS(base, channel)  
DMA\_ENABLE\_ERROR\_INT(base, channel)  
    **enable/disable error interrupt**  
DMA\_DISABLE\_ERROR\_INT(base, channel)  
DMA\_CLEAR\_ERROR\_STATUS(base, channel)  
    **get/clear error status**

```

DMA_GET_ERROR_STATUS(base, channel)
DMA_CLEAR_INT_STATUS(base, channel)
    get/clear INT status
DMA_GET_INT_STATUS(base, channel)
DMA_ENABLE_MAJOR_INT(base, channel)
    enable/dsiable MAJOR/HALF INT
DMA_ENABLE_HALF_INT(base, channel)
DMA_DISABLE_MAJOR_INT(base, channel)
DMA_DISABLE_HALF_INT(base, channel)
EDMA_TCD_ALIGN_SIZE
    EDMA tcd align size.
EDMA_CORE_BASE(base)
    EDMA base address convert macro.
EDMA_MP_BASE(base)
EDMA_CHANNEL_BASE(base, channel)
EDMA_TCD_BASE(base, channel)
EDMA_TCD_TYPE(x)
    EDMA TCD type macro.
EDMA_TCD_SADDR(tcd, flag)
    EDMA TCD address convert macro.
EDMA_TCD_SOFF(tcd, flag)
EDMA_TCD_ATTR(tcd, flag)
EDMA_TCD_NBYTES(tcd, flag)
EDMA_TCD_SLAST(tcd, flag)
EDMA_TCD_DADDR(tcd, flag)
EDMA_TCD_DOFF(tcd, flag)
EDMA_TCD_CITER(tcd, flag)
EDMA_TCD_DLAST_SGA(tcd, flag)
EDMA_TCD_CSR(tcd, flag)
EDMA_TCD_BITER(tcd, flag)
struct _edma_core_mp
    #include <fsl_edma_core.h> edma core channel struture definition

```

### Public Members

```

__IO uint32_t MP_CSR
    Channel Control and Status, array offset: 0x10000, array step: 0x10000

```

```
__IO uint32_t MP_ES
    Channel Error Status, array offset: 0x10004, array step: 0x10000
struct __edma_core_channel
    #include <fsl_edma_core.h> edma core channel struture definition
```

### Public Members

```
__IO uint32_t CH_CSR
    Channel Control and Status, array offset: 0x10000, array step: 0x10000
__IO uint32_t CH_ES
    Channel Error Status, array offset: 0x10004, array step: 0x10000
__IO uint32_t CH_INT
    Channel Interrupt Status, array offset: 0x10008, array step: 0x10000
__IO uint32_t CH_SBR
    Channel System Bus, array offset: 0x1000C, array step: 0x10000
__IO uint32_t CH_PRI
    Channel Priority, array offset: 0x10010, array step: 0x10000
struct __edma5_core_tcd
    #include <fsl_edma_core.h> edma5 core TCD struture definition
```

### Public Members

```
__IO uint32_t SADDR
    SADDR register, used to save source address
__IO uint32_t SADDR_HIGH
    SADDR HIGH register, used to save source address
__IO uint16_t SOFF
    SOFF register, save offset bytes every transfer
__IO uint16_t ATTR
    ATTR register, source/destination transfer size and modulo
__IO uint32_t NBYTES
    Nbytes register, minor loop length in bytes
__IO uint32_t SLAST
    SLAST register
__IO uint32_t SLAST_SDA_HIGH
    SLAST SDA HIGH register
__IO uint32_t DADDR
    DADDR register, used for destination address
__IO uint32_t DADDR_HIGH
    DADDR HIGH register, used for destination address
__IO uint32_t DLAST_SGA
    DLASTSGA register, next tcd address used in scatter-gather mode
__IO uint32_t DLAST_SGA_HIGH
    DLASTSGA HIGH register, next tcd address used in scatter-gather mode
```



```
__IO uint16_t DOFF
    DOFF register, used for destination offset
__IO uint16_t CITER
    CITER register, current minor loop numbers, for unfinished minor loop.
__IO uint16_t CSR
    CSR register, for TCD control status
__IO uint16_t BITER
    BITER register, begin minor loop count.
uint8_t RESERVED[16]
    Aligned 64 bytes
struct _edma4_core_tcd
    #include <fsl_edma_core.h> edma4 core TCD struture definition
```

### Public Members

```
__IO uint32_t SADDR
    SADDR register, used to save source address
__IO uint16_t SOFF
    SOFF register, save offset bytes every transfer
__IO uint16_t ATTR
    ATTR register, source/destination transfer size and modulo
__IO uint32_t NBYTES
    Nbytes register, minor loop length in bytes
__IO uint32_t SLAST
    SLAST register
__IO uint32_t DADDR
    DADDR register, used for destination address
__IO uint16_t DOFF
    DOFF register, used for destination offset
__IO uint16_t CITER
    CITER register, current minor loop numbers, for unfinished minor loop.
__IO uint32_t DLAST_SGA
    DLASTSGA register, next tcd address used in scatter-gather mode
__IO uint16_t CSR
    CSR register, for TCD control status
__IO uint16_t BITER
    BITER register, begin minor loop count.
struct _edma_core_tcd
    #include <fsl_edma_core.h> edma core TCD struture definition
union MP_REGS
```

**Public Members**

```
struct _edma_core_mp EDMA5_REG
```

```
struct EDMA5_REG
```

**Public Members**

```
__IO uint32_t MP_INT_LOW
    Channel Control and Status, array offset: 0x10008, array step: 0x10000
__IO uint32_t MP_INT_HIGH
    Channel Control and Status, array offset: 0x1000C, array step: 0x10000
__IO uint32_t MP_HRS_LOW
    Channel Control and Status, array offset: 0x10010, array step: 0x10000
__IO uint32_t MP_HRS_HIGH
    Channel Control and Status, array offset: 0x10014, array step: 0x10000
__IO uint32_t MP_STOPCH
    Channel Control and Status, array offset: 0x10020, array step: 0x10000
__IO uint32_t MP_SSR_LOW
    Channel Control and Status, array offset: 0x10030, array step: 0x10000
__IO uint32_t MP_SSR_HIGH
    Channel Control and Status, array offset: 0x10034, array step: 0x10000
__IO uint32_t CH_GRPRI [64]
    Channel Control and Status, array offset: 0x10100, array step: 0x10000
__IO uint32_t CH_MUX [64]
    Channel Control and Status, array offset: 0x10200, array step: 0x10000
__IO uint32_t CH_PROT [64]
    Channel Control and Status, array offset: 0x10400, array step: 0x10000
```

```
union CH_REGS
```

**Public Members**

```
struct _edma_core_channel EDMA5_REG
```

```
struct _edma_core_channel EDMA4_REG
```

```
struct EDMA5_REG
```

**Public Members**

```
__IO uint32_t CH_MATTR
    Memory Attributes Register, array offset: 0x10018, array step: 0x8000
```

```
struct EDMA4_REG
```

### Public Members

\_\_IO uint32\_t CH\_MUX  
Channel Multiplexor Configuration, array offset: 0x10014, array step: 0x10000

\_\_IO uint16\_t CH\_MATTR  
Memory Attributes Register, array offset: 0x10018, array step: 0x8000

union TCD\_REGS

### Public Members

*edma4\_core\_tcd\_t* edma4\_tcd

## 2.8 eDMA soc Driver

FSL\_EDMA\_SOC\_DRIVER\_VERSION  
Driver version 2.0.0.

FSL\_EDMA\_SOC\_IP\_DMA3  
DMA IP version.

FSL\_EDMA\_SOC\_IP\_DMA4

EDMA\_BASE\_PTRS  
DMA base table.

EDMA\_CHN\_IRQS

EDMA\_CHANNEL\_OFFSET  
EDMA base address convert macro.

EDMA\_CHANNEL\_ARRAY\_STEP(base)

## 2.9 EIM: error injection module

FSL\_ERM\_DRIVER\_VERSION  
Driver version.

void EIM\_Init(EIM\_Type \*base)  
EIM module initialization function.

**Parameters**

- base – EIM base address.

void EIM\_Deinit(EIM\_Type \*base)  
De-initializes the EIM.

## 2.10 EQDC: Enhanced Quadrature Encoder/Decoder

```
void EQDC_Init(EQDC_Type *base, const eqdc_config_t *psConfig)
```

Initializes the EQDC module.

This function initializes the EQDC by enabling the IP bus clock (optional).

#### Parameters

- base – EQDC peripheral base address.
- psConfig – Pointer to configuration structure.

```
void EQDC_GetDefaultConfig(eqdc_config_t *psConfig)
```

Gets an available pre-defined configuration.

The default value are:

```
psConfig->enableReverseDirection      = false;
psConfig->countOnce                    = false;
psConfig->operateMode                  = kEQDC_QuadratureDecodeOperationMode;
psConfig->countMode                    = kEQDC_QuadratureX4;
psConfig->homeEnableInitPosCounterMode = kEQDC_HomeInitPosCounterDisabled;
psConfig->indexPresetInitPosCounterMode = kEQDC_IndexInitPosCounterDisabled;
psConfig->enableIndexInitPositionCounter = false;
psConfig->enableDma                    = false;
psConfig->bufferedRegisterLoadMode     = false;
psConfig->enableTriggerInitPositionCounter = false;
psConfig->enableTriggerClearPositionRegisters = false;
psConfig->enableTriggerHoldPositionRegisters = false;
psConfig->enableWatchdog                = false;
psConfig->watchdogTimeoutValue          = 0xFFFFU;
psConfig->filterPhaseA                  = 0U;
psConfig->filterPhaseB                  = 0U;
psConfig->filterIndPre                  = 0U;
psConfig->filterHomEna                  = 0U;
psConfig->filterClockSourceSelection    = false;
psConfig->filterSampleCount             = kEQDC_Filter3Samples;
psConfig->filterSamplePeriod            = 0U;
psConfig->outputPulseMode               = kEQDC_OutputPulseOnCounterEqualCompare;
psConfig->positionCompareValue[0]       = 0xFFFFFFFFU;
psConfig->positionCompareValue[1]       = 0xFFFFFFFFU;
psConfig->positionCompareValue[2]       = 0xFFFFFFFFU;
psConfig->positionCompareValue[3]       = 0xFFFFFFFFU;
psConfig->revolutionCountCondition      = kEQDC_RevolutionCountOnIndexPulse;
psConfig->positionModulusValue          = 0U;
psConfig->positionInitialValue          = 0U;
psConfig->positionCounterValue          = 0U;
psConfig->enablePeriodMeasurement       = false;
psConfig->prescaler                     = kEQDC_Prescaler1;
psConfig->enabledInterruptsMask         = 0U;
```

#### Parameters

- psConfig – Pointer to configuration structure.

```
void EQDC_Deinit(EQDC_Type *base)
```

De-initializes the EQDC module.

This function deinitializes the EQDC by disabling the IP bus clock (optional).

#### Parameters

- base – EQDC peripheral base address.

```
void EQDC_SetOperateMode(EQDC_Type *base, eqdc_operate_mode_t operateMode)
```

Initializes the mode of operation.

This function initializes mode of operation by enabling the IP bus clock (optional).

#### Parameters

- `base` – EQDC peripheral base address.
- `operateMode` – Select operation mode.

static inline void EQDC\_SetCountMode(EQDC\_Type \*base, *eqdc\_count\_mode\_t* countMode)

Initializes the mode of count.

These bits control the basic counting and behavior of Position Counter and Position Difference Counter. Setting CTRL[REV] to 1 can reverse the counting direction. 1.In quadrature Mode (CTRL[PH1] = 0): 00b - CM0: Normal/Reverse Quadrature X4 01b - CM1: Normal/Reverse Quadrature X2 10b - CM2: Normal/Reverse Quadrature X1 11b - CM3: Reserved 2.In Single Phase Mode (CTRL[PH1] = 1): 00b - CM0: UP/DOWN Pulse Count Mode 01b - CM1: Signed Mode, count PHASEA rising/falling edge, position counter counts up when PHASEB is low and counts down when PHASEB is high 10b - CM2: Signed Count Mode, count PHASEA rising edge only, position counter counts up when PHASEB is low and counts down when PHASEB is high 11b - CM3: Reserved

#### Parameters

- `base` – EQDC peripheral base address.
- `countMode` – Select count mode.

static inline void EQDC\_EnableWatchdog(EQDC\_Type \*base, bool bEnable)

Enable watchdog for EQDC module.

#### Parameters

- `base` – EQDC peripheral base address
- `bEnable` – Enables or disables the watchdog

static inline void EQDC\_SetWatchdogTimeout(EQDC\_Type \*base, uint16\_t u16Timeout)

Set watchdog timeout value.

#### Parameters

- `base` – EQDC peripheral base address
- `u16Timeout` – Number of clock cycles, plus one clock cycle that the watchdog timer counts before timing out

static inline void EQDC\_EnableDMA(EQDC\_Type \*base, bool bEnable)

Enable DMA for EQDC module.

#### Parameters

- `base` – EQDC peripheral base address
- `bEnable` – Enables or disables the DMA

static inline void EQDC\_SetBufferedRegisterLoadUpdateMode(EQDC\_Type \*base)

Set Buffered Register Load (Update) Mode.

This bit selects the loading time point of the buffered compare registers UCOMPx/LCOMPx, x=0~3, initial register (UNIT/LINIT), and modulus register (UMOD/LMOD). Buffered registers are loaded and take effect at the next roll-over or roll-under if CTRL[LDOK] is set.

#### Parameters

- `base` – EQDC peripheral base address

static inline void EQDC\_ClearBufferedRegisterLoadUpdateMode(EQDC\_Type \*base)

Clear Buffered Register Load (Update) Mode.

Buffered Register Load (Update) Mode bit selects the loading time point of the buffered compare registers UCOMPx/LCOMPx, x=0~3, initial register (UINIT/LINIT), and modulus register (UMOD/LMOD). Buffered registers are loaded and take effect immediately upon CTRL[LDOK] is set.

**Parameters**

- base – EQDC peripheral base address

static inline void EQDC\_SetEqdcLdok(EQDC\_Type \*base)

Set load okay.

Load okay enables that the outer-set values of buffered compare registers (UCOMPx/LCOMPx, x=0~3), initial register(UINIT/LINIT) and modulus register(UMOD/LMOD) can be loaded into their inner-sets and take effect. When LDOK is set, this loading action occurs at the next position counter roll-over or roll-under if CTRL2[LDMOD] is set, or it occurs immediately if CTRL2[LDMOD] is cleared. LDOK is automatically cleared after the values in outer-set is loaded into the inner-set.

**Parameters**

- base – EQDC peripheral base address.

static inline uint8\_t EQDC\_GetEqdcLdok(EQDC\_Type \*base)

Get load okay.

**Parameters**

- base – EQDC peripheral base address.

static inline void EQDC\_ClearEqdcLdok(EQDC\_Type \*base)

Clear load okay.

**Parameters**

- base – EQDC peripheral base address.

static inline uint32\_t EQDC\_GetStatusFlags(EQDC\_Type \*base)

Get the status flags.

**Parameters**

- base – EQDC peripheral base address.

**Returns**

Logical OR'ed value of the status flags, \_eqdc\_status\_flags.

static inline void EQDC\_ClearStatusFlags(EQDC\_Type \*base, uint32\_t u32Flags)

Clear the status flags.

**Parameters**

- base – EQDC peripheral base address.
- u32Flags – Logical OR'ed value of the flags to clear, \_eqdc\_status\_flags.

static inline uint16\_t EQDC\_GetSignalStatusFlags(EQDC\_Type \*base)

Get the signals' real-time status.

**Parameters**

- base – EQDC peripheral base address.

**Returns**

Logical OR'ed value of the real-time signal status, \_eqdc\_signal\_status.

```
static inline eqdc_count_direction_flag_t EQDC_GetLastCountDirection(EQDC_Type *base)
```

Get the direction of the last count.

**Parameters**

- base – EQDC peripheral base address.

**Returns**

Direction of the last count.

```
static inline void EQDC_EnableInterrupts(EQDC_Type *base, uint32_t u32Interrupts)
```

Enable the interrupts.

**Parameters**

- base – EQDC peripheral base address.
- u32Interrupts – Logical OR'ed value of the interrupts, \_eqdc\_interrupt\_enable.

```
static inline void EQDC_DisableInterrupts(EQDC_Type *base, uint32_t u32Interrupts)
```

Disable the interrupts.

**Parameters**

- base – EQDC peripheral base address.
- u32Interrupts – Logical OR'ed value of the interrupts, \_eqdc\_interrupt\_enable.

```
static inline void EQDC_DoSoftwareLoadInitialPositionValue(EQDC_Type *base)
```

Load the initial position value to position counter.

Software trigger to load the initial position value (UNIT and LINIT) contents to position counter (UPOS and LPOS), so that to provide the consistent operation the position counter registers.

**Parameters**

- base – EQDC peripheral base address.

```
static inline void EQDC_SetInitialPositionValue(EQDC_Type *base, uint32_t  
u32PositionInitValue)
```

Set initial position value for EQDC module.

Set the position counter initial value (UNIT, LINIT). After writing values to the UNIT and LINIT registers, the values are “buffered” into outer-set registers temporarily. Values will be loaded into inner-set registers and take effect using the following two methods:

- a. If CTRL2[LDMODE] is 1, “buffered” values are loaded into inner-set and take effect at the next roll-over or roll-under if CTRL[LDOK] is set.
- b. If CTRL2[LDMODE] is 0, “buffered” values are loaded into inner-set and take effect immediately when CTRL[LDOK] is set.

**Parameters**

- base – EQDC peripheral base address
- u32PositionInitValue – Position initial value

```
static inline void EQDC_SetPositionCounterValue(EQDC_Type *base, uint32_t  
positionCounterValue)
```

Set position counter value.

Set the position counter value (POS or UPOS, LPOS).

**Parameters**

- base – EQDC peripheral base address
- positionCounterValue – Position counter value

```
static inline void EQDC_SetPositionModulusValue(EQDC_Type *base, uint32_t
                                              positionModulusValue)
```

Set position counter modulus value.

Set the position counter modulus value (UMOD, LMOD). After writing values to the UMOD and LMOD registers, the values are “buffered” into outer-set registers temporarily. Values will be loaded into inner-set registers and take effect using the following two methods:

- a. If CTRL2[LDMODE] is 1, “buffered” values are loaded into inner-set and take effect at the next roll-over or roll-under if CTRL[LDOK] is set.
- b. If CTRL2[LDMODE] is 0, “buffered” values are loaded into inner-set and take effect immediately when CTRL[LDOK] is set.

#### Parameters

- base – EQDC peripheral base address
- positionModulusValue – Position modulus value

```
static inline void EQDC_SetPositionCompare0Value(EQDC_Type *base, uint32_t
                                              u32PositionComp0Value)
```

Set position counter compare 0 value.

Set the position counter compare 0 value (UCOMP0, LCOMP0). After writing values to the UCOMP0 and LCOMP0 registers, the values are “buffered” into outer-set registers temporarily. Values will be loaded into inner-set registers and take effect using the following two methods:

- a. If CTRL2[LDMODE] is 1, “buffered” values are loaded into inner-set and take effect at the next roll-over or roll-under if CTRL[LDOK] is set.
- b. If CTRL2[LDMODE] is 0, “buffered” values are loaded into inner-set and take effect immediately when CTRL[LDOK] is set.

#### Parameters

- base – EQDC peripheral base address
- u32PositionComp0Value – Position modulus value

```
static inline void EQDC_SetPositionCompare1Value(EQDC_Type *base, uint32_t
                                              u32PositionComp1Value)
```

Set position counter compare 1 value.

Set the position counter compare 1 value (UCOMP1, LCOMP1). After writing values to the UCOMP1 and LCOMP1 registers, the values are “buffered” into outer-set registers temporarily. Values will be loaded into inner-set registers and take effect using the following two methods:

- a. If CTRL2[LDMODE] is 1, “buffered” values are loaded into inner-set and take effect at the next roll-over or roll-under if CTRL[LDOK] is set.
- b. If CTRL2[LDMODE] is 0, “buffered” values are loaded into inner-set and take effect immediately when CTRL[LDOK] is set.

#### Parameters

- base – EQDC peripheral base address
- u32PositionComp1Value – Position modulus value



```
static inline void EQDC_SetPositionCompare2Value(EQDC_Type *base, uint32_t
                                              u32PositionComp2Value)
```

Set position counter compare 2 value.

Set the position counter compare 2 value (UCOMP2, LCOMP2). After writing values to the UCOMP2 and LCOMP2 registers, the values are “buffered” into outer-set registers temporarily. Values will be loaded into inner-set registers and take effect using the following two methods:

- a. If CTRL2[LDMODE] is 1, “buffered” values are loaded into inner-set and take effect at the next roll-over or roll-under if CTRL[LDOK] is set.
- b. If CTRL2[LDMODE] is 0, “buffered” values are loaded into inner-set and take effect immediately when CTRL[LDOK] is set.

#### Parameters

- base – EQDC peripheral base address
- u32PositionComp2Value – Position modulus value

```
static inline void EQDC_SetPositionCompare3Value(EQDC_Type *base, uint32_t
                                              u32PositionComp3Value)
```

Set position counter compare 3 value.

Set the position counter compare 3 value (UCOMP3, LCOMP3). After writing values to the UCOMP3 and LCOMP3 registers, the values are “buffered” into outer-set registers temporarily. Values will be loaded into inner-set registers and take effect using the following two methods:

- a. If CTRL2[LDMODE] is 1, “buffered” values are loaded into inner-set and take effect at the next roll-over or roll-under if CTRL[LDOK] is set.
- b. If CTRL2[LDMODE] is 0, “buffered” values are loaded into inner-set and take effect immediately when CTRL[LDOK] is set.

#### Parameters

- base – EQDC peripheral base address
- u32PositionComp3Value – Position modulus value

```
static inline uint32_t EQDC_GetPosition(EQDC_Type *base)
```

Get the current position counter’s value.

#### Parameters

- base – EQDC peripheral base address.

#### Returns

Current position counter’s value.

```
static inline uint32_t EQDC_GetHoldPosition(EQDC_Type *base)
```

Get the hold position counter’s value.

The position counter (POS or UPOS, LPOS) value is loaded to hold position (POSH or UPOSH, LPOSH) when:

- a. Position register (POS or UPOS, LPOS), or position difference register (POSD), or revolution register (REV) is read.
- b. TRIGGER happens and TRIGGER is enabled to update the hold registers.

#### Parameters

- base – EQDC peripheral base address.

**Returns**

Hold position counter's value.

```
static inline uint32_t EQDC_GetHoldPosition1(EQDC_Type *base)
```

Get the hold position counter1's value.

The Upper Position Counter Hold Register 1(UPOSH1) shares the same address with UCOMP1. When read, this register means the value of UPOSH1, which is the upper 16 bits of POSH1. The Lower Position Counter Hold Register 1(LPOSH1) shares the same address with LCOMP1. When read, this register means the value of LPOSH1, which is the lower 16 bits of POSH1. Position counter is captured into POSH1 on the rising edge of ICAP[1].

**Parameters**

- base – EQDC peripheral base address.

**Returns**

Hold position counter1's value.

```
static inline uint32_t EQDC_GetHoldPosition2(EQDC_Type *base)
```

Get the hold position counter2's value.

The Upper Position Counter Hold Register 2(UPOSH2) shares the same address with UCOMP2. When read, this register means the value of UPOSH2, which is the upper 16 bits of POSH2. The Lower Position Counter Hold Register 2(LPOSH2) shares the same address with LCOMP2. When read, this register means the value of LPOSH2, which is the lower 16 bits of POSH2. Position counter is captured into POSH2 on the rising edge of ICAP[2].

**Parameters**

- base – EQDC peripheral base address.

**Returns**

Hold position counter2's value.

```
static inline uint32_t EQDC_GetHoldPosition3(EQDC_Type *base)
```

Get the hold position counter3's value.

The Upper Position Counter Hold Register 3(UPOSH3) shares the same address with UCOMP3. When read, this register means the value of UPOSH3, which is the upper 16 bits of POSH3. The Lower Position Counter Hold Register 3(LPOSH3) shares the same address with LCOMP3. When read, this register means the value of LPOSH3, which is the lower 16 bits of POSH3. Position counter is captured into POSH3 on the rising edge of ICAP[3].

**Parameters**

- base – EQDC peripheral base address.

**Returns**

Hold position counter3's value.

```
static inline uint16_t EQDC_GetPositionDifference(EQDC_Type *base)
```

Get the position difference counter's value.

**Parameters**

- base – EQDC peripheral base address.

**Returns**

The position difference counter's value.

```
static inline uint16_t EQDC_GetHoldPositionDifference(EQDC_Type *base)
```

Get the hold position difference counter's value.

The position difference (POSD) value is loaded to hold position difference (POSDH) when:

- a. Position register (POS or UPOS, LPOS), or position difference register (POSD), or revolution register (REV) is read. When Period Measurement is enabled (CTRL3[PMEN] = 1), POSDH will only be updated when reading POSD.
- b. TRIGGER happens and TRIGGER is enabled to update the hold registers.

**Parameters**

- base – EQDC peripheral base address.

**Returns**

Hold position difference counter's value.

```
static inline uint16_t EQDC_GetRevolution(EQDC_Type *base)
```

Get the revolution counter's value.

Get the revolution counter (REV) value.

**Parameters**

- base – EQDC peripheral base address.

**Returns**

The revolution counter's value.

```
static inline uint16_t EQDC_GetHoldRevolution(EQDC_Type *base)
```

Get the hold revolution counter's value.

The revolution counter (REV) value is loaded to hold revolution (REXH) when:

- a. Position register (POS or UPOS, LPOS), or position difference register (POSD), or revolution register (REV) is read.
- b. TRIGGER happens and TRIGGER is enabled to update the hold registers.

**Parameters**

- base – EQDC peripheral base address.

**Returns**

Hold position revolution counter's value.

```
static inline uint16_t EQDC_GetLastEdgeTime(EQDC_Type *base)
```

Get the last edge time.

Last edge time (LASTEDGE) is the time since the last edge occurred on PHASEA or PHASEB. The last edge time register counts up using the peripheral clock after prescaler. Any edge on PHASEA or PHASEB will reset this register to 0 and start counting. If the last edge timer count reaches 0xffff, the counting will stop in order to prevent an overflow. Counting will continue when an edge occurs on PHASEA or PHASEB.

**Parameters**

- base – EQDC peripheral base address.

**Returns**

The last edge time.

```
static inline uint16_t EQDC_GetHoldLastEdgeTime(EQDC_Type *base)
```

Get the hold last edge time.

The hold of last edge time (LASTEDGEH) is update to last edge time (LASTEDGE) when the position difference register (POSD) is read.

**Parameters**

- base – EQDC peripheral base address.

**Returns**

Hold of last edge time.

```
static inline uint16_t EQDC_GetPositionDifferencePeriod(EQDC_Type *base)
```

Get the Position Difference Period counter value.

The Position Difference Period counter (POSDPER) counts up using the prescaled peripheral clock. When reading the position difference register(POSD), the last edge time (LAST-EDGE) will be loaded to position difference period counter(POSDPER). If the POSDPER count reaches 0xffff, the counting will stop in order to prevent an overflow. Counting will continue when an edge occurs on PHASEA or PHASEB.

**Parameters**

- base – EQDC peripheral base address.

**Returns**

The position difference period counter value.

```
static inline uint16_t EQDC_GetBufferedPositionDifferencePeriod(EQDC_Type *base)
```

Get buffered Position Difference Period counter value.

The Bufferd Position Difference Period (POSDPERBFR) value is updated with the position difference period counter(POSDPER) when any edge occurs on PHASEA or PHASEB.

**Parameters**

- base – EQDC peripheral base address.

**Returns**

The buffered position difference period counter value.

```
static inline uint16_t EQDC_GetHoldPositionDifferencePeriod(EQDC_Type *base)
```

Get Hold Position Difference Period counter value.

The hold position difference period(POSDPERH) is updated with the value of buffered position difference period(POSDPERBFR) when the position difference(POSD) register is read.

**Parameters**

- base – EQDC peripheral base address.

**Returns**

The hold position difference period counter value.

```
enum _eqdc_status_flags
```

EQDC status flags, these flags indicate the counter's events. .

*Values:*

```
enumerator kEQDC_HomeEnableTransitionFlag
```

HOME/ENABLE signal transition occurred.

```
enumerator kEQDC_IndexPresetPulseFlag
```

INDEX/PRESET pulse occurred.

```
enumerator kEQDC_WatchdogTimeoutFlag
```

Watchdog timeout occurred.

```
enumerator kEQDC_SimultPhaseChangeFlag
```

Simultaneous change of PHASEA and PHASEB occurred.

```
enumerator kEQDC_CountDirectionChangeFlag
```

Count direction change interrupt enable.

```
enumerator kEQDC_PositionRollOverFlag
```

Position counter rolls over from 0xFFFFFFFF to 0, or from MOD value to INIT value.

enumerator kEQDC\_PositionRollUnderFlag

Position register roll under from 0 to 0xFFFFFFFF, or from INIT value to MOD value.

enumerator kEQDC\_PositionCompare0Flag

Position counter match the COMP0 value.

enumerator kEQDC\_PositionCompare1Flag

Position counter match the COMP1 value.

enumerator kEQDC\_PositionCompare2Flag

Position counter match the COMP2 value.

enumerator kEQDC\_PositionCompare3Flag

Position counter match the COMP3 value.

enumerator kEQDC\_StatusAllFlags

enum \_eqdc\_signal\_status

Signal status, these flags indicate the raw and filtered input signal status. .

*Values:*

enumerator kEQDC\_SignalStatusRawHomeEnable

Raw HOME/ENABLE input.

enumerator kEQDC\_SignalStatusRawIndexPreset

Raw INDEX/PRESET input.

enumerator kEQDC\_SignalStatusRawPhaseB

Raw PHASEB input.

enumerator kEQDC\_SignalStatusRawPhaseA

Raw PHASEA input.

enumerator kEQDC\_SignalStatusFilteredHomeEnable

The filtered HOME/ENABLE input.

enumerator kEQDC\_SignalStatusFilteredIndexPreset

The filtered INDEX/PRESET input.

enumerator kEQDC\_SignalStatusFilteredPhaseB

The filtered PHASEB input.

enumerator kEQDC\_SignalStatusFilteredPhaseA

The filtered PHASEA input.

enumerator kEQDC\_SignalStatusPositionCompare0Flag

Position Compare 0 Flag Output.

enumerator kEQDC\_SignalStatusPositionCompare1Flag

Position Compare 1 Flag Output.

enumerator kEQDC\_SignalStatusPositionCompare2Flag

Position Compare 2 Flag Output.

enumerator kEQDC\_SignalStatusPositionCompare3Flag

Position Compare 3 Flag Output.

enumerator kEQDC\_SignalStatusCountDirectionFlagHold

Count Direction Flag Hold.

enumerator kEQDC\_SignalStatusCountDirectionFlag

Count Direction Flag Output.

enumerator kEQDC\_SignalStatusAllFlags

enum \_eqdc\_interrupt\_enable

Interrupt enable/disable mask. .

*Values:*

enumerator kEQDC\_HomeEnableTransitionInterruptEnable

HOME/ENABLE signal transition interrupt enable.

enumerator kEQDC\_IndexPresetPulseInterruptEnable

INDEX/PRESET pulse interrupt enable.

enumerator kEQDC\_WatchdogTimeoutInterruptEnable

Watchdog timeout interrupt enable.

enumerator kEQDC\_SimultPhaseChangeInterruptEnable

Simultaneous PHASEA and PHASEB change interrupt enable.

enumerator kEQDC\_CountDirectionChangeInterruptEnable

Count direction change interrupt enable.

enumerator kEQDC\_PositionRollOverInterruptEnable

Roll-over interrupt enable.

enumerator kEQDC\_PositionRollUnderInterruptEnable

Roll-under interrupt enable.

enumerator kEQDC\_PositionCompare0InterruptEnable

Position compare 0 interrupt enable.

enumerator kEQDC\_PositionCompare1InterruptEnable

Position compare 1 interrupt enable.

enumerator kEQDC\_PositionCompare2InterruptEnable

Position compare 2 interrupt enable.

enumerator kEQDC\_PositionCompare3InterruptEnable

Position compare 3 interrupt enable.

enumerator kEQDC\_AllInterruptEnable

enum \_eqdc\_home\_enable\_init\_pos\_counter\_mode

Define HOME/ENABLE signal's trigger mode.

*Values:*

enumerator kEQDC\_HomeInitPosCounterDisabled

Don't use HOME/ENABLE signal to initialize the position counter.

enumerator kEQDC\_HomeInitPosCounterOnRisingEdge

Use positive going edge to trigger initialization of position counters.

enumerator kEQDC\_HomeInitPosCounterOnFallingEdge

Use negative going edge to trigger initialization of position counters.

enum \_eqdc\_index\_preset\_init\_pos\_counter\_mode

Define INDEX/PRESET signal's trigger mode.

*Values:*

enumerator kEQDC\_IndexInitPosCounterDisabled

INDEX/PRESET pulse does not initialize the position counter.

enumerator kEQDC\_IndexInitPosCounterOnRisingEdge

Use INDEX/PRESET pulse rising edge to initialize position counter.

enumerator kEQDC\_IndexInitPosCounterOnFallingEdge

Use INDEX/PRESET pulse falling edge to initialize position counter.

enum \_eqdc\_operate\_mode

Define type for decoder operation mode.

The Quadrature Decoder operates in following 4 operation modes: 1.Quadrature Decode(QDC) Operation Mode (CTRL[PH1] = 0,CTRL2[OPMODE] = 0) In QDC operation mode, Module uses PHASEA, PHASEB, INDEX, HOME, TRIGGER and ICAP[3:1] to decode the PHASEA and PHASEB signals from Speed/Position sensor. 2.Quadrature Count(QCT) Operation Mode (CTRL[PH1] = 0,CTRL2[OPMODE] = 1) In QCT operation mode, Module uses PHASEA, PHASEB, PRESET, ENABLE, TRIGGER and ICAP[3:1] to count the PHASEA and PHASEB signals from Speed/Position sensor. 3.Single Phase Decode(PH1DC) Operation Mode (CTRL[PH1] = 1,CTRL2[OPMODE] = 0) In PH1DC operation mode, the module uses PHASEA, PHASEB, INDEX, HOME, TRIGGER and ICAP[3:1] to decode the PHASEA and PHASEB signals from Speed/Position sensor. 4.Single Phase Count(PH1CT) Operation Mode (CTRL[PH1] = 1,CTRL2[OPMODE] = 1) In PH1CT operation mode, the module uses PHASEA, PHASEB, PRESET, ENABLE, TRIGGER and ICAP[3:1] to count the PHASEA and PHASEB signals from Speed/Position sensor.

*Values:*

enumerator kEQDC\_QuadratureDecodeOperationMode

Use standard quadrature decoder with PHASEA/PHASEB, INDEX/HOME.

enumerator kEQDC\_QuadratureCountOperationMode

Use quadrature count operation mode with PHASEA/PHASEB, PRESET/ENABLE.

enumerator kEQDC\_SinglePhaseDecodeOperationMode

Use single phase quadrature decoder with PHASEA/PHASEB, INDEX/HOME.

enumerator kEQDC\_SinglePhaseCountOperationMode

Use single phase count decoder with PHASEA/PHASEB, PRESET/ENABLE.

enum \_eqdc\_count\_mode

Define type for decoder count mode.

In decode mode, it uses the standard quadrature decoder with PHASEA and PHASEB, PHASEA = 0 and PHASEB = 0 mean reverse direction.

- If PHASEA leads PHASEB, then motion is in the positive direction.
- If PHASEA trails PHASEB, then motion is in the negative direction. In single phase mode, there are three count modes:
- In Signed Count mode (Single Edge). Both position counter (POS) and position difference counter (POSD) count on the input PHASEA rising edge while the input PHASEB provides the selected position counter direction (up/down). If CTRL[REV] is 1, then the position counter will count in the opposite direction.
- In Signed Count mode (double edge), both position counter (POS) and position difference counter (POSD) count the input PHASEA on both rising edge and falling edge while the input PHASEB provides the selected position counter direction (up/down).
- In UP/DOWN Pulse Count mode. Both position counter (POS) and position difference counter (POSD) count in the up direction when input PHASEA rising edge occurs. Both counters count in the down direction when input PHASEB rising edge occurs. If CTRL[REV] is 1, then the position counter will count in the opposite direction.

*Values:*

enumerator kEQDC\_QuadratureX4

Active on kEQDC\_QuadratureDecodeOperationMode/kEQDC\_QuadratureCountOperationMode.

enumerator kEQDC\_QuadratureX2

Active on kEQDC\_QuadratureDecodeOperationMode/kEQDC\_QuadratureCountOperationMode.

enumerator kEQDC\_QuadratureX1

Active on kEQDC\_QuadratureDecodeOperationMode/kEQDC\_QuadratureCountOperationMode.

enumerator kEQDC\_UpDownPulseCount

Active on kEQDC\_SinglePhaseDecodeOperationMode/kEQDC\_SinglePhaseCountOperationMode.

enumerator kEQDC\_SignedCountDoubleEdge

Active on kEQDC\_SinglePhaseDecodeOperationMode/kEQDC\_SinglePhaseCountOperationMode.

enumerator kEQDC\_SignedCountSingleEdge

Active on kEQDC\_SinglePhaseDecodeOperationMode/kEQDC\_SinglePhaseCountOperationMode.

enum \_eqdc\_output\_pulse\_mode

Define type for the condition of POSMATCH pulses.

*Values:*

enumerator kEQDC\_OutputPulseOnCounterEqualCompare

POSMATCH pulses when a match occurs between the position counters (POS) and the compare value (UCOMPx/LCOMPx)(x range is 0-3).

enumerator kEQDC\_OutputPulseOnReadingPositionCounter

POSMATCH pulses when reading position counter(POS and LPOS), revolution counter(REV), position difference counter(POSD).

enum \_eqdc\_revolution\_count\_condition

Define type for determining how the revolution counter (REV) is incremented/decremented.

*Values:*

enumerator kEQDC\_RevolutionCountOnIndexPulse

Use INDEX pulse to increment/decrement revolution counter.

enumerator kEQDC\_RevolutionCountOnRollOverModulus

Use modulus counting roll-over/under to increment/decrement revolution counter.

enum \_eqdc\_filter\_sample\_count

Input Filter Sample Count.

The Input Filter Sample Count represents the number of consecutive samples that must agree, before the input filter accepts an input transition

*Values:*

enumerator kEQDC\_Filter3Samples

3 samples.

enumerator kEQDC\_Filter4Samples

4 samples.

enumerator kEQDC\_Filter5Samples

5 samples.

enumerator kEQDC\_Filter6Samples

6 samples.



enumerator kEQDC\_Filter7Samples  
7 samples.

enumerator kEQDC\_Filter8Samples  
8 samples.

enumerator kEQDC\_Filter9Samples  
9 samples.

enumerator kEQDC\_Filter10Samples  
10 samples.

enum \_eqdc\_count\_direction\_flag  
Count direction.

*Values:*

enumerator kEQDC\_CountDirectionDown  
Last count was in down direction.

enumerator kEQDC\_CountDirectionUp  
Last count was in up direction.

enum \_eqdc\_prescaler  
Prescaler used by Last Edge Time (LASTEDGE) and Position Difference Period Counter (POS-DPER).

*Values:*

enumerator kEQDC\_Prescaler1  
Prescaler value 1.

enumerator kEQDC\_Prescaler2  
Prescaler value 2.

enumerator kEQDC\_Prescaler4  
Prescaler value 4.

enumerator kEQDC\_Prescaler8  
Prescaler value 8.

enumerator kEQDC\_Prescaler16  
Prescaler value 16.

enumerator kEQDC\_Prescaler32  
Prescaler value 32.

enumerator kEQDC\_Prescaler64  
Prescaler value 64.

enumerator kEQDC\_Prescaler128  
Prescaler value 128.

enumerator kEQDC\_Prescaler256  
Prescaler value 256.

enumerator kEQDC\_Prescaler512  
Prescaler value 512.

enumerator kEQDC\_Prescaler1024  
Prescaler value 1024.

enumerator kEQDC\_Prescaler2048  
Prescaler value 2048.

enumerator kEQDC\_Prescaler4096

Prescaler value 4096.

enumerator kEQDC\_Prescaler8192

Prescaler value 8192.

enumerator kEQDC\_Prescaler16384

Prescaler value 16384.

enumerator kEQDC\_Prescaler32768

Prescaler value 32768.

typedef enum *\_eqdc\_home\_enable\_init\_pos\_counter\_mode*

*eqdc\_home\_enable\_init\_pos\_counter\_mode\_t*

Define HOME/ENABLE signal's trigger mode.

typedef enum *\_eqdc\_index\_preset\_init\_pos\_counter\_mode*

*eqdc\_index\_preset\_init\_pos\_counter\_mode\_t*

Define INDEX/PRESET signal's trigger mode.

typedef enum *\_eqdc\_operate\_mode* *eqdc\_operate\_mode\_t*

Define type for decoder operation mode.

The Quadrature Decoder operates in following 4 operation modes: 1.Quadrature Decode(QDC) Operation Mode (CTRL[PH1] = 0,CTRL2[OPMODE] = 0) In QDC operation mode, Module uses PHASEA, PHASEB, INDEX, HOME, TRIGGER and ICAP[3:1] to decode the PHASEA and PHASEB signals from Speed/Position sensor. 2.Quadrature Count(QCT) Operation Mode (CTRL[PH1] = 0,CTRL2[OPMODE] = 1) In QCT operation mode, Module uses PHASEA, PHASEB, PRESET, ENABLE, TRIGGER and ICAP[3:1] to count the PHASEA and PHASEB signals from Speed/Position sensor. 3.Single Phase Decode(PH1DC) Operation Mode (CTRL[PH1] = 1,CTRL2[OPMODE] = 0) In PH1DC operation mode, the module uses PHASEA, PHASEB, INDEX, HOME, TRIGGER and ICAP[3:1] to decode the PHASEA and PHASEB signals from Speed/Position sensor. 4.Single Phase Count(PH1CT) Operation Mode (CTRL[PH1] = 1,CTRL2[OPMODE] = 1) In PH1CT operation mode, the module uses PHASEA, PHASEB, PRESET, ENABLE, TRIGGER and ICAP[3:1] to count the PHASEA and PHASEB signals from Speed/Position sensor.

typedef enum *\_eqdc\_count\_mode* *eqdc\_count\_mode\_t*

Define type for decoder count mode.

In decode mode, it uses the standard quadrature decoder with PHASEA and PHASEB, PHASEA = 0 and PHASEB = 0 mean reverse direction.

- If PHASEA leads PHASEB, then motion is in the positive direction.
- If PHASEA trails PHASEB, then motion is in the negative direction. In single phase mode, there are three count modes:
- In Signed Count mode (Single Edge). Both position counter (POS) and position difference counter (POSD) count on the input PHASEA rising edge while the input PHASEB provides the selected position counter direction (up/down). If CTRL[REV] is 1, then the position counter will count in the opposite direction.
- In Signed Count mode (double edge), both position counter (POS) and position difference counter (POSD) count the input PHASEA on both rising edge and falling edge while the input PHASEB provides the selected position counter direction (up/down).
- In UP/DOWN Pulse Count mode. Both position counter (POS) and position difference counter (POSD) count in the up direction when input PHASEA rising edge occurs. Both counters count in the down direction when input PHASEB rising edge occurs. If CTRL[REV] is 1, then the position counter will count in the opposite direction.

`typedef enum _eqdc_output_pulse_mode eqdc_output_pulse_mode_t`

Define type for the condition of POSMATCH pulses.

`typedef enum _eqdc_revolution_count_condition eqdc_revolution_count_condition_t`

Define type for determining how the revolution counter (REV) is incremented/decremented.

`typedef enum _eqdc_filter_sample_count eqdc_filter_sample_count_t`

Input Filter Sample Count.

The Input Filter Sample Count represents the number of consecutive samples that must agree, before the input filter accepts an input transition

`typedef enum _eqdc_count_direction_flag eqdc_count_direction_flag_t`

Count direction.

`typedef enum _eqdc_prescaler eqdc_prescaler_t`

Prescaler used by Last Edge Time (LASTEDGE) and Position Difference Period Counter (POS-DPER).

`typedef struct _eqdc_config eqdc_config_t`

Define user configuration structure for EQDC module.

`FSL_EQDC_DRIVER_VERSION`

`EQDC_CTRL_W1C_FLAGS`

W1C bits in EQDC CTRL registers.

`EQDC_INTCTRL_W1C_FLAGS`

W1C bits in EQDC INTCTRL registers.

`EQDC_CTRL_INT_EN`

Interrupt enable bits in EQDC CTRL registers.

`EQDC_INTCTRL_INT_EN`

Interrupt enable bits in EQDC INTCTRL registers.

`EQDC_CTRL_INT_FLAGS`

Interrupt flag bits in EQDC CTRL registers.

`EQDC_INTCTRL_INT_FLAGS`

Interrupt flag bits in EQDC INTCTRL registers.

`kEQDC_PositionCompare0InterruptEnable`

`kEQDC_PositionCompare1InterruptEnable`

`kEQDC_PositionCompare2InterruptEnable`

`kEQDC_PositionCompare3InterruptEnable`

`struct _eqdc_config`

`#include <fsl_eqdc.h>` Define user configuration structure for EQDC module.

### Public Members

`bool enableReverseDirection`

Enable reverse direction counting.

`bool countOnce`

Selects modulo loop or one shot counting mode.

**bool** enableDma

Enable DMA for new written buffer values of COMPx/INIT/MOD(x range is 0-3)

**bool** bufferedRegisterLoadMode

selects the loading time point of the buffered compare registers UCOMPx/LCOMPx, x=0~3, initial register (UINIT/LINIT), and modulus register (UMOD/LMOD).

**bool** enableTriggerInitPositionCounter

Initialize position counter with initial register(UINIT, LINIT) value on TRIGGER's rising edge.

**bool** enableIndexInitPositionCounter

Enables the feature that the position counter to be initialized by Index Event Edge Mark.

This option works together with `_eqdc_index_preset_init_pos_counter_mode` and `enableReverseDirection`; If enabled, the behavior is like this:

When PHA leads PHB (Clockwise): If `_eqdc_index_preset_init_pos_counter_mode` is `kEQDC_IndexInitPosCounterOnRisingEdge`, then INDEX rising edge reset position counter. If `_eqdc_index_preset_init_pos_counter_mode` is `kEQDC_IndexInitPosCounterOnFallingEdge`, then INDEX falling edge reset position counter. If `enableReverseDirection` is false, then Reset position counter to initial value. If `enableReverseDirection` is true, then reset position counter to modulus value.

When PHA lags PHB (Counter Clockwise): If `_eqdc_index_preset_init_pos_counter_mode` is `kEQDC_IndexInitPosCounterOnRisingEdge`, then INDEX falling edge reset position counter. If `_eqdc_index_preset_init_pos_counter_mode` is `kEQDC_IndexInitPosCounterOnFallingEdge`, then INDEX rising edge reset position counter. If `enableReverseDirection` is false, then Reset position counter to modulus value. If `enableReverseDirection` is true, then reset position counter to initial value.

**bool** enableTriggerClearPositionRegisters

Clear position counter(POS), revolution counter(REV), position difference counter(POSD) on TRIGGER's rising edge.

**bool** enableTriggerHoldPositionRegisters

Load position counter(POS), revolution counter(REV), position difference counter(POSD) values to hold registers on TRIGGER's rising edge.

**bool** filterPhaseA

Filter operation on PHASEA input, when write 1, it means filter for PHASEA input is bypassed.

**bool** filterPhaseB

Filter operation on PHASEB input, when write 1, it means filter for PHASEB input is bypassed.

**bool** filterIndPre

Filter operation on INDEX/PRESET input, when write 1, it means filter for INDEX/PRESET input is bypassed.

**bool** filterHomEna

Filter operation on HOME/ENABLE input, when write 1, it means filter for HOME/ENABLE input is bypassed.

**bool** enableWatchdog

Enable the watchdog to detect if the target is moving or not.

**uint16\_t** watchdogTimeoutValue

Watchdog timeout count value. It stores the timeout count for the quadrature decoder module watchdog timer.

*eqdc\_prescaler\_t* prescaler

Prescaler.

*bool filterClockSourceselection*

Filter Clock Source selection.

*eqdc\_filter\_sample\_count\_t* filterSampleCount

Input Filter Sample Count. This value should be chosen to reduce the probability of noisy samples causing an incorrect transition to be recognized. The value represent the number of consecutive samples that must agree prior to the input filter accepting an input transition.

*uint8\_t filterSamplePeriod*

Input Filter Sample Period. This value should be set such that the sampling period is larger than the period of the expected noise. This value represents the sampling period (in IPBus clock cycles) of the decoder input signals. The available range is 0 - 255.

*eqdc\_operate\_mode\_t* operateMode

Selects operation mode.

*eqdc\_count\_mode\_t* countMode

Selects count mode.

*eqdc\_home\_enable\_init\_pos\_counter\_mode\_t* homeEnableInitPosCounterMode

Select how HOME/Enable signal used to initialize position counters.

*eqdc\_index\_preset\_init\_pos\_counter\_mode\_t* indexPresetInitPosCounterMode

Select how INDEX/Preset signal used to initialize position counters.

*eqdc\_output\_pulse\_mode\_t* outputPulseMode

The condition of POSMATCH pulses.

*uint32\_t positionCompareValue[4]*

Position compare 0 ~ 3 value. The available value is a 32-bit number.

*eqdc\_revolution\_count\_condition\_t* revolutionCountCondition

Revolution Counter Modulus Enable.

*uint32\_t positionModulusValue*

Position modulus value. The available value is a 32-bit number.

*uint32\_t positionInitialValue*

Position initial value. The available value is a 32-bit number.

*uint32\_t positionCounterValue*

Position counter value. When Modulo mode enabled, the positionCounterValue should be in the range of positionInitialValue and positionModulusValue.

*bool enablePeriodMeasurement*

Enable period measurement. When enabled, the position difference hold register (POSDH) is only updated when position difference register (POSD) is read.

*uint16\_t enabledInterruptsMask*

Mask of interrupts to be enabled, should be OR'ed value of *\_eqdc\_interrupt\_enable*.

## 2.11 ERM: error recording module

void ERM\_Init(ERM\_Type \*base)

ERM module initialization function.

**Parameters**

- base – ERM base address.

void ERM\_Deinit(ERM\_Type \*base)

De-initializes the ERM.

static inline void ERM\_EnableInterrupts(ERM\_Type \*base, uint32\_t channel, uint32\_t mask)

ERM enable interrupts.

**Parameters**

- base – ERM peripheral base address.
- channel – memory channel.
- mask – single correction interrupt or non-correction interrupt enable to disable for one specific memory region. Refer to “\_erm\_interrupt\_enable” enumeration.

static inline void ERM\_DisableInterrupts(ERM\_Type \*base, uint32\_t channel, uint32\_t mask)

ERM module disable interrupts.

**Parameters**

- base – ERM base address.
- channel – memory channel.
- mask – single correction interrupt or non-correction interrupt enable to disable for one specific memory region. Refer to “\_erm\_interrupt\_enable” enumeration.

static inline uint32\_t ERM\_GetInterruptStatus(ERM\_Type \*base, uint32\_t channel)

Gets ERM interrupt flags.

**Parameters**

- base – ERM peripheral base address.

**Returns**

ERM event flags.

static inline void ERM\_ClearInterruptStatus(ERM\_Type \*base, uint32\_t channel, uint32\_t mask)

ERM module clear interrupt status flag.

**Parameters**

- base – ERM base address.
- mask – event flag to clear. Refer to “\_erm\_interrupt\_flag” enumeration.

uint32\_t ERM\_GetMemoryErrorAddr(ERM\_Type \*base, uint32\_t channel)

ERM get memory error absolute address, which capturing the address of the last ECC event in Memory n.

**Parameters**

- base – ERM base address.
- channel – memory channel.

**Return values**

memory – error absolute address.

uint32\_t ERM\_GetSyndrome(ERM\_Type \*base, uint32\_t channel)

ERM get syndrome, which identifies the pertinent bit position on a correctable, single-bit data inversion or a non-correctable, single-bit address inversion. The syndrome value does not provide any additional diagnostic information on non-correctable, multi-bit inversions.

**Parameters**

- base – ERM base address.
- channel – memory channel.

**Return values**

syndrome – value.

uint32\_t ERM\_GetErrorCount(ERM\_Type \*base, uint32\_t channel)

ERM get error count, which records the count value of the number of correctable ECC error events for Memory n. Non-correctable errors are considered a serious fault, so the ERM does not provide any mechanism to count non-correctable errors. Only correctable errors are counted.

**Parameters**

- base – ERM base address.
- channel – memory channel.

**Return values**

error – count.

void ERM\_ResetErrorCount(ERM\_Type \*base, uint32\_t channel)

ERM reset error count.

**Parameters**

- base – ERM base address.
- channel – memory channel.

FSL\_ERM\_DRIVER\_VERSION

Driver version.

ERM interrupt configuration structure, default settings all disabled, `_erm_interrupt_enable`.

This structure contains the settings for all of the ERM interrupt configurations.

*Values:*

enumerator `kERM_SingleCorrectionIntEnable`

Single Correction Interrupt Notification enable.

enumerator `kERM_NonCorrectableIntEnable`

Non-Correction Interrupt Notification enable.

enumerator `kERM_AllInterruptsEnable`

All Interrupts enable

ERM interrupt status, `_erm_interrupt_flag`.

This provides constants for the ERM event status for use in the ERM functions.

*Values:*

enumerator `kERM_SingleBitCorrectionIntFlag`

Single-Bit Correction Event.

enumerator kERM\_NonCorrectableErrorIntFlag  
Non-Correctable Error Event.

enumerator kERM\_AllIntsFlag  
All Events.

## 2.12 FGPIO Driver

## 2.13 FREQME: Frequency Measurement

## 2.14 GLIKEY

## 2.15 GLIKEY

*Values:*

enumerator kStatus\_GLIKEY\_LockedError  
GLIKEY status for locked SFR registers (unexpected) .

enumerator kStatus\_GLIKEY\_NotLocked  
GLIKEY status for unlocked SFR registers.

enumerator kStatus\_GLIKEY\_Locked  
GLIKEY status for locked SFR registers.

enumerator kStatus\_GLIKEY\_DisabledError  
GLIKEY status for disabled error.

FSL\_GLIKEY\_DRIVER\_VERSION  
Defines GLIKEY driver version 2.0.1.

Change log:

- Version 2.0.1
  - Implement INIT state recovery from the LOCKED state after a reset when the previous index was locked.
- Version 2.0.0
  - Initial version

GLIKEY\_CODEWORD\_STEP1

GLIKEY\_CODEWORD\_STEP2

GLIKEY\_CODEWORD\_STEP3

GLIKEY\_CODEWORD\_STEP4

GLIKEY\_CODEWORD\_STEP5

GLIKEY\_CODEWORD\_STEP6

GLIKEY\_CODEWORD\_STEP7

GLIKEY\_CODEWORD\_STEP\_EN



GLIKEY\_FSM\_WR\_DIS

GLIKEY\_FSM\_INIT

GLIKEY\_FSM\_STEP1

GLIKEY\_FSM\_STEP2

GLIKEY\_FSM\_STEP3

GLIKEY\_FSM\_STEP4

GLIKEY\_FSM\_LOCKED

GLIKEY\_FSM\_WR\_EN

GLIKEY\_FSM\_SSR\_RESET

uint32\_t GLIKEY\_GetStatus(GLIKEY\_Type \*base)

Retreives the current status of Glikey.

#### Parameters

- base – **[in]** The base address of the Glikey instance

#### Returns

Glikey status information

*status\_t* GLIKEY\_IsLocked(GLIKEY\_Type \*base)

Get if Glikey is locked.

This operation returns the locking status of Glikey.

#### Return values

- kStatus\_GLIKEY\_Locked – if locked
- kStatus\_GLIKEY\_NotLocked – if unlocked

#### Returns

Status

*status\_t* GLIKEY\_CheckLock(GLIKEY\_Type \*base)

Check if Glikey is locked.

This operation returns the locking status of Glikey.

#### Return values

- kStatus\_GLIKEY\_LockedError – if locked
- kStatus\_GLIKEY\_NotLocked – if unlocked

#### Returns

Status kStatus\_Success if success

*status\_t* GLIKEY\_SyncReset(GLIKEY\_Type \*base)

Perform a synchronous reset of Glikey.

This function performs a synchronounous reset of the Glikey. This results in:

- Glikey will return to the INIT state, unless it is in the LOCK state

#### Parameters

- base – **[in]** The base address of the Glikey instance

#### Returns

Status    kStatus\_Success    if    success    Possible    errors:    kSta-  
tus\_GLIKEY\_LockedError

*status\_t* GLIKEY\_SetIntEnable(GLIKEY\_Type \*base, uint32\_t value)

Set interrupt enable flag of Glikey.

#### Parameters

- base – **[in]** The base address of the Glikey instance
- value – **[in]** Value to set the interrupt enable flag to, see #[TODO: add reference to constants]

#### Returns

Status kStatus\_Success if success Possible errors: kStatus\_GLIKEY\_LockedError

*status\_t* GLIKEY\_GetIntEnable(GLIKEY\_Type \*base, uint32\_t \*value)

Get interrupt enable flag of Glikey.

#### Parameters

- base – **[in]** The base address of the Glikey instance
- value – **[out]** Pointer which will be filled with the interrupt enable status, see #[TODO: add reference to constants]

#### Returns

Status kStatus\_Success if success

*status\_t* GLIKEY\_ClearIntStatus(GLIKEY\_Type \*base)

Clear the interrupt status flag of Glikey.

#### Parameters

- base – **[in]** The base address of the Glikey instance

#### Returns

Status kStatus\_Success if success Possible errors: kStatus\_GLIKEY\_LockedError

*status\_t* GLIKEY\_SetIntStatus(GLIKEY\_Type \*base)

Set the interrupt status flag of Glikey.

#### Parameters

- base – **[in]** The base address of the Glikey instance

#### Returns

Status kStatus\_Success if success Possible errors: kStatus\_GLIKEY\_LockedError

*status\_t* GLIKEY\_Lock(GLIKEY\_Type \*base)

Lock Glikey SFR (Special Function Registers) interface.

This operation locks the Glikey SFR interface if it is not locked yet.

#### Parameters

- base – **[in]** The base address of the Glikey instance

#### Returns

Status kStatus\_Success if success

*status\_t* GLIKEY\_LockIndex(GLIKEY\_Type \*base)

Lock Glikey index.

This operation is used to lock a Glikey index. It can only be executed from the WR\_EN state, executing it from any other state will result in Glikey entering WR\_DIS state. When this happens Glikey requires a reset (synchronous or asynchronous) to go back to INIT state. If the Glikey SFR lock is active this operation will return an error.

**Parameters**

- base – **[in]** The base address of the Glikey instance

**Returns**

Status    kStatus\_Success    if    success    Possible    errors:    kSta-  
tus\_GLIKEY\_LockedError, kStatus\_GLIKEY\_DisabledError

*status\_t* GLIKEY\_IsIndexLocked(GLIKEY\_Type \*base, uint32\_t index)

Check if Glikey index is locked.

This operation returns the locking status of Glikey index.

**Parameters**

- base – **[in]** The base address of the Glikey instance
- index – **[in]** The index of the Glikey instance

**Returns**

kStatus\_GLIKEY\_Locked if locked, kStatus\_GLIKEY\_NotLocked if unlocked  
Possible errors: kStatus\_Fail

*status\_t* GLIKEY\_StartEnable(GLIKEY\_Type \*base, uint32\_t index)

Start Glikey enable.

This operation is used to set a new index and start a the sequence to enable it. It needs to be started from the INIT state. If the new index is already locked Glikey will go to LOCKED state, otherwise it will go to STEP1 state. If this operation is used when Glikey is in any state other than INIT Glikey will go to WR\_DIS state. It can only recover from this state through a reset (synchronous or asynchronous). If the Glikey SFR lock is active this operation will return an error.

**Parameters**

- base – **[in]** The base address of the Glikey instance
- index – **[in]** The index of the Glikey instance

**Returns**

Status    kStatus\_Success    if    success    Possible    errors:    kSta-  
tus\_GLIKEY\_LockedError, kStatus\_Fail

*status\_t* GLIKEY\_ContinueEnable(GLIKEY\_Type \*base, uint32\_t codeword)

Continue Glikey enable.

This operation is used to progress through the different states of the state machine, starting from STEP1 until the state WR\_EN is reached. Each next state of the state machine can only be reached by providing the right codeword to this function. If anything goes wrong the state machine will go to WR\_DIS state and can only recover from it through a reset (synchronous or asynchronous). If the Glikey SFR lock is active this operation will return an error.

**Parameters**

- base – **[in]** The base address of the Glikey instance
- codeword – **[in]** Encoded word for progressing to next FSM state (see GLIKEY\_CODEWORD\_STEPx/EN)

**Returns**

Status    kStatus\_Success    if    success    Possible    errors:    kSta-  
tus\_GLIKEY\_LockedError, kStatus\_Fail, kStatus\_GLIKEY\_DisabledError

*status\_t* GLIKEY\_EndOperation(GLIKEY\_Type \*base)

End Glikey operation.

This operation is used to end a Glikey operation. It can only be executed from the WR\_EN, LOCKED and RESET states. Executing it from any other state will result in Glikey entering WR\_DIS state. When this happens Glikey requires a reset (synchronous or asynchronous) to go back to INIT state. After this operation Glikey will go to INIT state or stay in LOCKED state when the index was locked. If the Glikey SFR lock is active this operation will return an error.

#### Parameters

- base – **[in]** The base address of the Glikey instance

#### Returns

A code-flow protected error code (see nxpCsslFlowProtection)

#### Returns

Status kStatus\_Success if success, kStatus\_GLIKEY\_Locked if index is still locked Possible errors: kStatus\_GLIKEY\_LockedError, kStatus\_GLIKEY\_DisabledError

*status\_t* GLIKEY\_ResetIndex(GLIKEY\_Type \*base, uint32\_t index)

Reset Glikey index.

This operation is used to reset a Glikey index. It can only be executed from the INIT state, executing it from any other state will result in Glikey entering WR\_DIS state. When this happens Glikey requires a reset (synchronous or asynchronous) to go back to INIT state. If the Glikey SFR lock is active or the index is locked this operation will return an error.

#### Returns

A code-flow protected error code (see nxpCsslFlowProtection)

#### Returns

Status kStatus\_Success if success, kStatus\_GLIKEY\_Locked if index is still locked Possible errors: kStatus\_GLIKEY\_LockedError, kStatus\_GLIKEY\_DisabledError

## 2.16 GPIO: General-Purpose Input/Output Driver

FSL\_GPIO\_DRIVER\_VERSION

GPIO driver version.

enum \_gpio\_pin\_direction

GPIO direction definition.

*Values:*

enumerator kGPIO\_DigitalInput

Set current pin as digital input

enumerator kGPIO\_DigitalOutput

Set current pin as digital output

enum \_gpio\_checker\_attribute

GPIO checker attribute.

*Values:*

enumerator kGPIO\_UsernonsecureRWUsersecureRWPrivilegedsecureRW

User nonsecure:Read+Write; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO\_UsernonsecureRUsersecureRWPrivilegedsecureRW

User nonsecure:Read; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO\_\_UsernonsecureNUsersecureRWPrivilegedsecureRW  
User nonsecure:None; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO\_\_UsernonsecureRUsersecureRPrivilegedsecureRW  
User nonsecure:Read; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO\_\_UsernonsecureNUsersecureRPrivilegedsecureRW  
User nonsecure:None; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO\_\_UsernonsecureNUsersecureNPrivilegedsecureRW  
User nonsecure:None; User Secure:None; Privileged Secure:Read+Write

enumerator kGPIO\_\_UsernonsecureNUsersecureNPrivilegedsecureR  
User nonsecure:None; User Secure:None; Privileged Secure:Read

enumerator kGPIO\_\_UsernonsecureNUsersecureNPrivilegedsecureN  
User nonsecure:None; User Secure:None; Privileged Secure:None

enumerator kGPIO\_\_IgnoreAttributeCheck  
Ignores the attribute check

enum \_gpio\_interrupt\_config  
Configures the interrupt generation condition.

*Values:*

enumerator kGPIO\_\_InterruptStatusFlagDisabled  
Interrupt status flag is disabled.

enumerator kGPIO\_\_DMARisingEdge  
ISF flag and DMA request on rising edge.

enumerator kGPIO\_\_DMAFallingEdge  
ISF flag and DMA request on falling edge.

enumerator kGPIO\_\_DMAEitherEdge  
ISF flag and DMA request on either edge.

enumerator kGPIO\_\_FlagRisingEdge  
Flag sets on rising edge.

enumerator kGPIO\_\_FlagFallingEdge  
Flag sets on falling edge.

enumerator kGPIO\_\_FlagEitherEdge  
Flag sets on either edge.

enumerator kGPIO\_\_InterruptLogicZero  
Interrupt when logic zero.

enumerator kGPIO\_\_InterruptRisingEdge  
Interrupt on rising edge.

enumerator kGPIO\_\_InterruptFallingEdge  
Interrupt on falling edge.

enumerator kGPIO\_\_InterruptEitherEdge  
Interrupt on either edge.

enumerator kGPIO\_\_InterruptLogicOne  
Interrupt when logic one.

enumerator kGPIO\_\_ActiveHighTriggerOutputEnable

Enable active high-trigger output.

enumerator kGPIO\_\_ActiveLowTriggerOutputEnable

Enable active low-trigger output.

enum \_\_gpio\_interrupt\_selection

Configures the selection of interrupt/DMA request/trigger output.

*Values:*

enumerator kGPIO\_\_InterruptOutput0

Interrupt/DMA request/trigger output 0.

enumerator kGPIO\_\_InterruptOutput1

Interrupt/DMA request/trigger output 1.

enum gpio\_pin\_interrupt\_control\_t

GPIO pin and interrupt control.

*Values:*

enumerator kGPIO\_\_PinControlNonSecure

Pin Control Non-Secure.

enumerator kGPIO\_\_InterruptControlNonSecure

Interrupt Control Non-Secure.

enumerator kGPIO\_\_PinControlNonPrivilege

Pin Control Non-Privilege.

enumerator kGPIO\_\_InterruptControlNonPrivilege

Interrupt Control Non-Privilege.

typedef enum \_\_gpio\_pin\_direction gpio\_pin\_direction\_t

GPIO direction definition.

typedef enum \_\_gpio\_checker\_attribute gpio\_checker\_attribute\_t

GPIO checker attribute.

typedef struct \_\_gpio\_pin\_config gpio\_pin\_config\_t

The GPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT\_SetPinConfig().

typedef enum \_\_gpio\_interrupt\_config gpio\_interrupt\_config\_t

Configures the interrupt generation condition.

typedef enum \_\_gpio\_interrupt\_selection gpio\_interrupt\_selection\_t

Configures the selection of interrupt/DMA request/trigger output.

typedef struct \_\_gpio\_version\_info gpio\_version\_info\_t

GPIO version information.

GPIO\_FIT\_REG(value)

struct \_\_gpio\_pin\_config

*#include <fsl\_gpio.h>* The GPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT\_SetPinConfig().

### Public Members

*gpio\_pin\_direction\_t* pinDirection

GPIO direction, input or output

uint8\_t outputLogic

Set a default output logic, which has no use in input

struct \_\_gpio\_version\_info

*#include <fsl\_gpio.h>* GPIO version information.

### Public Members

uint16\_t feature

Feature Specification Number.

uint8\_t minor

Minor Version Number.

uint8\_t major

Major Version Number.

## 2.17 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\_PortDeinit(GPIO\_Type \*base)

Denitalizes the GPIO peripheral.

### Parameters

- base – GPIO peripheral base pointer.

void GPIO\_PinInit(GPIO\_Type \*base, uint32\_t pin, const *gpio\_pin\_config\_t* \*config)

Initializes a GPIO pin used by the board.

To initialize the GPIO, define a pin configuration, as either input or output, in the user file. Then, call the GPIO\_PinInit() function.

This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
gpio_pin_config_t config =
{
    kGPIO_DigitalInput,
    0,
}
Define a digital output pin configuration,
gpio_pin_config_t config =
{
    kGPIO_DigitalOutput,
    0,
}
```

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO port pin number
- config – GPIO pin configuration pointer

```
void GPIO_GetVersionInfo(GPIO_Type *base, gpio_version_info_t *info)
```

Get GPIO version information.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- info – GPIO version information

```
static inline void GPIO_SecurePrivilegeLock(GPIO_Type *base, gpio_pin_interrupt_control_t mask)
```

lock or unlock secure privilege.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – pin or interrupt macro

```
static inline void GPIO_EnablePinControlNonSecure(GPIO_Type *base, uint32_t mask)
```

Enable Pin Control Non-Secure.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline void GPIO_DisablePinControlNonSecure(GPIO_Type *base, uint32_t mask)
```

Disable Pin Control Non-Secure.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline void GPIO_EnablePinControlNonPrivilege(GPIO_Type *base, uint32_t mask)
```

Enable Pin Control Non-Privilege.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline void GPIO_DisablePinControlNonPrivilege(GPIO_Type *base, uint32_t mask)
```

Disable Pin Control Non-Privilege.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline void GPIO_EnableInterruptControlNonSecure(GPIO_Type *base, uint32_t mask)
```

Enable Interrupt Control Non-Secure.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro



static inline void GPIO\_DisableInterruptControlNonSecure(GPIO\_Type \*base, uint32\_t mask)  
Disable Interrupt Control Non-Secure.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO\_EnableInterruptControlNonPrivilege(GPIO\_Type \*base, uint32\_t mask)  
Enable Interrupt Control Non-Privilege.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO\_DisableInterruptControlNonPrivilege(GPIO\_Type \*base, uint32\_t mask)  
Disable Interrupt Control Non-Privilege.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO\_PortInputEnable(GPIO\_Type \*base, uint32\_t mask)  
Enable port input.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO\_PortInputDisable(GPIO\_Type \*base, uint32\_t mask)  
Disable port input.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO\_PinWrite(GPIO\_Type \*base, uint32\_t pin, uint8\_t output)  
Sets the output level of the multiple GPIO pins to the logic 1 or 0.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO pin number
- output – GPIO pin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

static inline void GPIO\_PortSet(GPIO\_Type \*base, uint32\_t mask)  
Sets the output level of the multiple GPIO pins to the logic 1.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline void GPIO_PortClear(GPIO_Type *base, uint32_t mask)
```

Sets the output level of the multiple GPIO pins to the logic 0.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline void GPIO_PortToggle(GPIO_Type *base, uint32_t mask)
```

Reverses the current output logic of the multiple GPIO pins.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline uint32_t GPIO_PinRead(GPIO_Type *base, uint32_t pin)
```

Reads the current input value of the GPIO port.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO pin number

**Return values**

GPIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

```
static inline void GPIO_SetPinInterruptConfig(GPIO_Type *base, uint32_t pin,  
                                              gpio_interrupt_config_t config)
```

Configures the gpio pin interrupt/DMA request.

**Parameters**

- base – GPIO peripheral base pointer.
- pin – GPIO pin number.
- config – GPIO pin interrupt configuration.
  - kGPIO\_InterruptStatusFlagDisabled: Interrupt/DMA request disabled.
  - kGPIO\_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).
  - kGPIO\_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit).
  - kGPIO\_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).
  - kGPIO\_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).
  - kGPIO\_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).
  - kGPIO\_FlagEitherEdge : Flag sets on either edge(if the Flag states exit).
  - kGPIO\_InterruptLogicZero : Interrupt when logic zero.
  - kGPIO\_InterruptRisingEdge : Interrupt on rising edge.
  - kGPIO\_InterruptFallingEdge: Interrupt on falling edge.
  - kGPIO\_InterruptEitherEdge : Interrupt on either edge.
  - kGPIO\_InterruptLogicOne : Interrupt when logic one.



`void GPIO_GpioClearInterruptChannelFlags(GPIO_Type *base, uint32_t mask, uint32_t channel)`  
Clears GPIO pin interrupt status flags based on selected interrupt channel(IRQS).

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro
- channel – ‘0’ means selete interrupt channel 0, ‘1’ means selete interrupt channel 1.

`void GPIO_PinClearInterruptFlag(GPIO_Type *base, uint32_t pin)`  
Clear GPIO individual pin’s interrupt status flag.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO specific pin number.

`static inline void GPIO_SetMultipleInterruptPinsConfig(GPIO_Type *base, uint32_t mask, gpio_interrupt_config_t config)`

Sets the GPIO interrupt configuration in PCR register for multiple pins.

**Parameters**

- base – GPIO peripheral base pointer.
- mask – GPIO pin number macro.
- config – GPIO pin interrupt configuration.
  - kGPIO\_InterruptStatusFlagDisabled: Interrupt disabled.
  - kGPIO\_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).
  - kGPIO\_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit).
  - kGPIO\_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).
  - kGPIO\_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).
  - kGPIO\_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).
  - kGPIO\_FlagEitherEdge : Flag sets on either edge(if the Flag states exit).
  - kGPIO\_InterruptLogicZero : Interrupt when logic zero.
  - kGPIO\_InterruptRisingEdge : Interrupt on rising edge.
  - kGPIO\_InterruptFallingEdge: Interrupt on falling edge.
  - kGPIO\_InterruptEitherEdge : Interrupt on either edge.
  - kGPIO\_InterruptLogicOne : Interrupt when logic one.
  - kGPIO\_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit).
  - kGPIO\_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit)..

`void GPIO_CheckAttributeBytes(GPIO_Type *base, gpio_checker_attribute_t attribute)`

brief The GPIO module supports a device-specific number of data ports, organized as 32-bit words/8-bit Bytes. Each 32-bit/8-bit data port includes a GACR register, which defines the byte-level attributes required for a successful access to the GPIO programming model. If

the GPIO module's GACR register organized as 32-bit words, the attribute controls for the 4 data bytes in the GACR follow a standard little endian data convention.

#### Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- attribute – GPIO checker attribute

## 2.18 I3C: I3C Driver

FSL\_I3C\_DRIVER\_VERSION

I3C driver version.

I3C status return codes.

*Values:*

enumerator kStatus\_I3C\_Busy

The master is already performing a transfer.

enumerator kStatus\_I3C\_Idle

The slave driver is idle.

enumerator kStatus\_I3C\_Nak

The slave device sent a NAK in response to an address.

enumerator kStatus\_I3C\_WriteAbort

The slave device sent a NAK in response to a write.

enumerator kStatus\_I3C\_Term

The master terminates slave read.

enumerator kStatus\_I3C\_HdrParityError

Parity error from DDR read.

enumerator kStatus\_I3C\_CrcError

CRC error from DDR read.

enumerator kStatus\_I3C\_ReadFifoError

Read from M/SRDATAB register when FIFO empty.

enumerator kStatus\_I3C\_WriteFifoError

Write to M/SWDATAB register when FIFO full.

enumerator kStatus\_I3C\_MsgError

Message SDR/DDR mismatch or read/write message in wrong state

enumerator kStatus\_I3C\_InvalidReq

Invalid use of request.

enumerator kStatus\_I3C\_Timeout

The module has stalled too long in a frame.

enumerator kStatus\_I3C\_SlaveCountExceed

The I3C slave count has exceed the definition in I3C\_MAX\_DEVCNT.

enumerator kStatus\_I3C\_IBIWon

The I3C slave event IBI or MR or HJ won the arbitration on a header address.

enumerator kStatus\_I3C\_OverrunError

Slave internal from-bus buffer/FIFO overrun.

enumerator kStatus\_I3C\_UnderrunError

Slave internal to-bus buffer/FIFO underrun

enumerator kStatus\_I3C\_UnderrunNak

Slave internal from-bus buffer/FIFO underrun and NACK error

enumerator kStatus\_I3C\_InvalidStart

Slave invalid start flag

enumerator kStatus\_I3C\_SdrParityError

SDR parity error

enumerator kStatus\_I3C\_S0S1Error

S0 or S1 error

enum \_i3c\_hdr\_mode

I3C HDR modes.

*Values:*

enumerator kI3C\_HDRModeNone

enumerator kI3C\_HDRModeDDR

enumerator kI3C\_HDRModeTSP

enumerator kI3C\_HDRModeTSL

typedef enum \_i3c\_hdr\_mode i3c\_hdr\_mode\_t

I3C HDR modes.

typedef struct \_i3c\_device\_info i3c\_device\_info\_t

I3C device information.

I3C\_RETRY\_TIMES

Max loops to wait for I3C operation status complete.

This is the maximum number of loops to wait for I3C operation status complete. If set to 0, it will wait indefinitely.

I3C\_MAX\_DEVCNT

I3C\_IBI\_BUFF\_SIZE

struct \_i3c\_device\_info

*#include <fsl\_i3c.h>* I3C device information.

### Public Members

uint8\_t dynamicAddr

Device dynamic address.

uint8\_t staticAddr

Static address.

uint8\_t dcr

Device characteristics register information.

uint8\_t bcr

Bus characteristics register information.

`uint16_t vendorID`  
Device vendor ID(manufacture ID).

`uint32_t partNumber`  
Device part number info

`uint16_t maxReadLength`  
Maximum read length.

`uint16_t maxWriteLength`  
Maximum write length.

`uint8_t hdrMode`  
Support hdr mode, could be OR logic in `i3c_hdr_mode`.

## 2.19 I3C Common Driver

`typedef struct _i3c_config i3c_config_t`

Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_GetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

`uint32_t I3C_GetInstance(I3C_Type *base)`

Get which instance current I3C is used.

### Parameters

- `base` – The I3C peripheral base address.

`void I3C_GetDefaultConfig(i3c_config_t *config)`

Provides a default configuration for the I3C peripheral, the configuration covers both master functionality and slave functionality.

This function provides the following default configuration for I3C:

```
config->enableMaster          = kI3C_MasterCapable;
config->disableTimeout        = false;
config->hKeep                  = kI3C_MasterHighKeeperNone;
config->enableOpenDrainStop    = true;
config->enableOpenDrainHigh    = true;
config->baudRate_Hz.i2cBaud     = 400000U;
config->baudRate_Hz.i3cPushPullBaud = 12500000U;
config->baudRate_Hz.i3cOpenDrainBaud = 2500000U;
config->masterDynamicAddress    = 0x0AU;
config->slowClock_Hz           = 1000000U;
config->enableSlave            = true;
config->vendorID                = 0x11BU;
config->enableRandomPart        = false;
config->partNumber              = 0;
config->dcr                     = 0;
config->bcr = 0;
config->hdrMode                 = (uint8_t)kI3C_HDRModeDDR;
config->nakAllRequest           = false;
config->ignoreS0S1Error        = false;
config->offline                  = false;
config->matchSlaveStartStop    = false;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the common I3C driver with I3C\_Init().

### Parameters

- `config` – **[out]** User provided configuration structure for default values. Refer to `i3c_config_t`.

`void I3C_Init(I3C_Type *base, const i3c_config_t *config, uint32_t sourceClock_Hz)`

Initializes the I3C peripheral. This function enables the peripheral clock and initializes the I3C peripheral as described by the user provided configuration. This will initialize both the master peripheral and slave peripheral so that I3C module could work as pure master, pure slave or secondary master, etc. A software reset is performed prior to configuration.

### Parameters

- `base` – The I3C peripheral base address.
- `config` – User provided peripheral configuration. Use `I3C_GetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

`struct _i3c_config`

`#include <fsl_i3c.h>` Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_GetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

### Public Members

`i3c_master_enable_t` enableMaster

Enable master mode.

`bool` disableTimeout

Whether to disable timeout to prevent the ERRWARN.

`i3c_master_hkeep_t` hKeep

High keeper mode setting.

`bool` enableOpenDrainStop

Whether to emit open-drain speed STOP.

`bool` enableOpenDrainHigh

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

`i3c_baudrate_hz_t` baudRate\_Hz

Desired baud rate settings.

`i3c_start_scl_delay_t` startSclDelay

I3C SCL delay after START.

`i3c_start_scl_delay_t` restartSclDelay

I3C SCL delay after Repeated START.

`uint8_t` masterDynamicAddress

Main master dynamic address configuration.



`uint32_t maxWriteLength`  
Maximum write length.

`uint32_t maxReadLength`  
Maximum read length.

`bool enableSlave`  
Whether to enable slave.

`uint8_t staticAddr`  
Static address.

`uint16_t vendorID`  
Device vendor ID(manufacture ID).

`uint32_t partNumber`  
Device part number info

`uint8_t dcr`  
Device characteristics register information.

`uint8_t bcr`  
Bus characteristics register information.

`uint8_t hdrMode`  
Support hdr mode, could be OR logic in enumeration:`i3c_hdr_mode_t`.

`bool nakAllRequest`  
Whether to reply NAK to all requests except broadcast CCC.

`bool ignoreS0S1Error`  
Whether to ignore S0/S1 error in SDR mode.

`bool offline`  
Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

`bool matchSlaveStartStop`  
Whether to assert start/stop status only the time slave is addressed.

## 2.20 I3C Master Driver

`void I3C_MasterGetDefaultConfig(i3c_master_config_t *masterConfig)`

Provides a default configuration for the I3C master peripheral.

This function provides the following default configuration for the I3C master peripheral:

```
masterConfig->enableMaster      = kI3C_MasterOn;
masterConfig->disableTimeout    = false;
masterConfig->hKeep              = kI3C_MasterHighKeeperNone;
masterConfig->enableOpenDrainStop = true;
masterConfig->enableOpenDrainHigh = true;
masterConfig->baudRate_Hz        = 100000U;
masterConfig->busType            = kI3C_TypeI2C;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with `I3C_MasterInit()`.

### Parameters

- `masterConfig` – **[out]** User provided configuration structure for default values. Refer to `i3c_master_config_t`.

```
void I3C_MasterInit(I3C_Type *base, const i3c_master_config_t *masterConfig, uint32_t
    sourceClock_Hz)
```

Initializes the I3C master peripheral.

This function enables the peripheral clock and initializes the I3C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

#### Parameters

- `base` – The I3C peripheral base address.
- `masterConfig` – User provided peripheral configuration. Use `I3C_MasterGetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

```
void I3C_MasterDeinit(I3C_Type *base)
```

Deinitializes the I3C master peripheral.

This function disables the I3C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

#### Parameters

- `base` – The I3C peripheral base address.

```
status_t I3C_MasterCheckAndClearError(I3C_Type *base, uint32_t status)
```

```
status_t I3C_MasterWaitForCtrlDone(I3C_Type *base, bool waitIdle)
```

```
status_t I3C_CheckForBusyBus(I3C_Type *base)
```

```
static inline void I3C_MasterEnable(I3C_Type *base, i3c_master_enable_t enable)
```

Set I3C module master mode.

#### Parameters

- `base` – The I3C peripheral base address.
- `enable` – Enable master mode.

```
void I3C_SlaveGetDefaultConfig(i3c_slave_config_t *slaveConfig)
```

Provides a default configuration for the I3C slave peripheral.

This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableslave = true;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with `I3C_SlaveInit()`.

#### Parameters

- `slaveConfig` – **[out]** User provided configuration structure for default values. Refer to `i3c_slave_config_t`.

```
void I3C_SlaveInit(I3C_Type *base, const i3c_slave_config_t *slaveConfig, uint32_t
    slowClock_Hz)
```

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

**Parameters**

- base – The I3C peripheral base address.
- slaveConfig – User provided peripheral configuration. Use I3C\_SlaveGetDefaultConfig() to get a set of defaults that you can override.
- slowClock\_Hz – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If FSL\_FEATURE\_I3C\_HAS\_NO\_SCONFIG\_BAMATCH defines as 1, this parameter is useless.

void I3C\_SlaveDeinit(I3C\_Type \*base)

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

**Parameters**

- base – The I3C peripheral base address.

static inline void I3C\_SlaveEnable(I3C\_Type \*base, bool isEnabled)

Enable/Disable Slave.

**Parameters**

- base – The I3C peripheral base address.
- isEnabled – Enable or disable.

static inline uint32\_t I3C\_MasterGetStatusFlags(I3C\_Type \*base)

Gets the I3C master status flags.

A bit mask with the state of all I3C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

**See also:**

[\\_i3c\\_master\\_flags](#)

**Parameters**

- base – The I3C peripheral base address.

**Returns**

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C\_MasterClearStatusFlags(I3C\_Type \*base, uint32\_t statusMask)

Clears the I3C master status flag state.

The following status register flags can be cleared:

- kI3C\_MasterSlaveStartFlag
- kI3C\_MasterControlDoneFlag
- kI3C\_MasterCompleteFlag
- kI3C\_MasterArbitrationWonFlag
- kI3C\_MasterSlave2MasterFlag

Attempts to clear other flags has no effect.

**See also:**

`_i3c_master_flags`.

**Parameters**

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_master_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_MasterGetStatusFlags()`.

```
static inline uint32_t I3C_MasterGetErrorStatusFlags(I3C_Type *base)
```

Gets the I3C master error status flags.

A bit mask with the state of all I3C master error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

**See also:**

`_i3c_master_error_flags`

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_MasterClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C master error status flag state.

**See also:**

`_i3c_master_error_flags`.

**Parameters**

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_master_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_MasterGetStatusFlags()`.

```
i3c_master_state_t I3C_MasterGetState(I3C_Type *base)
```

Gets the I3C master state.

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

I3C master state.

```
static inline uint32_t I3C_SlaveGetStatusFlags(I3C_Type *base)
```

Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

**See also:**

`_i3c_slave_flags`

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_SlaveClearStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C slave status flag state.

The following status register flags can be cleared:

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`

Attempts to clear other flags has no effect.

**See also:**

`_i3c_slave_flags`.

**Parameters**

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetStatusFlags()`.

```
static inline uint32_t I3C_SlaveGetErrorStatusFlags(I3C_Type *base)
```

Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

**See also:**

`_i3c_slave_error_flags`

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

State of the error status flags:

- 1: related status flag is set.

- 0: related status flag is not set.

static inline void I3C\_SlaveClearErrorStatusFlags(I3C\_Type \*base, uint32\_t statusMask)

Clears the I3C slave error status flag state.

**See also:**

`_i3c_slave_error_flags`.

#### Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_slave_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetErrorStatusFlags()`.

*i3c\_slave\_activity\_state\_t* I3C\_SlaveGetActivityState(I3C\_Type \*base)

Gets the I3C slave state.

#### Parameters

- `base` – The I3C peripheral base address.

#### Returns

I3C slave activity state, refer `i3c_slave_activity_state_t`.

*status\_t* I3C\_SlaveCheckAndClearError(I3C\_Type \*base, uint32\_t status)

static inline void I3C\_MasterEnableInterrupts(I3C\_Type \*base, uint32\_t interruptMask)

Enables the I3C master interrupt requests.

All flags except `kI3C_MasterBetweenFlag` and `kI3C_MasterNackDetectFlag` can be enabled as interrupts.

#### Parameters

- `base` – The I3C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_i3c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline void I3C\_MasterDisableInterrupts(I3C\_Type \*base, uint32\_t interruptMask)

Disables the I3C master interrupt requests.

All flags except `kI3C_MasterBetweenFlag` and `kI3C_MasterNackDetectFlag` can be enabled as interrupts.

#### Parameters

- `base` – The I3C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_i3c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline uint32\_t I3C\_MasterGetEnabledInterrupts(I3C\_Type \*base)

Returns the set of currently enabled I3C master interrupt requests.

#### Parameters

- `base` – The I3C peripheral base address.

#### Returns

A bitmask composed of `_i3c_master_flags` enumerators OR'd together to indicate the set of enabled interrupts.

```
static inline uint32_t I3C_MasterGetPendingInterrupts(I3C_Type *base)
```

Returns the set of pending I3C master interrupt requests.

**Parameters**

- base – The I3C peripheral base address.

**Returns**

A bitmask composed of `_i3c_master_flags` enumerators OR'd together to indicate the set of pending interrupts.

```
static inline void I3C_SlaveEnableInterrupts(I3C_Type *base, uint32_t interruptMask)
```

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`
- `kI3C_SlaveErrorFlag`
- `kI3C_SlaveHDRCommandMatchFlag`
- `kI3C_SlaveCCCHandledFlag`
- `kI3C_SlaveEventSentFlag`

**Parameters**

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline void I3C_SlaveDisableInterrupts(I3C_Type *base, uint32_t interruptMask)
```

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`
- `kI3C_SlaveErrorFlag`
- `kI3C_SlaveHDRCommandMatchFlag`
- `kI3C_SlaveCCCHandledFlag`
- `kI3C_SlaveEventSentFlag`

**Parameters**

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline uint32\_t I3C\_SlaveGetEnabledInterrupts(I3C\_Type \*base)

Returns the set of currently enabled I3C slave interrupt requests.

**Parameters**

- base – The I3C peripheral base address.

**Returns**

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

static inline uint32\_t I3C\_SlaveGetPendingInterrupts(I3C\_Type \*base)

Returns the set of pending I3C slave interrupt requests.

**Parameters**

- base – The I3C peripheral base address.

**Returns**

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of pending interrupts.

static inline void I3C\_MasterEnableDMA(I3C\_Type \*base, bool enableTx, bool enableRx, uint32\_t width)

Enables or disables I3C master DMA requests.

**Parameters**

- base – The I3C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.
- width – DMA read/write unit in bytes.

static inline uint32\_t I3C\_MasterGetTxFifoAddress(I3C\_Type \*base, uint32\_t width)

Gets I3C master transmit data register address for DMA transfer.

**Parameters**

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

**Returns**

The I3C Master Transmit Data Register address.

static inline uint32\_t I3C\_MasterGetRxFifoAddress(I3C\_Type \*base, uint32\_t width)

Gets I3C master receive data register address for DMA transfer.

**Parameters**

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

**Returns**

The I3C Master Receive Data Register address.



```
static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)
```

Enables or disables I3C slave DMA requests.

#### Parameters

- *base* – The I3C peripheral base address.
- *enableTx* – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- *enableRx* – Enable flag for receive DMA request. Pass true for enable, false for disable.
- *width* – DMA read/write unit in bytes.

```
static inline uint32_t I3C_SlaveGetTxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave transmit data register address for DMA transfer.

#### Parameters

- *base* – The I3C peripheral base address.
- *width* – DMA read/write unit in bytes.

#### Returns

The I3C Slave Transmit Data Register address.

```
static inline uint32_t I3C_SlaveGetRxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave receive data register address for DMA transfer.

#### Parameters

- *base* – The I3C peripheral base address.
- *width* – DMA read/write unit in bytes.

#### Returns

The I3C Slave Receive Data Register address.

```
static inline void I3C_MasterSetWatermarks(I3C_Type *base, i3c_tx_trigger_level_t txLvl,
                                           i3c_rx_trigger_level_t rxLvl, bool flushTx, bool flushRx)
```

Sets the watermarks for I3C master FIFOs.

#### Parameters

- *base* – The I3C peripheral base address.
- *txLvl* – Transmit FIFO watermark level. The `kI3C_MasterTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
- *rxLvl* – Receive FIFO watermark level. The `kI3C_MasterRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
- *flushTx* – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- *flushRx* – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

```
static inline void I3C_MasterGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of bytes in the I3C master FIFOs.

#### Parameters

- *base* – The I3C peripheral base address.
- *txCount* – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.

- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

```
static inline void I3C_SlaveSetWatermarks(I3C_Type *base, i3c_tx_trigger_level_t txLvl,  
                                         i3c_rx_trigger_level_t rxLvl, bool flushTx, bool  
                                         flushRx)
```

Sets the watermarks for I3C slave FIFOs.

#### Parameters

- `base` – The I3C peripheral base address.
- `txLvl` – Transmit FIFO watermark level. The `kI3C_SlaveTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches `txLvl`.
- `rxLvl` – Receive FIFO watermark level. The `kI3C_SlaveRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches `rxLvl`.
- `flushTx` – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- `flushRx` – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

```
static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of bytes in the I3C slave FIFOs.

#### Parameters

- `base` – The I3C peripheral base address.
- `txCount` – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

```
void I3C_MasterSetBaudRate(I3C_Type *base, const i3c_baudrate_hz_t *baudRate_Hz, uint32_t  
                           sourceClock_Hz)
```

Sets the I3C bus frequency for master transactions.

The I3C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

#### Parameters

- `base` – The I3C peripheral base address.
- `baudRate_Hz` – Pointer to structure of requested bus frequency in Hertz.
- `sourceClock_Hz` – I3C functional clock frequency in Hertz.

```
static inline bool I3C_MasterGetBusIdleState(I3C_Type *base)
```

Returns whether the bus is idle.

Requires the master mode to be enabled.

#### Parameters

- `base` – The I3C peripheral base address.

#### Return values

- `true` – Bus is busy.
- `false` – Bus is idle.

*status\_t* I3C\_MasterStartWithRxSize(I3C\_Type \*base, *i3c\_bus\_type\_t* type, uint8\_t address, *i3c\_direction\_t* dir, uint8\_t rxSize)

Sends a START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

#### Parameters

- *base* – The I3C peripheral base address.
- *type* – The bus type to use in this transaction.
- *address* – 7-bit slave device address, in bits [6:0].
- *dir* – Master transfer direction, either *kI3C\_Read* or *kI3C\_Write*. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- *rxSize* – Read terminate size for the followed read transfer, limit to 255 bytes.

#### Return values

- *kStatus\_Success* – START signal and address were successfully enqueued in the transmit FIFO.
- *kStatus\_I3C\_Busy* – Another master is currently utilizing the bus.

*status\_t* I3C\_MasterStart(I3C\_Type \*base, *i3c\_bus\_type\_t* type, uint8\_t address, *i3c\_direction\_t* dir)

Sends a START signal and slave address on the I2C/I3C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

#### Parameters

- *base* – The I3C peripheral base address.
- *type* – The bus type to use in this transaction.
- *address* – 7-bit slave device address, in bits [6:0].
- *dir* – Master transfer direction, either *kI3C\_Read* or *kI3C\_Write*. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

#### Return values

- *kStatus\_Success* – START signal and address were successfully enqueued in the transmit FIFO.
- *kStatus\_I3C\_Busy* – Another master is currently utilizing the bus.

*status\_t* I3C\_MasterRepeatedStartWithRxSize(I3C\_Type \*base, *i3c\_bus\_type\_t* type, uint8\_t address, *i3c\_direction\_t* dir, uint8\_t rxSize)

Sends a repeated START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to send a Repeated START signal when a transfer is already in progress. Like *I3C\_MasterStart()*, it also sends the specified 7-bit address. Call this API also configures the read terminate size for the following read transfer. For example, set the *rxSize* = 2, the

following read transfer will be terminated after two bytes of data received. Write transfer will not be affected by the rxSize configuration.

---

**Note:** This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

---

#### Parameters

- `base` – The I3C peripheral base address.
- `type` – The bus type to use in this transaction.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- `rxSize` – Read terminate size for the followed read transfer, limit to 255 bytes.

#### Return values

`kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.

```
static inline status_t I3C_MasterRepeatedStart(I3C_Type *base, i3c_bus_type_t type, uint8_t  
                                              address, i3c_direction_t dir)
```

Sends a repeated START signal and slave address on the I2C/I3C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like `I3C_MasterStart()`, it also sends the specified 7-bit address.

---

**Note:** This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

---

#### Parameters

- `base` – The I3C peripheral base address.
- `type` – The bus type to use in this transaction.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

#### Return values

`kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.

```
status_t I3C_MasterSend(I3C_Type *base, const void *txBuff, size_t txSize, uint32_t flags)
```

Performs a polling send transfer on the I2C/I3C bus.

Sends up to `txSize` number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns `kStatus_I3C_Nak`.

#### Parameters

- `base` – The I3C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.

- `flags` – Bit mask of options for the transfer. See enumeration `_i3c_master_transfer_flags` for available options.

**Return values**

- `kStatus_Success` – Data was sent successfully.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.
- `kStatus_I3C_Nak` – The slave device sent a NAK in response to an address.
- `kStatus_I3C_WriteAbort` – The slave device sent a NAK in response to a write.
- `kStatus_I3C_MsgError` – Message SDR/DDR mismatch or read/write message in wrong state.
- `kStatus_I3C_WriteFifoError` – Write to M/SWDATAB register when FIFO full.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

*status\_t* I3C\_MasterReceive(I3C\_Type \*base, void \*rxBuff, size\_t rxSize, uint32\_t flags)

Performs a polling receive transfer on the I2C/I3C bus.

**Parameters**

- `base` – The I3C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.
- `flags` – Bit mask of options for the transfer. See enumeration `_i3c_master_transfer_flags` for available options.

**Return values**

- `kStatus_Success` – Data was received successfully.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.
- `kStatus_I3C_Term` – The master terminates slave read.
- `kStatus_I3C_HdrParityError` – Parity error from DDR read.
- `kStatus_I3C_CrcError` – CRC error from DDR read.
- `kStatus_I3C_MsgError` – Message SDR/DDR mismatch or read/write message in wrong state.
- `kStatus_I3C_ReadFifoError` – Read from M/SRDATAB register when FIFO empty.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

*status\_t* I3C\_MasterStop(I3C\_Type \*base)

Sends a STOP signal on the I2C/I3C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

**Parameters**

- `base` – The I3C peripheral base address.

**Return values**

- `kStatus_Success` – The STOP signal was successfully sent on the bus and the transaction terminated.

- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

`void I3C_MasterEmitRequest(I3C_Type *base, i3c_bus_request_t masterReq)`  
I3C master emit request.

**Parameters**

- `base` – The I3C peripheral base address.
- `masterReq` – I3C master request of type `i3c_bus_request_t`

`static inline void I3C_MasterEmitIBIResponse(I3C_Type *base, i3c_ibi_response_t ibiResponse)`  
I3C master emit request.

**Parameters**

- `base` – The I3C peripheral base address.
- `ibiResponse` – I3C master emit IBI response of type `i3c_ibi_response_t`

`void I3C_MasterRegisterIBI(I3C_Type *base, i3c_register_ibi_addr_t *ibiRule)`  
I3C master register IBI rule.

**Parameters**

- `base` – The I3C peripheral base address.
- `ibiRule` – Pointer to ibi rule description of type `i3c_register_ibi_addr_t`

`void I3C_MasterGetIBIRules(I3C_Type *base, i3c_register_ibi_addr_t *ibiRule)`  
I3C master get IBI rule.

**Parameters**

- `base` – The I3C peripheral base address.
- `ibiRule` – Pointer to store the read out ibi rule description.

`i3c_ibi_type_t I3C_GetIBIType(I3C_Type *base)`  
I3C master get IBI Type.

**Parameters**

- `base` – The I3C peripheral base address.

**Return values**

`i3c_ibi_type_t` – Type of `i3c_ibi_type_t`.

`static inline uint8_t I3C_GetIBIAddress(I3C_Type *base)`  
I3C master get IBI Address.

**Parameters**

- `base` – The I3C peripheral base address.

**Return values**

The – 8-bit IBI address.

`status_t I3C_MasterProcessDAASpecifiedBaudrate(I3C_Type *base, uint8_t *addressList, uint32_t count, i3c_master_daa_baudrate_t *daaBaudRate)`

Performs a DAA in the i3c bus with specified temporary baud rate.

**Parameters**

- `base` – The I3C peripheral base address.

- `addressList` – The pointer for address list which is used to do DAA.
- `count` – The address count in the address list.
- `daaBaudRate` – The temporary baud rate in DAA process, NULL for using initial setting. The initial setting is set back between the completion of the DAA and the return of this function.

#### Return values

- `kStatus__Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- `kStatus_I3C_SlaveCountExceed` – The I3C slave count has exceed the definition in `I3C_MAX_DEVCNT`.

static inline *status\_t* I3C\_MasterProcessDAA(I3C\_Type \*base, uint8\_t \*addressList, uint32\_t count)

Performs a DAA in the i3c bus.

#### Parameters

- `base` – The I3C peripheral base address.
- `addressList` – The pointer for address list which is used to do DAA.
- `count` – The address count in the address list. The initial setting is set back between the completion of the DAA and the return of this function.

#### Return values

- `kStatus__Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- `kStatus_I3C_SlaveCountExceed` – The I3C slave count has exceed the definition in `I3C_MAX_DEVCNT`.

*i3c\_device\_info\_t* \*I3C\_MasterGetDeviceListAfterDAA(I3C\_Type \*base, uint8\_t \*count)

Get device information list after DAA process is done.

#### Parameters

- `base` – The I3C peripheral base address.
- `count` – **[out]** The pointer to store the available device count.

#### Returns

Pointer to the `i3c_device_info_t` array.

void I3C\_MasterClearDeviceCount(I3C\_Type \*base)

Clear the global device count which represents current devices number on the bus. When user resets all dynamic addresses on the bus, should call this API.

#### Parameters

- `base` – The I3C peripheral base address.

*status\_t* I3C\_MasterTransferBlocking(I3C\_Type \*base, *i3c\_master\_transfer\_t* \*transfer)

Performs a master polling transfer on the I2C/I3C bus.

---

**Note:** The API does not return until the transfer succeeds or fails due to error happens during transfer.

---

#### Parameters

- base – The I3C peripheral base address.
- transfer – Pointer to the transfer structure.

**Return values**

- kStatus\_Success – Data was received successfully.
- kStatus\_I3C\_Busy – Another master is currently utilizing the bus.
- kStatus\_I3C\_IBIWon – The I3C slave event IBI or MR or HJ won the arbitration on a header address.
- kStatus\_I3C\_Timeout – The module has stalled too long in a frame.
- kStatus\_I3C\_Nak – The slave device sent a NAK in response to an address.
- kStatus\_I3C\_WriteAbort – The slave device sent a NAK in response to a write.
- kStatus\_I3C\_Term – The master terminates slave read.
- kStatus\_I3C\_HdrParityError – Parity error from DDR read.
- kStatus\_I3C\_CrcError – CRC error from DDR read.
- kStatus\_I3C\_MsgError – Message SDR/DDR mismatch or read/write message in wrong state.
- kStatus\_I3C\_ReadFifoError – Read from M/SRDATA register when FIFO empty.
- kStatus\_I3C\_WriteFifoError – Write to M/SWDATA register when FIFO full.
- kStatus\_I3C\_InvalidReq – Invalid use of request.

*status\_t* I3C\_SlaveSend(I3C\_Type \*base, const void \*txBuff, size\_t txSize)

Performs a polling send transfer on the I3C bus.

**Parameters**

- base – The I3C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

**Returns**

Error or success status returned by API.

*status\_t* I3C\_SlaveReceive(I3C\_Type \*base, void \*rxBuff, size\_t rxSize)

Performs a polling receive transfer on the I3C bus.

**Parameters**

- base – The I3C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.

**Returns**

Error or success status returned by API.

void I3C\_MasterTransferCreateHandle(I3C\_Type \*base, *i3c\_master\_handle\_t* \*handle, const *i3c\_master\_transfer\_callback\_t* \*callback, void \*userData)

Creates a new handle for the I3C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the I3C\_MasterTransferAbort() API shall be called.



---

**Note:** The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

---

#### Parameters

- base – The I3C peripheral base address.
- handle – **[out]** Pointer to the I3C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

*status\_t* I3C\_MasterTransferNonBlocking(I3C\_Type \*base, *i3c\_master\_handle\_t* \*handle, *i3c\_master\_transfer\_t* \*transfer)

Performs a non-blocking transaction on the I2C/I3C bus.

#### Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- transfer – The pointer to the transfer descriptor.

#### Return values

- kStatus\_Success – The transaction was started successfully.
- kStatus\_I3C\_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

*status\_t* I3C\_MasterTransferGetCount(I3C\_Type \*base, *i3c\_master\_handle\_t* \*handle, *size\_t* \*count)

Returns number of bytes transferred so far.

#### Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

#### Return values

- kStatus\_Success –
- kStatus\_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

*void* I3C\_MasterTransferAbort(I3C\_Type \*base, *i3c\_master\_handle\_t* \*handle)

Terminates a non-blocking I3C master transmission early.

---

**Note:** It is not safe to call this function from an IRQ handler that has a higher priority than the I3C peripheral's IRQ priority.

---

#### Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.

void I3C\_MasterTransferHandleIRQ(I3C\_Type \*base, void \*intHandle)

Reusable routine to handle master interrupts.

---

**Note:** This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

---

#### Parameters

- base – The I3C peripheral base address.
- intHandle – Pointer to the I3C master driver handle.

enum \_i3c\_master\_flags

I3C master peripheral flags.

The following status register flags can be cleared:

- kI3C\_MasterSlaveStartFlag
- kI3C\_MasterControlDoneFlag
- kI3C\_MasterCompleteFlag
- kI3C\_MasterArbitrationWonFlag
- kI3C\_MasterSlave2MasterFlag

All flags except kI3C\_MasterBetweenFlag and kI3C\_MasterNackDetectFlag can be enabled as interrupts.

---

**Note:** These enums are meant to be OR'd together to form a bit mask.

---

#### Values:

enumerator kI3C\_MasterBetweenFlag

Between messages/DAA's flag

enumerator kI3C\_MasterNackDetectFlag

NACK detected flag

enumerator kI3C\_MasterSlaveStartFlag

Slave request start flag

enumerator kI3C\_MasterControlDoneFlag

Master request complete flag

enumerator kI3C\_MasterCompleteFlag

Transfer complete flag

enumerator kI3C\_MasterRxReadyFlag

Rx data ready in Rx buffer flag

enumerator kI3C\_MasterTxReadyFlag

Tx buffer ready for Tx data flag

enumerator kI3C\_MasterArbitrationWonFlag

Header address won arbitration flag

enumerator kI3C\_MasterErrorFlag

Error occurred flag

enumerator kI3C\_MasterSlave2MasterFlag

Switch from slave to master flag

enumerator kI3C\_MasterClearFlags

enum \_\_i3c\_master\_error\_flags

I3C master error flags to indicate the causes.

---

**Note:** These enums are meant to be OR'd together to form a bit mask.

---

*Values:*

enumerator kI3C\_MasterErrorNackFlag

Slave NACKed the last address

enumerator kI3C\_MasterErrorWriteAbortFlag

Slave NACKed the write data

enumerator kI3C\_MasterErrorParityFlag

Parity error from DDR read

enumerator kI3C\_MasterErrorCrcFlag

CRC error from DDR read

enumerator kI3C\_MasterErrorReadFlag

Read from MRDATAB register when FIFO empty

enumerator kI3C\_MasterErrorWriteFlag

Write to MWDATAB register when FIFO full

enumerator kI3C\_MasterErrorMsgFlag

Message SDR/DDR mismatch or read/write message in wrong state

enumerator kI3C\_MasterErrorInvalidReqFlag

Invalid use of request

enumerator kI3C\_MasterErrorTimeoutFlag

The module has stalled too long in a frame

enumerator kI3C\_MasterAllErrorFlags

All error flags

enum \_\_i3c\_master\_state

I3C working master state.

*Values:*

enumerator kI3C\_MasterStateIdle

Bus stopped.

enumerator kI3C\_MasterStateSlvReq

Bus stopped but slave holding SDA low.

enumerator kI3C\_MasterStateMsgSdr

In SDR Message mode from using MWMSG\_SDR.

enumerator kI3C\_MasterStateNormAct

In normal active SDR mode.

enumerator kI3C\_MasterStateDdr

In DDR Message mode.

enumerator kI3C\_MasterStateDaa

In ENTDAAs mode.

enumerator kI3C\_MasterStateIbiAck

Waiting on IBI ACK/NACK decision.

enumerator kI3C\_MasterStateIbiRcv

Receiving IBI.

enum \_i3c\_master\_enable

I3C master enable configuration.

*Values:*

enumerator kI3C\_MasterOff

Master off.

enumerator kI3C\_MasterOn

Master on.

enumerator kI3C\_MasterCapable

Master capable.

enum \_i3c\_master\_hkeep

I3C high keeper configuration.

*Values:*

enumerator kI3C\_MasterHighKeeperNone

Use PUR to hold SCL high.

enumerator kI3C\_MasterHighKeeperWiredIn

Use pin\_HK controls.

enumerator kI3C\_MasterPassiveSDA

Hi-Z for Bus Free and hold SDA.

enumerator kI3C\_MasterPassiveSDASCL

Hi-Z both for Bus Free, and can Hi-Z SDA for hold.

enum \_i3c\_bus\_request

Emits the requested operation when doing in pieces vs. by message.

*Values:*

enumerator kI3C\_RequestNone

No request.

enumerator kI3C\_RequestEmitStartAddr

Request to emit start and address on bus.

enumerator kI3C\_RequestEmitStop

Request to emit stop on bus.

enumerator kI3C\_RequestIbiAckNack

Manual IBI ACK or NACK.

enumerator kI3C\_RequestProcessDAA

Process DAA.

enumerator kI3C\_RequestForceExit

Request to force exit.

enumerator kI3C\_RequestAutoIbi

Hold in stopped state, but Auto-emit START,7E.

enum \_i3c\_bus\_type

Bus type with EmitStartAddr.

*Values:*

enumerator kI3C\_TypeI3CSdr

SDR mode of I3C.

enumerator kI3C\_TypeI2C

Standard i2c protocol.

enumerator kI3C\_TypeI3CDdr

HDR-DDR mode of I3C.

enum \_i3c\_ibi\_response

IBI response.

*Values:*

enumerator kI3C\_IbiRespAck

ACK with no mandatory byte.

enumerator kI3C\_IbiRespNack

NACK.

enumerator kI3C\_IbiRespAckMandatory

ACK with mandatory byte.

enumerator kI3C\_IbiRespManual

Reserved.

enum \_i3c\_ibi\_type

IBI type.

*Values:*

enumerator kI3C\_IbiNormal

In-band interrupt.

enumerator kI3C\_IbiHotJoin

slave hot join.

enumerator kI3C\_IbiMasterRequest

slave master ship request.

enum \_i3c\_ibi\_state

IBI state.

*Values:*

enumerator kI3C\_IbiReady

In-band interrupt ready state, ready for user to handle.

enumerator kI3C\_IbiDataBuffNeed

In-band interrupt need data buffer for data receive.

enumerator kI3C\_IbiAckNackPending

In-band interrupt Ack/Nack pending for decision.

enum \_i3c\_direction

Direction of master and slave transfers.

*Values:*

enumerator kI3C\_Write

Master transmit.

enumerator kI3C\_Read

Master receive.

enum \_i3c\_tx\_trigger\_level

Watermark of TX int/dma trigger level.

*Values:*

enumerator kI3C\_TxTriggerOnEmpty

Trigger on empty.

enumerator kI3C\_TxTriggerUntilOneQuarterOrLess

Trigger on 1/4 full or less.

enumerator kI3C\_TxTriggerUntilOneHalfOrLess

Trigger on 1/2 full or less.

enumerator kI3C\_TxTriggerUntilOneLessThanFull

Trigger on 1 less than full or less.

enum \_i3c\_rx\_trigger\_level

Watermark of RX int/dma trigger level.

*Values:*

enumerator kI3C\_RxTriggerOnNotEmpty

Trigger on not empty.

enumerator kI3C\_RxTriggerUntilOneQuarterOrMore

Trigger on 1/4 full or more.

enumerator kI3C\_RxTriggerUntilOneHalfOrMore

Trigger on 1/2 full or more.

enumerator kI3C\_RxTriggerUntilThreeQuarterOrMore

Trigger on 3/4 full or more.

enum \_i3c\_rx\_term\_ops

I3C master read termination operations.

*Values:*

enumerator kI3C\_RxTermDisable

Master doesn't terminate read, used for CCC transfer.

enumerator kI3C\_RxAutoTerm

Master auto terminate read after receiving specified bytes(<=255).

enumerator kI3C\_RxTermLastByte

Master terminates read at any time after START, no length limitation.

enum \_i3c\_start\_scl\_delay

I3C start SCL delay options.

*Values:*

enumerator kI3C\_NoDelay

No delay.

enumerator kI3C\_IncreaseSclHalfPeriod

Increases SCL clock period by 1/2.

enumerator kI3C\_IncreaseSclOnePeriod

Increases SCL clock period by 1.

enumerator kI3C\_IncreaseSclOneAndHalfPeriod

Increases SCL clock period by 1 1/2

enum \_i3c\_master\_transfer\_flags

Transfer option flags.

---

**Note:** These enumerations are intended to be OR'd together to form a bit mask of options for the `_i3c_master_transfer::flags` field.

---

*Values:*

enumerator kI3C\_TransferDefaultFlag

Transfer starts with a start signal, stops with a stop signal.

enumerator kI3C\_TransferNoStartFlag

Don't send a start condition, address, and sub address

enumerator kI3C\_TransferRepeatedStartFlag

Send a repeated start condition

enumerator kI3C\_TransferNoStopFlag

Don't send a stop condition.

enumerator kI3C\_TransferWordsFlag

Transfer in words, else transfer in bytes.

enumerator kI3C\_TransferDisableRxTermFlag

Disable Rx termination. Note: It's for I3C CCC transfer.

enumerator kI3C\_TransferRxAutoTermFlag

Set Rx auto-termination. Note: It's adaptive based on Rx size(<=255 bytes) except in I3C\_MasterReceive.

enumerator kI3C\_TransferStartWithBroadcastAddr

Start transfer with 0x7E, then read/write data with device address.

typedef enum \_i3c\_master\_state i3c\_master\_state\_t

I3C working master state.

typedef enum \_i3c\_master\_enable i3c\_master\_enable\_t

I3C master enable configuration.

typedef enum \_i3c\_master\_hkeep i3c\_master\_hkeep\_t

I3C high keeper configuration.

typedef enum \_i3c\_bus\_request i3c\_bus\_request\_t

Emits the requested operation when doing in pieces vs. by message.

typedef enum \_i3c\_bus\_type i3c\_bus\_type\_t

Bus type with EmitStartAddr.

```
typedef enum _i3c_ibi_response i3c_ibi_response_t
    IBI response.
```

```
typedef enum _i3c_ibi_type i3c_ibi_type_t
    IBI type.
```

```
typedef enum _i3c_ibi_state i3c_ibi_state_t
    IBI state.
```

```
typedef enum _i3c_direction i3c_direction_t
    Direction of master and slave transfers.
```

```
typedef enum _i3c_tx_trigger_level i3c_tx_trigger_level_t
    Watermark of TX int/dma trigger level.
```

```
typedef enum _i3c_rx_trigger_level i3c_rx_trigger_level_t
    Watermark of RX int/dma trigger level.
```

```
typedef enum _i3c_rx_term_ops i3c_rx_term_ops_t
    I3C master read termination operations.
```

```
typedef enum _i3c_start_scl_delay i3c_start_scl_delay_t
    I3C start SCL delay options.
```

```
typedef struct _i3c_register_ibi_addr i3c_register_ibi_addr_t
    Structure with setting master IBI rules and slave registry.
```

```
typedef struct _i3c_baudrate i3c_baudrate_hz_t
    Structure with I3C baudrate settings.
```

```
typedef struct _i3c_master_daa_baudrate i3c_master_daa_baudrate_t
    I3C DAA baud rate configuration.
```

```
typedef struct _i3c_master_config i3c_master_config_t
    Structure with settings to initialize the I3C master module.
```

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_MasterGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

```
typedef struct _i3c_master_transfer i3c_master_transfer_t
```

```
typedef struct _i3c_master_handle i3c_master_handle_t
```

```
typedef struct _i3c_master_transfer_callback i3c_master_transfer_callback_t
    i3c master callback functions.
```

```
typedef void (*i3c_master_isr_t)(I3C_Type *base, void *handle)
    Typedef for master interrupt handler.
```

```
struct _i3c_register_ibi_addr
    #include <fsl_i3c.h> Structure with setting master IBI rules and slave registry.
```

### Public Members

```
uint8_t address[5]
    Address array for registry.
```

```
bool i3cFastStart
    Allow the START header to run as push-pull speed if all dynamic addresses take MSB 0.
```



bool ibiHasPayload

Whether the address array has mandatory IBI byte.

struct \_\_i3c\_\_baudrate

*#include <fsl\_i3c.h>* Structure with I3C baudrate settings.

### Public Members

uint32\_t i2cBaud

Desired I2C baud rate in Hertz.

uint32\_t i3cPushPullBaud

Desired I3C push-pull baud rate in Hertz.

uint32\_t i3cOpenDrainBaud

Desired I3C open-drain baud rate in Hertz.

struct \_\_i3c\_\_master\_\_daa\_\_baudrate

*#include <fsl\_i3c.h>* I3C DAA baud rate configuration.

### Public Members

uint32\_t sourceClock\_Hz

FCLK, function clock in Hertz.

uint32\_t i3cPushPullBaud

Desired I3C push-pull baud rate in Hertz.

uint32\_t i3cOpenDrainBaud

Desired I3C open-drain baud rate in Hertz.

struct \_\_i3c\_\_master\_\_config

*#include <fsl\_i3c.h>* Structure with settings to initialize the I3C master module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C\_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

### Public Members

i3c\_master\_enable\_t enableMaster

Enable master mode.

bool disableTimeout

Whether to disable timeout to prevent the ERRWARN.

i3c\_master\_hkeep\_t hKeep

High keeper mode setting.

bool enableOpenDrainStop

Whether to emit open-drain speed STOP.

bool enableOpenDrainHigh

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

i3c\_baudrate\_hz\_t baudRate\_Hz

Desired baud rate settings.

*i3c\_start\_scl\_delay\_t* startSclDelay

I3C SCL delay after START.

*i3c\_start\_scl\_delay\_t* restartSclDelay

I3C SCL delay after Repeated START.

struct *\_i3c\_master\_transfer\_callback*

#include <fsl\_i3c.h> i3c master callback functions.

### Public Members

void (\*slave2Master)(I3C\_Type \*base, void \*userData)

Transfer complete callback

void (\*ibiCallback)(I3C\_Type \*base, *i3c\_master\_handle\_t* \*handle, *i3c\_ibi\_type\_t* ibiType, *i3c\_ibi\_state\_t* ibiState)

IBI event callback

void (\*transferComplete)(I3C\_Type \*base, *i3c\_master\_handle\_t* \*handle, *status\_t* completionStatus, void \*userData)

Transfer complete callback

struct *\_i3c\_master\_transfer*

#include <fsl\_i3c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the I3C\_MasterTransferNonBlocking() API.

### Public Members

uint32\_t flags

Bit mask of options for the transfer. See enumeration *\_i3c\_master\_transfer\_flags* for available options. Set to 0 or *kI3C\_TransferDefaultFlag* for normal transfers.

uint8\_t slaveAddress

The 7-bit slave address.

*i3c\_direction\_t* direction

Either *kI3C\_Read* or *kI3C\_Write*.

uint32\_t subaddress

Sub address. Transferred MSB first.

size\_t subaddressSize

Length of sub address to send in bytes. Maximum size is 4 bytes.

void \*data

Pointer to data to transfer.

size\_t dataSize

Number of bytes to transfer.

*i3c\_bus\_type\_t* busType

bus type.

*i3c\_ibi\_response\_t* ibiResponse

ibi response during transfer.

```
struct _i3c_master_handle
    #include <fsl_i3c.h> Driver handle for master non-blocking APIs.
```

---

**Note:** The contents of this structure are private and subject to change.

---

### Public Members

`uint8_t` state  
Transfer state machine current state.

`uint32_t` remainingBytes  
Remaining byte count in current state.

`i3c_rx_term_ops_t` rxTermOps  
Read termination operation.

`i3c_master_transfer_t` transfer  
Copy of the current transfer info.

`uint8_t` ibiAddress  
Slave address which request IBI.

`uint8_t *`ibiBuff  
Pointer to IBI buffer to keep ibi bytes.

`size_t` ibiPayloadSize  
IBI payload size.

`i3c_ibi_type_t` ibiType  
IBI type.

`i3c_master_transfer_callback_t` callback  
Callback functions pointer.

`void *`userData  
Application data passed to callback.

## 2.21 I3C Master DMA Driver

```
void I3C_MasterTransferCreateHandleEDMA(I3C_Type *base, i3c_master_edma_handle_t
                                         *handle, const i3c_master_edma_callback_t
                                         *callback, void *userData, edma_handle_t
                                         *rxDmaHandle, edma_handle_t *txDmaHandle)
```

Create a new handle for the I3C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_MasterTransferAbortDMA()` API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

### Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C master driver handle.
- *callback* – User provided pointer to the asynchronous callback function.

- `userData` – User provided pointer to the application callback data.
- `rxDmaHandle` – Handle for the DMA receive channel. Created by the user prior to calling this function.
- `txDmaHandle` – Handle for the DMA transmit channel. Created by the user prior to calling this function.

`status_t I3C_MasterTransferEDMA(I3C_Type *base, i3c_master_edma_handle_t *handle, i3c_master_transfer_t *transfer)`

Performs a non-blocking DMA-based transaction on the I3C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.
- `transfer` – The pointer to the transfer descriptor.

#### Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

`status_t I3C_MasterTransferGetCountEDMA(I3C_Type *base, i3c_master_edma_handle_t *handle, size_t *count)`

Returns number of bytes transferred so far.

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.
- `count` – **[out]** Number of bytes transferred so far by the non-blocking transaction.

#### Return values

- `kStatus_Success` –
- `kStatus_NoTransferInProgress` – There is not a DMA transaction currently in progress.

`void I3C_MasterTransferAbortEDMA(I3C_Type *base, i3c_master_edma_handle_t *handle)`

Terminates a non-blocking I3C master transmission early.

---

**Note:** It is not safe to call this function from an IRQ handler that has a higher priority than the DMA peripheral's IRQ priority.

---

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.

`void I3C_MasterTransferEDMAHandleIRQ(I3C_Type *base, void *i3cHandle)`

Reusable routine to handle master interrupts.

---

**Note:** This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

---

### Parameters

- base – The I3C peripheral base address.
- i3cHandle – Pointer to the I3C master DMA driver handle.

```
typedef struct _i3c_master_edma_handle i3c_master_edma_handle_t
```

```
typedef struct _i3c_master_edma_callback i3c_master_edma_callback_t  
i3c master callback functions.
```

```
struct _i3c_master_edma_callback  
#include <fsl_i3c_edma.h> i3c master callback functions.
```

### Public Members

```
void (*slave2Master)(I3C_Type *base, void *userData)
```

Target asks for controller request.

```
void (*ibiCallback)(I3C_Type *base, i3c_master_edma_handle_t *handle, i3c_ibi_type_t  
ibiType, i3c_ibi_state_t ibiState)
```

IBI event callback.

```
void (*transferComplete)(I3C_Type *base, i3c_master_edma_handle_t *handle, status_t  
status, void *userData)
```

Transfer complete callback.

```
struct _i3c_master_edma_handle  
#include <fsl_i3c_edma.h> Driver handle for master EDMA APIs.
```

---

**Note:** The contents of this structure are private and subject to change.

---

### Public Members

```
I3C_Type *base
```

I3C base pointer.

```
uint8_t state
```

Transfer state machine current state.

```
uint32_t transferCount
```

Indicates progress of the transfer

```
uint8_t subaddressBuffer[4]
```

Saving subaddress command.

```
uint8_t subaddressCount
```

Saving command count.

```
i3c_master_transfer_t transfer
```

Copy of the current transfer info.

```
i3c_master_edma_callback_t callback
```

Callback function pointer.

`void *userData`  
Application data passed to callback.

`edma_handle_t *rxDmaHandle`  
Handle for receive DMA channel.

`edma_handle_t *txDmaHandle`  
Handle for transmit DMA channel.

`bool ibiFlag`  
IBIWON flag.

`uint8_t ibiAddress`  
Slave address which request IBI.

`uint8_t *ibiBuff`  
Pointer to IBI buffer to keep ibi bytes.

`size_t ibiPayloadSize`  
IBI payload size.

`i3c_ibi_type_t ibiType`  
IBI type.

`status_t result`  
Transfer result.

## 2.22 I3C Slave Driver

`void I3C_SlaveGetDefaultConfig(i3c_slave_config_t *slaveConfig)`

Provides a default configuration for the I3C slave peripheral.

This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableSlave = true;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with `I3C_SlaveInit()`.

### Parameters

- `slaveConfig` – **[out]** User provided configuration structure for default values. Refer to `i3c_slave_config_t`.

`void I3C_SlaveInit(I3C_Type *base, const i3c_slave_config_t *slaveConfig, uint32_t slowClock_Hz)`

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

### Parameters

- `base` – The I3C peripheral base address.
- `slaveConfig` – User provided peripheral configuration. Use `I3C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- `slowClock_Hz` – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If `FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH` defines as 1, this parameter is useless.

`void I3C_SlaveDeinit(I3C_Type *base)`

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

**Parameters**

- `base` – The I3C peripheral base address.

`static inline void I3C_SlaveEnable(I3C_Type *base, bool isEnabled)`

Enable/Disable Slave.

**Parameters**

- `base` – The I3C peripheral base address.
- `isEnabled` – Enable or disable.

`static inline uint32_t I3C_SlaveGetStatusFlags(I3C_Type *base)`

Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

**See also:**

`_i3c_slave_flags`

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

`static inline void I3C_SlaveClearStatusFlags(I3C_Type *base, uint32_t statusMask)`

Clears the I3C slave status flag state.

The following status register flags can be cleared:

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`

Attempts to clear other flags has no effect.

**See also:**

`_i3c_slave_flags`.

**Parameters**

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetStatusFlags()`.

`static inline uint32_t I3C_SlaveGetErrorStatusFlags(I3C_Type *base)`

Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

**See also:**

`_i3c_slave_error_flags`

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

`static inline void I3C_SlaveClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)`

Clears the I3C slave error status flag state.

**See also:**

`_i3c_slave_error_flags`.

**Parameters**

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_slave_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetErrorStatusFlags()`.

`i3c_slave_activity_state_t I3C_SlaveGetActivityState(I3C_Type *base)`

Gets the I3C slave state.

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

I3C slave activity state, refer `i3c_slave_activity_state_t`.

`status_t I3C_SlaveCheckAndClearError(I3C_Type *base, uint32_t status)`

`static inline void I3C_SlaveEnableInterrupts(I3C_Type *base, uint32_t interruptMask)`

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`



- kI3C\_SlaveErrorFlag
- kI3C\_SlaveHDRCommandMatchFlag
- kI3C\_SlaveCCCHandledFlag
- kI3C\_SlaveEventSentFlag

**Parameters**

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline void I3C\_SlaveDisableInterrupts(I3C\_Type \*base, uint32\_t interruptMask)

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- kI3C\_SlaveBusStartFlag
- kI3C\_SlaveMatchedFlag
- kI3C\_SlaveBusStopFlag
- kI3C\_SlaveRxReadyFlag
- kI3C\_SlaveTxReadyFlag
- kI3C\_SlaveDynamicAddrChangedFlag
- kI3C\_SlaveReceivedCCCFlag
- kI3C\_SlaveErrorFlag
- kI3C\_SlaveHDRCommandMatchFlag
- kI3C\_SlaveCCCHandledFlag
- kI3C\_SlaveEventSentFlag

**Parameters**

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline uint32\_t I3C\_SlaveGetEnabledInterrupts(I3C\_Type \*base)

Returns the set of currently enabled I3C slave interrupt requests.

**Parameters**

- base – The I3C peripheral base address.

**Returns**

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

static inline uint32\_t I3C\_SlaveGetPendingInterrupts(I3C\_Type \*base)

Returns the set of pending I3C slave interrupt requests.

**Parameters**

- base – The I3C peripheral base address.

**Returns**

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of pending interrupts.

```
static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)
```

Enables or disables I3C slave DMA requests.

#### Parameters

- *base* – The I3C peripheral base address.
- *enableTx* – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- *enableRx* – Enable flag for receive DMA request. Pass true for enable, false for disable.
- *width* – DMA read/write unit in bytes.

```
static inline uint32_t I3C_SlaveGetTxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave transmit data register address for DMA transfer.

#### Parameters

- *base* – The I3C peripheral base address.
- *width* – DMA read/write unit in bytes.

#### Returns

The I3C Slave Transmit Data Register address.

```
static inline uint32_t I3C_SlaveGetRxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave receive data register address for DMA transfer.

#### Parameters

- *base* – The I3C peripheral base address.
- *width* – DMA read/write unit in bytes.

#### Returns

The I3C Slave Receive Data Register address.

```
static inline void I3C_SlaveSetWatermarks(I3C_Type *base, i3c_tx_trigger_level_t txLvl, i3c_rx_trigger_level_t rxLvl, bool flushTx, bool flushRx)
```

Sets the watermarks for I3C slave FIFOs.

#### Parameters

- *base* – The I3C peripheral base address.
- *txLvl* – Transmit FIFO watermark level. The `kI3C_SlaveTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
- *rxLvl* – Receive FIFO watermark level. The `kI3C_SlaveRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
- *flushTx* – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- *flushRx* – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

```
static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of bytes in the I3C slave FIFOs.

#### Parameters

- *base* – The I3C peripheral base address.
- *txCount* – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.

- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

`status_t I3C_SlaveSend(I3C_Type *base, const void *txBuff, size_t txSize)`

Performs a polling send transfer on the I3C bus.

#### Parameters

- `base` – The I3C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.

#### Returns

Error or success status returned by API.

`status_t I3C_SlaveReceive(I3C_Type *base, void *rxBuff, size_t rxSize)`

Performs a polling receive transfer on the I3C bus.

#### Parameters

- `base` – The I3C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.

#### Returns

Error or success status returned by API.

`void I3C_SlaveTransferCreateHandle(I3C_Type *base, i3c_slave_handle_t *handle, i3c_slave_transfer_callback_t callback, void *userData)`

Creates a new handle for the I3C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_SlaveTransferAbort()` API shall be called.

---

**Note:** The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

---

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – **[out]** Pointer to the I3C slave driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

`status_t I3C_SlaveTransferNonBlocking(I3C_Type *base, i3c_slave_handle_t *handle, uint32_t eventMask)`

Starts accepting slave transfers.

Call this API after calling `I2C_SlaveInit()` and `I3C_SlaveTransferCreateHandle()` to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to `I3C_SlaveTransferCreateHandle()`. The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the `eventMask` parameter to the OR'd combination of `i3c_slave_transfer_event_t` enumerators for the events you wish to receive. The `kI3C_SlaveTransmitEvent` and `kI3C_SlaveReceiveEvent` events are always enabled and do not need to be included in the mask. Alternatively, you can pass

0 to get a default set of only the transmit and receive events that are always enabled. In addition, the `kI3C_SlaveAllEvents` constant is provided as a convenient way to enable all events.

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to struct: `_i3c_slave_handle` structure which stores the transfer state.
- `eventMask` – Bit mask formed by OR'ing together `i3c_slave_transfer_event_t` enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and `kI3C_SlaveAllEvents` to enable all events.

#### Return values

- `kStatus_Success` – Slave transfers were successfully started.
- `kStatus_I3C_Busy` – Slave transfers have already been started on this handle.

`status_t I3C_SlaveTransferGetCount(I3C_Type *base, i3c_slave_handle_t *handle, size_t *count)`

Gets the slave transfer status during a non-blocking transfer.

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to `i2c_slave_handle_t` structure.
- `count` – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

#### Return values

- `kStatus_Success` –
- `kStatus_NoTransferInProgress` –

`void I3C_SlaveTransferAbort(I3C_Type *base, i3c_slave_handle_t *handle)`

Aborts the slave non-blocking transfers.

---

**Note:** This API could be called at any time to stop slave for handling the bus events.

---

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to struct: `_i3c_slave_handle` structure which stores the transfer state.

`void I3C_SlaveTransferHandleIRQ(I3C_Type *base, void *intHandle)`

Reusable routine to handle slave interrupts.

---

**Note:** This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

---

#### Parameters

- `base` – The I3C peripheral base address.
- `intHandle` – Pointer to struct: `_i3c_slave_handle` structure which stores the transfer state.

enum `_i3c_slave_flags`

I3C slave peripheral flags.

The following status register flags can be cleared:

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`

Only below flags can be enabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`
- `kI3C_SlaveErrorFlag`
- `kI3C_SlaveHDRCommandMatchFlag`
- `kI3C_SlaveCCCHandledFlag`
- `kI3C_SlaveEventSentFlag`

---

**Note:** These enums are meant to be OR'd together to form a bit mask.

---

*Values:*

enumerator `kI3C_SlaveNotStopFlag`

Slave status not stop flag

enumerator `kI3C_SlaveMessageFlag`

Slave status message, indicating slave is listening to the bus traffic or responding

enumerator `kI3C_SlaveRequiredReadFlag`

Slave status required, either is master doing SDR read from slave, or is IBI pushing out.

enumerator `kI3C_SlaveRequiredWriteFlag`

Slave status request write, master is doing SDR write to slave, except slave in ENTDAAMode

enumerator `kI3C_SlaveBusDAAModeFlag`

I3C bus is in ENTDAAMode

enumerator `kI3C_SlaveBusHDRModeFlag`

I3C bus is in HDR mode

enumerator `kI3C_SlaveBusStartFlag`

Start/Re-start event is seen since the bus was last cleared

enumerator `kI3C_SlaveMatchedFlag`

Slave address(dynamic/static) matched since last cleared

enumerator `kI3C_SlaveBusStopFlag`

Stop event is seen since the bus was last cleared

enumerator kI3C\_SlaveRxReadyFlag

Rx data ready in rx buffer flag

enumerator kI3C\_SlaveTxReadyFlag

Tx buffer ready for Tx data flag

enumerator kI3C\_SlaveDynamicAddrChangedFlag

Slave dynamic address has been assigned, re-assigned, or lost

enumerator kI3C\_SlaveReceivedCCCFlag

Slave received Common command code

enumerator kI3C\_SlaveErrorFlag

Error occurred flag

enumerator kI3C\_SlaveHDRCommandMatchFlag

High data rate command match

enumerator kI3C\_SlaveCCCHandledFlag

Slave received Common command code is handled by I3C module

enumerator kI3C\_SlaveEventSentFlag

Slave IBI/P2P/MR/HJ event has been sent

enumerator kI3C\_SlaveIbiDisableFlag

Slave in band interrupt is disabled.

enumerator kI3C\_SlaveMasterRequestDisabledFlag

Slave master request is disabled.

enumerator kI3C\_SlaveHotJoinDisabledFlag

Slave Hot-Join is disabled.

enumerator kI3C\_SlaveClearFlags

All flags which are cleared by the driver upon starting a transfer.

enumerator kI3C\_SlaveAllIrqFlags

enum \_i3c\_slave\_error\_flags

I3C slave error flags to indicate the causes.

---

**Note:** These enums are meant to be OR'd together to form a bit mask.

---

*Values:*

enumerator kI3C\_SlaveErrorOverrunFlag

Slave internal from-bus buffer/FIFO overrun.

enumerator kI3C\_SlaveErrorUnderrunFlag

Slave internal to-bus buffer/FIFO underrun

enumerator kI3C\_SlaveErrorUnderrunNakFlag

Slave internal from-bus buffer/FIFO underrun and NACK error

enumerator kI3C\_SlaveErrorTermFlag

Terminate error from master

enumerator kI3C\_SlaveErrorInvalidStartFlag

Slave invalid start flag

enumerator kI3C\_SlaveErrorSdrParityFlag

SDR parity error

enumerator kI3C\_SlaveErrorHdrParityFlag

HDR parity error

enumerator kI3C\_SlaveErrorHdrCRCFlag

HDR-DDR CRC error

enumerator kI3C\_SlaveErrorS0S1Flag

S0 or S1 error

enumerator kI3C\_SlaveErrorOverreadFlag

Over-read error

enumerator kI3C\_SlaveErrorOverwriteFlag

Over-write error

enum \_i3c\_slave\_event

I3C slave.event.

*Values:*

enumerator kI3C\_SlaveEventNormal

Normal mode.

enumerator kI3C\_SlaveEventIBI

In band interrupt event.

enumerator kI3C\_SlaveEventMasterReq

Master request event.

enumerator kI3C\_SlaveEventHotJoinReq

Hot-join event.

enum \_i3c\_slave\_activity\_state

I3C slave.activity state.

*Values:*

enumerator kI3C\_SlaveNoLatency

Normal bus operation

enumerator kI3C\_SlaveLatency1Ms

1ms of latency.

enumerator kI3C\_SlaveLatency100Ms

100ms of latency.

enumerator kI3C\_SlaveLatency10S

10s latency.

enum \_i3c\_slave\_transfer\_event

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to `I3C_SlaveTransferNonBlocking()` in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

---

**Note:** These enumerations are meant to be OR'd together to form a bit mask of events.

---

*Values:*

enumerator `kI3C_SlaveAddressMatchEvent`

Received the slave address after a start or repeated start.

enumerator `kI3C_SlaveTransmitEvent`

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator `kI3C_SlaveReceiveEvent`

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator `kI3C_SlaveRequiredTransmitEvent`

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator `kI3C_SlaveStartEvent`

A start/repeated start was detected.

enumerator `kI3C_SlaveHDRCommandMatchEvent`

Slave Match HDR Command.

enumerator `kI3C_SlaveCompletionEvent`

A stop was detected, completing the transfer.

enumerator `kI3C_SlaveRequestSentEvent`

Slave request event sent.

enumerator `kI3C_SlaveReceivedCCCEvent`

Slave received CCC event, need to handle by application.

enumerator `kI3C_SlaveAllEvents`

Bit mask of all available events.

typedef enum `_i3c_slave_event` `i3c_slave_event_t`

I3C slave.event.

typedef enum `_i3c_slave_activity_state` `i3c_slave_activity_state_t`

I3C slave.activity state.

typedef struct `_i3c_slave_config` `i3c_slave_config_t`

Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_SlaveGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum `_i3c_slave_transfer_event` `i3c_slave_transfer_event_t`

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to `I3C_SlaveTransferNonBlocking()` in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

---

**Note:** These enumerations are meant to be OR'd together to form a bit mask of events.

---

typedef struct `_i3c_slave_transfer` `i3c_slave_transfer_t`

I3C slave transfer structure.

typedef struct `_i3c_slave_handle` `i3c_slave_handle_t`



```
typedef void (*i3c_slave_transfer_callback_t)(I3C_Type *base, i3c_slave_transfer_t *transfer, void *userData)
```

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the I3C\_SlaveSetCallback() function after you have created a handle.

**Param base**

Base address for the I3C instance on which the event occurred.

**Param transfer**

Pointer to transfer descriptor containing values passed to and/or from the callback.

**Param userData**

Arbitrary pointer-sized value passed from the application.

```
typedef void (*i3c_slave_isr_t)(I3C_Type *base, void *handle)
```

Typedef for slave interrupt handler.

```
struct _i3c_slave_config
```

*#include <fsl\_i3c.h>* Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C\_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

## Public Members

bool enableSlave

Whether to enable slave.

uint8\_t staticAddr

Static address.

uint16\_t vendorID

Device vendor ID(manufacture ID).

uint32\_t partNumber

Device part number info

uint8\_t dcr

Device characteristics register information.

uint8\_t bcr

Bus characteristics register information.

uint8\_t hdrMode

Support hdr mode, could be OR logic in enumeration:i3c\_hdr\_mode\_t.

bool nakAllRequest

Whether to reply NAK to all requests except broadcast CCC.

bool ignoreS0S1Error

Whether to ignore S0/S1 error in SDR mode.

bool offline

Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

bool matchSlaveStartStop

Whether to assert start/stop status only the time slave is addressed.

uint32\_t maxWriteLength

Maximum write length.

uint32\_t maxReadLength

Maximum read length.

struct \_\_i3c\_slave\_transfer

*#include <fsl\_i3c.h>* I3C slave transfer structure.

### Public Members

uint32\_t event

Reason the callback is being invoked.

uint8\_t \*txData

Transfer buffer

size\_t txDataSize

Transfer size

uint8\_t \*rxData

Transfer buffer

size\_t rxDataSize

Transfer size

status\_t completionStatus

Success or error code describing how the transfer completed. Only applies for kI3C\_SlaveCompletionEvent.

size\_t transferredCount

Number of bytes actually transferred since start or last repeated start.

struct \_\_i3c\_slave\_handle

*#include <fsl\_i3c.h>* I3C slave handle structure.

---

**Note:** The contents of this structure are private and subject to change.

---

### Public Members

i3c\_slave\_transfer\_t transfer

I3C slave transfer copy.

bool isBusy

Whether transfer is busy.

bool wasTransmit

Whether the last transfer was a transmit.

uint32\_t eventMask

Mask of enabled events.

uint32\_t transferredCount

Count of bytes transferred.

*i3c\_slave\_transfer\_callback\_t* callback

Callback function called at transfer event.

void \*userData

Callback parameter passed to callback.

size\_t txFifoSize

Tx Fifo size

## 2.23 I3C Slave DMA Driver

```
void I3C_SlaveTransferCreateHandleEDMA(I3C_Type *base, i3c_slave_edma_handle_t *handle,  
                                       i3c_slave_edma_callback_t callback, void *userData,  
                                       edma_handle_t *rxDmaHandle, edma_handle_t  
                                       *txDmaHandle)
```

Create a new handle for the I3C slave DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_SlaveTransferAbortDMA()` API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

### Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C slave driver handle.
- *callback* – User provided pointer to the asynchronous callback function.
- *userData* – User provided pointer to the application callback data.
- *rxDmaHandle* – Handle for the DMA receive channel. Created by the user prior to calling this function.
- *txDmaHandle* – Handle for the DMA transmit channel. Created by the user prior to calling this function.

```
status_t I3C_SlaveTransferEDMA(I3C_Type *base, i3c_slave_edma_handle_t *handle,  
                               i3c_slave_edma_transfer_t *transfer, uint32_t eventMask)
```

Prepares for a non-blocking DMA-based transaction on the I3C bus.

The API will do DMA configuration according to the input transfer descriptor, and the data will be transferred when there's bus master requesting transfer from/to this slave. So the timing of call to this API need be aligned with master application to ensure the transfer is executed as expected. Callback specified when the *handle* was created is invoked when the transaction has completed.

### Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C slave driver handle.
- *transfer* – The pointer to the transfer descriptor.
- *eventMask* – Bit mask formed by OR'ing together `i3c_slave_transfer_event_t` enumerators to specify which events to send to the callback. The transmit and receive events is not allowed to be enabled.

### Return values

- `kStatus_Success` – The transaction was started successfully.

- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.
- `kStatus_Fail` – The transaction can't be set.

`void I3C_SlaveTransferAbortEDMA(I3C_Type *base, i3c_slave_edma_handle_t *handle)`

Abort a slave edma non-blocking transfer in a early time.

#### Parameters

- `base` – I3C peripheral base address
- `handle` – pointer to `i3c_slave_edma_handle_t` structure

`void I3C_SlaveTransferEDMAHandleIRQ(I3C_Type *base, void *i3cHandle)`

Reusable routine to handle slave interrupts.

---

**Note:** This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

---

#### Parameters

- `base` – The I3C peripheral base address.
- `i3cHandle` – Pointer to the I3C slave DMA driver handle.

`typedef struct _i3c_slave_edma_handle i3c_slave_edma_handle_t`

`typedef struct _i3c_slave_edma_transfer i3c_slave_edma_transfer_t`

I3C slave transfer structure.

`typedef void (*i3c_slave_edma_callback_t)(I3C_Type *base, i3c_slave_edma_transfer_t *transfer, void *userData)`

Slave event callback function pointer type.

This callback is used only for the slave DMA transfer API.

#### Param base

Base address for the I3C instance on which the event occurred.

#### Param handle

Pointer to slave DMA transfer handle.

#### Param transfer

Pointer to transfer descriptor containing values passed to and/or from the callback.

#### Param userData

Arbitrary pointer-sized value passed from the application.

`struct _i3c_slave_edma_transfer`

`#include <fsl_i3c_edma.h>` I3C slave transfer structure.

#### Public Members

`uint32_t event`

Reason the callback is being invoked.

`uint8_t *txData`

Transfer buffer

`size_t txDataSize`

Transfer size

```
uint8_t *rxData
    Transfer buffer

size_t rxDataSize
    Transfer size

status_t completionStatus
    Success or error code describing how the transfer completed. Only applies for
    kI3C_SlaveCompletionEvent.

struct _i3c_slave_edma_handle
    #include <fsl_i3c_edma.h> I3C slave edma handle structure.
```

---

**Note:** The contents of this structure are private and subject to change.

---

### Public Members

```
I3C_Type *base
    I3C base pointer.

i3c_slave_edma_transfer_t transfer
    I3C slave transfer copy.

bool isBusy
    Whether transfer is busy.

bool wasTransmit
    Whether the last transfer was a transmit.

uint32_t eventMask
    Mask of enabled events.

i3c_slave_edma_callback_t callback
    Callback function called at transfer event.

edma_handle_t *rxDmaHandle
    Handle for receive DMA channel.

edma_handle_t *txDmaHandle
    Handle for transmit DMA channel.

void *userData
    Callback parameter passed to callback.
```

## 2.24 INPUTMUX: Input Multiplexing Driver

```
enum _inputmux_index_t
    Values:

    enumerator kINPUTMUX_INDEX_CTIMER0CAPTSEL0
    enumerator kINPUTMUX_INDEX_CTIMER0CAPTSEL1
    enumerator kINPUTMUX_INDEX_CTIMER0CAPTSEL2
    enumerator kINPUTMUX_INDEX_CTIMER0CAPTSEL3
```

enumerator kINPUTMUX\_INDEX\_CTIMER1CAPTSEL0  
enumerator kINPUTMUX\_INDEX\_CTIMER1CAPTSEL1  
enumerator kINPUTMUX\_INDEX\_CTIMER1CAPTSEL2  
enumerator kINPUTMUX\_INDEX\_CTIMER1CAPTSEL3  
enumerator kINPUTMUX\_INDEX\_CTIMER2CAPTSEL0  
enumerator kINPUTMUX\_INDEX\_CTIMER2CAPTSEL1  
enumerator kINPUTMUX\_INDEX\_CTIMER2CAPTSEL2  
enumerator kINPUTMUX\_INDEX\_CTIMER2CAPTSEL3  
enumerator kINPUTMUX\_INDEX\_ADC0\_TRIGSEL0  
enumerator kINPUTMUX\_INDEX\_ADC0\_TRIGSEL1  
enumerator kINPUTMUX\_INDEX\_ADC0\_TRIGSEL2  
enumerator kINPUTMUX\_INDEX\_ADC0\_TRIGSEL3  
enumerator kINPUTMUX\_INDEX\_QDC0\_ICAPSEL1  
enumerator kINPUTMUX\_INDEX\_QDC0\_ICAPSEL2  
enumerator kINPUTMUX\_INDEX\_QDC0\_ICAPSEL3  
enumerator kINPUTMUX\_INDEX\_FLEXPWM0\_FAULTSEL0  
enumerator kINPUTMUX\_INDEX\_FLEXPWM0\_FAULTSEL1  
enumerator kINPUTMUX\_INDEX\_FLEXPWM0\_FAULTSEL2  
enumerator kINPUTMUX\_INDEX\_FLEXPWM0\_FAULTSEL3  
enumerator kINPUTMUX\_INDEX\_AOI0\_TRIGSEL0  
enumerator kINPUTMUX\_INDEX\_AOI0\_TRIGSEL1  
enumerator kINPUTMUX\_INDEX\_AOI0\_TRIGSEL2  
enumerator kINPUTMUX\_INDEX\_AOI0\_TRIGSEL3  
enumerator kINPUTMUX\_INDEX\_AOI0\_TRIGSEL4  
enumerator kINPUTMUX\_INDEX\_AOI0\_TRIGSEL5  
enumerator kINPUTMUX\_INDEX\_AOI0\_TRIGSEL6  
enumerator kINPUTMUX\_INDEX\_AOI0\_TRIGSEL7  
enumerator kINPUTMUX\_INDEX\_AOI0\_TRIGSEL8  
enumerator kINPUTMUX\_INDEX\_AOI0\_TRIGSEL9  
enumerator kINPUTMUX\_INDEX\_AOI0\_TRIGSEL10  
enumerator kINPUTMUX\_INDEX\_AOI0\_TRIGSEL11  
enumerator kINPUTMUX\_INDEX\_AOI0\_TRIGSEL12  
enumerator kINPUTMUX\_INDEX\_AOI0\_TRIGSEL13

enumerator kINPUTMUX\_INDEX\_AOI0\_TRIGSEL14  
enumerator kINPUTMUX\_INDEX\_AOI0\_TRIGSEL15  
enumerator kINPUTMUX\_INDEX\_EXT\_TRIGSEL0  
enumerator kINPUTMUX\_INDEX\_EXT\_TRIGSEL1  
enumerator kINPUTMUX\_INDEX\_EXT\_TRIGSEL2  
enumerator kINPUTMUX\_INDEX\_EXT\_TRIGSEL3  
enumerator kINPUTMUX\_INDEX\_EXT\_TRIGSEL4  
enumerator kINPUTMUX\_INDEX\_EXT\_TRIGSEL6  
enumerator kINPUTMUX\_INDEX\_EXT\_TRIGSEL7

enum \_inputmux\_connection\_t  
INPUTMUX connections type.

*Values:*

enumerator kINPUTMUX\_CtimerInp0ToTimer0Cptsel  
TIMER0 CAPTSEL.  
enumerator kINPUTMUX\_CtimerInp1ToTimer0Cptsel  
enumerator kINPUTMUX\_CtimerInp2ToTimer0Cptsel  
enumerator kINPUTMUX\_CtimerInp3ToTimer0Cptsel  
enumerator kINPUTMUX\_CtimerInp4ToTimer0Cptsel  
enumerator kINPUTMUX\_CtimerInp5ToTimer0Cptsel  
enumerator kINPUTMUX\_CtimerInp6ToTimer0Cptsel  
enumerator kINPUTMUX\_CtimerInp7ToTimer0Cptsel  
enumerator kINPUTMUX\_CtimerInp8ToTimer0Cptsel  
enumerator kINPUTMUX\_CtimerInp9ToTimer0Cptsel  
enumerator kINPUTMUX\_CtimerInp12ToTimer0Cptsel  
enumerator kINPUTMUX\_CtimerInp13ToTimer0Cptsel  
enumerator kINPUTMUX\_CtimerInp14ToTimer0Cptsel  
enumerator kINPUTMUX\_CtimerInp15ToTimer0Cptsel  
enumerator kINPUTMUX\_CtimerInp16ToTimer0Cptsel  
enumerator kINPUTMUX\_CtimerInp17ToTimer0Cptsel  
enumerator kINPUTMUX\_CtimerInp18ToTimer0Cptsel  
enumerator kINPUTMUX\_CtimerInp19ToTimer0Cptsel  
enumerator kINPUTMUX\_Usb0StartOffFrameToTimer0Cptsel  
enumerator kINPUTMUX\_Aoi0Out0ToTimer0Cptsel  
enumerator kINPUTMUX\_Aoi0Out1ToTimer0Cptsel

enumerator kINPUTMUX\_\_Aoi0Out2ToTimer0Captsel  
enumerator kINPUTMUX\_\_Aoi0Out3ToTimer0Captsel  
enumerator kINPUTMUX\_\_Adc0Tcomp0ToTimer0Captsel  
enumerator kINPUTMUX\_\_Adc0Tcomp1ToTimer0Captsel  
enumerator kINPUTMUX\_\_Adc0Tcomp2ToTimer0Captsel  
enumerator kINPUTMUX\_\_Adc0Tcomp3ToTimer0Captsel  
enumerator kINPUTMUX\_\_Cmp0OutToTimer0Captsel  
enumerator kINPUTMUX\_\_Cmp1OutToTimer0Captsel  
enumerator kINPUTMUX\_\_Ctimer1M1ToTimer0Captsel  
enumerator kINPUTMUX\_\_Ctimer1M2ToTimer0Captsel  
enumerator kINPUTMUX\_\_Ctimer1M3ToTimer0Captsel  
enumerator kINPUTMUX\_\_Ctimer2M1ToTimer0Captsel  
enumerator kINPUTMUX\_\_Ctimer2M2ToTimer0Captsel  
enumerator kINPUTMUX\_\_Ctimer2M3ToTimer0Captsel  
enumerator kINPUTMUX\_\_Qdc0CmpFlag0ToTimer0Captsel  
enumerator kINPUTMUX\_\_Qdc0CmpFlag1ToTimer0Captsel  
enumerator kINPUTMUX\_\_Qdc0CmpFlag2ToTimer0Captsel  
enumerator kINPUTMUX\_\_Qdc0CmpFlag3ToTimer0Captsel  
enumerator kINPUTMUX\_\_Qdc0PosMatch0ToTimer0Captsel  
enumerator kINPUTMUX\_\_Pwm0Sm0OutTrig0ToTimer0Captsel  
enumerator kINPUTMUX\_\_Pwm0Sm1OutTrig0ToTimer0Captsel  
enumerator kINPUTMUX\_\_Pwm0Sm2OutTrig0ToTimer0Captsel  
enumerator kINPUTMUX\_\_Lpi2c0MasterEndOfPacketToTimer0Captsel  
enumerator kINPUTMUX\_\_Lpi2c0SlaveEndOfPacketToTimer0Captsel  
enumerator kINPUTMUX\_\_Lpspi0EndOfFrameToTimer0Captsel  
enumerator kINPUTMUX\_\_Lpspi0ReceivedDataWordToTimer0Captsel  
enumerator kINPUTMUX\_\_Lpspi1EndOfFrameToTimer0Captsel  
enumerator kINPUTMUX\_\_Lpspi1ReceivedDataWordToTimer0Captsel  
enumerator kINPUTMUX\_\_Lpuart0ReceivedDataWordToTimer0Captsel  
enumerator kINPUTMUX\_\_Lpuart0TransmittedDataWordToTimer0Captsel  
enumerator kINPUTMUX\_\_Lpuart0ReceiveLineIdleToTimer0Captsel  
enumerator kINPUTMUX\_\_Lpuart1ReceivedDataWordToTimer0Captsel  
enumerator kINPUTMUX\_\_Lpuart1TransmittedDataWordToTimer0Captsel



enumerator kINPUTMUX\_Lpuart1ReceiveLineIdleToTimer0Captsel  
enumerator kINPUTMUX\_Lpuart2ReceivedDataWordToTimer0Captsel  
enumerator kINPUTMUX\_Lpuart2TransmittedDataWordToTimer0Captsel  
enumerator kINPUTMUX\_Lpuart2ReceiveLineIdleToTimer0Captsel  
Timer1 CAPTSEL.  
enumerator kINPUTMUX\_CtimerInp0ToTimer1Captsel  
enumerator kINPUTMUX\_CtimerInp1ToTimer1Captsel  
enumerator kINPUTMUX\_CtimerInp2ToTimer1Captsel  
enumerator kINPUTMUX\_CtimerInp3ToTimer1Captsel  
enumerator kINPUTMUX\_CtimerInp4ToTimer1Captsel  
enumerator kINPUTMUX\_CtimerInp5ToTimer1Captsel  
enumerator kINPUTMUX\_CtimerInp6ToTimer1Captsel  
enumerator kINPUTMUX\_CtimerInp7ToTimer1Captsel  
enumerator kINPUTMUX\_CtimerInp8ToTimer1Captsel  
enumerator kINPUTMUX\_CtimerInp9ToTimer1Captsel  
enumerator kINPUTMUX\_CtimerInp12ToTimer1Captsel  
enumerator kINPUTMUX\_CtimerInp13ToTimer1Captsel  
enumerator kINPUTMUX\_CtimerInp14ToTimer1Captsel  
enumerator kINPUTMUX\_CtimerInp15ToTimer1Captsel  
enumerator kINPUTMUX\_CtimerInp16ToTimer1Captsel  
enumerator kINPUTMUX\_CtimerInp17ToTimer1Captsel  
enumerator kINPUTMUX\_CtimerInp18ToTimer1Captsel  
enumerator kINPUTMUX\_CtimerInp19ToTimer1Captsel  
enumerator kINPUTMUX\_Usb0StartOffFrameToTimer1Captsel  
enumerator kINPUTMUX\_Aoi0Out0ToTimer1Captsel  
enumerator kINPUTMUX\_Aoi0Out1ToTimer1Captsel  
enumerator kINPUTMUX\_Aoi0Out2ToTimer1Captsel  
enumerator kINPUTMUX\_Aoi0Out3ToTimer1Captsel  
enumerator kINPUTMUX\_Adc0Tcomp0ToTimer1Captsel  
enumerator kINPUTMUX\_Adc0Tcomp1ToTimer1Captsel  
enumerator kINPUTMUX\_Adc0Tcomp2ToTimer1Captsel  
enumerator kINPUTMUX\_Adc0Tcomp3ToTimer1Captsel  
enumerator kINPUTMUX\_Cmp0OutToTimer1Captsel

enumerator kINPUTMUX\_\_Cmp1OutToTimer1Captsel  
enumerator kINPUTMUX\_\_Ctimer0M1ToTimer1Captsel  
enumerator kINPUTMUX\_\_Ctimer0M2ToTimer1Captsel  
enumerator kINPUTMUX\_\_Ctimer0M3ToTimer1Captsel  
enumerator kINPUTMUX\_\_Ctimer2M1ToTimer1Captsel  
enumerator kINPUTMUX\_\_Ctimer2M2ToTimer1Captsel  
enumerator kINPUTMUX\_\_Ctimer2M3ToTimer1Captsel  
enumerator kINPUTMUX\_\_Qdc0CmpFlag0ToTimer1Captsel  
enumerator kINPUTMUX\_\_Qdc0CmpFlag1ToTimer1Captsel  
enumerator kINPUTMUX\_\_Qdc0CmpFlag2ToTimer1Captsel  
enumerator kINPUTMUX\_\_Qdc0CmpFlag3ToTimer1Captsel  
enumerator kINPUTMUX\_\_Qdc0PosMatch0ToTimer1Captsel  
enumerator kINPUTMUX\_\_Pwm0Sm0OutTrig0ToTimer1Captsel  
enumerator kINPUTMUX\_\_Pwm0Sm1OutTrig0ToTimer1Captsel  
enumerator kINPUTMUX\_\_Pwm0Sm2OutTrig0ToTimer1Captsel  
enumerator kINPUTMUX\_\_Lpi2c0MasterEndOfPacketToTimer1Captsel  
enumerator kINPUTMUX\_\_Lpi2c0SlaveEndOfPacketToTimer1Captsel  
enumerator kINPUTMUX\_\_Lpspi0EndOfFrameToTimer1Captsel  
enumerator kINPUTMUX\_\_Lpspi0ReceivedDataWordToTimer1Captsel  
enumerator kINPUTMUX\_\_Lpspi1EndOfFrameToTimer1Captsel  
enumerator kINPUTMUX\_\_Lpspi1ReceivedDataWordToTimer1Captsel  
enumerator kINPUTMUX\_\_Lpuart0ReceivedDataWordToTimer1Captsel  
enumerator kINPUTMUX\_\_Lpuart0TransmittedDataWordToTimer1Captsel  
enumerator kINPUTMUX\_\_Lpuart0ReceiveLineIdleToTimer1Captsel  
enumerator kINPUTMUX\_\_Lpuart1ReceivedDataWordToTimer1Captsel  
enumerator kINPUTMUX\_\_Lpuart1TransmittedDataWordToTimer1Captsel  
enumerator kINPUTMUX\_\_Lpuart1ReceiveLineIdleToTimer1Captsel  
enumerator kINPUTMUX\_\_Lpuart2ReceivedDataWordToTimer1Captsel  
enumerator kINPUTMUX\_\_Lpuart2TransmittedDataWordToTimer1Captsel  
enumerator kINPUTMUX\_\_Lpuart2ReceiveLineIdleToTimer1Captsel  
Timer2 CAPTSEL.  
enumerator kINPUTMUX\_\_CtimerInp0ToTimer2Captsel  
enumerator kINPUTMUX\_\_CtimerInp1ToTimer2Captsel

enumerator kINPUTMUX\_CtimerInp2ToTimer2Cptsel  
enumerator kINPUTMUX\_CtimerInp3ToTimer2Cptsel  
enumerator kINPUTMUX\_CtimerInp4ToTimer2Cptsel  
enumerator kINPUTMUX\_CtimerInp5ToTimer2Cptsel  
enumerator kINPUTMUX\_CtimerInp6ToTimer2Cptsel  
enumerator kINPUTMUX\_CtimerInp7ToTimer2Cptsel  
enumerator kINPUTMUX\_CtimerInp8ToTimer2Cptsel  
enumerator kINPUTMUX\_CtimerInp9ToTimer2Cptsel  
enumerator kINPUTMUX\_CtimerInp12ToTimer2Cptsel  
enumerator kINPUTMUX\_CtimerInp13ToTimer2Cptsel  
enumerator kINPUTMUX\_CtimerInp14ToTimer2Cptsel  
enumerator kINPUTMUX\_CtimerInp15ToTimer2Cptsel  
enumerator kINPUTMUX\_CtimerInp16ToTimer2Cptsel  
enumerator kINPUTMUX\_CtimerInp17ToTimer2Cptsel  
enumerator kINPUTMUX\_CtimerInp18ToTimer2Cptsel  
enumerator kINPUTMUX\_CtimerInp19ToTimer2Cptsel  
enumerator kINPUTMUX\_Usb0StartOffFrameToTimer2Cptsel  
enumerator kINPUTMUX\_Aoi0Out0ToTimer2Cptsel  
enumerator kINPUTMUX\_Aoi0Out1ToTimer2Cptsel  
enumerator kINPUTMUX\_Aoi0Out2ToTimer2Cptsel  
enumerator kINPUTMUX\_Aoi0Out3ToTimer2Cptsel  
enumerator kINPUTMUX\_Adc0Tcomp0ToTimer2Cptsel  
enumerator kINPUTMUX\_Adc0Tcomp1ToTimer2Cptsel  
enumerator kINPUTMUX\_Adc0Tcomp2ToTimer2Cptsel  
enumerator kINPUTMUX\_Adc0Tcomp3ToTimer2Cptsel  
enumerator kINPUTMUX\_Cmp0OutToTimer2Cptsel  
enumerator kINPUTMUX\_Cmp1OutToTimer2Cptsel  
enumerator kINPUTMUX\_Ctimer0M1ToTimer2Cptsel  
enumerator kINPUTMUX\_Ctimer0M2ToTimer2Cptsel  
enumerator kINPUTMUX\_Ctimer0M3ToTimer2Cptsel  
enumerator kINPUTMUX\_Ctimer1M1ToTimer2Cptsel  
enumerator kINPUTMUX\_Ctimer1M2ToTimer2Cptsel  
enumerator kINPUTMUX\_Ctimer1M3ToTimer2Cptsel

enumerator kINPUTMUX\_Qdc0CmpFlag0ToTimer2Cptsel  
enumerator kINPUTMUX\_Qdc0CmpFlag1ToTimer2Cptsel  
enumerator kINPUTMUX\_Qdc0CmpFlag2ToTimer2Cptsel  
enumerator kINPUTMUX\_Qdc0CmpFlag3ToTimer2Cptsel  
enumerator kINPUTMUX\_Qdc0PosMatch0ToTimer2Cptsel  
enumerator kINPUTMUX\_Pwm0Sm0OutTrig0ToTimer2Cptsel  
enumerator kINPUTMUX\_Pwm0Sm1OutTrig0ToTimer2Cptsel  
enumerator kINPUTMUX\_Pwm0Sm2OutTrig0ToTimer2Cptsel  
enumerator kINPUTMUX\_Lpi2c0MasterEndOfPacketToTimer2Cptsel  
enumerator kINPUTMUX\_Lpi2c0SlaveEndOfPacketToTimer2Cptsel  
enumerator kINPUTMUX\_Lpspi0EndOfFrameToTimer2Cptsel  
enumerator kINPUTMUX\_Lpspi0ReceivedDataWordToTimer2Cptsel  
enumerator kINPUTMUX\_Lpspi1EndOfFrameToTimer2Cptsel  
enumerator kINPUTMUX\_Lpspi1ReceivedDataWordToTimer2Cptsel  
enumerator kINPUTMUX\_Lpuart0ReceivedDataWordToTimer2Cptsel  
enumerator kINPUTMUX\_Lpuart0TransmittedDataWordToTimer2Cptsel  
enumerator kINPUTMUX\_Lpuart0ReceiveLineIdleToTimer2Cptsel  
enumerator kINPUTMUX\_Lpuart1ReceivedDataWordToTimer2Cptsel  
enumerator kINPUTMUX\_Lpuart1TransmittedDataWordToTimer2Cptsel  
enumerator kINPUTMUX\_Lpuart1ReceiveLineIdleToTimer2Cptsel  
enumerator kINPUTMUX\_Lpuart2ReceivedDataWordToTimer2Cptsel  
enumerator kINPUTMUX\_Lpuart2TransmittedDataWordToTimer2Cptsel  
enumerator kINPUTMUX\_Lpuart2ReceiveLineIdleToTimer2Cptsel

**TIMER0 Trigger.**

enumerator kINPUTMUX\_CtimerInp0ToTimer0Trigger  
enumerator kINPUTMUX\_CtimerInp1ToTimer0Trigger  
enumerator kINPUTMUX\_CtimerInp2ToTimer0Trigger  
enumerator kINPUTMUX\_CtimerInp3ToTimer0Trigger  
enumerator kINPUTMUX\_CtimerInp4ToTimer0Trigger  
enumerator kINPUTMUX\_CtimerInp5ToTimer0Trigger  
enumerator kINPUTMUX\_CtimerInp6ToTimer0Trigger  
enumerator kINPUTMUX\_CtimerInp7ToTimer0Trigger  
enumerator kINPUTMUX\_CtimerInp8ToTimer0Trigger

enumerator kINPUTMUX\_CtimerInp9ToTimer0Trigger  
enumerator kINPUTMUX\_CtimerInp12ToTimer0Trigger  
enumerator kINPUTMUX\_CtimerInp13ToTimer0Trigger  
enumerator kINPUTMUX\_CtimerInp14ToTimer0Trigger  
enumerator kINPUTMUX\_CtimerInp15ToTimer0Trigger  
enumerator kINPUTMUX\_CtimerInp16ToTimer0Trigger  
enumerator kINPUTMUX\_CtimerInp17ToTimer0Trigger  
enumerator kINPUTMUX\_CtimerInp18ToTimer0Trigger  
enumerator kINPUTMUX\_CtimerInp19ToTimer0Trigger  
enumerator kINPUTMUX\_Usb0StartOffFrameToTimer0Trigger  
enumerator kINPUTMUX\_Aoi0Out0ToTimer0Trigger  
enumerator kINPUTMUX\_Aoi0Out1ToTimer0Trigger  
enumerator kINPUTMUX\_Aoi0Out2ToTimer0Trigger  
enumerator kINPUTMUX\_Aoi0Out3ToTimer0Trigger  
enumerator kINPUTMUX\_Adc0Tcomp0ToTimer0Trigger  
enumerator kINPUTMUX\_Adc0Tcomp1ToTimer0Trigger  
enumerator kINPUTMUX\_Adc0Tcomp2ToTimer0Trigger  
enumerator kINPUTMUX\_Adc0Tcomp3ToTimer0Trigger  
enumerator kINPUTMUX\_Cmp0OutToTimer0Trigger  
enumerator kINPUTMUX\_Cmp1OutToTimer0Trigger  
enumerator kINPUTMUX\_Ctimer1M1ToTimer0Trigger  
enumerator kINPUTMUX\_Ctimer1M2ToTimer0Trigger  
enumerator kINPUTMUX\_Ctimer1M3ToTimer0Trigger  
enumerator kINPUTMUX\_Ctimer2M1ToTimer0Trigger  
enumerator kINPUTMUX\_Ctimer2M2ToTimer0Trigger  
enumerator kINPUTMUX\_Ctimer2M3ToTimer0Trigger  
enumerator kINPUTMUX\_Qdc0CmpFlag0ToTimer0Trigger  
enumerator kINPUTMUX\_Qdc0CmpFlag1ToTimer0Trigger  
enumerator kINPUTMUX\_Qdc0CmpFlag2ToTimer0Trigger  
enumerator kINPUTMUX\_Qdc0CmpFlag3ToTimer0Trigger  
enumerator kINPUTMUX\_Qdc0PosMatch0ToTimer0Trigger  
enumerator kINPUTMUX\_Pwm0Sm0OutTrig0ToTimer0Trigger  
enumerator kINPUTMUX\_Pwm0Sm1OutTrig0ToTimer0Trigger

enumerator kINPUTMUX\_Pwm0Sm2OutTrig0ToTimer0Trigger  
enumerator kINPUTMUX\_Lpi2c0MasterEndOfPacketToTimer0Trigger  
enumerator kINPUTMUX\_Lpi2c0SlaveEndOfPacketToTimer0Trigger  
enumerator kINPUTMUX\_Lpspi0EndOfFrameToTimer0Trigger  
enumerator kINPUTMUX\_Lpspi0ReceivedDataWordToTimer0Trigger  
enumerator kINPUTMUX\_Lpspi1EndOfFrameToTimer0Trigger  
enumerator kINPUTMUX\_Lpspi1ReceivedDataWordToTimer0Trigger  
enumerator kINPUTMUX\_Lpuart0ReceivedDataWordToTimer0Trigger  
enumerator kINPUTMUX\_Lpuart0TransmittedDataWordToTimer0Trigger  
enumerator kINPUTMUX\_Lpuart0ReceiveLineIdleToTimer0Trigger  
enumerator kINPUTMUX\_Lpuart1ReceivedDataWordToTimer0Trigger  
enumerator kINPUTMUX\_Lpuart1TransmittedDataWordToTimer0Trigger  
enumerator kINPUTMUX\_Lpuart1ReceiveLineIdleToTimer0Trigger  
enumerator kINPUTMUX\_Lpuart2ReceivedDataWordToTimer0Trigger  
enumerator kINPUTMUX\_Lpuart2TransmittedDataWordToTimer0Trigger  
enumerator kINPUTMUX\_Lpuart2ReceiveLineIdleToTimer0Trigger  
Timer1 Trigger.  
enumerator kINPUTMUX\_CtimerInp0ToTimer1Trigger  
enumerator kINPUTMUX\_CtimerInp1ToTimer1Trigger  
enumerator kINPUTMUX\_CtimerInp2ToTimer1Trigger  
enumerator kINPUTMUX\_CtimerInp3ToTimer1Trigger  
enumerator kINPUTMUX\_CtimerInp4ToTimer1Trigger  
enumerator kINPUTMUX\_CtimerInp5ToTimer1Trigger  
enumerator kINPUTMUX\_CtimerInp6ToTimer1Trigger  
enumerator kINPUTMUX\_CtimerInp7ToTimer1Trigger  
enumerator kINPUTMUX\_CtimerInp8ToTimer1Trigger  
enumerator kINPUTMUX\_CtimerInp9ToTimer1Trigger  
enumerator kINPUTMUX\_CtimerInp12ToTimer1Trigger  
enumerator kINPUTMUX\_CtimerInp13ToTimer1Trigger  
enumerator kINPUTMUX\_CtimerInp14ToTimer1Trigger  
enumerator kINPUTMUX\_CtimerInp15ToTimer1Trigger  
enumerator kINPUTMUX\_CtimerInp16ToTimer1Trigger  
enumerator kINPUTMUX\_CtimerInp17ToTimer1Trigger

enumerator kINPUTMUX\_\_CtimerInp18ToTimer1Trigger  
enumerator kINPUTMUX\_\_CtimerInp19ToTimer1Trigger  
enumerator kINPUTMUX\_\_Usb0StartOfFrameToTimer1Trigger  
enumerator kINPUTMUX\_\_Aoi0Out0ToTimer1Trigger  
enumerator kINPUTMUX\_\_Aoi0Out1ToTimer1Trigger  
enumerator kINPUTMUX\_\_Aoi0Out2ToTimer1Trigger  
enumerator kINPUTMUX\_\_Aoi0Out3ToTimer1Trigger  
enumerator kINPUTMUX\_\_Adc0Tcomp0ToTimer1Trigger  
enumerator kINPUTMUX\_\_Adc0Tcomp1ToTimer1Trigger  
enumerator kINPUTMUX\_\_Adc0Tcomp2ToTimer1Trigger  
enumerator kINPUTMUX\_\_Adc0Tcomp3ToTimer1Trigger  
enumerator kINPUTMUX\_\_Cmp0OutToTimer1Trigger  
enumerator kINPUTMUX\_\_Cmp1OutToTimer1Trigger  
enumerator kINPUTMUX\_\_Ctimer0M1ToTimer1Trigger  
enumerator kINPUTMUX\_\_Ctimer0M2ToTimer1Trigger  
enumerator kINPUTMUX\_\_Ctimer0M3ToTimer1Trigger  
enumerator kINPUTMUX\_\_Ctimer2M1ToTimer1Trigger  
enumerator kINPUTMUX\_\_Ctimer2M2ToTimer1Trigger  
enumerator kINPUTMUX\_\_Ctimer2M3ToTimer1Trigger  
enumerator kINPUTMUX\_\_Qdc0CmpFlag0ToTimer1Trigger  
enumerator kINPUTMUX\_\_Qdc0CmpFlag1ToTimer1Trigger  
enumerator kINPUTMUX\_\_Qdc0CmpFlag2ToTimer1Trigger  
enumerator kINPUTMUX\_\_Qdc0CmpFlag3ToTimer1Trigger  
enumerator kINPUTMUX\_\_Qdc0PosMatch0ToTimer1Trigger  
enumerator kINPUTMUX\_\_Pwm0Sm0OutTrig0ToTimer1Trigger  
enumerator kINPUTMUX\_\_Pwm0Sm1OutTrig0ToTimer1Trigger  
enumerator kINPUTMUX\_\_Pwm0Sm2OutTrig0ToTimer1Trigger  
enumerator kINPUTMUX\_\_Lpi2c0MasterEndOfPacketToTimer1Trigger  
enumerator kINPUTMUX\_\_Lpi2c0SlaveEndOfPacketToTimer1Trigger  
enumerator kINPUTMUX\_\_Lpspi0EndOfFrameToTimer1Trigger  
enumerator kINPUTMUX\_\_Lpspi0ReceivedDataWordToTimer1Trigger  
enumerator kINPUTMUX\_\_Lpspi1EndOfFrameToTimer1Trigger  
enumerator kINPUTMUX\_\_Lpspi1ReceivedDataWordToTimer1Trigger

enumerator kINPUTMUX\_Lpuart0ReceivedDataWordToTimer1Trigger  
enumerator kINPUTMUX\_Lpuart0TransmittedDataWordToTimer1Trigger  
enumerator kINPUTMUX\_Lpuart0ReceiveLineIdleToTimer1Trigger  
enumerator kINPUTMUX\_Lpuart1ReceivedDataWordToTimer1Trigger  
enumerator kINPUTMUX\_Lpuart1TransmittedDataWordToTimer1Trigger  
enumerator kINPUTMUX\_Lpuart1ReceiveLineIdleToTimer1Trigger  
enumerator kINPUTMUX\_Lpuart2ReceivedDataWordToTimer1Trigger  
enumerator kINPUTMUX\_Lpuart2TransmittedDataWordToTimer1Trigger  
enumerator kINPUTMUX\_Lpuart2ReceiveLineIdleToTimer1Trigger  
Timer2 Trigger.  
enumerator kINPUTMUX\_CtimerInp0ToTimer2Trigger  
enumerator kINPUTMUX\_CtimerInp1ToTimer2Trigger  
enumerator kINPUTMUX\_CtimerInp2ToTimer2Trigger  
enumerator kINPUTMUX\_CtimerInp3ToTimer2Trigger  
enumerator kINPUTMUX\_CtimerInp4ToTimer2Trigger  
enumerator kINPUTMUX\_CtimerInp5ToTimer2Trigger  
enumerator kINPUTMUX\_CtimerInp6ToTimer2Trigger  
enumerator kINPUTMUX\_CtimerInp7ToTimer2Trigger  
enumerator kINPUTMUX\_CtimerInp8ToTimer2Trigger  
enumerator kINPUTMUX\_CtimerInp9ToTimer2Trigger  
enumerator kINPUTMUX\_CtimerInp12ToTimer2Trigger  
enumerator kINPUTMUX\_CtimerInp13ToTimer2Trigger  
enumerator kINPUTMUX\_CtimerInp14ToTimer2Trigger  
enumerator kINPUTMUX\_CtimerInp15ToTimer2Trigger  
enumerator kINPUTMUX\_CtimerInp16ToTimer2Trigger  
enumerator kINPUTMUX\_CtimerInp17ToTimer2Trigger  
enumerator kINPUTMUX\_CtimerInp18ToTimer2Trigger  
enumerator kINPUTMUX\_CtimerInp19ToTimer2Trigger  
enumerator kINPUTMUX\_Usb0StartOfFrameToTimer2Trigger  
enumerator kINPUTMUX\_Aoi0Out0ToTimer2Trigger  
enumerator kINPUTMUX\_Aoi0Out1ToTimer2Trigger  
enumerator kINPUTMUX\_Aoi0Out2ToTimer2Trigger  
enumerator kINPUTMUX\_Aoi0Out3ToTimer2Trigger



enumerator kINPUTMUX\_\_Adc0Tcomp0ToTimer2Trigger  
enumerator kINPUTMUX\_\_Adc0Tcomp1ToTimer2Trigger  
enumerator kINPUTMUX\_\_Adc0Tcomp2ToTimer2Trigger  
enumerator kINPUTMUX\_\_Adc0Tcomp3ToTimer2Trigger  
enumerator kINPUTMUX\_\_Cmp0OutToTimer2Trigger  
enumerator kINPUTMUX\_\_Cmp1OutToTimer2Trigger  
enumerator kINPUTMUX\_\_Ctimer0M1ToTimer2Trigger  
enumerator kINPUTMUX\_\_Ctimer0M2ToTimer2Trigger  
enumerator kINPUTMUX\_\_Ctimer0M3ToTimer2Trigger  
enumerator kINPUTMUX\_\_Ctimer1M1ToTimer2Trigger  
enumerator kINPUTMUX\_\_Ctimer1M2ToTimer2Trigger  
enumerator kINPUTMUX\_\_Ctimer1M3ToTimer2Trigger  
enumerator kINPUTMUX\_\_Qdc0CmpFlag0ToTimer2Trigger  
enumerator kINPUTMUX\_\_Qdc0CmpFlag1ToTimer2Trigger  
enumerator kINPUTMUX\_\_Qdc0CmpFlag2ToTimer2Trigger  
enumerator kINPUTMUX\_\_Qdc0CmpFlag3ToTimer2Trigger  
enumerator kINPUTMUX\_\_Qdc0PosMatch0ToTimer2Trigger  
enumerator kINPUTMUX\_\_Pwm0Sm0OutTrig0ToTimer2Trigger  
enumerator kINPUTMUX\_\_Pwm0Sm1OutTrig0ToTimer2Trigger  
enumerator kINPUTMUX\_\_Pwm0Sm2OutTrig0ToTimer2Trigger  
enumerator kINPUTMUX\_\_Lpi2c0MasterEndOfPacketToTimer2Trigger  
enumerator kINPUTMUX\_\_Lpi2c0SlaveEndOfPacketToTimer2Trigger  
enumerator kINPUTMUX\_\_Lpspi0EndOfFrameToTimer2Trigger  
enumerator kINPUTMUX\_\_Lpspi0ReceivedDataWordToTimer2Trigger  
enumerator kINPUTMUX\_\_Lpspi1EndOfFrameToTimer2Trigger  
enumerator kINPUTMUX\_\_Lpspi1ReceivedDataWordToTimer2Trigger  
enumerator kINPUTMUX\_\_Lpuart0ReceivedDataWordToTimer2Trigger  
enumerator kINPUTMUX\_\_Lpuart0TransmittedDataWordToTimer2Trigger  
enumerator kINPUTMUX\_\_Lpuart0ReceiveLineIdleToTimer2Trigger  
enumerator kINPUTMUX\_\_Lpuart1ReceivedDataWordToTimer2Trigger  
enumerator kINPUTMUX\_\_Lpuart1TransmittedDataWordToTimer2Trigger  
enumerator kINPUTMUX\_\_Lpuart1ReceiveLineIdleToTimer2Trigger  
enumerator kINPUTMUX\_\_Lpuart2ReceivedDataWordToTimer2Trigger

enumerator kINPUTMUX\_Lpuart2TransmittedDataWordToTimer2Trigger

enumerator kINPUTMUX\_Lpuart2ReceiveLineIdleToTimer2Trigger

Selection for frequency measurement reference clock.

enumerator kINPUTMUX\_ClkInToFreqmeasRef

enumerator kINPUTMUX\_FroOsc12MToFreqmeasRef

enumerator kINPUTMUX\_FroHfDivToFreqmeasRef

enumerator kINPUTMUX\_Clk16K1ToFreqmeasRef

enumerator kINPUTMUX\_SlowClkToFreqmeasRef

enumerator kINPUTMUX\_FreqmeClkIn0ToFreqmeasRef

enumerator kINPUTMUX\_FreqmeClkIn1ToFreqmeasRef

enumerator kINPUTMUX\_Aoi0Out0ToFreqmeasRef

enumerator kINPUTMUX\_Aoi0Out1ToFreqmeasRef

enumerator kINPUTMUX\_Pwm0Sm0OutTrig0ToFreqmeasRef

enumerator kINPUTMUX\_Pwm0Sm0OutTrig1ToFreqmeasRef

enumerator kINPUTMUX\_Pwm0Sm1OutTrig0ToFreqmeasRef

enumerator kINPUTMUX\_Pwm0Sm1OutTrig1ToFreqmeasRef

enumerator kINPUTMUX\_Pwm0Sm2OutTrig0ToFreqmeasRef

enumerator kINPUTMUX\_Pwm0Sm2OutTrig1ToFreqmeasRef

Selection for frequency measurement target clock.

enumerator kINPUTMUX\_ClkInToFreqmeasTar

enumerator kINPUTMUX\_FroOsc12MToFreqmeasTar

enumerator kINPUTMUX\_FroHfDivToFreqmeasTar

enumerator kINPUTMUX\_Clk16K1ToFreqmeasTar

enumerator kINPUTMUX\_SlowClkToFreqmeasTar

enumerator kINPUTMUX\_FreqmeClkIn0ToFreqmeasTar

enumerator kINPUTMUX\_FreqmeClkIn1ToFreqmeasTar

enumerator kINPUTMUX\_Aoi0Out0ToFreqmeasTar

enumerator kINPUTMUX\_Aoi0Out1ToFreqmeasTar

enumerator kINPUTMUX\_Pwm0Sm0OutTrig0ToFreqmeasTar

enumerator kINPUTMUX\_Pwm0Sm0OutTrig1ToFreqmeasTar

enumerator kINPUTMUX\_Pwm0Sm1OutTrig0ToFreqmeasTar

enumerator kINPUTMUX\_Pwm0Sm1OutTrig1ToFreqmeasTar

enumerator kINPUTMUX\_Pwm0Sm2OutTrig0ToFreqmeasTar

enumerator kINPUTMUX\_Pwm0Sm2OutTrig1ToFreqmeasTar

Cmp0 Trigger.

enumerator kINPUTMUX\_\_ArmTxevToCmp0Trigger  
enumerator kINPUTMUX\_\_Aoi0Out0ToCmp0Trigger  
enumerator kINPUTMUX\_\_Aoi0Out1ToCmp0Trigger  
enumerator kINPUTMUX\_\_Aoi0Out2ToCmp0Trigger  
enumerator kINPUTMUX\_\_Aoi0Out3ToCmp0Trigger  
enumerator kINPUTMUX\_\_Cmp1OutToCmp0Trigger  
enumerator kINPUTMUX\_\_Ctimer0M0ToCmp0Trigger  
enumerator kINPUTMUX\_\_Ctimer0M2ToCmp0Trigger  
enumerator kINPUTMUX\_\_Ctimer1M0ToCmp0Trigger  
enumerator kINPUTMUX\_\_Ctimer1M2ToCmp0Trigger  
enumerator kINPUTMUX\_\_Ctimer2M0ToCmp0Trigger  
enumerator kINPUTMUX\_\_Ctimer2M2ToCmp0Trigger  
enumerator kINPUTMUX\_\_Lptmr0ToCmp0Trigger  
enumerator kINPUTMUX\_\_Qdc0PosMatch0ToCmp0Trigger  
enumerator kINPUTMUX\_\_Pwm0Sm0OutTrig0ToCmp0Trigger  
enumerator kINPUTMUX\_\_Pwm0Sm0OutTrig1ToCmp0Trigger  
enumerator kINPUTMUX\_\_Pwm0Sm1OutTrig0ToCmp0Trigger  
enumerator kINPUTMUX\_\_Pwm0Sm1OutTrig1ToCmp0Trigger  
enumerator kINPUTMUX\_\_Pwm0Sm2OutTrig0ToCmp0Trigger  
enumerator kINPUTMUX\_\_Pwm0Sm2OutTrig1ToCmp0Trigger  
enumerator kINPUTMUX\_\_Gpio0PinEventTrig0ToCmp0Trigger  
enumerator kINPUTMUX\_\_Gpio1PinEventTrig0ToCmp0Trigger  
enumerator kINPUTMUX\_\_Gpio2PinEventTrig0ToCmp0Trigger  
enumerator kINPUTMUX\_\_Gpio3PinEventTrig0ToCmp0Trigger  
enumerator kINPUTMUX\_\_WuuToCmp0Trigger

**Cmp1 Trigger.**

enumerator kINPUTMUX\_\_ArmTxevToCmp1Trigger  
enumerator kINPUTMUX\_\_Aoi0Out0ToCmp1Trigger  
enumerator kINPUTMUX\_\_Aoi0Out1ToCmp1Trigger  
enumerator kINPUTMUX\_\_Aoi0Out2ToCmp1Trigger  
enumerator kINPUTMUX\_\_Aoi0Out3ToCmp1Trigger  
enumerator kINPUTMUX\_\_Cmp0OutToCmp1Trigger  
enumerator kINPUTMUX\_\_Ctimer0M0ToCmp1Trigger

enumerator kINPUTMUX\_\_Ctimer0M2ToCmp1Trigger  
enumerator kINPUTMUX\_\_Ctimer1M0ToCmp1Trigger  
enumerator kINPUTMUX\_\_Ctimer1M2ToCmp1Trigger  
enumerator kINPUTMUX\_\_Ctimer2M0ToCmp1Trigger  
enumerator kINPUTMUX\_\_Ctimer2M2ToCmp1Trigger  
enumerator kINPUTMUX\_\_Lptmr0ToCmp1Trigger  
enumerator kINPUTMUX\_\_Qdc0PosMatch0ToCmp1Trigger  
enumerator kINPUTMUX\_\_Pwm0Sm0OutTrig0ToCmp1Trigger  
enumerator kINPUTMUX\_\_Pwm0Sm0OutTrig1ToCmp1Trigger  
enumerator kINPUTMUX\_\_Pwm0Sm1OutTrig0ToCmp1Trigger  
enumerator kINPUTMUX\_\_Pwm0Sm1OutTrig1ToCmp1Trigger  
enumerator kINPUTMUX\_\_Pwm0Sm2OutTrig0ToCmp1Trigger  
enumerator kINPUTMUX\_\_Pwm0Sm2OutTrig1ToCmp1Trigger  
enumerator kINPUTMUX\_\_Gpio0PinEventTrig0ToCmp1Trigger  
enumerator kINPUTMUX\_\_Gpio1PinEventTrig0ToCmp1Trigger  
enumerator kINPUTMUX\_\_Gpio2PinEventTrig0ToCmp1Trigger  
enumerator kINPUTMUX\_\_Gpio3PinEventTrig0ToCmp1Trigger  
enumerator kINPUTMUX\_\_WuuToCmp1Trigger  
    Adc0 Trigger.  
enumerator kINPUTMUX\_\_ArmTxevToAdc0Trigger  
enumerator kINPUTMUX\_\_Aoi0Out0ToAdc0Trigger  
enumerator kINPUTMUX\_\_Aoi0Out1ToAdc0Trigger  
enumerator kINPUTMUX\_\_Aoi0Out2ToAdc0Trigger  
enumerator kINPUTMUX\_\_Aoi0Out3ToAdc0Trigger  
enumerator kINPUTMUX\_\_Cmp0OutToAdc0Trigger  
enumerator kINPUTMUX\_\_Cmp1OutToAdc0Trigger  
enumerator kINPUTMUX\_\_Ctimer0M0ToAdc0Trigger  
enumerator kINPUTMUX\_\_Ctimer0M1ToAdc0Trigger  
enumerator kINPUTMUX\_\_Ctimer1M0ToAdc0Trigger  
enumerator kINPUTMUX\_\_Ctimer1M1ToAdc0Trigger  
enumerator kINPUTMUX\_\_Ctimer2M0ToAdc0Trigger  
enumerator kINPUTMUX\_\_Ctimer2M1ToAdc0Trigger  
enumerator kINPUTMUX\_\_Lptmr0ToAdc0Trigger

enumerator kINPUTMUX\_Qdc0PosMatch0ToAdc0Trigger  
enumerator kINPUTMUX\_Pwm0Sm0OutTrig0ToAdc0Trigger  
enumerator kINPUTMUX\_Pwm0Sm0OutTrig1ToAdc0Trigger  
enumerator kINPUTMUX\_Pwm0Sm1OutTrig0ToAdc0Trigger  
enumerator kINPUTMUX\_Pwm0Sm1OutTrig1ToAdc0Trigger  
enumerator kINPUTMUX\_Pwm0Sm2OutTrig0ToAdc0Trigger  
enumerator kINPUTMUX\_Pwm0Sm2OutTrig1ToAdc0Trigger  
enumerator kINPUTMUX\_Gpio0PinEventTrig0ToAdc0Trigger  
enumerator kINPUTMUX\_Gpio1PinEventTrig0ToAdc0Trigger  
enumerator kINPUTMUX\_Gpio2PinEventTrig0ToAdc0Trigger  
enumerator kINPUTMUX\_Gpio3PinEventTrig0ToAdc0Trigger  
enumerator kINPUTMUX\_WuuToAdc0Trigger  
    Qdc0 Trigger.  
enumerator kINPUTMUX\_ArmTxevToQdc0Trigger  
enumerator kINPUTMUX\_Aoi0Out0ToQdc0Trigger  
enumerator kINPUTMUX\_Aoi0Out1ToQdc0Trigger  
enumerator kINPUTMUX\_Aoi0Out2ToQdc0Trigger  
enumerator kINPUTMUX\_Aoi0Out3ToQdc0Trigger  
enumerator kINPUTMUX\_Cmp0OutToQdc0Trigger  
enumerator kINPUTMUX\_Cmp1OutToQdc0Trigger  
enumerator kINPUTMUX\_Ctimer0M2ToQdc0Trigger  
enumerator kINPUTMUX\_Ctimer0M3ToQdc0Trigger  
enumerator kINPUTMUX\_Ctimer1M2ToQdc0Trigger  
enumerator kINPUTMUX\_Ctimer1M3ToQdc0Trigger  
enumerator kINPUTMUX\_Ctimer2M2ToQdc0Trigger  
enumerator kINPUTMUX\_Ctimer2M3ToQdc0Trigger  
enumerator kINPUTMUX\_Qdc0PosMatch0ToQdc0Trigger  
enumerator kINPUTMUX\_Pwm0Sm0OutTrig0ToQdc0Trigger  
enumerator kINPUTMUX\_Pwm0Sm0OutTrig1ToQdc0Trigger  
enumerator kINPUTMUX\_Pwm0Sm1OutTrig0ToQdc0Trigger  
enumerator kINPUTMUX\_Pwm0Sm1OutTrig1ToQdc0Trigger  
enumerator kINPUTMUX\_Pwm0Sm2OutTrig0ToQdc0Trigger  
enumerator kINPUTMUX\_Pwm0Sm2OutTrig1ToQdc0Trigger

enumerator kINPUTMUX\_\_TrigIn0ToQdc0Trigger  
enumerator kINPUTMUX\_\_TrigIn1ToQdc0Trigger  
enumerator kINPUTMUX\_\_TrigIn2ToQdc0Trigger  
enumerator kINPUTMUX\_\_TrigIn3ToQdc0Trigger  
enumerator kINPUTMUX\_\_TrigIn4ToQdc0Trigger  
enumerator kINPUTMUX\_\_TrigIn5ToQdc0Trigger  
enumerator kINPUTMUX\_\_TrigIn6ToQdc0Trigger  
enumerator kINPUTMUX\_\_TrigIn7ToQdc0Trigger  
enumerator kINPUTMUX\_\_TrigIn8ToQdc0Trigger  
enumerator kINPUTMUX\_\_TrigIn9ToQdc0Trigger  
enumerator kINPUTMUX\_\_TrigIn10ToQdc0Trigger  
enumerator kINPUTMUX\_\_TrigIn11ToQdc0Trigger  
enumerator kINPUTMUX\_\_Gpio0PinEventTrig0ToQdc0Trigger  
enumerator kINPUTMUX\_\_Gpio1PinEventTrig0ToQdc0Trigger  
enumerator kINPUTMUX\_\_Gpio2PinEventTrig0ToQdc0Trigger  
enumerator kINPUTMUX\_\_Gpio3PinEventTrig0ToQdc0Trigger  
Qdc0 Home.  
enumerator kINPUTMUX\_\_ArmTxevToQdc0Home  
enumerator kINPUTMUX\_\_Aoi0Out0ToQdc0Home  
enumerator kINPUTMUX\_\_Aoi0Out1ToQdc0Home  
enumerator kINPUTMUX\_\_Aoi0Out2ToQdc0Home  
enumerator kINPUTMUX\_\_Aoi0Out3ToQdc0Home  
enumerator kINPUTMUX\_\_Cmp0OutToQdc0Home  
enumerator kINPUTMUX\_\_Cmp1OutToQdc0Home  
enumerator kINPUTMUX\_\_Ctimer0M2ToQdc0Home  
enumerator kINPUTMUX\_\_Ctimer0M3ToQdc0Home  
enumerator kINPUTMUX\_\_Ctimer1M2ToQdc0Home  
enumerator kINPUTMUX\_\_Ctimer1M3ToQdc0Home  
enumerator kINPUTMUX\_\_Ctimer2M2ToQdc0Home  
enumerator kINPUTMUX\_\_Ctimer2M3ToQdc0Home  
enumerator kINPUTMUX\_\_Qdc0PosMatch0ToQdc0Home  
enumerator kINPUTMUX\_\_Pwm0Sm0OutTrig0ToQdc0Home  
enumerator kINPUTMUX\_\_Pwm0Sm0OutTrig1ToQdc0Home

enumerator kINPUTMUX\_Pwm0Sm1OutTrig0ToQdc0Home  
enumerator kINPUTMUX\_Pwm0Sm1OutTrig1ToQdc0Home  
enumerator kINPUTMUX\_Pwm0Sm2OutTrig0ToQdc0Home  
enumerator kINPUTMUX\_Pwm0Sm2OutTrig1ToQdc0Home  
enumerator kINPUTMUX\_TrigIn0ToQdc0Home  
enumerator kINPUTMUX\_TrigIn1ToQdc0Home  
enumerator kINPUTMUX\_TrigIn2ToQdc0Home  
enumerator kINPUTMUX\_TrigIn3ToQdc0Home  
enumerator kINPUTMUX\_TrigIn4ToQdc0Home  
enumerator kINPUTMUX\_TrigIn5ToQdc0Home  
enumerator kINPUTMUX\_TrigIn6ToQdc0Home  
enumerator kINPUTMUX\_TrigIn7ToQdc0Home  
enumerator kINPUTMUX\_TrigIn8ToQdc0Home  
enumerator kINPUTMUX\_TrigIn9ToQdc0Home  
enumerator kINPUTMUX\_TrigIn10ToQdc0Home  
enumerator kINPUTMUX\_TrigIn11ToQdc0Home  
enumerator kINPUTMUX\_Gpio0PinEventTrig0ToQdc0Home  
enumerator kINPUTMUX\_Gpio1PinEventTrig0ToQdc0Home  
enumerator kINPUTMUX\_Gpio2PinEventTrig0ToQdc0Home  
enumerator kINPUTMUX\_Gpio3PinEventTrig0ToQdc0Home

#### Qdc0 Index.

enumerator kINPUTMUX\_ArmTxevToQdc0Index  
enumerator kINPUTMUX\_Aoi0Out0ToQdc0Index  
enumerator kINPUTMUX\_Aoi0Out1ToQdc0Index  
enumerator kINPUTMUX\_Aoi0Out2ToQdc0Index  
enumerator kINPUTMUX\_Aoi0Out3ToQdc0Index  
enumerator kINPUTMUX\_Cmp0OutToQdc0Index  
enumerator kINPUTMUX\_Cmp1OutToQdc0Index  
enumerator kINPUTMUX\_Ctimer0M2ToQdc0Index  
enumerator kINPUTMUX\_Ctimer0M3ToQdc0Index  
enumerator kINPUTMUX\_Ctimer1M2ToQdc0Index  
enumerator kINPUTMUX\_Ctimer1M3ToQdc0Index  
enumerator kINPUTMUX\_Ctimer2M2ToQdc0Index

enumerator kINPUTMUX\_\_Ctimer2M3ToQdc0Index  
enumerator kINPUTMUX\_\_Qdc0PosMatch0ToQdc0Index  
enumerator kINPUTMUX\_\_Pwm0Sm0OutTrig0ToQdc0Index  
enumerator kINPUTMUX\_\_Pwm0Sm0OutTrig1ToQdc0Index  
enumerator kINPUTMUX\_\_Pwm0Sm1OutTrig0ToQdc0Index  
enumerator kINPUTMUX\_\_Pwm0Sm1OutTrig1ToQdc0Index  
enumerator kINPUTMUX\_\_Pwm0Sm2OutTrig0ToQdc0Index  
enumerator kINPUTMUX\_\_Pwm0Sm2OutTrig1ToQdc0Index  
enumerator kINPUTMUX\_\_TrigIn0ToQdc0Index  
enumerator kINPUTMUX\_\_TrigIn1ToQdc0Index  
enumerator kINPUTMUX\_\_TrigIn2ToQdc0Index  
enumerator kINPUTMUX\_\_TrigIn3ToQdc0Index  
enumerator kINPUTMUX\_\_TrigIn4ToQdc0Index  
enumerator kINPUTMUX\_\_TrigIn5ToQdc0Index  
enumerator kINPUTMUX\_\_TrigIn6ToQdc0Index  
enumerator kINPUTMUX\_\_TrigIn7ToQdc0Index  
enumerator kINPUTMUX\_\_TrigIn8ToQdc0Index  
enumerator kINPUTMUX\_\_TrigIn9ToQdc0Index  
enumerator kINPUTMUX\_\_TrigIn10ToQdc0Index  
enumerator kINPUTMUX\_\_TrigIn11ToQdc0Index  
enumerator kINPUTMUX\_\_Gpio0PinEventTrig0ToQdc0Index  
enumerator kINPUTMUX\_\_Gpio1PinEventTrig0ToQdc0Index  
enumerator kINPUTMUX\_\_Gpio2PinEventTrig0ToQdc0Index  
enumerator kINPUTMUX\_\_Gpio3PinEventTrig0ToQdc0Index  
Qdc0 Phaseb.  
enumerator kINPUTMUX\_\_ArmTxevToQdc0Phaseb  
enumerator kINPUTMUX\_\_Aoi0Out0ToQdc0Phaseb  
enumerator kINPUTMUX\_\_Aoi0Out1ToQdc0Phaseb  
enumerator kINPUTMUX\_\_Aoi0Out2ToQdc0Phaseb  
enumerator kINPUTMUX\_\_Aoi0Out3ToQdc0Phaseb  
enumerator kINPUTMUX\_\_Cmp0OutToQdc0Phaseb  
enumerator kINPUTMUX\_\_Cmp1OutToQdc0Phaseb  
enumerator kINPUTMUX\_\_Ctimer0M2ToQdc0Phaseb



enumerator kINPUTMUX\_\_Ctimer0M3ToQdc0Phaseb  
enumerator kINPUTMUX\_\_Ctimer1M2ToQdc0Phaseb  
enumerator kINPUTMUX\_\_Ctimer1M3ToQdc0Phaseb  
enumerator kINPUTMUX\_\_Ctimer2M2ToQdc0Phaseb  
enumerator kINPUTMUX\_\_Ctimer2M3ToQdc0Phaseb  
enumerator kINPUTMUX\_\_Qdc0PosMatch0ToQdc0Phaseb  
enumerator kINPUTMUX\_\_Pwm0Sm0OutTrig0ToQdc0Phaseb  
enumerator kINPUTMUX\_\_Pwm0Sm0OutTrig1ToQdc0Phaseb  
enumerator kINPUTMUX\_\_Pwm0Sm1OutTrig0ToQdc0Phaseb  
enumerator kINPUTMUX\_\_Pwm0Sm1OutTrig1ToQdc0Phaseb  
enumerator kINPUTMUX\_\_Pwm0Sm2OutTrig0ToQdc0Phaseb  
enumerator kINPUTMUX\_\_Pwm0Sm2OutTrig1ToQdc0Phaseb  
enumerator kINPUTMUX\_\_TrigIn0ToQdc0Phaseb  
enumerator kINPUTMUX\_\_TrigIn1ToQdc0Phaseb  
enumerator kINPUTMUX\_\_TrigIn2ToQdc0Phaseb  
enumerator kINPUTMUX\_\_TrigIn3ToQdc0Phaseb  
enumerator kINPUTMUX\_\_TrigIn4ToQdc0Phaseb  
enumerator kINPUTMUX\_\_TrigIn5ToQdc0Phaseb  
enumerator kINPUTMUX\_\_TrigIn6ToQdc0Phaseb  
enumerator kINPUTMUX\_\_TrigIn7ToQdc0Phaseb  
enumerator kINPUTMUX\_\_TrigIn8ToQdc0Phaseb  
enumerator kINPUTMUX\_\_TrigIn9ToQdc0Phaseb  
enumerator kINPUTMUX\_\_TrigIn10ToQdc0Phaseb  
enumerator kINPUTMUX\_\_TrigIn11ToQdc0Phaseb  
enumerator kINPUTMUX\_\_Gpio0PinEventTrig0ToQdc0Phaseb  
enumerator kINPUTMUX\_\_Gpio1PinEventTrig0ToQdc0Phaseb  
enumerator kINPUTMUX\_\_Gpio2PinEventTrig0ToQdc0Phaseb  
enumerator kINPUTMUX\_\_Gpio3PinEventTrig0ToQdc0Phaseb  
Qdc0 Phasea.  
enumerator kINPUTMUX\_\_ArmTxevToQdc0Phasea  
enumerator kINPUTMUX\_\_Aoi0Out0ToQdc0Phasea  
enumerator kINPUTMUX\_\_Aoi0Out1ToQdc0Phasea  
enumerator kINPUTMUX\_\_Aoi0Out2ToQdc0Phasea

enumerator kINPUTMUX\_\_Aoi0Out3ToQdc0Phasea  
enumerator kINPUTMUX\_\_Cmp0OutToQdc0Phasea  
enumerator kINPUTMUX\_\_Cmp1OutToQdc0Phasea  
enumerator kINPUTMUX\_\_Ctimer0M2ToQdc0Phasea  
enumerator kINPUTMUX\_\_Ctimer0M3ToQdc0Phasea  
enumerator kINPUTMUX\_\_Ctimer1M2ToQdc0Phasea  
enumerator kINPUTMUX\_\_Ctimer1M3ToQdc0Phasea  
enumerator kINPUTMUX\_\_Ctimer2M2ToQdc0Phasea  
enumerator kINPUTMUX\_\_Ctimer2M3ToQdc0Phasea  
enumerator kINPUTMUX\_\_Qdc0PosMatch0ToQdc0Phasea  
enumerator kINPUTMUX\_\_Pwm0Sm0OutTrig0ToQdc0Phasea  
enumerator kINPUTMUX\_\_Pwm0Sm0OutTrig1ToQdc0Phasea  
enumerator kINPUTMUX\_\_Pwm0Sm1OutTrig0ToQdc0Phasea  
enumerator kINPUTMUX\_\_Pwm0Sm1OutTrig1ToQdc0Phasea  
enumerator kINPUTMUX\_\_Pwm0Sm2OutTrig0ToQdc0Phasea  
enumerator kINPUTMUX\_\_Pwm0Sm2OutTrig1ToQdc0Phasea  
enumerator kINPUTMUX\_\_TrigIn0ToQdc0Phasea  
enumerator kINPUTMUX\_\_TrigIn1ToQdc0Phasea  
enumerator kINPUTMUX\_\_TrigIn2ToQdc0Phasea  
enumerator kINPUTMUX\_\_TrigIn3ToQdc0Phasea  
enumerator kINPUTMUX\_\_TrigIn4ToQdc0Phasea  
enumerator kINPUTMUX\_\_TrigIn5ToQdc0Phasea  
enumerator kINPUTMUX\_\_TrigIn6ToQdc0Phasea  
enumerator kINPUTMUX\_\_TrigIn7ToQdc0Phasea  
enumerator kINPUTMUX\_\_TrigIn8ToQdc0Phasea  
enumerator kINPUTMUX\_\_TrigIn9ToQdc0Phasea  
enumerator kINPUTMUX\_\_TrigIn10ToQdc0Phasea  
enumerator kINPUTMUX\_\_TrigIn11ToQdc0Phasea  
enumerator kINPUTMUX\_\_Gpio0PinEventTrig0ToQdc0Phasea  
enumerator kINPUTMUX\_\_Gpio1PinEventTrig0ToQdc0Phasea  
enumerator kINPUTMUX\_\_Gpio2PinEventTrig0ToQdc0Phasea  
enumerator kINPUTMUX\_\_Gpio3PinEventTrig0ToQdc0Phasea  
Qdc0 Icap1-3.

enumerator kINPUTMUX\_ArmTxeVToQdc0Icap1  
enumerator kINPUTMUX\_Aoi0Out0ToQdc0Icap1  
enumerator kINPUTMUX\_Aoi0Out1ToQdc0Icap1  
enumerator kINPUTMUX\_Aoi0Out2ToQdc0Icap1  
enumerator kINPUTMUX\_Aoi0Out3ToQdc0Icap1  
enumerator kINPUTMUX\_Cmp0OutToQdc0Icap1  
enumerator kINPUTMUX\_Cmp1OutToQdc0Icap1  
enumerator kINPUTMUX\_Ctimer0M2ToQdc0Icap1  
enumerator kINPUTMUX\_Ctimer0M3ToQdc0Icap1  
enumerator kINPUTMUX\_Ctimer1M2ToQdc0Icap1  
enumerator kINPUTMUX\_Ctimer1M3ToQdc0Icap1  
enumerator kINPUTMUX\_Ctimer2M2ToQdc0Icap1  
enumerator kINPUTMUX\_Ctimer2M3ToQdc0Icap1  
enumerator kINPUTMUX\_Qdc0PosMatch0ToQdc0Icap1  
enumerator kINPUTMUX\_Pwm0Sm0OutTrig0ToQdc0Icap1  
enumerator kINPUTMUX\_Pwm0Sm0OutTrig1ToQdc0Icap1  
enumerator kINPUTMUX\_Pwm0Sm1OutTrig0ToQdc0Icap1  
enumerator kINPUTMUX\_Pwm0Sm1OutTrig1ToQdc0Icap1  
enumerator kINPUTMUX\_Pwm0Sm2OutTrig0ToQdc0Icap1  
enumerator kINPUTMUX\_Pwm0Sm2OutTrig1ToQdc0Icap1  
enumerator kINPUTMUX\_TrigIn0ToQdc0Icap1  
enumerator kINPUTMUX\_TrigIn1ToQdc0Icap1  
enumerator kINPUTMUX\_TrigIn2ToQdc0Icap1  
enumerator kINPUTMUX\_TrigIn3ToQdc0Icap1  
enumerator kINPUTMUX\_TrigIn4ToQdc0Icap1  
enumerator kINPUTMUX\_TrigIn5ToQdc0Icap1  
enumerator kINPUTMUX\_TrigIn6ToQdc0Icap1  
enumerator kINPUTMUX\_TrigIn7ToQdc0Icap1  
enumerator kINPUTMUX\_TrigIn8ToQdc0Icap1  
enumerator kINPUTMUX\_TrigIn9ToQdc0Icap1  
enumerator kINPUTMUX\_TrigIn10ToQdc0Icap1  
enumerator kINPUTMUX\_TrigIn11ToQdc0Icap1  
enumerator kINPUTMUX\_Gpio0PinEventTrig0ToQdc0Icap1

enumerator kINPUTMUX\_Gpio1PinEventTrig0ToQdc0Icap1  
enumerator kINPUTMUX\_Gpio2PinEventTrig0ToQdc0Icap1  
enumerator kINPUTMUX\_Gpio3PinEventTrig0ToQdc0Icap1  
FlexPWM0\_SM0\_EXTa0 input trigger connections.  
enumerator kINPUTMUX\_ArmTxevToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_Aoi0Out0ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_Aoi0Out1ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_Aoi0Out2ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_Aoi0Out3ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_Cmp0OutToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_Cmp1OutToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_Ctimer0M2ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_Ctimer0M3ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_Ctimer1M2ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_Ctimer1M3ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_Ctimer2M2ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_Ctimer2M3ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_Qdc0CmpFlag0ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_Qdc0CmpFlag1ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_Qdc0CmpFlag2ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_Qdc0CmpFlag3ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_Qdc0PosMatch0ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_TrigIn0ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_TrigIn1ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_TrigIn2ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_TrigIn3ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_TrigIn4ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_TrigIn5ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_TrigIn6ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_TrigIn7ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_TrigIn8ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_TrigIn9ToFlexPwm0Sm0Ext0  
enumerator kINPUTMUX\_TrigIn10ToFlexPwm0Sm0Ext0

enumerator kINPUTMUX\_\_TrigIn11ToFlexPwm0Sm0Exta0  
enumerator kINPUTMUX\_\_Gpio0PinEventTrig0ToFlexPwm0Sm0Exta0  
enumerator kINPUTMUX\_\_Gpio1PinEventTrig0ToFlexPwm0Sm0Exta0  
enumerator kINPUTMUX\_\_Gpio2PinEventTrig0ToFlexPwm0Sm0Exta0  
enumerator kINPUTMUX\_\_Gpio3PinEventTrig0ToFlexPwm0Sm0Exta0  
FlexPWM0\_SM1\_EXT\_A1 input trigger connections.  
enumerator kINPUTMUX\_\_ArmTxevToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Aoi0Out0ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Aoi0Out1ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Aoi0Out2ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Aoi0Out3ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Cmp0OutToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Cmp1OutToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Ctimer0M2ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Ctimer0M3ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Ctimer1M2ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Ctimer1M3ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Ctimer2M2ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Ctimer2M3ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Qdc0CmpFlag0ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Qdc0CmpFlag1ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Qdc0CmpFlag2ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Qdc0CmpFlag3ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Qdc0PosMatch0ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_TrigIn0ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_TrigIn1ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_TrigIn2ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_TrigIn3ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_TrigIn4ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_TrigIn5ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_TrigIn6ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_TrigIn7ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_TrigIn8ToFlexPwm0Sm1Exta1

enumerator kINPUTMUX\_\_TrigIn9ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_TrigIn10ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_TrigIn11ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Gpio0PinEventTrig0ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Gpio1PinEventTrig0ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Gpio2PinEventTrig0ToFlexPwm0Sm1Exta1  
enumerator kINPUTMUX\_\_Gpio3PinEventTrig0ToFlexPwm0Sm1Exta1  
FlexPWM0\_SM2\_EXT2 input trigger connections.  
enumerator kINPUTMUX\_\_ArmTxevToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Aoi0Out0ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Aoi0Out1ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Aoi0Out2ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Aoi0Out3ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Cmp0OutToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Cmp1OutToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Ctimer0M2ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Ctimer0M3ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Ctimer1M2ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Ctimer1M3ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Ctimer2M2ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Ctimer2M3ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Qdc0CmpFlag0ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Qdc0CmpFlag1ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Qdc0CmpFlag2ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Qdc0CmpFlag3ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Qdc0PosMatch0ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_TrigIn0ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_TrigIn1ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_TrigIn2ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_TrigIn3ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_TrigIn4ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_TrigIn5ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_TrigIn6ToFlexPwm0Sm2Exta2

enumerator kINPUTMUX\_\_TrigIn7ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_TrigIn8ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_TrigIn9ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_TrigIn10ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_TrigIn11ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Gpio0PinEventTrig0ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Gpio1PinEventTrig0ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Gpio2PinEventTrig0ToFlexPwm0Sm2Exta2  
enumerator kINPUTMUX\_\_Gpio3PinEventTrig0ToFlexPwm0Sm2Exta2  
FlexPWM0\_SM0\_EXTSYNC0 input trigger connections.  
enumerator kINPUTMUX\_\_ArmTxevToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Aoi0Out0ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Aoi0Out1ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Aoi0Out2ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Aoi0Out3ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Cmp0OutToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Cmp1OutToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Ctimer0M2ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Ctimer0M3ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Ctimer1M2ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Ctimer1M3ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Ctimer2M2ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Ctimer2M3ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Qdc0CmpFlag0ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Qdc0CmpFlag1ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Qdc0CmpFlag2ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Qdc0CmpFlag3ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Qdc0PosMatch0ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_TrigIn0ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_TrigIn1ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_TrigIn2ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_TrigIn3ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_TrigIn4ToFlexPwm0Sm0Extsync0

enumerator kINPUTMUX\_\_TrigIn5ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_TrigIn6ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_TrigIn7ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_TrigIn8ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_TrigIn9ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_TrigIn10ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_TrigIn11ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Gpio0PinEventTrig0ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Gpio1PinEventTrig0ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Gpio2PinEventTrig0ToFlexPwm0Sm0Extsync0  
enumerator kINPUTMUX\_\_Gpio3PinEventTrig0ToFlexPwm0Sm0Extsync0  
FlexPWM0\_SM1\_EXTSYNC1 input trigger connections.  
enumerator kINPUTMUX\_\_ArmTxevToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Aoi0Out0ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Aoi0Out1ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Aoi0Out2ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Aoi0Out3ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Cmp0OutToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Cmp1OutToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Ctimer0M2ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Ctimer0M3ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Ctimer1M2ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Ctimer1M3ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Ctimer2M2ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Ctimer2M3ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Qdc0CmpFlag0ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Qdc0CmpFlag1ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Qdc0CmpFlag2ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Qdc0CmpFlag3ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Qdc0PosMatch0ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_TrigIn0ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_TrigIn1ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_TrigIn2ToFlexPwm0Sm1Extsync1



enumerator kINPUTMUX\_\_TrigIn3ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_TrigIn4ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_TrigIn5ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_TrigIn6ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_TrigIn7ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_TrigIn8ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_TrigIn9ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_TrigIn10ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_TrigIn11ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Gpio0PinEventTrig0ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Gpio1PinEventTrig0ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Gpio2PinEventTrig0ToFlexPwm0Sm1Extsync1  
enumerator kINPUTMUX\_\_Gpio3PinEventTrig0ToFlexPwm0Sm1Extsync1  
FlexPWM0\_SM2\_EXTSYNC2 input trigger connections.  
enumerator kINPUTMUX\_\_ArmTxevToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Aoi0Out0ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Aoi0Out1ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Aoi0Out2ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Aoi0Out3ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Cmp0OutToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Cmp1OutToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Ctimer0M2ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Ctimer0M3ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Ctimer1M2ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Ctimer1M3ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Ctimer2M2ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Ctimer2M3ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Qdc0CmpFlag0ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Qdc0CmpFlag1ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Qdc0CmpFlag2ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Qdc0CmpFlag3ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Qdc0PosMatch0ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_TrigIn0ToFlexPwm0Sm2Extsync2

enumerator kINPUTMUX\_\_TrigIn1ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_TrigIn2ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_TrigIn3ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_TrigIn4ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_TrigIn5ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_TrigIn6ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_TrigIn7ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_TrigIn8ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_TrigIn9ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_TrigIn10ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_TrigIn11ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Gpio0PinEventTrig0ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Gpio1PinEventTrig0ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Gpio2PinEventTrig0ToFlexPwm0Sm2Extsync2  
enumerator kINPUTMUX\_\_Gpio3PinEventTrig0ToFlexPwm0Sm2Extsync2  
FlexPWM0\_FAULT input trigger connections.  
enumerator kINPUTMUX\_\_ArmTxevToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Aoi0Out0ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Aoi0Out1ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Aoi0Out2ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Aoi0Out3ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Cmp0OutToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Cmp1OutToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Ctimer0M2ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Ctimer0M3ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Ctimer1M2ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Ctimer1M3ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Ctimer2M2ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Ctimer2M3ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Qdc0CmpFlag0ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Qdc0CmpFlag1ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Qdc0CmpFlag2ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Qdc0CmpFlag3ToFlexPwm0Fault

enumerator kINPUTMUX\_\_Qdc0PosMatch0ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_TrigIn0ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_TrigIn1ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_TrigIn2ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_TrigIn3ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_TrigIn4ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_TrigIn5ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_TrigIn6ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_TrigIn7ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_TrigIn8ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_TrigIn9ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_TrigIn10ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_TrigIn11ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Gpio0PinEventTrig0ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Gpio1PinEventTrig0ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Gpio2PinEventTrig0ToFlexPwm0Fault  
enumerator kINPUTMUX\_\_Gpio3PinEventTrig0ToFlexPwm0Fault  
FlexPWM0\_FORCE input trigger connections.  
enumerator kINPUTMUX\_\_ArmTxevToFlexPwm0Force  
enumerator kINPUTMUX\_\_Aoi0Out0ToFlexPwm0Force  
enumerator kINPUTMUX\_\_Aoi0Out1ToFlexPwm0Force  
enumerator kINPUTMUX\_\_Aoi0Out2ToFlexPwm0Force  
enumerator kINPUTMUX\_\_Aoi0Out3ToFlexPwm0Force  
enumerator kINPUTMUX\_\_Cmp0OutToFlexPwm0Force  
enumerator kINPUTMUX\_\_Cmp1OutToFlexPwm0Force  
enumerator kINPUTMUX\_\_Ctimer0M2ToFlexPwm0Force  
enumerator kINPUTMUX\_\_Ctimer0M3ToFlexPwm0Force  
enumerator kINPUTMUX\_\_Ctimer1M2ToFlexPwm0Force  
enumerator kINPUTMUX\_\_Ctimer1M3ToFlexPwm0Force  
enumerator kINPUTMUX\_\_Ctimer2M2ToFlexPwm0Force  
enumerator kINPUTMUX\_\_Ctimer2M3ToFlexPwm0Force  
enumerator kINPUTMUX\_\_Qdc0CmpFlag0ToFlexPwm0Force  
enumerator kINPUTMUX\_\_Qdc0CmpFlag1ToFlexPwm0Force

enumerator kINPUTMUX\_Qdc0CmpFlag2ToFlexPwm0Force  
enumerator kINPUTMUX\_Qdc0CmpFlag3ToFlexPwm0Force  
enumerator kINPUTMUX\_Qdc0PosMatch0ToFlexPwm0Force  
enumerator kINPUTMUX\_TrigIn0ToFlexPwm0Force  
enumerator kINPUTMUX\_TrigIn1ToFlexPwm0Force  
enumerator kINPUTMUX\_TrigIn2ToFlexPwm0Force  
enumerator kINPUTMUX\_TrigIn3ToFlexPwm0Force  
enumerator kINPUTMUX\_TrigIn4ToFlexPwm0Force  
enumerator kINPUTMUX\_TrigIn5ToFlexPwm0Force  
enumerator kINPUTMUX\_TrigIn6ToFlexPwm0Force  
enumerator kINPUTMUX\_TrigIn7ToFlexPwm0Force  
enumerator kINPUTMUX\_TrigIn8ToFlexPwm0Force  
enumerator kINPUTMUX\_TrigIn9ToFlexPwm0Force  
enumerator kINPUTMUX\_TrigIn10ToFlexPwm0Force  
enumerator kINPUTMUX\_TrigIn11ToFlexPwm0Force  
enumerator kINPUTMUX\_Gpio0PinEventTrig0ToFlexPwm0Force  
enumerator kINPUTMUX\_Gpio1PinEventTrig0ToFlexPwm0Force  
enumerator kINPUTMUX\_Gpio2PinEventTrig0ToFlexPwm0Force  
enumerator kINPUTMUX\_Gpio3PinEventTrig0ToFlexPwm0Force  
PWM0 external clock trigger.  
enumerator kINPUTMUX\_Clk16K1ToPwm0ExtClk  
enumerator kINPUTMUX\_ClkInToPwm0ExtClk  
enumerator kINPUTMUX\_Aoi0Out0ToPwm0ExtClk  
enumerator kINPUTMUX\_Aoi0Out1ToPwm0ExtClk  
enumerator kINPUTMUX\_ExttrigIn0ToPwm0ExtClk  
enumerator kINPUTMUX\_ExttrigIn7ToPwm0ExtClk  
AOI0 trigger input connections.  
enumerator kINPUTMUX\_Adc0Tcomp0ToAoi0Mux  
enumerator kINPUTMUX\_Adc0Tcomp1ToAoi0Mux  
enumerator kINPUTMUX\_Adc0Tcomp2ToAoi0Mux  
enumerator kINPUTMUX\_Adc0Tcomp3ToAoi0Mux  
enumerator kINPUTMUX\_Cmp0OutToAoi0Mux  
enumerator kINPUTMUX\_Cmp1OutToAoi0Mux  
enumerator kINPUTMUX\_Ctimer0M0ToAoi0Mux

enumerator kINPUTMUX\_\_Ctimer0M1ToAoi0Mux  
enumerator kINPUTMUX\_\_Ctimer0M2ToAoi0Mux  
enumerator kINPUTMUX\_\_Ctimer0M3ToAoi0Mux  
enumerator kINPUTMUX\_\_Ctimer1M0ToAoi0Mux  
enumerator kINPUTMUX\_\_Ctimer1M1ToAoi0Mux  
enumerator kINPUTMUX\_\_Ctimer1M2ToAoi0Mux  
enumerator kINPUTMUX\_\_Ctimer1M3ToAoi0Mux  
enumerator kINPUTMUX\_\_Ctimer2M0ToAoi0Mux  
enumerator kINPUTMUX\_\_Ctimer2M1ToAoi0Mux  
enumerator kINPUTMUX\_\_Ctimer2M2ToAoi0Mux  
enumerator kINPUTMUX\_\_Ctimer2M3ToAoi0Mux  
enumerator kINPUTMUX\_\_Lptmr0ToAoi0Mux  
enumerator kINPUTMUX\_\_Qdc0CmpFlag0ToAoi0Mux  
enumerator kINPUTMUX\_\_Qdc0CmpFlag1ToAoi0Mux  
enumerator kINPUTMUX\_\_Qdc0CmpFlag2ToAoi0Mux  
enumerator kINPUTMUX\_\_Qdc0CmpFlag3ToAoi0Mux  
enumerator kINPUTMUX\_\_Qdc0PosMatchToAoi0Mux  
enumerator kINPUTMUX\_\_Pwm0Sm0OutTrig0ToAoi0Mux  
enumerator kINPUTMUX\_\_Pwm0Sm0OutTrig1ToAoi0Mux  
enumerator kINPUTMUX\_\_Pwm0Sm1OutTrig0ToAoi0Mux  
enumerator kINPUTMUX\_\_Pwm0Sm1OutTrig1ToAoi0Mux  
enumerator kINPUTMUX\_\_Pwm0Sm2OutTrig0ToAoi0Mux  
enumerator kINPUTMUX\_\_Pwm0Sm2OutTrig1ToAoi0Mux  
enumerator kINPUTMUX\_\_TrigIn0ToAoi0Mux  
enumerator kINPUTMUX\_\_TrigIn1ToAoi0Mux  
enumerator kINPUTMUX\_\_TrigIn2ToAoi0Mux  
enumerator kINPUTMUX\_\_TrigIn3ToAoi0Mux  
enumerator kINPUTMUX\_\_TrigIn4ToAoi0Mux  
enumerator kINPUTMUX\_\_TrigIn5ToAoi0Mux  
enumerator kINPUTMUX\_\_TrigIn6ToAoi0Mux  
enumerator kINPUTMUX\_\_TrigIn7ToAoi0Mux  
enumerator kINPUTMUX\_\_TrigIn8ToAoi0Mux  
enumerator kINPUTMUX\_\_TrigIn9ToAoi0Mux

enumerator kINPUTMUX\_\_TrigIn10ToAoi0Mux

enumerator kINPUTMUX\_\_TrigIn11ToAoi0Mux

enumerator kINPUTMUX\_\_Gpio0PinEventTrig0ToAoi0Mux

enumerator kINPUTMUX\_\_Gpio1PinEventTrig0ToAoi0Mux

enumerator kINPUTMUX\_\_Gpio2PinEventTrig0ToAoi0Mux

enumerator kINPUTMUX\_\_Gpio3PinEventTrig0ToAoi0Mux

USB-FS trigger input connections.

enumerator kINPUTMUX\_\_Lpuart0TrgTxdataToUsbfsTrigger

enumerator kINPUTMUX\_\_Lpuart1TrgTxdataToUsbfsTrigger

enumerator kINPUTMUX\_\_Lpuart2TrgTxdataToUsbfsTrigger

EXT trigger connections.

enumerator kINPUTMUX\_\_ArmTxevToExtTrigger

enumerator kINPUTMUX\_\_Aoi0Out0ToExtTrigger

enumerator kINPUTMUX\_\_Aoi0Out1ToExtTrigger

enumerator kINPUTMUX\_\_Aoi0Out2ToExtTrigger

enumerator kINPUTMUX\_\_Aoi0Out3ToExtTrigger

enumerator kINPUTMUX\_\_Cmp0OutToExtTrigger

enumerator kINPUTMUX\_\_Cmp1OutToExtTrigger

enumerator kINPUTMUX\_\_Lpuart0ToExtTrigger

enumerator kINPUTMUX\_\_Lpuart1ToExtTrigger

enumerator kINPUTMUX\_\_Lpuart2ToExtTrigger

LPI2C0 trigger input connections.

enumerator kINPUTMUX\_\_ArmTxevToLpi2c0Trigger

enumerator kINPUTMUX\_\_Aoi0Out0ToLpi2c0Trigger

enumerator kINPUTMUX\_\_Aoi0Out1ToLpi2c0Trigger

enumerator kINPUTMUX\_\_Aoi0Out2ToLpi2c0Trigger

enumerator kINPUTMUX\_\_Aoi0Out3ToLpi2c0Trigger

enumerator kINPUTMUX\_\_Cmp0OutToLpi2c0Trigger

enumerator kINPUTMUX\_\_Cmp1OutToLpi2c0Trigger

enumerator kINPUTMUX\_\_Ctimer0M0ToLpi2c0Trigger

enumerator kINPUTMUX\_\_Ctimer0M1ToLpi2c0Trigger

enumerator kINPUTMUX\_\_Ctimer1M0ToLpi2c0Trigger

enumerator kINPUTMUX\_\_Ctimer1M1ToLpi2c0Trigger

enumerator kINPUTMUX\_\_Ctimer2M0ToLpi2c0Trigger

enumerator kINPUTMUX\_\_Ctimer2M1ToLpi2c0Trigger  
enumerator kINPUTMUX\_\_Lptmr0ToLpi2c0Trigger  
enumerator kINPUTMUX\_\_TrigIn0ToLpi2c0Trigger  
enumerator kINPUTMUX\_\_TrigIn1ToLpi2c0Trigger  
enumerator kINPUTMUX\_\_TrigIn2ToLpi2c0Trigger  
enumerator kINPUTMUX\_\_TrigIn3ToLpi2c0Trigger  
enumerator kINPUTMUX\_\_TrigIn4ToLpi2c0Trigger  
enumerator kINPUTMUX\_\_TrigIn5ToLpi2c0Trigger  
enumerator kINPUTMUX\_\_TrigIn6ToLpi2c0Trigger  
enumerator kINPUTMUX\_\_TrigIn7ToLpi2c0Trigger  
enumerator kINPUTMUX\_\_Gpio0PinEventTrig0ToLpi2c0Trigger  
enumerator kINPUTMUX\_\_Gpio1PinEventTrig0ToLpi2c0Trigger  
enumerator kINPUTMUX\_\_Gpio2PinEventTrig0ToLpi2c0Trigger  
enumerator kINPUTMUX\_\_Gpio3PinEventTrig0ToLpi2c0Trigger

LPSPiO trigger input connections.

enumerator kINPUTMUX\_\_ArmTxevToLpspi0Trigger  
enumerator kINPUTMUX\_\_Aoi0Out0ToLpspi0Trigger  
enumerator kINPUTMUX\_\_Aoi0Out1ToLpspi0Trigger  
enumerator kINPUTMUX\_\_Aoi0Out2ToLpspi0Trigger  
enumerator kINPUTMUX\_\_Aoi0Out3ToLpspi0Trigger  
enumerator kINPUTMUX\_\_Cmp0OutToLpspi0Trigger  
enumerator kINPUTMUX\_\_Cmp1OutToLpspi0Trigger  
enumerator kINPUTMUX\_\_Ctimer0M0ToLpspi0Trigger  
enumerator kINPUTMUX\_\_Ctimer0M1ToLpspi0Trigger  
enumerator kINPUTMUX\_\_Ctimer1M0ToLpspi0Trigger  
enumerator kINPUTMUX\_\_Ctimer1M1ToLpspi0Trigger  
enumerator kINPUTMUX\_\_Ctimer2M0ToLpspi0Trigger  
enumerator kINPUTMUX\_\_Ctimer2M1ToLpspi0Trigger  
enumerator kINPUTMUX\_\_Lptmr0ToLpspi0Trigger  
enumerator kINPUTMUX\_\_TrigIn0ToLpspi0Trigger  
enumerator kINPUTMUX\_\_TrigIn1ToLpspi0Trigger  
enumerator kINPUTMUX\_\_TrigIn2ToLpspi0Trigger  
enumerator kINPUTMUX\_\_TrigIn3ToLpspi0Trigger

enumerator kINPUTMUX\_\_TrigIn4ToLpspi0Trigger  
enumerator kINPUTMUX\_\_TrigIn5ToLpspi0Trigger  
enumerator kINPUTMUX\_\_TrigIn6ToLpspi0Trigger  
enumerator kINPUTMUX\_\_TrigIn7ToLpspi0Trigger  
enumerator kINPUTMUX\_\_Gpio0PinEventTrig0ToLpspi0Trigger  
enumerator kINPUTMUX\_\_Gpio1PinEventTrig0ToLpspi0Trigger  
enumerator kINPUTMUX\_\_Gpio2PinEventTrig0ToLpspi0Trigger  
enumerator kINPUTMUX\_\_Gpio3PinEventTrig0ToLpspi0Trigger

**LPSPI1 trigger input connections.**

enumerator kINPUTMUX\_\_ArmTxevToLpspi1Trigger  
enumerator kINPUTMUX\_\_Aoi0Out0ToLpspi1Trigger  
enumerator kINPUTMUX\_\_Aoi0Out1ToLpspi1Trigger  
enumerator kINPUTMUX\_\_Aoi0Out2ToLpspi1Trigger  
enumerator kINPUTMUX\_\_Aoi0Out3ToLpspi1Trigger  
enumerator kINPUTMUX\_\_Cmp0OutToLpspi1Trigger  
enumerator kINPUTMUX\_\_Cmp1OutToLpspi1Trigger  
enumerator kINPUTMUX\_\_Ctimer0M0ToLpspi1Trigger  
enumerator kINPUTMUX\_\_Ctimer0M1ToLpspi1Trigger  
enumerator kINPUTMUX\_\_Ctimer1M0ToLpspi1Trigger  
enumerator kINPUTMUX\_\_Ctimer1M1ToLpspi1Trigger  
enumerator kINPUTMUX\_\_Ctimer2M0ToLpspi1Trigger  
enumerator kINPUTMUX\_\_Ctimer2M1ToLpspi1Trigger  
enumerator kINPUTMUX\_\_Lptmr0ToLpspi1Trigger  
enumerator kINPUTMUX\_\_TrigIn0ToLpspi1Trigger  
enumerator kINPUTMUX\_\_TrigIn1ToLpspi1Trigger  
enumerator kINPUTMUX\_\_TrigIn2ToLpspi1Trigger  
enumerator kINPUTMUX\_\_TrigIn3ToLpspi1Trigger  
enumerator kINPUTMUX\_\_TrigIn4ToLpspi1Trigger  
enumerator kINPUTMUX\_\_TrigIn5ToLpspi1Trigger  
enumerator kINPUTMUX\_\_TrigIn6ToLpspi1Trigger  
enumerator kINPUTMUX\_\_TrigIn7ToLpspi1Trigger  
enumerator kINPUTMUX\_\_Gpio0PinEventTrig0ToLpspi1Trigger  
enumerator kINPUTMUX\_\_Gpio1PinEventTrig0ToLpspi1Trigger



enumerator kINPUTMUX\_Gpio2PinEventTrig0ToLpspi1Trigger

enumerator kINPUTMUX\_Gpio3PinEventTrig0ToLpspi1Trigger

LPUART0 trigger input connections.

enumerator kINPUTMUX\_ArmTxevToLpuart0Trigger

enumerator kINPUTMUX\_Aoi0Out0ToLpuart0Trigger

enumerator kINPUTMUX\_Aoi0Out1ToLpuart0Trigger

enumerator kINPUTMUX\_Aoi0Out2ToLpuart0Trigger

enumerator kINPUTMUX\_Aoi0Out3ToLpuart0Trigger

enumerator kINPUTMUX\_Cmp0OutToLpuart0Trigger

enumerator kINPUTMUX\_Cmp1OutToLpuart0Trigger

enumerator kINPUTMUX\_Ctimer0M2ToLpuart0Trigger

enumerator kINPUTMUX\_Ctimer0M3ToLpuart0Trigger

enumerator kINPUTMUX\_Ctimer1M2ToLpuart0Trigger

enumerator kINPUTMUX\_Ctimer1M3ToLpuart0Trigger

enumerator kINPUTMUX\_Ctimer2M2ToLpuart0Trigger

enumerator kINPUTMUX\_Ctimer2M3ToLpuart0Trigger

enumerator kINPUTMUX\_Lptmr0ToLpuart0Trigger

enumerator kINPUTMUX\_TrigIn0ToLpuart0Trigger

enumerator kINPUTMUX\_TrigIn1ToLpuart0Trigger

enumerator kINPUTMUX\_TrigIn2ToLpuart0Trigger

enumerator kINPUTMUX\_TrigIn3ToLpuart0Trigger

enumerator kINPUTMUX\_TrigIn4ToLpuart0Trigger

enumerator kINPUTMUX\_TrigIn5ToLpuart0Trigger

enumerator kINPUTMUX\_TrigIn6ToLpuart0Trigger

enumerator kINPUTMUX\_TrigIn7ToLpuart0Trigger

enumerator kINPUTMUX\_TrigIn8ToLpuart0Trigger

enumerator kINPUTMUX\_TrigIn9ToLpuart0Trigger

enumerator kINPUTMUX\_TrigIn10ToLpuart0Trigger

enumerator kINPUTMUX\_TrigIn11ToLpuart0Trigger

enumerator kINPUTMUX\_Gpio0PinEventTrig0ToLpuart0Trigger

enumerator kINPUTMUX\_Gpio1PinEventTrig0ToLpuart0Trigger

enumerator kINPUTMUX\_Gpio2PinEventTrig0ToLpuart0Trigger

enumerator kINPUTMUX\_Gpio3PinEventTrig0ToLpuart0Trigger

enumerator kINPUTMUX\_\_WuuToLpuart0Trigger

enumerator kINPUTMUX\_\_Usb0IppIndUartRxdUsbmuxToLpuart0Trigger

LPUART1 trigger input connections.

enumerator kINPUTMUX\_\_ArmTxevToLpuart1Trigger

enumerator kINPUTMUX\_\_Aoi0Out0ToLpuart1Trigger

enumerator kINPUTMUX\_\_Aoi0Out1ToLpuart1Trigger

enumerator kINPUTMUX\_\_Aoi0Out2ToLpuart1Trigger

enumerator kINPUTMUX\_\_Aoi0Out3ToLpuart1Trigger

enumerator kINPUTMUX\_\_Cmp0OutToLpuart1Trigger

enumerator kINPUTMUX\_\_Cmp1OutToLpuart1Trigger

enumerator kINPUTMUX\_\_Ctimer0M2ToLpuart1Trigger

enumerator kINPUTMUX\_\_Ctimer0M3ToLpuart1Trigger

enumerator kINPUTMUX\_\_Ctimer1M2ToLpuart1Trigger

enumerator kINPUTMUX\_\_Ctimer1M3ToLpuart1Trigger

enumerator kINPUTMUX\_\_Ctimer2M2ToLpuart1Trigger

enumerator kINPUTMUX\_\_Ctimer2M3ToLpuart1Trigger

enumerator kINPUTMUX\_\_Lptmr0ToLpuart1Trigger

enumerator kINPUTMUX\_\_TrigIn0ToLpuart1Trigger

enumerator kINPUTMUX\_\_TrigIn1ToLpuart1Trigger

enumerator kINPUTMUX\_\_TrigIn2ToLpuart1Trigger

enumerator kINPUTMUX\_\_TrigIn3ToLpuart1Trigger

enumerator kINPUTMUX\_\_TrigIn4ToLpuart1Trigger

enumerator kINPUTMUX\_\_TrigIn5ToLpuart1Trigger

enumerator kINPUTMUX\_\_TrigIn6ToLpuart1Trigger

enumerator kINPUTMUX\_\_TrigIn7ToLpuart1Trigger

enumerator kINPUTMUX\_\_TrigIn8ToLpuart1Trigger

enumerator kINPUTMUX\_\_TrigIn9ToLpuart1Trigger

enumerator kINPUTMUX\_\_TrigIn10ToLpuart1Trigger

enumerator kINPUTMUX\_\_TrigIn11ToLpuart1Trigger

enumerator kINPUTMUX\_\_Gpio0PinEventTrig0ToLpuart1Trigger

enumerator kINPUTMUX\_\_Gpio1PinEventTrig0ToLpuart1Trigger

enumerator kINPUTMUX\_\_Gpio2PinEventTrig0ToLpuart1Trigger

enumerator kINPUTMUX\_\_Gpio3PinEventTrig0ToLpuart1Trigger

enumerator kINPUTMUX\_\_WuuToLpuart1Trigger

enumerator kINPUTMUX\_\_Usb0IppIndUartRxdUsbmuxToLpuart1Trigger

LPUART2 trigger input connections.

enumerator kINPUTMUX\_\_ArmTxevToLpuart2Trigger

enumerator kINPUTMUX\_\_Aoi0Out0ToLpuart2Trigger

enumerator kINPUTMUX\_\_Aoi0Out1ToLpuart2Trigger

enumerator kINPUTMUX\_\_Aoi0Out2ToLpuart2Trigger

enumerator kINPUTMUX\_\_Aoi0Out3ToLpuart2Trigger

enumerator kINPUTMUX\_\_Cmp0OutToLpuart2Trigger

enumerator kINPUTMUX\_\_Cmp1OutToLpuart2Trigger

enumerator kINPUTMUX\_\_Ctimer0M2ToLpuart2Trigger

enumerator kINPUTMUX\_\_Ctimer0M3ToLpuart2Trigger

enumerator kINPUTMUX\_\_Ctimer1M2ToLpuart2Trigger

enumerator kINPUTMUX\_\_Ctimer1M3ToLpuart2Trigger

enumerator kINPUTMUX\_\_Ctimer2M2ToLpuart2Trigger

enumerator kINPUTMUX\_\_Ctimer2M3ToLpuart2Trigger

enumerator kINPUTMUX\_\_Lptmr0ToLpuart2Trigger

enumerator kINPUTMUX\_\_TrigIn0ToLpuart2Trigger

enumerator kINPUTMUX\_\_TrigIn1ToLpuart2Trigger

enumerator kINPUTMUX\_\_TrigIn2ToLpuart2Trigger

enumerator kINPUTMUX\_\_TrigIn3ToLpuart2Trigger

enumerator kINPUTMUX\_\_TrigIn4ToLpuart2Trigger

enumerator kINPUTMUX\_\_TrigIn5ToLpuart2Trigger

enumerator kINPUTMUX\_\_TrigIn6ToLpuart2Trigger

enumerator kINPUTMUX\_\_TrigIn7ToLpuart2Trigger

enumerator kINPUTMUX\_\_TrigIn8ToLpuart2Trigger

enumerator kINPUTMUX\_\_TrigIn9ToLpuart2Trigger

enumerator kINPUTMUX\_\_TrigIn10ToLpuart2Trigger

enumerator kINPUTMUX\_\_TrigIn11ToLpuart2Trigger

enumerator kINPUTMUX\_\_Gpio0PinEventTrig0ToLpuart2Trigger

enumerator kINPUTMUX\_\_Gpio1PinEventTrig0ToLpuart2Trigger

enumerator kINPUTMUX\_\_Gpio2PinEventTrig0ToLpuart2Trigger

enumerator kINPUTMUX\_\_Gpio3PinEventTrig0ToLpuart2Trigger

```
enumerator kINPUTMUX_WuuToLpuart2Trigger
enumerator kINPUTMUX_Usb0IppIndUartRxdUsbmuxToLpuart2Trigger
typedef enum _inputmux_index_t inputmux_index_t
typedef enum _inputmux_connection_t inputmux_connection_t
    INPUTMUX connections type.
TIMER0CAPTSEL0
    Periphinmux IDs.
TIMER0TRIGIN
TIMER1CAPTSEL0
TIMER1TRIGIN
TIMER2CAPTSEL0
TIMER2TRIGIN
FREQMEAS_REF_REG
FREQMEAS_TAR_REG
CMP0_TRIG_REG
ADC0_TRIG0_REG
QDC0_TRIG_REG
QDC0_HOME_REG
QDC0_INDEX_REG
QDC0_PHASEB_REG
QDC0_PHASEA_REG
QDC0_ICAP0_REG
FlexPWM0_SM0_EXT_A0_REG
FlexPWM0_SM0_EXTSYNC0_REG
FlexPWM0_SM1_EXT_A1_REG
FlexPWM0_SM1_EXTSYNC1_REG
FlexPWM0_SM2_EXT_A2_REG
FlexPWM0_SM2_EXTSYNC2_REG
FlexPWM0_FAULT_REG
FlexPWM0_FORCE_REG
PWM0_EXT_CLK_REG
AOI0_MUX_REG
USBFS_TRIG_REG
EXT_TRIG0_REG
```

CMP1\_TRIG\_REG

LPI2C0\_TRIG\_REG

LPSPi0\_TRIG\_REG

LPSPi1\_TRIG\_REG

LPUART0\_TRIG\_REG

LPUART1\_TRIG\_REG

LPUART2\_TRIG\_REG

PMUX\_SHIFT

FSL\_INPUTMUX\_DRIVER\_VERSION

Group interrupt driver version for SDK.

void INPUTMUX\_Init(void \*base)

Initialize INPUTMUX peripheral.

This function enables the INPUTMUX clock.

#### Parameters

- base – Base address of the INPUTMUX peripheral.

#### Return values

None. –

void INPUTMUX\_AttachSignal(void \*base, uint16\_t index, *inputmux\_connection\_t* connection)

Attaches a signal.

This function attaches multiplexed signals from INPUTMUX to target signals. For example, to attach GPIO PORT0 Pin 5 to PINT peripheral, do the following:

```
INPUTMUX_AttachSignal(INPUTMUX, 2, kINPUTMUX_GpioPort0Pin5ToPintsel);
```

In this example, INTMUX has 8 registers for PINT, PINT\_SEL0~PINT\_SEL7. With parameter index specified as 2, this function configures register PINT\_SEL2.

#### Parameters

- base – Base address of the INPUTMUX peripheral.
- index – The serial number of destination register in the group of INPUTMUX registers with same name.
- connection – Applies signal from source signals collection to target signal.

#### Return values

None. –

void INPUTMUX\_Deinit(void \*base)

Deinitialize INPUTMUX peripheral.

This function disables the INPUTMUX clock.

#### Parameters

- base – Base address of the INPUTMUX peripheral.

#### Return values

None. –

## 2.25 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.

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 specified by *bits* to the variable at address *address*.

SDK\_ATOMIC\_LOCAL\_CLEAR(*addr*, *bits*)

Clear the bits specified by *bits* to the variable at address *address*.

SDK\_ATOMIC\_LOCAL\_TOGGLE(addr, bits)

Toggle the bits specified 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 specified by *clearBits* and set the bits specified 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

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*.

`RAMFUNCTION_SECTION_CODE(func)`

Place function in ram.

`enum __status_groups`

Status group numbers.

*Values:*

enumerator `kStatusGroup_Generic`

Group number for generic status codes.

enumerator `kStatusGroup_FLASH`

Group number for FLASH status codes.

enumerator `kStatusGroup_LPSPI`

Group number for LPSPI status codes.

enumerator `kStatusGroup_FLEXIO_SPI`

Group number for FLEXIO SPI status codes.

enumerator `kStatusGroup_DSPI`

Group number for DSPI status codes.

enumerator `kStatusGroup_FLEXIO_UART`

Group number for FLEXIO UART status codes.

enumerator `kStatusGroup_FLEXIO_I2C`

Group number for FLEXIO I2C status codes.

enumerator `kStatusGroup_LPI2C`

Group number for LPI2C status codes.

enumerator `kStatusGroup_UART`

Group number for UART status codes.

enumerator `kStatusGroup_I2C`

Group number for I2C 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_CODEEC`  
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.

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.

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

SUPPRESS\_FALL\_THROUGH\_WARNING()

For switch case code block, if case section ends without “break;” statement, there will 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.

## 2.26 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

#### 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 timeout (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

#### **Returns**

`void`

`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`

## 2.27 LPADC: 12-bit SAR Analog-to-Digital Converter Driver

enum `_lpadc_status_flags`

Define hardware flags of the module.

*Values:*

enumerator `kLPADC_ResultFIFO0OverflowFlag`

Indicates that more data has been written to the Result FIFO 0 than it can hold.

enumerator `kLPADC_ResultFIFO0ReadyFlag`

Indicates when the number of valid datawords in the result FIFO 0 is greater than the setting watermark level.

enumerator `kLPADC_TriggerExceptionFlag`

Indicates that a trigger exception event has occurred.

enumerator `kLPADC_TriggerCompletionFlag`

Indicates that a trigger completion event has occurred.

enumerator `kLPADC_CalibrationReadyFlag`

Indicates that the calibration process is done.

enumerator `kLPADC_ActiveFlag`

Indicates that the ADC is in active state.

enumerator `kLPADC_ResultFIFOOverflowFlag`

To compilitable with old version, do not recommend using this, please use `kLPADC_ResultFIFO0OverflowFlag` as instead.

enumerator `kLPADC_ResultFIFOReadyFlag`

To compilitable with old version, do not recommend using this, please use `kLPADC_ResultFIFO0ReadyFlag` as instead.

enum `_lpadc_interrupt_enable`

Define interrupt switchers of the module.

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

*Values:*

enumerator `kLPADC_ResultFIFO0OverflowInterruptEnable`

Configures ADC to generate overflow interrupt requests when FOF0 flag is asserted.

enumerator `kLPADC_FIFO0WatermarkInterruptEnable`

Configures ADC to generate watermark interrupt requests when RDY0 flag is asserted.

enumerator `kLPADC_ResultFIFOOverflowInterruptEnable`

To compilitable with old version, do not recommend using this, please use `kLPADC_ResultFIFO0OverflowInterruptEnable` as instead.

enumerator `kLPADC_FIFOWatermarkInterruptEnable`

To compilitable with old version, do not recommend using this, please use `kLPADC_FIFO0WatermarkInterruptEnable` as instead.

enumerator `kLPADC_TriggerExceptionInterruptEnable`

Configures ADC to generate trigger exception interrupt.

enumerator `kLPADC_Trigger0CompletionInterruptEnable`

Configures ADC to generate interrupt when trigger 0 completion.

enumerator kLPADC\_Trigger1CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 1 completion.

enumerator kLPADC\_Trigger2CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 2 completion.

enumerator kLPADC\_Trigger3CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 3 completion.

enumerator kLPADC\_Trigger4CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 4 completion.

enumerator kLPADC\_Trigger5CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 5 completion.

enumerator kLPADC\_Trigger6CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 6 completion.

enumerator kLPADC\_Trigger7CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 7 completion.

enumerator kLPADC\_Trigger8CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 8 completion.

enumerator kLPADC\_Trigger9CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 9 completion.

enumerator kLPADC\_Trigger10CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 10 completion.

enumerator kLPADC\_Trigger11CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 11 completion.

enumerator kLPADC\_Trigger12CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 12 completion.

enumerator kLPADC\_Trigger13CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 13 completion.

enumerator kLPADC\_Trigger14CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 14 completion.

enumerator kLPADC\_Trigger15CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 15 completion.

enum \_\_lpadc\_trigger\_status\_flags

The enumerator of lpadc trigger status flags, including interrupted flags and completed flags.

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

*Values:*

enumerator kLPADC\_Trigger0InterruptedFlag

Trigger 0 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger1InterruptedFlag

Trigger 1 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger2InterruptedFlag

Trigger 2 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger3InterruptedFlag

Trigger 3 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger4InterruptedFlag

Trigger 4 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger5InterruptedFlag

Trigger 5 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger6InterruptedFlag

Trigger 6 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger7InterruptedFlag

Trigger 7 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger8InterruptedFlag

Trigger 8 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger9InterruptedFlag

Trigger 9 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger10InterruptedFlag

Trigger 10 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger11InterruptedFlag

Trigger 11 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger12InterruptedFlag

Trigger 12 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger13InterruptedFlag

Trigger 13 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger14InterruptedFlag

Trigger 14 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger15InterruptedFlag

Trigger 15 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger0CompletedFlag

Trigger 0 is completed and trigger 0 has enabled completion interrupts.

enumerator kLPADC\_Trigger1CompletedFlag

Trigger 1 is completed and trigger 1 has enabled completion interrupts.

enumerator kLPADC\_Trigger2CompletedFlag

Trigger 2 is completed and trigger 2 has enabled completion interrupts.

enumerator kLPADC\_Trigger3CompletedFlag

Trigger 3 is completed and trigger 3 has enabled completion interrupts.

enumerator kLPADC\_Trigger4CompletedFlag

Trigger 4 is completed and trigger 4 has enabled completion interrupts.

enumerator kLPADC\_Trigger5CompletedFlag

Trigger 5 is completed and trigger 5 has enabled completion interrupts.

enumerator kLPADC\_Trigger6CompletedFlag

Trigger 6 is completed and trigger 6 has enabled completion interrupts.

enumerator kLPADC\_Trigger7CompletedFlag

Trigger 7 is completed and trigger 7 has enabled completion interrupts.

enumerator kLPADC\_Trigger8CompletedFlag

Trigger 8 is completed and trigger 8 has enabled completion interrupts.

enumerator kLPADC\_Trigger9CompletedFlag

Trigger 9 is completed and trigger 9 has enabled completion interrupts.

enumerator kLPADC\_Trigger10CompletedFlag

Trigger 10 is completed and trigger 10 has enabled completion interrupts.

enumerator kLPADC\_Trigger11CompletedFlag

Trigger 11 is completed and trigger 11 has enabled completion interrupts.

enumerator kLPADC\_Trigger12CompletedFlag

Trigger 12 is completed and trigger 12 has enabled completion interrupts.

enumerator kLPADC\_Trigger13CompletedFlag

Trigger 13 is completed and trigger 13 has enabled completion interrupts.

enumerator kLPADC\_Trigger14CompletedFlag

Trigger 14 is completed and trigger 14 has enabled completion interrupts.

enumerator kLPADC\_Trigger15CompletedFlag

Trigger 15 is completed and trigger 15 has enabled completion interrupts.

enum \_lpadc\_sample\_scale\_mode

Define enumeration of sample scale mode.

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

*Values:*

enumerator kLPADC\_SamplePartScale

Use divided input voltage signal. (For scale select, please refer to the reference manual).

enumerator kLPADC\_SampleFullScale

Full scale (Factor of 1).

enum \_lpadc\_sample\_channel\_mode

Define enumeration of channel sample mode.

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

*Values:*

enumerator kLPADC\_SampleChannelSingleEndSideA

Single-end mode, only A-side channel is converted.

enumerator kLPADC\_SampleChannelSingleEndSideB

Single-end mode, only B-side channel is converted.

enumerator kLPADC\_SampleChannelDiffBothSideAB

Differential mode, the ADC result is (CHnA-CHnB).

enumerator kLPADC\_SampleChannelDiffBothSideBA

Differential mode, the ADC result is (CHnB-CHnA).

enumerator kLPADC\_SampleChannelDiffBothSide

Differential mode, the ADC result is (CHnA-CHnB).

enumerator kLPADC\_SampleChannelDualSingleEndBothSide

Dual-Single-Ended Mode. Both A side and B side channels are converted independently.

enum \_lpadc\_hardware\_average\_mode

Define enumeration of hardware average selection.

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

---

**Note:** Some enumerator values are not available on some devices, mainly depends on the size of AVGS field in CMDH register.

---

*Values:*

enumerator kLPADC\_HardwareAverageCount1

Single conversion.

enumerator kLPADC\_HardwareAverageCount2

2 conversions averaged.

enumerator kLPADC\_HardwareAverageCount4

4 conversions averaged.

enumerator kLPADC\_HardwareAverageCount8

8 conversions averaged.

enumerator kLPADC\_HardwareAverageCount16

16 conversions averaged.

enumerator kLPADC\_HardwareAverageCount32

32 conversions averaged.

enumerator kLPADC\_HardwareAverageCount64

64 conversions averaged.

enumerator kLPADC\_HardwareAverageCount128

128 conversions averaged.

enum \_lpadc\_sample\_time\_mode

Define enumeration of sample time selection.

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

*Values:*

enumerator kLPADC\_SampleTimeADCK3

3 ADCK cycles total sample time.

enumerator kLPADC\_SampleTimeADCK5

5 ADCK cycles total sample time.

enumerator kLPADC\_SampleTimeADCK7

7 ADCK cycles total sample time.

enumerator kLPADC\_SampleTimeADCK11

11 ADCK cycles total sample time.

enumerator kLPADC\_SampleTimeADCK19  
19 ADCK cycles total sample time.

enumerator kLPADC\_SampleTimeADCK35  
35 ADCK cycles total sample time.

enumerator kLPADC\_SampleTimeADCK67  
69 ADCK cycles total sample time.

enumerator kLPADC\_SampleTimeADCK131  
131 ADCK cycles total sample time.

enum \_lpadc\_hardware\_compare\_mode

Define enumeration of hardware compare mode.

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

*Values:*

enumerator kLPADC\_HardwareCompareDisabled  
Compare disabled.

enumerator kLPADC\_HardwareCompareStoreOnTrue  
Compare enabled. Store on true.

enumerator kLPADC\_HardwareCompareRepeatUntilTrue  
Compare enabled. Repeat channel acquisition until true.

enum \_lpadc\_conversion\_resolution\_mode

Define enumeration of conversion resolution mode.

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to lpadc\_sample\_channel\_mode\_t

*Values:*

enumerator kLPADC\_ConversionResolutionStandard  
Standard resolution. Single-ended 12-bit conversion, Differential 13-bit conversion with 2's complement output.

enumerator kLPADC\_ConversionResolutionHigh  
High resolution. Single-ended 16-bit conversion; Differential 16-bit conversion with 2's complement output.

enum \_lpadc\_conversion\_average\_mode

Define enumeration of conversion averages mode.

Configure the conversion average number for auto-calibration.

---

**Note:** Some enumerator values are not available on some devices, mainly depends on the size of CAL\_AVGS field in CTRL register.

---

*Values:*

enumerator kLPADC\_ConversionAverage1  
Single conversion.

enumerator kLPADC\_ConversionAverage2  
2 conversions averaged.

enumerator kLPADC\_ConversionAverage4

4 conversions averaged.

enumerator kLPADC\_ConversionAverage8

8 conversions averaged.

enumerator kLPADC\_ConversionAverage16

16 conversions averaged.

enumerator kLPADC\_ConversionAverage32

32 conversions averaged.

enumerator kLPADC\_ConversionAverage64

64 conversions averaged.

enumerator kLPADC\_ConversionAverage128

128 conversions averaged.

enumerator kLPADC\_ConversionAverageMax

enum \_lpadc\_reference\_voltage\_mode

Define enumeration of reference voltage source.

For detail information, need to check the SoC's specification.

*Values:*

enumerator kLPADC\_ReferenceVoltageAlt1

Option 1 setting.

enumerator kLPADC\_ReferenceVoltageAlt2

Option 2 setting.

enumerator kLPADC\_ReferenceVoltageAlt3

Option 3 setting.

enum \_lpadc\_power\_level\_mode

Define enumeration of power configuration.

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

*Values:*

enumerator kLPADC\_PowerLevelAlt1

Lowest power setting.

enumerator kLPADC\_PowerLevelAlt2

Next lowest power setting.

enumerator kLPADC\_PowerLevelAlt3

...

enumerator kLPADC\_PowerLevelAlt4

Highest power setting.

enum \_lpadc\_offset\_calibration\_mode

Define enumeration of offset calibration mode.

*Values:*

enumerator kLPADC\_OffsetCalibration12bitMode

12 bit offset calibration mode.



enumerator kLPADC\_OffsetCalibration16bitMode  
16 bit offset calibration mode.

enum \_\_lpadc\_trigger\_priority\_policy

Define enumeration of trigger priority policy.

This selection controls how higher priority triggers are handled.

---

**Note:** **kLPADC\_TriggerPriorityPreemptSubsequently** is not available on some devices, mainly depends on the size of TPRICTRL field in CFG register.

---

*Values:*

enumerator kLPADC\_ConvPreemptImmediatelyNotAutoResumed

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion is not automatically resumed or restarted.

enumerator kLPADC\_ConvPreemptSoftlyNotAutoResumed

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion is not resumed or restarted.

enumerator kLPADC\_ConvPreemptImmediatelyAutoRestarted

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

enumerator kLPADC\_ConvPreemptSoftlyAutoRestarted

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

enumerator kLPADC\_ConvPreemptImmediatelyAutoResumed

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be resumed.

enumerator kLPADC\_ConvPreemptSoftlyAutoResumed

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will be automatically be resumed.

enumerator kLPADC\_TriggerPriorityPreemptImmediately

Legacy support is not recommended as it only ensures compatibility with older versions.

enumerator kLPADC\_TriggerPriorityPreemptSoftly

Legacy support is not recommended as it only ensures compatibility with older versions.

enumerator kLPADC\_TriggerPriorityExceptionDisabled

High priority trigger exception disabled.

enum \_lpadc\_tune\_value

Define enumeration of tune value.

*Values:*

enumerator kLPADC\_TuneValue0

Tune value 0.

enumerator kLPADC\_TuneValue1

Tune value 1.

enumerator kLPADC\_TuneValue2

Tune value 2.

enumerator kLPADC\_TuneValue3

Tune value 3.

typedef enum \_lpadc\_sample\_scale\_mode lpadc\_sample\_scale\_mode\_t

Define enumeration of sample scale mode.

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

typedef enum \_lpadc\_sample\_channel\_mode lpadc\_sample\_channel\_mode\_t

Define enumeration of channel sample mode.

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

typedef enum \_lpadc\_hardware\_average\_mode lpadc\_hardware\_average\_mode\_t

Define enumeration of hardware average selection.

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

---

**Note:** Some enumerator values are not available on some devices, mainly depends on the size of AVGS field in CMDH register.

---

typedef enum \_lpadc\_sample\_time\_mode lpadc\_sample\_time\_mode\_t

Define enumeration of sample time selection.

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

typedef enum \_lpadc\_hardware\_compare\_mode lpadc\_hardware\_compare\_mode\_t

Define enumeration of hardware compare mode.

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

`typedef enum _lpadc_conversion_resolution_mode` `lpadc_conversion_resolution_mode_t`

Define enumeration of conversion resolution mode.

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to `lpadc_sample_channel_mode_t`

`typedef enum _lpadc_conversion_average_mode` `lpadc_conversion_average_mode_t`

Define enumeration of conversion averages mode.

Configure the conversion average number for auto-calibration.

---

**Note:** Some enumerator values are not available on some devices, mainly depends on the size of `CAL_AVGS` field in `CTRL` register.

---

`typedef enum _lpadc_reference_voltage_mode` `lpadc_reference_voltage_source_t`

Define enumeration of reference voltage source.

For detail information, need to check the SoC's specification.

`typedef enum _lpadc_power_level_mode` `lpadc_power_level_mode_t`

Define enumeration of power configuration.

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

`typedef enum _lpadc_offset_calibration_mode` `lpadc_offset_calibration_mode_t`

Define enumeration of offset calibration mode.

`typedef enum _lpadc_trigger_priority_policy` `lpadc_trigger_priority_policy_t`

Define enumeration of trigger priority policy.

This selection controls how higher priority triggers are handled.

---

**Note:** `kLPADC_TriggerPriorityPreemptSubsequently` is not available on some devices, mainly depends on the size of `TPRCTRL` field in `CFG` register.

---

`typedef enum _lpadc_tune_value` `lpadc_tune_value_t`

Define enumeration of tune value.

`typedef struct _lpadc_calibration_value` `lpadc_calibration_value_t`

A structure of calibration value.

`LPADC_CONVERSION_COMPLETE_TIMEOUT`

Max loops to wait for LPADC conversion complete.

When doing calibration, driver will wait for the completion of conversion. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

`LPADC_CALIBRATION_READY_TIMEOUT`

Max loops to wait for LPADC calibration ready.

Before doing calibration, driver will wait for the calibration ready. This parameter defines how many loops to check the calibration ready. If defined as 0, driver will wait forever until ready.

`LPADC_GAIN_CAL_READY_TIMEOUT`

Max loops to wait for LPADC gain calibration `GAIN_CAL` ready.

Before doing calibration, driver will wait for the gain calibration GAIN\_CAL ready. This parameter defines how many loops to check the gain calibration GAIN\_CAL ready. If defined as 0, driver will wait forever until ready.

ADC\_OFSTRIM\_OFSTRIM\_MAX

ADC\_OFSTRIM\_OFSTRIM\_SIGN

LPADC\_GET\_ACTIVE\_COMMAND\_STATUS(statusVal)

Define the MACRO function to get command status from status value.

The statusVal is the return value from LPADC\_GetStatusFlags().

LPADC\_GET\_ACTIVE\_TRIGGER\_STATUE(statusVal)

Define the MACRO function to get trigger status from status value.

The statusVal is the return value from LPADC\_GetStatusFlags().

void LPADC\_Init(ADC\_Type \*base, const *lpadc\_config\_t* \*config)

Initializes the LPADC module.

#### Parameters

- base – LPADC peripheral base address.
- config – Pointer to configuration structure. See “*lpadc\_config\_t*”.

void LPADC\_GetDefaultConfig(*lpadc\_config\_t* \*config)

Gets an available pre-defined settings for initial configuration.

This function initializes the converter configuration structure with an available settings. The default values are:

```
config->enableInDozeMode      = true;
config->enableAnalogPreliminary = false;
config->powerUpDelay           = 0x80;
config->referenceVoltageSource = kLPADC_ReferenceVoltageAlt1;
config->powerLevelMode         = kLPADC_PowerLevelAlt1;
config->triggerPriorityPolicy   = kLPADC_TriggerPriorityPreemptImmediately;
config->enableConvPause        = false;
config->convPauseDelay          = 0U;
config->FIFOWatermark           = 0U;
```

#### Parameters

- config – Pointer to configuration structure.

void LPADC\_Deinit(ADC\_Type \*base)

De-initializes the LPADC module.

#### Parameters

- base – LPADC peripheral base address.

static inline void LPADC\_Enable(ADC\_Type \*base, bool enable)

Switch on/off the LPADC module.

#### Parameters

- base – LPADC peripheral base address.
- enable – switcher to the module.

static inline void LPADC\_DoResetFIFO(ADC\_Type \*base)

Do reset the conversion FIFO.

#### Parameters

- base – LPADC peripheral base address.

static inline void LPADC\_DoResetConfig(ADC\_Type \*base)

Do reset the module's configuration.

Reset all ADC internal logic and registers, except the Control Register (ADCx\_CTRL).

#### Parameters

- base – LPADC peripheral base address.

static inline uint32\_t LPADC\_GetStatusFlags(ADC\_Type \*base)

Get status flags.

#### Parameters

- base – LPADC peripheral base address.

#### Returns

status flags' mask. See to `_lpadc_status_flags`.

static inline void LPADC\_ClearStatusFlags(ADC\_Type \*base, uint32\_t mask)

Clear status flags.

Only the flags can be cleared by writing ADCx\_STATUS register would be cleared by this API.

#### Parameters

- base – LPADC peripheral base address.
- mask – Mask value for flags to be cleared. See to `_lpadc_status_flags`.

static inline uint32\_t LPADC\_GetTriggerStatusFlags(ADC\_Type \*base)

Get trigger status flags to indicate which trigger sequences have been completed or interrupted by a high priority trigger exception.

#### Parameters

- base – LPADC peripheral base address.

#### Returns

The OR'ed value of `_lpadc_trigger_status_flags`.

static inline void LPADC\_ClearTriggerStatusFlags(ADC\_Type \*base, uint32\_t mask)

Clear trigger status flags.

#### Parameters

- base – LPADC peripheral base address.
- mask – The mask of trigger status flags to be cleared, should be the OR'ed value of `_lpadc_trigger_status_flags`.

static inline void LPADC\_EnableInterrupts(ADC\_Type \*base, uint32\_t mask)

Enable interrupts.

#### Parameters

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to `_lpadc_interrupt_enable`.

static inline void LPADC\_DisableInterrupts(ADC\_Type \*base, uint32\_t mask)

Disable interrupts.

#### Parameters

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to `_lpadc_interrupt_enable`.

```
static inline void LPADC_EnableFIFOWatermarkDMA(ADC_Type *base, bool enable)
```

Switch on/off the DMA trigger for FIFO watermark event.

#### Parameters

- base – LPADC peripheral base address.
- enable – Switcher to the event.

```
static inline uint32_t LPADC_GetConvResultCount(ADC_Type *base)
```

Get the count of result kept in conversion FIFO.

#### Parameters

- base – LPADC peripheral base address.

#### Returns

The count of result kept in conversion FIFO.

```
bool LPADC_GetConvResult(ADC_Type *base, lpadc_conv_result_t *result)
```

Get the result in conversion FIFO.

#### Parameters

- base – LPADC peripheral base address.
- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

#### Returns

Status whether FIFO entry is valid.

```
void LPADC_GetConvResultBlocking(ADC_Type *base, lpadc_conv_result_t *result)
```

Get the result in conversion FIFO using blocking method.

#### Parameters

- base – LPADC peripheral base address.
- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

```
void LPADC_SetConvTriggerConfig(ADC_Type *base, uint32_t triggerId, const  
                                lpadc_conv_trigger_config_t *config)
```

Configure the conversion trigger source.

Each programmable trigger can launch the conversion command in command buffer.

#### Parameters

- base – LPADC peripheral base address.
- triggerId – ID for each trigger. Typically, the available value range is from 0.
- config – Pointer to configuration structure. See to `lpadc_conv_trigger_config_t`.

```
void LPADC_GetDefaultConvTriggerConfig(lpadc_conv_trigger_config_t *config)
```

Gets an available pre-defined settings for trigger's configuration.

This function initializes the trigger's configuration structure with an available settings. The default values are:

```
config->targetCommandId    = 0U;  
config->delayPower         = 0U;  
config->priority           = 0U;  
config->channelAFIFOSelect = 0U;
```

(continues on next page)

(continued from previous page)

```
config->channelBFIFOSelect    = 0U;
config->enableHardwareTrigger = false;
```

### Parameters

- config – Pointer to configuration structure.

static inline void LPADC\_DoSoftwareTrigger(ADC\_Type \*base, uint32\_t triggerIdMask)

Do software trigger to conversion command.

### Parameters

- base – LPADC peripheral base address.
- triggerIdMask – Mask value for software trigger indexes, which count from zero.

void LPADC\_SetConvCommandConfig(ADC\_Type \*base, uint32\_t commandId, const *lpadc\_conv\_command\_config\_t* \*config)

Configure conversion command.

---

**Note:** The number of compare value register on different chips is different, that is mean in some chips, some command buffers do not have the compare functionality.

---

### Parameters

- base – LPADC peripheral base address.
- commandId – ID for command in command buffer. Typically, the available value range is 1 - 15.
- config – Pointer to configuration structure. See to *lpadc\_conv\_command\_config\_t*.

void LPADC\_GetDefaultConvCommandConfig(*lpadc\_conv\_command\_config\_t* \*config)

Gets an available pre-defined settings for conversion command's configuration.

This function initializes the conversion command's configuration structure with an available settings. The default values are:

```
config->sampleScaleMode      = kLPADC_SampleFullScale;
config->channelBScaleMode    = kLPADC_SampleFullScale;
config->sampleChannelMode    = kLPADC_SampleChannelSingleEndSideA;
config->channelNumber        = 0U;
config->channelBNumber       = 0U;
config->chainedNextCommandNumber = 0U;
config->enableAutoChannelIncrement = false;
config->loopCount            = 0U;
config->hardwareAverageMode   = kLPADC_HardwareAverageCount1;
config->sampleTimeMode       = kLPADC_SampleTimeADCK3;
config->hardwareCompareMode   = kLPADC_HardwareCompareDisabled;
config->hardwareCompareValueHigh = 0U;
config->hardwareCompareValueLow  = 0U;
config->conversionResolutionMode = kLPADC_ConversionResolutionStandard;
config->enableWaitTrigger     = false;
config->enableChannelB       = false;
```

### Parameters

- config – Pointer to configuration structure.

`void LPADC_EnableCalibration(ADC_Type *base, bool enable)`

Enable the calibration function.

When CALOFS is set, the ADC is configured to perform a calibration function anytime the ADC executes a conversion. Any channel selected is ignored and the value returned in the RESFIFO is a signed value between -31 and 31. -32 is not a valid and is never a returned value. Software should copy the lower 6-bits of the conversion result stored in the RESFIFO after a completed calibration conversion to the OFSTRIM field. The OFSTRIM field is used in normal operation for offset correction.

#### Parameters

- base – LPADC peripheral base address.
- enable – switcher to the calibration function.

`static inline void LPADC_SetOffsetValue(ADC_Type *base, uint32_t value)`

Set proper offset value to trim ADC.

To minimize the offset during normal operation, software should read the conversion result from the RESFIFO calibration operation and write the lower 6 bits to the OFSTRIM register.

#### Parameters

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

`status_t LPADC_DoAutoCalibration(ADC_Type *base)`

Do auto calibration.

Calibration function should be executed before using converter in application. It used the software trigger and a dummy conversion, get the offset and write them into the OFSTRIM register. It called some of functional API including: -LPADC\_EnableCalibration(...) -LPADC\_LPADC\_SetOffsetValue(...) -LPADC\_SetConvCommandConfig(...) -LPADC\_SetConvTriggerConfig(...)

#### Parameters

- base – LPADC peripheral base address.
- base – LPADC peripheral base address.

#### Return values

- kStatus\_Success – Successfully configured.
- kStatus\_Timeout – Timeout occurs while waiting completion.

`static inline void LPADC_SetOffsetValue(ADC_Type *base, int16_t value)`

Set trim value for offset.

---

**Note:** For 16-bit conversions, each increment is 1/2 LSB resulting in a programmable offset range of -256 LSB to 255.5 LSB; For 12-bit conversions, each increment is 1/32 LSB resulting in a programmable offset range of -16 LSB to 15.96875 LSB.

---

#### Parameters

- base – LPADC peripheral base address.
- value – Offset trim value, is a 10-bit signed value between -512 and 511.

`static inline void LPADC_GetOffsetValue(ADC_Type *base, int16_t *pValue)`

Get trim value of offset.

#### Parameters



- base – LPADC peripheral base address.
- pValue – Pointer to the variable in type of int16\_t to store offset value.

static inline void LPADC\_\_EnableOffsetCalibration(ADC\_Type \*base, bool enable)

Enable the offset calibration function.

#### Parameters

- base – LPADC peripheral base address.
- enable – switcher to the calibration function.

static inline void LPADC\_\_SetOffsetCalibrationMode(ADC\_Type \*base,  
lpadc\_offset\_calibration\_mode\_t mode)

Set offset calibration mode.

#### Parameters

- base – LPADC peripheral base address.
- mode – set offset calibration mode.see to lpadc\_offset\_calibration\_mode\_t .

status\_t LPADC\_\_DoOffsetCalibration(ADC\_Type \*base)

Do offset calibration.

#### Parameters

- base – LPADC peripheral base address.

#### Return values

- kStatus\_Success – Successfully configured.
- kStatus\_Timeout – Timeout occurs while waiting completion.

void LPADC\_\_PrepareAutoCalibration(ADC\_Type \*base)

Prepare auto calibration, LPADC\_FinishAutoCalibration has to be called before using the LPADC. LPADC\_DoAutoCalibration has been split in two API to avoid to be stuck too long in the function.

#### Parameters

- base – LPADC peripheral base address.

status\_t LPADC\_\_FinishAutoCalibration(ADC\_Type \*base)

Finish auto calibration start with LPADC\_\_PrepareAutoCalibration.

---

**Note:** This feature is used for LPADC with CTRL[CALOFSMODE].

---

#### Parameters

- base – LPADC peripheral base address.

#### Return values

- kStatus\_Success – Successfully configured.
- kStatus\_Timeout – Timeout occurs while waiting completion.

void LPADC\_\_GetCalibrationValue(ADC\_Type \*base, lpadc\_calibration\_value\_t  
\*ptrCalibrationValue)

Get calibration value into the memory which is defined by invoker.

---

**Note:** Please note the ADC will be disabled temporary.

---

---

**Note:** This function should be used after finish calibration.

---

#### Parameters

- base – LPADC peripheral base address.
- ptrCalibrationValue – Pointer to `lpadc_calibration_value_t` structure, this memory block should be always powered on even in low power modes.

`status_t` LPADC\_SetCalibrationValue(ADC\_Type \*base, const *lpadc\_calibration\_value\_t* \*ptrCalibrationValue)

Set calibration value into ADC calibration registers.

---

**Note:** Please note the ADC will be disabled temporary.

---

#### Parameters

- base – LPADC peripheral base address.
- ptrCalibrationValue – Pointer to `lpadc_calibration_value_t` structure which contains ADC's calibration value.

#### Return values

- kStatus\_Success – Successfully configured.
- kStatus\_Timeout – Timeout occurs while waiting completion.

static inline void LPADC\_RequestHighSpeedModeTrim(ADC\_Type \*base)

Request high speed mode trim calculation.

#### Parameters

- base – LPADC peripheral base address.

static inline int8\_t LPADC\_GetHighSpeedTrimValue(ADC\_Type \*base)

Get high speed mode trim value, the result is a 5-bit signed value between -16 and 15.

---

**Note:** The high speed mode trim value is used to minimize offset for high speed conversion.

---

#### Parameters

- base – LPADC peripheral base address.

#### Returns

The calculated high speed mode trim value.

static inline void LPADC\_SetHighSpeedTrimValue(ADC\_Type \*base, int8\_t trimValue)

Set high speed mode trim value.

---

**Note:** If is possible to set the trim value manually, but it is recommended to use the LPADC\_RequestHighSpeedModeTrim.

---

#### Parameters

- base – LPADC peripheral base address.
- trimValue – The trim value to be set.

static inline void LPADC\_EnableHighSpeedConversionMode(ADC\_Type \*base, bool enable)

Enable/disable high speed conversion mode, if enabled conversions complete 2 or 3 ADCK cycles sooner compared to conversion cycle counts when high speed mode is disabled.

#### Parameters

- base – LPADC peripheral base address.
- enable – Used to enable/disable high speed conversion mode:
  - **true** Enable high speed conversion mode;
  - **false** Disable high speed conversion mode.

static inline void LPADC\_EnableExtraCycle(ADC\_Type \*base, bool enable)

Enable/disable an additional ADCK cycle to conversion.

#### Parameters

- base – LPADC peripheral base address.
- enable – Used to enable/disable an additional ADCK cycle to conversion:
  - **true** Enable an additional ADCK cycle to conversion;
  - **false** Disable an additional ADCK cycle to conversion.

static inline void LPADC\_SetTuneValue(ADC\_Type \*base, *lpadc\_tune\_value\_t* tuneValue)

Set tune value which provides some variability in how many cycles are needed to complete a conversion.

#### Parameters

- base – LPADC peripheral base address.
- tuneValue – The tune value to be set, please refer to *lpadc\_tune\_value\_t*.

static inline *lpadc\_tune\_value\_t* LPADC\_GetTuneValue(ADC\_Type \*base)

Get tune value which provides some variability in how many cycles are needed to complete a conversion.

#### Parameters

- base – LPADC peripheral base address.

#### Returns

The tune value, please refer to *lpadc\_tune\_value\_t*.

FSL\_LPADC\_DRIVER\_VERSION

LPADC driver version 2.9.4.

struct *lpadc\_config\_t*

*#include <fsl\_lpadc.h>* LPADC global configuration.

This structure would used to keep the settings for initialization.

#### Public Members

bool enableInternalClock

Enables the internally generated clock source. The clock source is used in clock selection logic at the chip level and is optionally used for the ADC clock source.

bool enableVref1LowVoltage

If voltage reference option1 input is below 1.8V, it should be “true”. If voltage reference option1 input is above 1.8V, it should be “false”.

`bool enableInDozeMode`

Control system transition to Stop and Wait power modes while ADC is converting. When enabled in Doze mode, immediate entries to Wait or Stop are allowed. When disabled, the ADC will wait for the current averaging iteration/FIFO storage to complete before acknowledging stop or wait mode entry.

`lpadc_conversion_average_mode_t` conversionAverageMode

Auto-Calibration Averages.

`bool enableAnalogPreliminary`

ADC analog circuits are pre-enabled and ready to execute conversions without startup delays(at the cost of higher DC current consumption).

`uint32_t powerUpDelay`

When the analog circuits are not pre-enabled, the ADC analog circuits are only powered while the ADC is active and there is a counted delay defined by this field after an initial trigger transitions the ADC from its Idle state to allow time for the analog circuits to stabilize. The startup delay count of (`powerUpDelay * 4`) ADCK cycles must result in a longer delay than the analog startup time.

`lpadc_reference_voltage_source_t` referenceVoltageSource

Selects the voltage reference high used for conversions.

`lpadc_power_level_mode_t` powerLevelMode

Power Configuration Selection.

`lpadc_trigger_priority_policy_t` triggerPriorityPolicy

Control how higher priority triggers are handled, see to `lpadc_trigger_priority_t`.

`bool enableConvPause`

Enables the ADC pausing function. When enabled, a programmable delay is inserted during command execution sequencing between LOOP iterations, between commands in a sequence, and between conversions when command is executing in “Compare Until True” configuration.

`uint32_t convPauseDelay`

Controls the duration of pausing during command execution sequencing. The pause delay is a count of (`convPauseDelay*4`) ADCK cycles. Only available when ADC pausing function is enabled. The available value range is in 9-bit.

`uint32_t FIFOWatermark`

FIFOWatermark is a programmable threshold setting. When the number of datawords stored in the ADC Result FIFO is greater than the value in this field, the ready flag would be asserted to indicate stored data has reached the programmable threshold.

`struct lpadc_conv_command_config_t`

`#include <fsl_lpadc.h>` Define structure to keep the configuration for conversion command.

## Public Members

`lpadc_sample_scale_mode_t` sampleScaleMode

Sample scale mode.

`lpadc_sample_scale_mode_t` channelBScaleMode

Alternate channel B Scale mode.

`lpadc_sample_channel_mode_t` sampleChannelMode

Channel sample mode.

uint32\_t channelNumber

Channel number, select the channel or channel pair.

uint32\_t channelBNumber

Alternate Channel B number, select the channel.

uint32\_t chainedNextCommandNumber

Selects the next command to be executed after this command completes. 1-15 is available, 0 is to terminate the chain after this command.

bool enableAutoChannelIncrement

Loop with increment: when disabled, the “loopCount” field selects the number of times the selected channel is converted consecutively; when enabled, the “loopCount” field defines how many consecutive channels are converted as part of the command execution.

uint32\_t loopCount

Selects how many times this command executes before finish and transition to the next command or Idle state. Command executes LOOP+1 times. 0-15 is available.

*lpadc\_hardware\_average\_mode\_t* hardwareAverageMode

Hardware average selection.

*lpadc\_sample\_time\_mode\_t* sampleTimeMode

Sample time selection.

*lpadc\_hardware\_compare\_mode\_t* hardwareCompareMode

Hardware compare selection.

uint32\_t hardwareCompareValueHigh

Compare Value High. The available value range is in 16-bit.

uint32\_t hardwareCompareValueLow

Compare Value Low. The available value range is in 16-bit.

*lpadc\_conversion\_resolution\_mode\_t* conversionResolutionMode

Conversion resolution mode.

bool enableWaitTrigger

Wait for trigger assertion before execution: when disabled, this command will be automatically executed; when enabled, the active trigger must be asserted again before executing this command.

struct lpadc\_conv\_trigger\_config\_t

*#include <fsl\_lpadc.h>* Define structure to keep the configuration for conversion trigger.

## Public Members

uint32\_t targetCommandId

Select the command from command buffer to execute upon detect of the associated trigger event.

uint32\_t delayPower

Select the trigger delay duration to wait at the start of servicing a trigger event. When this field is clear, then no delay is incurred. When this field is set to a non-zero value, the duration for the delay is  $2^{\text{delayPower}}$  ADCK cycles. The available value range is 4-bit.

uint32\_t priority

Sets the priority of the associated trigger source. If two or more triggers have the same priority level setting, the lower order trigger event has the higher priority. The lower value for this field is for the higher priority, the available value range is 1-bit.

bool enableHardwareTrigger

Enable hardware trigger source to initiate conversion on the rising edge of the input trigger source or not. The software trigger is always available.

struct lpadc\_conv\_result\_t

#include <fsl\_lpadc.h> Define the structure to keep the conversion result.

### Public Members

uint32\_t commandIdSource

Indicate the command buffer being executed that generated this result.

uint32\_t loopCountIndex

Indicate the loop count value during command execution that generated this result.

uint32\_t triggerIdSource

Indicate the trigger source that initiated a conversion and generated this result.

uint16\_t convValue

Data result.

struct \_\_lpadc\_calibration\_value

#include <fsl\_lpadc.h> A structure of calibration value.

## 2.28 Lpc\_freqme

void FREQME\_Init(FREQME\_Type \*base, const *freq\_measure\_config\_t* \*config)

Initialize freqme module, set operate mode, operate mode attribute and initialize measurement cycle.

### Parameters

- base – FREQME peripheral base address.
- config – The pointer to module basic configuration, please refer to *freq\_measure\_config\_t*.

void FREQME\_GetDefaultConfig(*freq\_measure\_config\_t* \*config)

Get default configuration.

```
config->operateMode = kFREQME_FreqMeasurementMode;
config->operateModeAttribute.refClkScaleFactor = 0U;
config->enableContinuousMode = false;
config->startMeasurement = false;
```

### Parameters

- config – The pointer to module basic configuration, please refer to *freq\_measure\_config\_t*.

static inline void FREQME\_StartMeasurementCycle(FREQME\_Type \*base)

Start frequency or pulse width measurement process.

### Parameters

- base – FREQME peripheral base address.

static inline void FREQME\_TerminateMeasurementCycle(FREQME\_Type \*base)

Force the termination of any measurement cycle currently in progress and resets RESULT or just reset RESULT if the module in idle state.

#### Parameters

- base – FREQME peripheral base address.

static inline void FREQME\_EnableContinuousMode(FREQME\_Type \*base, bool enable)

Enable/disable Continuous mode.

#### Parameters

- base – FREQME peripheral base address.
- enable – Used to enable/disable continuous mode,
  - **true** Enable Continuous mode.
  - **false** Disable Continuous mode.

static inline bool FREQME\_CheckContinuousMode(FREQME\_Type \*base)

Check whether continuous mode is enabled.

#### Parameters

- base – FREQME peripheral base address.

#### Return values

- True – Continuous mode is enabled, the measurement is performed continuously.
- False – Continuous mode is disabled.

static inline void FREQME\_SetOperateMode(FREQME\_Type \*base, *freqme\_operate\_mode\_t* operateMode)

Set operate mode of freqme module.

#### Parameters

- base – FREQME peripheral base address.
- operateMode – The operate mode to be set, please refer to *freqme\_operate\_mode\_t*.

static inline bool FREQME\_CheckOperateMode(FREQME\_Type \*base)

Check module's operate mode.

#### Parameters

- base – FREQME peripheral base address.

#### Return values

- True – Pulse width measurement mode.
- False – Frequency measurement mode.

static inline void FREQME\_SetMinExpectedValue(FREQME\_Type \*base, uint32\_t minValue)

Set the minimum expected value for the measurement result.

#### Parameters

- base – FREQME peripheral base address.
- minValue – The minimum value to set, please note that this value is 31 bits width.

```
static inline void FREQME_SetMaxExpectedValue(FREQME_Type *base, uint32_t maxValue)
```

Set the maximum expected value for the measurement result.

**Parameters**

- base – FREQME peripheral base address.
- maxValue – The maximum value to set, please note that this value is 31 bits width.

```
uint32_t FREQME_CalculateTargetClkFreq(FREQME_Type *base, uint32_t refClkFrequency)
```

Calculate the frequency of selected target clock□

---

**Note:** The formula:  $F_{target} = (RESULT - 2) * F_{reference} / 2^{REF\_SCALE}$ .

---

---

**Note:** This function only useful when the operate mode is selected as frequency measurement mode.

---

**Parameters**

- base – FREQME peripheral base address.
- refClkFrequency – The frequency of reference clock.

**Returns**

The frequency of target clock the unit is Hz, if the output result is 0, please check the module's operate mode.

```
static inline uint8_t FREQME_GetReferenceClkScaleValue(FREQME_Type *base)
```

Get reference clock scaling factor.

**Parameters**

- base – FREQME peripheral base address.

**Returns**

Reference clock scaling factor, the reference count cycle is  $2^{ref\_scale}$ .

```
static inline void FREQME_SetPulsePolarity(FREQME_Type *base, freqme_pulse_polarity_t pulsePolarity)
```

Set pulse polarity when operate mode is selected as Pulse Width Measurement mode.

**Parameters**

- base – FREQME peripheral base address.
- pulsePolarity – The pulse polarity to be set, please refer to `freqme_pulse_polarity_t`.

```
static inline bool FREQME_CheckPulsePolarity(FREQME_Type *base)
```

Check pulse polarity when the operate mode is selected as pulse width measurement mode.

**Parameters**

- base – FREQME peripheral base address.

**Return values**

- True – Low period.
- False – High period.



```
static inline uint32_t FREQME_GetMeasurementResult(FREQME_Type *base)
```

Get measurement result, if operate mode is selected as pulse width measurement mode this function can be used to calculate pulse width.

---

**Note:** Pulse width = counter result / Frequency of target clock.

---

#### Parameters

- base – FREQME peripheral base address.

#### Returns

Measurement result.

```
static inline uint32_t FREQME_GetInterruptStatusFlags(FREQME_Type *base)
```

Get interrupt status flags, such as overflow interrupt status flag, underflow interrupt status flag, and so on.

#### Parameters

- base – FREQME peripheral base address.

#### Returns

Current interrupt status flags, should be the OR'ed value of `_freqme_interrupt_status_flags`.

```
static inline void FREQME_ClearInterruptStatusFlags(FREQME_Type *base, uint32_t statusFlags)
```

Clear interrupt status flags.

#### Parameters

- base – FREQME peripheral base address.
- statusFlags – The combination of interrupt status flags to clear, should be the OR'ed value of `_freqme_interrupt_status_flags`.

```
static inline void FREQME_EnableInterrupts(FREQME_Type *base, uint32_t masks)
```

Enable interrupts, such as result ready interrupt, overflow interrupt and so on.

#### Parameters

- base – FREQME peripheral base address.
- masks – The mask of interrupts to enable, should be the OR'ed value of `_freqme_interrupt_enable`.

```
static inline void FREQME_DisableInterrupts(FREQME_Type *base, uint32_t masks)
```

Disable interrupts, such as result ready interrupt, overflow interrupt and so on.

#### Parameters

- base – FREQME peripheral base address.
- masks – The mask of interrupts to disable, should be the OR'ed value of `_freqme_interrupt_enable`.

```
FSL_FREQME_DRIVER_VERSION
```

FREQME driver version 2.1.2.

```
enum _freqme_interrupt_status_flags
```

The enumeration of interrupt status flags. .

*Values:*

enumerator kFREQME\_\_UnderflowInterruptStatusFlag

Indicate the measurement is just done and the result is less than minimum value.

enumerator kFREQME\_\_OverflowInterruptStatusFlag

Indicate the measurement is just done and the result is greater than maximum value.

enumerator kFREQME\_\_ReadyInterruptStatusFlag

Indicate the measurement is just done and the result is ready to read.

enumerator kFREQME\_\_AllInterruptStatusFlags

All interrupt status flags.

enum \_freqme\_interrupt\_enable

The enumeration of interrupts, including underflow interrupt, overflow interrupt, and result ready interrupt. .

*Values:*

enumerator kFREQME\_\_UnderflowInterruptEnable

Enable interrupt when the result is less than minimum value.

enumerator kFREQME\_\_OverflowInterruptEnable

Enable interrupt when the result is greater than maximum value.

enumerator kFREQME\_\_ReadyInterruptEnable

Enable interrupt when a measurement completes and the result is ready.

enum \_freqme\_operate\_mode

FREQME module operate mode enumeration, including frequency measurement mode and pulse width measurement mode.

*Values:*

enumerator kFREQME\_\_FreqMeasurementMode

The module works in the frequency measurement mode.

enumerator kFREQME\_\_PulseWidthMeasurementMode

The module works in the pulse width measurement mode.

enum \_freqme\_pulse\_polarity

The enumeration of pulse polarity.

*Values:*

enumerator kFREQME\_\_PulseHighPeriod

Select high period of the reference clock.

enumerator kFREQME\_\_PulseLowPeriod

Select low period of the reference clock.

typedef enum \_freqme\_operate\_mode freqme\_operate\_mode\_t

FREQME module operate mode enumeration, including frequency measurement mode and pulse width measurement mode.

typedef enum \_freqme\_pulse\_polarity freqme\_pulse\_polarity\_t

The enumeration of pulse polarity.

typedef union \_freqme\_mode\_attribute freqme\_mode\_attribute\_t

The union of operate mode attribute.

---

**Note:** If the operate mode is selected as frequency measurement mode the member **refClkScaleFactor** should be used, if the operate mode is selected as pulse width measurement mode the member **pulsePolarity** should be used.

---

```
typedef struct _freq_measure_config freq_measure_config_t
```

The structure of freqme module basic configuration, including operate mode, operate mode attribute and so on.

```
union _freqme_mode_attribute
```

*#include <fsl\_freqme.h>* The union of operate mode attribute.

---

**Note:** If the operate mode is selected as frequency measurement mode the member **refClkScaleFactor** should be used, if the operate mode is selected as pulse width measurement mode the member **pulsePolarity** should be used.

---

### Public Members

```
uint8_t refClkScaleFactor
```

Only useful in frequency measurement operate mode, used to set the reference clock counter scaling factor.

```
freqme_pulse_polarity_t pulsePolarity
```

Only Useful in pulse width measurement operate mode, used to set period polarity.

```
struct _freq_measure_config
```

*#include <fsl\_freqme.h>* The structure of freqme module basic configuration, including operate mode, operate mode attribute and so on.

### Public Members

```
freqme_operate_mode_t operateMode
```

Select operate mode, please refer to *freqme\_operate\_mode\_t*.

```
freqme_mode_attribute_t operateModeAttribute
```

Used to set the attribute of the selected operate mode, if the operate mode is selected as kFREQME\_FreqMeasurementMode set *freqme\_mode\_attribute\_t::refClkScaleFactor*, if operate mode is selected as kFREOME\_PulseWidthMeasurementMode, please set *freqme\_mode\_attribute\_t::pulsePolarity*.

```
bool enableContinuousMode
```

Enable/disable continuous mode, if continuous mode is enable, the measurement is performed continuously and the result for the last completed measurement is available in the result register.

## 2.29 LPCMP: Low Power Analog Comparator Driver

```
void LPCMP_Init(LPCMP_Type *base, const lpcmp_config_t *config)
```

Initialize the LPCMP.

This function initializes the LPCMP module. The operations included are:

- Enabling the clock for LPCMP module.
- Configuring the comparator.
- Enabling the LPCMP module. Note: For some devices, multiple LPCMP instance share the same clock gate. In this case, to enable the clock for any instance enables all the LPCMPs. Check the chip reference manual for the clock assignment of the LPCMP.

### Parameters

- base – LPCMP peripheral base address.
- config – Pointer to “lpcmp\_config\_t” structure.

void LPCMP\_Deinit(LPCMP\_Type \*base)

De-initializes the LPCMP module.

This function de-initializes the LPCMP module. The operations included are:

- Disabling the LPCMP module.
- Disabling the clock for LPCMP module.

This function disables the clock for the LPCMP. Note: For some devices, multiple LPCMP instance shares the same clock gate. In this case, before disabling the clock for the LPCMP, ensure that all the LPCMP instances are not used.

#### Parameters

- base – LPCMP peripheral base address.

void LPCMP\_GetDefaultConfig(*lpcmp\_config\_t* \*config)

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

This function initializes the comparator configuration structure to these default values:

```
config->enableStopMode    = false;
config->enableOutputPin   = false;
config->enableCmpToDacLink = false;
config->useUnfilteredOutput = false;
config->enableInvertOutput = false;
config->hysteresisMode     = kLPCMP_HysteresisLevel0;
config->powerMode          = kLPCMP_LowSpeedPowerMode;
config->functionalSourceClock = kLPCMP_FunctionalClockSource0;
config->plusInputSrc       = kLPCMP_PlusInputSrcMux;
config->minusInputSrc      = kLPCMP_MinusInputSrcMux;
```

#### Parameters

- config – Pointer to “lpcmp\_config\_t” structure.

static inline void LPCMP\_Enable(LPCMP\_Type \*base, bool enable)

Enable/Disable LPCMP module.

#### Parameters

- base – LPCMP peripheral base address.
- enable – “true” means enable the module, and “false” means disable the module.

void LPCMP\_SetInputChannels(LPCMP\_Type \*base, uint32\_t positiveChannel, uint32\_t negativeChannel)

Select the input channels for LPCMP. This function determines which input is selected for the negative and positive mux.

#### Parameters

- base – LPCMP 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.

static inline void LPCMP\_EnableDMA(LPCMP\_Type \*base, bool enable)

Enables/disables the DMA request for rising/falling events. Normally, the LPCMP generates a CPU interrupt if there is a rising/falling event. When DMA support is enabled and the rising/falling interrupt is enabled, the rising/falling event forces a DMA transfer request rather than a CPU interrupt instead.

**Parameters**

- base – LPCMP peripheral base address.
- enable – “true” means enable DMA support, and “false” means disable DMA support.

void LPCMP\_SetFilterConfig(LPCMP\_Type \*base, const *lpcmp\_filter\_config\_t* \*config)

Configures the filter.

**Parameters**

- base – LPCMP peripheral base address.
- config – Pointer to “lpcmp\_filter\_config\_t” structure.

void LPCMP\_SetDACConfig(LPCMP\_Type \*base, const *lpcmp\_dac\_config\_t* \*config)

Configure the internal DAC module.

**Parameters**

- base – LPCMP peripheral base address.
- config – Pointer to “lpcmp\_dac\_config\_t” structure. If config is “NULL”, disable internal DAC.

static inline void LPCMP\_EnableInterrupts(LPCMP\_Type \*base, uint32\_t mask)

Enable the interrupts.

**Parameters**

- base – LPCMP peripheral base address.
- mask – Mask value for interrupts. See “\_lpcmp\_interrupt\_enable”.

static inline void LPCMP\_DisableInterrupts(LPCMP\_Type \*base, uint32\_t mask)

Disable the interrupts.

**Parameters**

- base – LPCMP peripheral base address.
- mask – Mask value for interrupts. See “\_lpcmp\_interrupt\_enable”.

static inline uint32\_t LPCMP\_GetStatusFlags(LPCMP\_Type \*base)

Get the LPCMP status flags.

**Parameters**

- base – LPCMP peripheral base address.

**Returns**

Mask value for the asserted flags. See “\_lpcmp\_status\_flags”.

static inline void LPCMP\_ClearStatusFlags(LPCMP\_Type \*base, uint32\_t mask)

Clear the LPCMP status flags.

**Parameters**

- base – LPCMP peripheral base address.
- mask – Mask value for the flags. See “\_lpcmp\_status\_flags”.

static inline void LPCMP\_EnableWindowMode(LPCMP\_Type \*base, bool enable)

Enable/Disable window mode. When any windowed mode is active, COUTA is clocked by the bus clock whenever WINDOW = 1. The last latched value is held when WINDOW = 0. The optionally inverted comparator output COUT\_RAW is sampled on every bus clock when WINDOW=1 to generate COUTA.

#### Parameters

- base – LPCMP peripheral base address.
- enable – “true” means enable window mode, and “false” means disable window mode.

void LPCMP\_SetWindowControl(LPCMP\_Type \*base, const *lpcmp\_window\_control\_config\_t* \*config)

Configure the window control, users can use this API to implement operations on the window, such as inverting the window signal, setting the window closing event (only valid in windowing mode), and setting the COUTA signal after the window is closed (only valid in windowing mode).

#### Parameters

- base – LPCMP peripheral base address.
- config – Pointer “lpcmp\_window\_control\_config\_t” structure.

void LPCMP\_SetRoundRobinConfig(LPCMP\_Type \*base, const *lpcmp\_roundrobin\_config\_t* \*config)

Configure the roundrobin mode.

#### Parameters

- base – LPCMP peripheral base address.
- config – Pointer “lpcmp\_roundrobin\_config\_t” structure.

static inline void LPCMP\_EnableRoundRobinMode(LPCMP\_Type \*base, bool enable)

Enable/Disable roundrobin mode.

#### Parameters

- base – LPCMP peripheral base address.
- enable – “true” means enable roundrobin mode, and “false” means disable roundrobin mode.

void LPCMP\_SetRoundRobinInternalTimer(LPCMP\_Type \*base, uint32\_t value)

brief Configure the roundrobin internal timer reload value.

param base LPCMP peripheral base address. param value RoundRobin internal timer reload value, allowed range: 0x0UL-0xFFFFFFFFUL.

static inline void LPCMP\_EnableRoundRobinInternalTimer(LPCMP\_Type \*base, bool enable)

Enable/Disable roundrobin internal timer, note that this function is only valid when using the internal trigger source.

#### Parameters

- base – LPCMP peripheral base address.
- enable – “true” means enable roundrobin internal timer, and “false” means disable roundrobin internal timer.

static inline void LPCMP\_SetPreSetValue(LPCMP\_Type \*base, uint8\_t mask)

Set preset value for all channels, users can set all channels' preset value through this API, for example, if the mask set to 0x03U means channel0 and channel2's preset value set to 1U and other channels' preset value set to 0U.

**Parameters**

- base – LPCMP peripheral base address.
- mask – Mask of channel index.

static inline uint8\_t LPCMP\_GetComparisonResult(LPCMP\_Type \*base)

Get comparison results for all channels, users can get all channels' comparison results through this API.

**Parameters**

- base – LPCMP peripheral base address.

**Returns**

return All channels' comparison result.

static inline void LPCMP\_ClearInputChangedFlags(LPCMP\_Type \*base, uint8\_t mask)

Clear input changed flags for single channel or multiple channels, users can clear input changed flag of a single channel or multiple channels through this API, for example, if the mask set to 0x03U means clear channel0 and channel2's input changed flags.

**Parameters**

- base – LPCMP peripheral base address.
- mask – Mask of channel index.

static inline uint8\_t LPCMP\_GetInputChangedFlags(LPCMP\_Type \*base)

Get input changed flags for all channels, Users can get all channels' input changed flags through this API.

**Parameters**

- base – LPCMP peripheral base address.

**Returns**

return All channels' changed flag.

FSL\_LPCMP\_DRIVER\_VERSION

LPCMP driver version 2.3.2.

enum \_lpcmp\_status\_flags

LPCMP status falgs mask.

*Values:*

enumerator kLPCMP\_OutputRisingEventFlag

Rising-edge on the comparison output has occurred.

enumerator kLPCMP\_OutputFallingEventFlag

Falling-edge on the comparison output has occurred.

enumerator kLPCMP\_OutputRoundRobinEventFlag

Detects when any channel's last comparison result is different from the pre-set value in trigger mode.

enumerator kLPCMP\_OutputAssertEventFlag

Return the current value of the analog comparator output. The flag does not support W1C.

enum \_lpcmp\_interrupt\_enable

LPCMP interrupt enable/disable mask.

*Values:*

enumerator kLPCMP\_OutputRisingInterruptEnable

Comparator interrupt enable rising.

enumerator kLPCMP\_OutputFallingInterruptEnable

Comparator interrupt enable falling.

enumerator kLPCMP\_RoundRobinInterruptEnable

Comparator round robin mode interrupt occurred when the comparison result changes for a given channel.

enum \_lpcmp\_hysteresis\_mode

LPCMP hysteresis mode. See chip data sheet to get the actual hysteresis value with each level.

*Values:*

enumerator kLPCMP\_HysteresisLevel0

The hard block output has level 0 hysteresis internally.

enumerator kLPCMP\_HysteresisLevel1

The hard block output has level 1 hysteresis internally.

enumerator kLPCMP\_HysteresisLevel2

The hard block output has level 2 hysteresis internally.

enumerator kLPCMP\_HysteresisLevel3

The hard block output has level 3 hysteresis internally.

enum \_lpcmp\_power\_mode

LPCMP nano mode.

*Values:*

enumerator kLPCMP\_LowSpeedPowerMode

Low speed comparison mode is selected.

enumerator kLPCMP\_HighSpeedPowerMode

High speed comparison mode is selected.

enumerator kLPCMP\_NanoPowerMode

Nano power comparator is enabled.

enum \_lpcmp\_dac\_reference\_voltage\_source

Internal DAC reference voltage source.

*Values:*

enumerator kLPCMP\_VrefSourceVin1

vrefh\_int is selected as resistor ladder network supply reference Vin.

enumerator kLPCMP\_VrefSourceVin2

vrefh\_ext is selected as resistor ladder network supply reference Vin.

enum \_lpcmp\_functional\_source\_clock

LPCMP functional mode clock source selection.

Note: In different devices, the functional mode clock source selection is different, please refer to specific device Reference Manual for details.

*Values:*

enumerator kLPCMP\_FunctionalClockSource0

Select functional mode clock source0.



enumerator kLPCMP\_FunctionalClockSource1

Select functional mode clock source1.

enumerator kLPCMP\_FunctionalClockSource2

Select functional mode clock source2.

enumerator kLPCMP\_FunctionalClockSource3

Select functional mode clock source3.

enum \_\_lpcmp\_couta\_signal

Set the COUTA signal value when the window is closed.

*Values:*

enumerator kLPCMP\_COUTASignalNoSet

NO set the COUTA signal value when the window is closed.

enumerator kLPCMP\_COUTASignalLow

Set COUTA signal low(0) when the window is closed.

enumerator kLPCMP\_COUTASignalHigh

Set COUTA signal high(1) when the window is closed.

enum \_\_lpcmp\_close\_window\_event

Set COUT event, which can close the active window in window mode.

*Values:*

enumerator kLPCMP\_CCloseWindowEventNoSet

No Set COUT event, which can close the active window in window mode.

enumerator kLPCMP\_CloseWindowEventRisingEdge

Set rising edge COUT signal as COUT event.

enumerator kLPCMP\_CloseWindowEventFallingEdge

Set falling edge COUT signal as COUT event.

enumerator kLPCMP\_CCloseWindowEventBothEdge

Set both rising and falling edge COUT signal as COUT event.

enum \_\_lpcmp\_roundrobin\_fixedmuxport

LPCMP round robin mode fixed mux port.

*Values:*

enumerator kLPCMP\_FixedPlusMuxPort

Fixed plus mux port.

enumerator kLPCMP\_FixedMinusMuxPort

Fixed minus mux port.

enum \_\_lpcmp\_roundrobin\_clock\_source

LPCMP round robin mode clock source selection.

Note: In different devices, the round robin mode clock source selection is different, please refer to the specific device Reference Manual for details.

*Values:*

enumerator kLPCMP\_RoundRobinClockSource0

Select roundrobin mode clock source0.

enumerator kLPCMP\_RoundRobinClockSource1

Select roundrobin mode clock source1.

enumerator kLPCMP\_RoundRobinClockSource2

Select roundrobin mode clock source2.

enumerator kLPCMP\_RoundRobinClockSource3

Select roundrobin mode clock source3.

enum \_lpcmp\_roundrobin\_trigger\_source

LPCMP round robin mode trigger source.

*Values:*

enumerator kLPCMP\_TriggerSourceExternally

Select external trigger source.

enumerator kLPCMP\_TriggerSourceInternally

Select internal trigger source.

typedef enum \_lpcmp\_hysteresis\_mode lpcmp\_hysteresis\_mode\_t

LPCMP hysteresis mode. See chip data sheet to get the actual hysteresis value with each level.

typedef enum \_lpcmp\_power\_mode lpcmp\_power\_mode\_t

LPCMP nano mode.

typedef enum \_lpcmp\_dac\_reference\_voltage\_source lpcmp\_dac\_reference\_voltage\_source\_t

Internal DAC reference voltage source.

typedef enum \_lpcmp\_functional\_source\_clock lpcmp\_functional\_source\_clock\_t

LPCMP functional mode clock source selection.

Note: In different devices, the functional mode clock source selection is different, please refer to specific device Reference Manual for details.

typedef enum \_lpcmp\_couta\_signal lpcmp\_couta\_signal\_t

Set the COUTA signal value when the window is closed.

typedef enum \_lpcmp\_close\_window\_event lpcmp\_close\_window\_event\_t

Set COUT event, which can close the active window in window mode.

typedef enum \_lpcmp\_roundrobin\_fixedmuxport lpcmp\_roundrobin\_fixedmuxport\_t

LPCMP round robin mode fixed mux port.

typedef enum \_lpcmp\_roundrobin\_clock\_source lpcmp\_roundrobin\_clock\_source\_t

LPCMP round robin mode clock source selection.

Note: In different devices, the round robin mode clock source selection is different, please refer to the specific device Reference Manual for details.

typedef enum \_lpcmp\_roundrobin\_trigger\_source lpcmp\_roundrobin\_trigger\_source\_t

LPCMP round robin mode trigger source.

typedef struct \_lpcmp\_filter\_config lpcmp\_filter\_config\_t

Configure the filter.

typedef struct \_lpcmp\_dac\_config lpcmp\_dac\_config\_t

configure the internal DAC.

typedef struct \_lpcmp\_config lpcmp\_config\_t

Configures the comparator.

typedef struct \_lpcmp\_window\_control\_config lpcmp\_window\_control\_config\_t

Configure the window mode control.

```
typedef struct _lpcmp_roundrobin_config lpcmp_roundrobin_config_t
```

Configure the round robin mode.

```
LPCMP_CCR1_COUTA_CFG_MASK
```

```
LPCMP_CCR1_COUTA_CFG_SHIFT
```

```
LPCMP_CCR1_COUTA_CFG(x)
```

```
LPCMP_CCR1_EVT_SEL_CFG_MASK
```

```
LPCMP_CCR1_EVT_SEL_CFG_SHIFT
```

```
LPCMP_CCR1_EVT_SEL_CFG(x)
```

```
struct _lpcmp_filter_config
```

*#include <fsl\_lpcmp.h>* Configure the filter.

### Public Members

bool enableSample

Decide whether to use the external SAMPLE as a sampling clock input.

uint8\_t filterSampleCount

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

uint8\_t filterSamplePeriod

Filter Sample Period. The divider to the bus clock. Available range is 0-255. The sampling clock must be at least 4 times slower than the system clock to the comparator. So if enableSample is “false”, filterSamplePeriod should be set greater than 4.

```
struct _lpcmp_dac_config
```

*#include <fsl\_lpcmp.h>* configure the internal DAC.

### Public Members

bool enableLowPowerMode

Decide whether to enable DAC low power mode.

*lpcmp\_dac\_reference\_voltage\_source\_t* referenceVoltageSource

Internal DAC supply voltage reference source.

uint8\_t DACValue

Value for the DAC Output Voltage. Different devices has different available range, for specific values, please refer to the reference manual.

```
struct _lpcmp_config
```

*#include <fsl\_lpcmp.h>* Configures the comparator.

### Public Members

bool enableStopMode

Decide whether to enable the comparator when in STOP modes.

bool enableOutputPin

Decide whether to enable the comparator is available in selected pin.

bool useUnfilteredOutput

Decide whether to use unfiltered output.

bool enableInvertOutput

Decide whether to invert the comparator output.

*lpcmp\_hysteresis\_mode\_t* hysteresisMode

LPCMP hysteresis mode.

*lpcmp\_power\_mode\_t* powerMode

LPCMP power mode.

*lpcmp\_functional\_source\_clock\_t* functionalSourceClock

Select LPCMP functional mode clock source.

struct *\_\_lpcmp\_window\_control\_config*

*#include <fsl\_lpcmp.h>* Configure the window mode control.

### Public Members

bool enableInvertWindowSignal

True: enable invert window signal, False: disable invert window signal.

*lpcmp\_couta\_signal\_t* COUTASignal

Decide whether to define the COUTA signal value when the window is closed.

*lpcmp\_close\_window\_event\_t* closeWindowEvent

Decide whether to select COUT event signal edge defines a COUT event to close window.

struct *\_\_lpcmp\_roundrobin\_config*

*#include <fsl\_lpcmp.h>* Configure the round robin mode.

### Public Members

uint8\_t initDelayModules

Comparator and DAC initialization delay modulus, See Reference Manual and DataSheet for specific value.

uint8\_t sampleClockNumbers

Specify the number of the round robin clock cycles(0~3) to wait after scanning the active channel before sampling the channel's comparison result.

uint8\_t channelSampleNumbers

Specify the number of samples for one channel, note that channelSampleNumbers must not smaller than sampleTimeThreshhold.

uint8\_t sampleTimeThreshhold

Specify that for one channel, when (sampleTimeThreshhold + 1) sample results are "1", the final result is "1", otherwise the final result is "0", note that the sampleTimeThreshhold must not be larger than channelSampleNumbers.

*lpcmp\_roundrobin\_clock\_source\_t* roundrobinClockSource

Decide which clock source to choose in round robin mode.

*lpcmp\_roundrobin\_trigger\_source\_t* roundrobinTriggerSource

Decide which trigger source to choose in round robin mode.

*lpcmp\_roundrobin\_fixedmuxport\_t* fixedMuxPort

Decide which mux port to choose as fixed channel in round robin mode.

uint8\_t fixedChannel

Indicate which channel of the fixed mux port is used in round robin mode.

uint8\_t checkerChannelMask

Indicate which channel of the non-fixed mux port to check its voltage value in round robin mode, for example, if checkerChannelMask set to 0x11U means select channel 0 and channel 4 as checker channel.

## 2.30 LPI2C: Low Power Inter-Integrated Circuit Driver

void LPI2C\_DriverIRQHandler(uint32\_t instance)

LPI2C driver IRQ handler common entry.

This function provides the common IRQ request entry for LPI2C.

### Parameters

- instance – LPI2C instance.

FSL\_LPI2C\_DRIVER\_VERSION

LPI2C driver version.

LPI2C status return codes.

*Values:*

enumerator kStatus\_LPI2C\_Busy

The master is already performing a transfer.

enumerator kStatus\_LPI2C\_Idle

The slave driver is idle.

enumerator kStatus\_LPI2C\_Nak

The slave device sent a NAK in response to a byte.

enumerator kStatus\_LPI2C\_FifoError

FIFO under run or overrun.

enumerator kStatus\_LPI2C\_BitError

Transferred bit was not seen on the bus.

enumerator kStatus\_LPI2C\_ArbitrationLost

Arbitration lost error.

enumerator kStatus\_LPI2C\_PinLowTimeout

SCL or SDA were held low longer than the timeout.

enumerator kStatus\_LPI2C\_NoTransferInProgress

Attempt to abort a transfer when one is not in progress.

enumerator kStatus\_LPI2C\_DmaRequestFail

DMA request failed.

enumerator kStatus\_LPI2C\_Timeout

Timeout polling status flags.

IRQn\_Type const kLpi2cIrqs[]

Array to map LPI2C instance number to IRQ number, used internally for LPI2C master interrupt and EDMA transactional APIs.

lpi2c\_master\_isr\_t s\_lpi2cMasterIsr

Pointer to master IRQ handler for each instance, used internally for LPI2C master interrupt and EDMA transactional APIs.

`void *s_lpi2cMasterHandle[]`

Pointers to master handles for each instance, used internally for LPI2C master interrupt and EDMA transactional APIs.

`uint32_t LPI2C_GetInstance(LPI2C_Type *base)`

Returns an instance number given a base address.

If an invalid base address is passed, debug builds will assert. Release builds will just return instance number 0.

#### Parameters

- `base` – The LPI2C peripheral base address.

#### Returns

LPI2C instance number starting from 0.

`I2C_RETRY_TIMES`

Retry times for waiting flag.

## 2.31 LPI2C Master Driver

`void LPI2C_MasterGetDefaultConfig(lpi2c_master_config_t *masterConfig)`

Provides a default configuration for the LPI2C master peripheral.

This function provides the following default configuration for the LPI2C master peripheral:

```
masterConfig->enableMaster      = true;
masterConfig->debugEnable       = false;
masterConfig->ignoreAck         = false;
masterConfig->pinConfig          = kLPI2C_2PinOpenDrain;
masterConfig->baudRate_Hz       = 100000U;
masterConfig->busIdleTimeout_ns = 0;
masterConfig->pinLowTimeout_ns  = 0;
masterConfig->sdaGlitchFilterWidth_ns = 0;
masterConfig->sclGlitchFilterWidth_ns = 0;
masterConfig->hostRequest.enable = false;
masterConfig->hostRequest.source  = kLPI2C_HostRequestExternalPin;
masterConfig->hostRequest.polarity = kLPI2C_HostRequestPinActiveHigh;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with `LPI2C_MasterInit()`.

#### Parameters

- `masterConfig` – **[out]** User provided configuration structure for default values. Refer to `lpi2c_master_config_t`.

`void LPI2C_MasterInit(LPI2C_Type *base, const lpi2c_master_config_t *masterConfig, uint32_t sourceClock_Hz)`

Initializes the LPI2C master peripheral.

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

#### Parameters

- `base` – The LPI2C peripheral base address.
- `masterConfig` – User provided peripheral configuration. Use `LPI2C_MasterGetDefaultConfig()` to get a set of defaults that you can override.

- `sourceClock_Hz` – Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

`void LPI2C_MasterDeinit(LPI2C_Type *base)`

Deinitializes the LPI2C master peripheral.

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

#### Parameters

- `base` – The LPI2C peripheral base address.

`void LPI2C_MasterConfigureDataMatch(LPI2C_Type *base, const lpi2c_data_match_config_t *matchConfig)`

Configures LPI2C master data match feature.

#### Parameters

- `base` – The LPI2C peripheral base address.
- `matchConfig` – Settings for the data match feature.

`status_t LPI2C_MasterCheckAndClearError(LPI2C_Type *base, uint32_t status)`

Convert provided flags to status code, and clear any errors if present.

#### Parameters

- `base` – The LPI2C peripheral base address.
- `status` – Current status flags value that will be checked.

#### Return values

- `kStatus_Success` –
- `kStatus_LPI2C_PinLowTimeout` –
- `kStatus_LPI2C_ArbitrationLost` –
- `kStatus_LPI2C_Nak` –
- `kStatus_LPI2C_FifoError` –

`status_t LPI2C_CheckForBusyBus(LPI2C_Type *base)`

Make sure the bus isn't already busy.

A busy bus is allowed if we are the one driving it.

#### Parameters

- `base` – The LPI2C peripheral base address.

#### Return values

- `kStatus_Success` –
- `kStatus_LPI2C_Busy` –

`static inline void LPI2C_MasterReset(LPI2C_Type *base)`

Performs a software reset.

Restores the LPI2C master peripheral to reset conditions.

#### Parameters

- `base` – The LPI2C peripheral base address.

```
static inline void LPI2C_MasterEnable(LPI2C_Type *base, bool enable)
```

Enables or disables the LPI2C module as master.

**Parameters**

- `base` – The LPI2C peripheral base address.
- `enable` – Pass true to enable or false to disable the specified LPI2C as master.

```
static inline uint32_t LPI2C_MasterGetStatusFlags(LPI2C_Type *base)
```

Gets the LPI2C master status flags.

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

**See also:**

`_lpi2c_master_flags`

**Parameters**

- `base` – The LPI2C peripheral base address.

**Returns**

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void LPI2C_MasterClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)
```

Clears the LPI2C master status flag state.

The following status register flags can be cleared:

- `kLPI2C_MasterEndOfPacketFlag`
- `kLPI2C_MasterStopDetectFlag`
- `kLPI2C_MasterNackDetectFlag`
- `kLPI2C_MasterArbitrationLostFlag`
- `kLPI2C_MasterFifoErrFlag`
- `kLPI2C_MasterPinLowTimeoutFlag`
- `kLPI2C_MasterDataMatchFlag`

Attempts to clear other flags has no effect.

**See also:**

`_lpi2c_master_flags`.

**Parameters**

- `base` – The LPI2C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_lpi2c_master_flags` enumerators OR'd together. You may pass the result of a previous call to `LPI2C_MasterGetStatusFlags()`.

```
static inline void LPI2C_MasterEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)
```

Enables the LPI2C master interrupt requests.

All flags except `kLPI2C_MasterBusyFlag` and `kLPI2C_MasterBusBusyFlag` can be enabled as interrupts.



**Parameters**

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See `_lpi2c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline void LPI2C_MasterDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)`  
Disables the LPI2C master interrupt requests.

All flags except `kLPI2C_MasterBusyFlag` and `kLPI2C_MasterBusBusyFlag` can be enabled as interrupts.

**Parameters**

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See `_lpi2c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline uint32_t LPI2C_MasterGetEnabledInterrupts(LPI2C_Type *base)`  
Returns the set of currently enabled LPI2C master interrupt requests.

**Parameters**

- base – The LPI2C peripheral base address.

**Returns**

A bitmask composed of `_lpi2c_master_flags` enumerators OR'd together to indicate the set of enabled interrupts.

`static inline void LPI2C_MasterEnableDMA(LPI2C_Type *base, bool enableTx, bool enableRx)`  
Enables or disables LPI2C master DMA requests.

**Parameters**

- base – The LPI2C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.

`static inline uint32_t LPI2C_MasterGetTxFifoAddress(LPI2C_Type *base)`  
Gets LPI2C master transmit data register address for DMA transfer.

**Parameters**

- base – The LPI2C peripheral base address.

**Returns**

The LPI2C Master Transmit Data Register address.

`static inline uint32_t LPI2C_MasterGetRxFifoAddress(LPI2C_Type *base)`  
Gets LPI2C master receive data register address for DMA transfer.

**Parameters**

- base – The LPI2C peripheral base address.

**Returns**

The LPI2C Master Receive Data Register address.

`static inline void LPI2C_MasterSetWatermarks(LPI2C_Type *base, size_t txWords, size_t rxWords)`

Sets the watermarks for LPI2C master FIFOs.

**Parameters**

- `base` – The LPI2C peripheral base address.
- `txWords` – Transmit FIFO watermark value in words. The `kLPI2C_MasterTxReadyFlag` flag is set whenever the number of words in the transmit FIFO is equal or less than `txWords`. Writing a value equal or greater than the FIFO size is truncated.
- `rxWords` – Receive FIFO watermark value in words. The `kLPI2C_MasterRxReadyFlag` flag is set whenever the number of words in the receive FIFO is greater than `rxWords`. Writing a value equal or greater than the FIFO size is truncated.

```
static inline void LPI2C_MasterGetFifoCounts(LPI2C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of words in the LPI2C master FIFOs.

#### Parameters

- `base` – The LPI2C peripheral base address.
- `txCount` – **[out]** Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.

```
void LPI2C_MasterSetBaudRate(LPI2C_Type *base, uint32_t sourceClock_Hz, uint32_t baudRate_Hz)
```

Sets the I2C bus frequency for master transactions.

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

---

**Note:** Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

---

#### Parameters

- `base` – The LPI2C peripheral base address.
- `sourceClock_Hz` – LPI2C functional clock frequency in Hertz.
- `baudRate_Hz` – Requested bus frequency in Hertz.

```
static inline bool LPI2C_MasterGetBusIdleState(LPI2C_Type *base)
```

Returns whether the bus is idle.

Requires the master mode to be enabled.

#### Parameters

- `base` – The LPI2C peripheral base address.

#### Return values

- `true` – Bus is busy.
- `false` – Bus is idle.

```
status_t LPI2C_MasterStart(LPI2C_Type *base, uint8_t address, lpi2c_direction_t dir)
```

Sends a START signal and slave address on the I2C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted,

followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

#### Parameters

- *base* – The LPI2C peripheral base address.
- *address* – 7-bit slave device address, in bits [6:0].
- *dir* – Master transfer direction, either `kLPI2C_Read` or `kLPI2C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

#### Return values

- `kStatus_Success` – START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.

```
static inline status_t LPI2C_MasterRepeatedStart(LPI2C_Type *base, uint8_t address,  
                                                lpi2c_direction_t dir)
```

Sends a repeated START signal and slave address on the I2C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like `LPI2C_MasterStart()`, it also sends the specified 7-bit address.

---

**Note:** This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

---

#### Parameters

- *base* – The LPI2C peripheral base address.
- *address* – 7-bit slave device address, in bits [6:0].
- *dir* – Master transfer direction, either `kLPI2C_Read` or `kLPI2C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

#### Return values

- `kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.

```
status_t LPI2C_MasterSend(LPI2C_Type *base, void *txBuff, size_t txSize)
```

Performs a polling send transfer on the I2C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns `kStatus_LPI2C_Nak`.

#### Parameters

- *base* – The LPI2C peripheral base address.
- *txBuff* – The pointer to the data to be transferred.
- *txSize* – The length in bytes of the data to be transferred.

#### Return values

- `kStatus_Success` – Data was sent successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.

- `kStatus_LPI2C_FifoError` – FIFO under run or over run.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

*status\_t* LPI2C\_MasterReceive(LPI2C\_Type \*base, void \*rxBuff, size\_t rxSize)

Performs a polling receive transfer on the I2C bus.

#### Parameters

- `base` – The LPI2C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.

#### Return values

- `kStatus_Success` – Data was received successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

*status\_t* LPI2C\_MasterStop(LPI2C\_Type \*base)

Sends a STOP signal on the I2C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

#### Parameters

- `base` – The LPI2C peripheral base address.

#### Return values

- `kStatus_Success` – The STOP signal was successfully sent on the bus and the transaction terminated.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

*status\_t* LPI2C\_MasterTransferBlocking(LPI2C\_Type \*base, *lpi2c\_master\_transfer\_t* \*transfer)

Performs a master polling transfer on the I2C bus.

---

**Note:** The API does not return until the transfer succeeds or fails due to error happens during transfer.

---

#### Parameters

- `base` – The LPI2C peripheral base address.
- `transfer` – Pointer to the transfer structure.

**Return values**

- `kStatus_Success` – Data was received successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

```
void LPI2C_MasterTransferCreateHandle(LPI2C_Type *base, lpi2c_master_handle_t *handle,  
                                     lpi2c_master_transfer_callback_t callback, void  
                                     *userData)
```

Creates a new handle for the LPI2C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `LPI2C_MasterTransferAbort()` API shall be called.

---

**Note:** The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

---

**Parameters**

- `base` – The LPI2C peripheral base address.
- `handle` – **[out]** Pointer to the LPI2C master driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

```
status_t LPI2C_MasterTransferNonBlocking(LPI2C_Type *base, lpi2c_master_handle_t *handle,  
                                         lpi2c_master_transfer_t *transfer)
```

Performs a non-blocking transaction on the I2C bus.

**Parameters**

- `base` – The LPI2C peripheral base address.
- `handle` – Pointer to the LPI2C master driver handle.
- `transfer` – The pointer to the transfer descriptor.

**Return values**

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_LPI2C_Busy` – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

```
status_t LPI2C_MasterTransferGetCount(LPI2C_Type *base, lpi2c_master_handle_t *handle,  
                                       size_t *count)
```

Returns number of bytes transferred so far.

**Parameters**

- `base` – The LPI2C peripheral base address.
- `handle` – Pointer to the LPI2C master driver handle.

- `count` – **[out]** Number of bytes transferred so far by the non-blocking transaction.

**Return values**

- `kStatus__Success` –
- `kStatus__NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

`void LPI2C_MasterTransferAbort(LPI2C_Type *base, lpi2c_master_handle_t *handle)`

Terminates a non-blocking LPI2C master transmission early.

---

**Note:** It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

---

**Parameters**

- `base` – The LPI2C peripheral base address.
- `handle` – Pointer to the LPI2C master driver handle.

`void LPI2C_MasterTransferHandleIRQ(LPI2C_Type *base, void *lpi2cMasterHandle)`

Reusable routine to handle master interrupts.

---

**Note:** This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

---

**Parameters**

- `base` – The LPI2C peripheral base address.
- `lpi2cMasterHandle` – Pointer to the LPI2C master driver handle.

`enum _lpi2c_master_flags`

LPI2C master peripheral flags.

The following status register flags can be cleared:

- `kLPI2C_MasterEndOfPacketFlag`
- `kLPI2C_MasterStopDetectFlag`
- `kLPI2C_MasterNackDetectFlag`
- `kLPI2C_MasterArbitrationLostFlag`
- `kLPI2C_MasterFifoErrFlag`
- `kLPI2C_MasterPinLowTimeoutFlag`
- `kLPI2C_MasterDataMatchFlag`

All flags except `kLPI2C_MasterBusyFlag` and `kLPI2C_MasterBusBusyFlag` can be enabled as interrupts.

---

**Note:** These enums are meant to be OR'd together to form a bit mask.

---

*Values:*

enumerator `kLPI2C_MasterTxReadyFlag`

Transmit data flag

enumerator kLPI2C\_MasterRxReadyFlag

Receive data flag

enumerator kLPI2C\_MasterEndOfPacketFlag

End Packet flag

enumerator kLPI2C\_MasterStopDetectFlag

Stop detect flag

enumerator kLPI2C\_MasterNackDetectFlag

NACK detect flag

enumerator kLPI2C\_MasterArbitrationLostFlag

Arbitration lost flag

enumerator kLPI2C\_MasterFifoErrFlag

FIFO error flag

enumerator kLPI2C\_MasterPinLowTimeoutFlag

Pin low timeout flag

enumerator kLPI2C\_MasterDataMatchFlag

Data match flag

enumerator kLPI2C\_MasterBusyFlag

Master busy flag

enumerator kLPI2C\_MasterBusBusyFlag

Bus busy flag

enumerator kLPI2C\_MasterClearFlags

All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C\_MasterIrqFlags

IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C\_MasterErrorFlags

Errors to check for.

enum \_lpi2c\_direction

Direction of master and slave transfers.

*Values:*

enumerator kLPI2C\_Write

Master transmit.

enumerator kLPI2C\_Read

Master receive.

enum \_lpi2c\_master\_pin\_config

LPI2C pin configuration.

*Values:*

enumerator kLPI2C\_2PinOpenDrain

LPI2C Configured for 2-pin open drain mode

enumerator kLPI2C\_2PinOutputOnly

LPI2C Configured for 2-pin output only mode (ultra-fast mode)

enumerator kLPI2C\_2PinPushPull

LPI2C Configured for 2-pin push-pull mode

enumerator kLPI2C\_4PinPushPull

LPI2C Configured for 4-pin push-pull mode

enumerator kLPI2C\_2PinOpenDrainWithSeparateSlave

LPI2C Configured for 2-pin open drain mode with separate LPI2C slave

enumerator kLPI2C\_2PinOutputOnlyWithSeparateSlave

LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave

enumerator kLPI2C\_2PinPushPullWithSeparateSlave

LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave

enumerator kLPI2C\_4PinPushPullWithInvertedOutput

LPI2C Configured for 4-pin push-pull mode(inverted outputs)

enum \_lpi2c\_host\_request\_source

LPI2C master host request selection.

*Values:*

enumerator kLPI2C\_HostRequestExternalPin

Select the LPI2C\_HREQ pin as the host request input

enumerator kLPI2C\_HostRequestInputTrigger

Select the input trigger as the host request input

enum \_lpi2c\_host\_request\_polarity

LPI2C master host request pin polarity configuration.

*Values:*

enumerator kLPI2C\_HostRequestPinActiveLow

Configure the LPI2C\_HREQ pin active low

enumerator kLPI2C\_HostRequestPinActiveHigh

Configure the LPI2C\_HREQ pin active high

enum \_lpi2c\_data\_match\_config\_mode

LPI2C master data match configuration modes.

*Values:*

enumerator kLPI2C\_MatchDisabled

LPI2C Match Disabled

enumerator kLPI2C\_1stWordEqualsM0OrM1

LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1

enumerator kLPI2C\_AnyWordEqualsM0OrM1

LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1

enumerator kLPI2C\_1stWordEqualsM0And2ndWordEqualsM1

LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1

enumerator kLPI2C\_AnyWordEqualsM0AndNextWordEqualsM1

LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1

enumerator kLPI2C\_1stWordAndM1EqualsM0AndM1

LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1

enumerator kLPI2C\_AnyWordAndM1EqualsM0AndM1

LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1



enum `_lpi2c_master_transfer_flags`

Transfer option flags.

---

**Note:** These enumerations are intended to be OR'd together to form a bit mask of options for the `_lpi2c_master_transfer::flags` field.

---

*Values:*

enumerator `kLPI2C_TransferDefaultFlag`

Transfer starts with a start signal, stops with a stop signal.

enumerator `kLPI2C_TransferNoStartFlag`

Don't send a start condition, address, and sub address

enumerator `kLPI2C_TransferRepeatedStartFlag`

Send a repeated start condition

enumerator `kLPI2C_TransferNoStopFlag`

Don't send a stop condition.

typedef enum `_lpi2c_direction` `lpi2c_direction_t`

Direction of master and slave transfers.

typedef enum `_lpi2c_master_pin_config` `lpi2c_master_pin_config_t`

LPI2C pin configuration.

typedef enum `_lpi2c_host_request_source` `lpi2c_host_request_source_t`

LPI2C master host request selection.

typedef enum `_lpi2c_host_request_polarity` `lpi2c_host_request_polarity_t`

LPI2C master host request pin polarity configuration.

typedef struct `_lpi2c_master_config` `lpi2c_master_config_t`

Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the `LPI2C_MasterGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum `_lpi2c_data_match_config_mode` `lpi2c_data_match_config_mode_t`

LPI2C master data match configuration modes.

typedef struct `_lpi2c_match_config` `lpi2c_data_match_config_t`

LPI2C master data match configuration structure.

typedef struct `_lpi2c_master_transfer` `lpi2c_master_transfer_t`

LPI2C master descriptor of the transfer.

typedef struct `_lpi2c_master_handle` `lpi2c_master_handle_t`

LPI2C master handle of the transfer.

typedef void (\*`lpi2c_master_transfer_callback_t`)(`LPI2C_Type` \*`base`, `lpi2c_master_handle_t` \*`handle`, `status_t` `completionStatus`, void \*`userData`)

Master completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to `LPI2C_MasterTransferCreateHandle()`.

**Param base**

The LPI2C peripheral base address.

**Param handle**

Pointer to the LPI2C master driver handle.

**Param completionStatus**

Either kStatus\_Success or an error code describing how the transfer completed.

**Param userData**

Arbitrary pointer-sized value passed from the application.

```
typedef void (*lpi2c_master_isr_t)(LPI2C_Type *base, void *handle)
```

Typedef for master interrupt handler, used internally for LPI2C master interrupt and EDMA transactional APIs.

```
struct _lpi2c_master_config
```

*#include <fsl\_lpi2c.h>* Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C\_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

**Public Members**

```
bool enableMaster
```

Whether to enable master mode.

```
bool enableDoze
```

Whether master is enabled in doze mode.

```
bool debugEnable
```

Enable transfers to continue when halted in debug mode.

```
bool ignoreAck
```

Whether to ignore ACK/NACK.

```
lpi2c_master_pin_config_t pinConfig
```

The pin configuration option.

```
uint32_t baudRate_Hz
```

Desired baud rate in Hertz.

```
uint32_t busIdleTimeout_ns
```

Bus idle timeout in nanoseconds. Set to 0 to disable.

```
uint32_t pinLowTimeout_ns
```

Pin low timeout in nanoseconds. Set to 0 to disable.

```
uint8_t sdaGlitchFilterWidth_ns
```

Width in nanoseconds of glitch filter on SDA pin. Set to 0 to disable.

```
uint8_t sclGlitchFilterWidth_ns
```

Width in nanoseconds of glitch filter on SCL pin. Set to 0 to disable.

```
struct _lpi2c_master_config hostRequest
```

Host request options.

```
struct _lpi2c_match_config
```

*#include <fsl\_lpi2c.h>* LPI2C master data match configuration structure.

### Public Members

*lpi2c\_data\_match\_config\_mode\_t* matchMode

Data match configuration setting.

bool rxDataMatchOnly

When set to true, received data is ignored until a successful match.

uint32\_t match0

Match value 0.

uint32\_t match1

Match value 1.

struct *\_lpi2c\_master\_transfer*

*#include <fsl\_lpi2c.h>* Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the LPI2C\_MasterTransferNonBlocking() API.

### Public Members

uint32\_t flags

Bit mask of options for the transfer. See enumeration *\_lpi2c\_master\_transfer\_flags* for available options. Set to 0 or *kLPI2C\_TransferDefaultFlag* for normal transfers.

uint16\_t slaveAddress

The 7-bit slave address.

*lpi2c\_direction\_t* direction

Either *kLPI2C\_Read* or *kLPI2C\_Write*.

uint32\_t subaddress

Sub address. Transferred MSB first.

size\_t subaddressSize

Length of sub address to send in bytes. Maximum size is 4 bytes.

void \*data

Pointer to data to transfer.

size\_t dataSize

Number of bytes to transfer.

struct *\_lpi2c\_master\_handle*

*#include <fsl\_lpi2c.h>* Driver handle for master non-blocking APIs.

---

**Note:** The contents of this structure are private and subject to change.

---

### Public Members

uint8\_t state

Transfer state machine current state.

uint16\_t remainingBytes

Remaining byte count in current state.

uint8\_t \*buf

Buffer pointer for current state.

uint16\_t commandBuffer[6]

LPI2C command sequence. When all 6 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word]

*lpi2c\_master\_transfer\_t* transfer

Copy of the current transfer info.

*lpi2c\_master\_transfer\_callback\_t* completionCallback

Callback function pointer.

void \*userData

Application data passed to callback.

struct hostRequest

### Public Members

bool enable

Enable host request.

*lpi2c\_host\_request\_source\_t* source

Host request source.

*lpi2c\_host\_request\_polarity\_t* polarity

Host request pin polarity.

## 2.32 LPI2C Master DMA Driver

```
void LPI2C_MasterCreateEDMAHandle(LPI2C_Type *base, lpi2c_master_edma_handle_t *handle,
                                   edma_handle_t *rxDmaHandle, edma_handle_t
                                   *txDmaHandle, lpi2c_master_edma_transfer_callback_t
                                   callback, void *userData)
```

Create a new handle for the LPI2C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C\_MasterTransferAbortEDMA() API shall be called.

For devices where the LPI2C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

### Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- rxDmaHandle – Handle for the eDMA receive channel. Created by the user prior to calling this function.
- txDmaHandle – Handle for the eDMA transmit channel. Created by the user prior to calling this function.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

*status\_t* LPI2C\_MasterTransferEDMA(LPI2C\_Type \*base, *lpi2c\_master\_edma\_handle\_t* \*handle, *lpi2c\_master\_transfer\_t* \*transfer)

Performs a non-blocking DMA-based transaction on the I2C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

#### Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

#### Return values

- kStatus\_Success – The transaction was started successfully.
- kStatus\_LPI2C\_Busy – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

*status\_t* LPI2C\_MasterTransferGetCountEDMA(LPI2C\_Type \*base, *lpi2c\_master\_edma\_handle\_t* \*handle, *size\_t* \*count)

Returns number of bytes transferred so far.

#### Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

#### Return values

- kStatus\_Success –
- kStatus\_NoTransferInProgress – There is not a DMA transaction currently in progress.

*status\_t* LPI2C\_MasterTransferAbortEDMA(LPI2C\_Type \*base, *lpi2c\_master\_edma\_handle\_t* \*handle)

Terminates a non-blocking LPI2C master transmission early.

---

**Note:** It is not safe to call this function from an IRQ handler that has a higher priority than the eDMA peripheral's IRQ priority.

---

#### Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

#### Return values

- kStatus\_Success – A transaction was successfully aborted.
- kStatus\_LPI2C\_Idle – There is not a DMA transaction currently in progress.

typedef struct *lpi2c\_master\_edma\_handle* *lpi2c\_master\_edma\_handle\_t*  
LPI2C master EDMA handle of the transfer.

```
typedef void (*lpi2c_master_edma_transfer_callback_t)(LPI2C_Type *base,
lpi2c_master_edma_handle_t *handle, status_t completionStatus, void *userData)
```

Master DMA completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C\_MasterCreateEDMAHandle().

**Param base**

The LPI2C peripheral base address.

**Param handle**

Handle associated with the completed transfer.

**Param completionStatus**

Either kStatus\_Success or an error code describing how the transfer completed.

**Param userData**

Arbitrary pointer-sized value passed from the application.

```
struct _lpi2c_master_edma_handle
#include <fsl_lpi2c_edma.h> Driver handle for master DMA APIs.
```

---

**Note:** The contents of this structure are private and subject to change.

---

## Public Members

LPI2C\_Type \*base

LPI2C base pointer.

bool isBusy

Transfer state machine current state.

uint8\_t nbytes

eDMA minor byte transfer count initially configured.

uint16\_t commandBuffer[20]

LPI2C command sequence. When all 10 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word] + receive&Size[4 words]

*lpi2c\_master\_transfer\_t* transfer

Copy of the current transfer info.

*lpi2c\_master\_edma\_transfer\_callback\_t* completionCallback

Callback function pointer.

void \*userData

Application data passed to callback.

*edma\_handle\_t* \*rx

Handle for receive DMA channel.

*edma\_handle\_t* \*tx

Handle for transmit DMA channel.

*edma\_tcd\_t* tcds[3]

Software TCD. Three are allocated to provide enough room to align to 32-bytes.

## 2.33 LPI2C Slave Driver

`void LPI2C_SlaveGetDefaultConfig(lpi2c_slave_config_t *slaveConfig)`

Provides a default configuration for the LPI2C slave peripheral.

This function provides the following default configuration for the LPI2C slave peripheral:

```
slaveConfig->enableSlave      = true;
slaveConfig->address0         = 0U;
slaveConfig->address1         = 0U;
slaveConfig->addressMatchMode = kLPI2C_MatchAddress0;
slaveConfig->filterDozeEnable  = true;
slaveConfig->filterEnable     = true;
slaveConfig->enableGeneralCall = false;
slaveConfig->sclStall.enableAck = false;
slaveConfig->sclStall.enableTx  = true;
slaveConfig->sclStall.enableRx  = true;
slaveConfig->sclStall.enableAddress = true;
slaveConfig->ignoreAck         = false;
slaveConfig->enableReceivedAddressRead = false;
slaveConfig->sdaGlitchFilterWidth_ns = 0;
slaveConfig->sclGlitchFilterWidth_ns = 0;
slaveConfig->dataValidDelay_ns  = 0;
slaveConfig->clockHoldTime_ns   = 0;
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with `LPI2C_SlaveInit()`. Be sure to override at least the *address0* member of the configuration structure with the desired slave address.

### Parameters

- *slaveConfig* – **[out]** User provided configuration structure that is set to default values. Refer to `lpi2c_slave_config_t`.

`void LPI2C_SlaveInit(LPI2C_Type *base, const lpi2c_slave_config_t *slaveConfig, uint32_t sourceClock_Hz)`

Initializes the LPI2C slave peripheral.

This function enables the peripheral clock and initializes the LPI2C slave peripheral as described by the user provided configuration.

### Parameters

- *base* – The LPI2C peripheral base address.
- *slaveConfig* – User provided peripheral configuration. Use `LPI2C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- *sourceClock\_Hz* – Frequency in Hertz of the LPI2C functional clock. Used to calculate the filter widths, data valid delay, and clock hold time.

`void LPI2C_SlaveDeinit(LPI2C_Type *base)`

Deinitializes the LPI2C slave peripheral.

This function disables the LPI2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

### Parameters

- *base* – The LPI2C peripheral base address.

`static inline void LPI2C_SlaveReset(LPI2C_Type *base)`

Performs a software reset of the LPI2C slave peripheral.

### Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C\_SlaveEnable(LPI2C\_Type \*base, bool enable)

Enables or disables the LPI2C module as slave.

#### Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as slave.

static inline uint32\_t LPI2C\_SlaveGetStatusFlags(LPI2C\_Type \*base)

Gets the LPI2C slave status flags.

A bit mask with the state of all LPI2C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

#### See also:

[\\_lpi2c\\_slave\\_flags](#)

#### Parameters

- base – The LPI2C peripheral base address.

#### Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C\_SlaveClearStatusFlags(LPI2C\_Type \*base, uint32\_t statusMask)

Clears the LPI2C status flag state.

The following status register flags can be cleared:

- kLPI2C\_SlaveRepeatedStartDetectFlag
- kLPI2C\_SlaveStopDetectFlag
- kLPI2C\_SlaveBitErrFlag
- kLPI2C\_SlaveFifoErrFlag

Attempts to clear other flags has no effect.

#### See also:

[\\_lpi2c\\_slave\\_flags](#).

#### Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of [\\_lpi2c\\_slave\\_flags](#) enumerators OR'd together. You may pass the result of a previous call to [LPI2C\\_SlaveGetStatusFlags\(\)](#).

static inline void LPI2C\_SlaveEnableInterrupts(LPI2C\_Type \*base, uint32\_t interruptMask)

Enables the LPI2C slave interrupt requests.

All flags except kLPI2C\_SlaveBusyFlag and kLPI2C\_SlaveBusBusyFlag can be enabled as interrupts.

#### Parameters

- base – The LPI2C peripheral base address.



- `interruptMask` – Bit mask of interrupts to enable. See `_lpi2c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline void LPI2C_SlaveDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)`

Disables the LPI2C slave interrupt requests.

All flags except `kLPI2C_SlaveBusyFlag` and `kLPI2C_SlaveBusBusyFlag` can be enabled as interrupts.

#### Parameters

- `base` – The LPI2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_lpi2c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline uint32_t LPI2C_SlaveGetEnabledInterrupts(LPI2C_Type *base)`

Returns the set of currently enabled LPI2C slave interrupt requests.

#### Parameters

- `base` – The LPI2C peripheral base address.

#### Returns

A bitmask composed of `_lpi2c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

`static inline void LPI2C_SlaveEnableDMA(LPI2C_Type *base, bool enableAddressValid, bool enableRx, bool enableTx)`

Enables or disables the LPI2C slave peripheral DMA requests.

#### Parameters

- `base` – The LPI2C peripheral base address.
- `enableAddressValid` – Enable flag for the address valid DMA request. Pass true for enable, false for disable. The address valid DMA request is shared with the receive data DMA request.
- `enableRx` – Enable flag for the receive data DMA request. Pass true for enable, false for disable.
- `enableTx` – Enable flag for the transmit data DMA request. Pass true for enable, false for disable.

`static inline bool LPI2C_SlaveGetBusIdleState(LPI2C_Type *base)`

Returns whether the bus is idle.

Requires the slave mode to be enabled.

#### Parameters

- `base` – The LPI2C peripheral base address.

#### Return values

- `true` – Bus is busy.
- `false` – Bus is idle.

`static inline void LPI2C_SlaveTransmitAck(LPI2C_Type *base, bool ackOrNack)`

Transmits either an ACK or NAK on the I2C bus in response to a byte from the master.

Use this function to send an ACK or NAK when the `kLPI2C_SlaveTransmitAckFlag` is asserted. This only happens if you enable the `sclStall.enableAck` field of the `lpi2c_slave_config_t` configuration structure used to initialize the slave peripheral.

#### Parameters

- `base` – The LPI2C peripheral base address.

- `ackOrNack` – Pass true for an ACK or false for a NAK.

`static inline void LPI2C_SlaveEnableAckStall(LPI2C_Type *base, bool enable)`

Enables or disables ACKSTALL.

When enables ACKSTALL, software can transmit either an ACK or NAK on the I2C bus in response to a byte from the master.

#### Parameters

- `base` – The LPI2C peripheral base address.
- `enable` – True will enable ACKSTALL, false will disable ACKSTALL.

`static inline uint32_t LPI2C_SlaveGetReceivedAddress(LPI2C_Type *base)`

Returns the slave address sent by the I2C master.

This function should only be called if the `kLPI2C_SlaveAddressValidFlag` is asserted.

#### Parameters

- `base` – The LPI2C peripheral base address.

#### Returns

The 8-bit address matched by the LPI2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

`status_t LPI2C_SlaveSend(LPI2C_Type *base, void *txBuff, size_t txSize, size_t *actualTxSize)`

Performs a polling send transfer on the I2C bus.

#### Parameters

- `base` – The LPI2C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.
- `actualTxSize` – **[out]**

#### Returns

Error or success status returned by API.

`status_t LPI2C_SlaveReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize, size_t *actualRxSize)`

Performs a polling receive transfer on the I2C bus.

#### Parameters

- `base` – The LPI2C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.
- `actualRxSize` – **[out]**

#### Returns

Error or success status returned by API.

`void LPI2C_SlaveTransferCreateHandle(LPI2C_Type *base, lpi2c_slave_handle_t *handle, lpi2c_slave_transfer_callback_t callback, void *userData)`

Creates a new handle for the LPI2C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `LPI2C_SlaveTransferAbort()` API shall be called.

---

**Note:** The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

---

### Parameters

- `base` – The LPI2C peripheral base address.
- `handle` – **[out]** Pointer to the LPI2C slave driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

*status\_t* LPI2C\_SlaveTransferNonBlocking(LPI2C\_Type \*base, *lpi2c\_slave\_handle\_t* \*handle, uint32\_t eventMask)

Starts accepting slave transfers.

Call this API after calling I2C\_SlaveInit() and LPI2C\_SlaveTransferCreateHandle() to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to LPI2C\_SlaveTransferCreateHandle(). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of *lpi2c\_slave\_transfer\_event\_t* enumerators for the events you wish to receive. The *kLPI2C\_SlaveTransmitEvent* and *kLPI2C\_SlaveReceiveEvent* events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the *kLPI2C\_SlaveAllEvents* constant is provided as a convenient way to enable all events.

### Parameters

- `base` – The LPI2C peripheral base address.
- `handle` – Pointer to *lpi2c\_slave\_handle\_t* structure which stores the transfer state.
- `eventMask` – Bit mask formed by OR'ing together *lpi2c\_slave\_transfer\_event\_t* enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and *kLPI2C\_SlaveAllEvents* to enable all events.

### Return values

- *kStatus\_Success* – Slave transfers were successfully started.
- *kStatus\_LPI2C\_Busy* – Slave transfers have already been started on this handle.

*status\_t* LPI2C\_SlaveTransferGetCount(LPI2C\_Type \*base, *lpi2c\_slave\_handle\_t* \*handle, size\_t \*count)

Gets the slave transfer status during a non-blocking transfer.

### Parameters

- `base` – The LPI2C peripheral base address.
- `handle` – Pointer to *i2c\_slave\_handle\_t* structure.
- `count` – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

### Return values

- kStatus\_Success –
- kStatus\_NoTransferInProgress –

void LPI2C\_SlaveTransferAbort(LPI2C\_Type \*base, *lpi2c\_slave\_handle\_t* \*handle)

Aborts the slave non-blocking transfers.

---

**Note:** This API could be called at any time to stop slave for handling the bus events.

---

#### Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to *lpi2c\_slave\_handle\_t* structure which stores the transfer state.

void LPI2C\_SlaveTransferHandleIRQ(LPI2C\_Type \*base, *lpi2c\_slave\_handle\_t* \*handle)

Reusable routine to handle slave interrupts.

---

**Note:** This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

---

#### Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to *lpi2c\_slave\_handle\_t* structure which stores the transfer state.

enum *\_lpi2c\_slave\_flags*

LPI2C slave peripheral flags.

The following status register flags can be cleared:

- kLPI2C\_SlaveRepeatedStartDetectFlag
- kLPI2C\_SlaveStopDetectFlag
- kLPI2C\_SlaveBitErrFlag
- kLPI2C\_SlaveFifoErrFlag

All flags except kLPI2C\_SlaveBusyFlag and kLPI2C\_SlaveBusBusyFlag can be enabled as interrupts.

---

**Note:** These enumerations are meant to be OR'd together to form a bit mask.

---

*Values:*

enumerator kLPI2C\_SlaveTxReadyFlag

Transmit data flag

enumerator kLPI2C\_SlaveRxReadyFlag

Receive data flag

enumerator kLPI2C\_SlaveAddressValidFlag

Address valid flag

enumerator kLPI2C\_SlaveTransmitAckFlag

Transmit ACK flag

enumerator kLPI2C\_SlaveRepeatedStartDetectFlag

Repeated start detect flag

enumerator kLPI2C\_SlaveStopDetectFlag

Stop detect flag

enumerator kLPI2C\_SlaveBitErrFlag

Bit error flag

enumerator kLPI2C\_SlaveFifoErrFlag

FIFO error flag

enumerator kLPI2C\_SlaveAddressMatch0Flag

Address match 0 flag

enumerator kLPI2C\_SlaveAddressMatch1Flag

Address match 1 flag

enumerator kLPI2C\_SlaveGeneralCallFlag

General call flag

enumerator kLPI2C\_SlaveBusyFlag

Master busy flag

enumerator kLPI2C\_SlaveBusBusyFlag

Bus busy flag

enumerator kLPI2C\_SlaveClearFlags

All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C\_SlaveIrqFlags

IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C\_SlaveErrorFlags

Errors to check for.

enum \_lpi2c\_slave\_address\_match

LPI2C slave address match options.

*Values:*

enumerator kLPI2C\_MatchAddress0

Match only address 0.

enumerator kLPI2C\_MatchAddress0OrAddress1

Match either address 0 or address 1.

enumerator kLPI2C\_MatchAddress0ThroughAddress1

Match a range of slave addresses from address 0 through address 1.

enum \_lpi2c\_slave\_transfer\_event

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C\_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

---

**Note:** These enumerations are meant to be OR'd together to form a bit mask of events.

---

*Values:*

enumerator kLPI2C\_SlaveAddressMatchEvent

Received the slave address after a start or repeated start.

enumerator kLPI2C\_SlaveTransmitEvent

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator kLPI2C\_SlaveReceiveEvent

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kLPI2C\_SlaveTransmitAckEvent

Callback needs to either transmit an ACK or NACK.

enumerator kLPI2C\_SlaveRepeatedStartEvent

A repeated start was detected.

enumerator kLPI2C\_SlaveCompletionEvent

A stop was detected, completing the transfer.

enumerator kLPI2C\_SlaveAllEvents

Bit mask of all available events.

typedef enum *\_lpi2c\_slave\_address\_match* lpi2c\_slave\_address\_match\_t

LPI2C slave address match options.

typedef struct *\_lpi2c\_slave\_config* lpi2c\_slave\_config\_t

Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the `LPI2C_SlaveGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *\_lpi2c\_slave\_transfer\_event* lpi2c\_slave\_transfer\_event\_t

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to `LPI2C_SlaveTransferNonBlocking()` in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

---

**Note:** These enumerations are meant to be OR'd together to form a bit mask of events.

---

typedef struct *\_lpi2c\_slave\_transfer* lpi2c\_slave\_transfer\_t

LPI2C slave transfer structure.

typedef struct *\_lpi2c\_slave\_handle* lpi2c\_slave\_handle\_t

LPI2C slave handle structure.

typedef void (\*lpi2c\_slave\_transfer\_callback\_t)(LPI2C\_Type \*base, lpi2c\_slave\_transfer\_t \*transfer, void \*userData)

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the `LPI2C_SlaveSetCallback()` function after you have created a handle.

**Param base**

Base address for the LPI2C instance on which the event occurred.

**Param transfer**

Pointer to transfer descriptor containing values passed to and/or from the callback.

**Param userData**

Arbitrary pointer-sized value passed from the application.

struct `_lpi2c_slave_config`

*#include <fsl\_lpi2c.h>* Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the `LPI2C_SlaveGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

**Public Members**

bool `enableSlave`

Enable slave mode.

uint8\_t `address0`

Slave's 7-bit address.

uint8\_t `address1`

Alternate slave 7-bit address.

*lpi2c\_slave\_address\_match\_t* `addressMatchMode`

Address matching options.

bool `filterDozeEnable`

Enable digital glitch filter in doze mode.

bool `filterEnable`

Enable digital glitch filter.

bool `enableGeneralCall`

Enable general call address matching.

struct *\_lpi2c\_slave\_config* `sclStall`

SCL stall enable options.

bool `ignoreAck`

Continue transfers after a NACK is detected.

bool `enableReceivedAddressRead`

Enable reading the address received address as the first byte of data.

uint32\_t `sdaGlitchFilterWidth_ns`

Width in nanoseconds of the digital filter on the SDA signal. Set to 0 to disable.

uint32\_t `sclGlitchFilterWidth_ns`

Width in nanoseconds of the digital filter on the SCL signal. Set to 0 to disable.

uint32\_t `dataValidDelay_ns`

Width in nanoseconds of the data valid delay.

uint32\_t `clockHoldTime_ns`

Width in nanoseconds of the clock hold time.

struct `_lpi2c_slave_transfer`

*#include <fsl\_lpi2c.h>* LPI2C slave transfer structure.

### Public Members

*lpi2c\_slave\_transfer\_event\_t* event

Reason the callback is being invoked.

uint8\_t receivedAddress

Matching address send by master.

uint8\_t \*data

Transfer buffer

size\_t dataSize

Transfer size

*status\_t* completionStatus

Success or error code describing how the transfer completed. Only applies for kLPI2C\_SlaveCompletionEvent.

size\_t transferredCount

Number of bytes actually transferred since start or last repeated start.

struct \_lpi2c\_slave\_handle

#include <fsl\_lpi2c.h> LPI2C slave handle structure.

---

**Note:** The contents of this structure are private and subject to change.

---

### Public Members

*lpi2c\_slave\_transfer\_t* transfer

LPI2C slave transfer copy.

bool isBusy

Whether transfer is busy.

bool wasTransmit

Whether the last transfer was a transmit.

uint32\_t eventMask

Mask of enabled events.

uint32\_t transferredCount

Count of bytes transferred.

*lpi2c\_slave\_transfer\_callback\_t* callback

Callback function called at transfer event.

void \*userData

Callback parameter passed to callback.

struct sclStall

### Public Members

bool enableAck

Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted. Clock stretching occurs when transmitting the 9th bit. When enableAckSCLStall is enabled, there is no need to set either enableRxDataSCLStall or enableAddressSCLStall.



bool enableTx

Enables SCL clock stretching when the transmit data flag is set during a slave-transmit transfer.

bool enableRx

Enables SCL clock stretching when receive data flag is set during a slave-receive transfer.

bool enableAddress

Enables SCL clock stretching when the address valid flag is asserted.

## 2.34 LPSPI: Low Power Serial Peripheral Interface

### 2.35 LPSPI Peripheral driver

```
void LPSPI_MasterInit(LPSPI_Type *base, const lpspi_master_config_t *masterConfig, uint32_t  
srcClock_Hz)
```

Initializes the LPSPI master.

#### Parameters

- base – LPSPI peripheral address.
- masterConfig – Pointer to structure *lpspi\_master\_config\_t*.
- srcClock\_Hz – Module source input clock in Hertz

```
void LPSPI_MasterGetDefaultConfig(lpspi_master_config_t *masterConfig)
```

Sets the *lpspi\_master\_config\_t* structure to default values.

This API initializes the configuration structure for LPSPI\_MasterInit(). The initialized structure can remain unchanged in LPSPI\_MasterInit(), or can be modified before calling the LPSPI\_MasterInit(). Example:

```
lpspi_master_config_t masterConfig;  
LPSPI_MasterGetDefaultConfig(&masterConfig);
```

#### Parameters

- masterConfig – pointer to *lpspi\_master\_config\_t* structure

```
void LPSPI_SlaveInit(LPSPI_Type *base, const lpspi_slave_config_t *slaveConfig)
```

LPSPI slave configuration.

#### Parameters

- base – LPSPI peripheral address.
- slaveConfig – Pointer to a structure *lpspi\_slave\_config\_t*.

```
void LPSPI_SlaveGetDefaultConfig(lpspi_slave_config_t *slaveConfig)
```

Sets the *lpspi\_slave\_config\_t* structure to default values.

This API initializes the configuration structure for LPSPI\_SlaveInit(). The initialized structure can remain unchanged in LPSPI\_SlaveInit() or can be modified before calling the LPSPI\_SlaveInit(). Example:

```
lpspi_slave_config_t slaveConfig;  
LPSPI_SlaveGetDefaultConfig(&slaveConfig);
```

#### Parameters

- slaveConfig – pointer to lpspi\_slave\_config\_t structure.

void LPSPI\_Deinit(LPSPI\_Type \*base)

De-initializes the LPSPI peripheral. Call this API to disable the LPSPI clock.

**Parameters**

- base – LPSPI peripheral address.

void LPSPI\_Reset(LPSPI\_Type \*base)

Restores the LPSPI peripheral to reset state. Note that this function sets all registers to reset state. As a result, the LPSPI module can't work after calling this API.

**Parameters**

- base – LPSPI peripheral address.

uint32\_t LPSPI\_GetInstance(LPSPI\_Type \*base)

Get the LPSPI instance from peripheral base address.

**Parameters**

- base – LPSPI peripheral base address.

**Returns**

LPSPI instance.

static inline void LPSPI\_Enable(LPSPI\_Type \*base, bool enable)

Enables the LPSPI peripheral and sets the MCR MDIS to 0.

**Parameters**

- base – LPSPI peripheral address.
- enable – Pass true to enable module, false to disable module.

static inline uint32\_t LPSPI\_GetStatusFlags(LPSPI\_Type \*base)

Gets the LPSPI status flag state.

**Parameters**

- base – LPSPI peripheral address.

**Returns**

The LPSPI status(in SR register).

static inline uint8\_t LPSPI\_GetTxFifoSize(LPSPI\_Type \*base)

Gets the LPSPI Tx FIFO size.

**Parameters**

- base – LPSPI peripheral address.

**Returns**

The LPSPI Tx FIFO size.

static inline uint8\_t LPSPI\_GetRxFifoSize(LPSPI\_Type \*base)

Gets the LPSPI Rx FIFO size.

**Parameters**

- base – LPSPI peripheral address.

**Returns**

The LPSPI Rx FIFO size.

static inline uint32\_t LPSPI\_GetTxFifoCount(LPSPI\_Type \*base)

Gets the LPSPI Tx FIFO count.

**Parameters**

- base – LPSPI peripheral address.

**Returns**

The number of words in the transmit FIFO.

```
static inline uint32_t LPSPI_GetRxFifoCount(LPSPI_Type *base)
```

Gets the LPSPI Rx FIFO count.

**Parameters**

- base – LPSPI peripheral address.

**Returns**

The number of words in the receive FIFO.

```
static inline void LPSPI_ClearStatusFlags(LPSPI_Type *base, uint32_t statusFlags)
```

Clears the LPSPI status flag.

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status flag bit to clear. The list of status flags is defined in the `_lpspi_flags`. Example usage:

```
LPSPI_ClearStatusFlags(base, kLPSPI_TxDataRequestFlag|kLPSPI_RxDataReadyFlag);
```

**Parameters**

- base – LPSPI peripheral address.
- statusFlags – The status flag used from type `_lpspi_flags`.

```
static inline uint32_t LPSPI_GetTcr(LPSPI_Type *base)
```

```
static inline void LPSPI_EnableInterrupts(LPSPI_Type *base, uint32_t mask)
```

Enables the LPSPI interrupts.

This function configures the various interrupt masks of the LPSPI. The parameters are base and an interrupt mask. Note that, for Tx fill and Rx FIFO drain requests, enabling the interrupt request disables the DMA request.

```
LPSPI_EnableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

**Parameters**

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_interrupt_enable`.

```
static inline void LPSPI_DisableInterrupts(LPSPI_Type *base, uint32_t mask)
```

Disables the LPSPI interrupts.

```
LPSPI_DisableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

**Parameters**

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_interrupt_enable`.

```
static inline void LPSPI_EnableDMA(LPSPI_Type *base, uint32_t mask)
```

Enables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
LPSPI_EnableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

**Parameters**

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_dma_enable`.

static inline void LPSPI\_DisableDMA(LPSPI\_Type \*base, uint32\_t mask)

Disables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
SPI_DisableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

**Parameters**

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_dma_enable`.

static inline uint32\_t LPSPI\_GetTxRegisterAddress(LPSPI\_Type \*base)

Gets the LPSPI Transmit Data Register address for a DMA operation.

This function gets the LPSPI Transmit Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

**Parameters**

- base – LPSPI peripheral address.

**Returns**

The LPSPI Transmit Data Register address.

static inline uint32\_t LPSPI\_GetRxRegisterAddress(LPSPI\_Type \*base)

Gets the LPSPI Receive Data Register address for a DMA operation.

This function gets the LPSPI Receive Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

**Parameters**

- base – LPSPI peripheral address.

**Returns**

The LPSPI Receive Data Register address.

bool LPSPI\_CheckTransferArgument(LPSPI\_Type \*base, *lpspi\_transfer\_t* \*transfer, bool isEdma)

Check the argument for transfer .

**Parameters**

- base – LPSPI peripheral address.
- transfer – the transfer struct to be used.
- isEdma – True to check for EDMA transfer, false to check interrupt non-blocking transfer

**Returns**

Return true for right and false for wrong.

static inline void LPSPI\_SetMasterSlaveMode(LPSPI\_Type \*base, *lpspi\_master\_slave\_mode\_t* mode)

Configures the LPSPI for either master or slave.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx\_CR\_MEN = 0).

**Parameters**

- base – LPSPI peripheral address.
- mode – Mode setting (master or slave) of type `lpspi_master_slave_mode_t`.

static inline void LPSPI\_SelectTransferPCS(LPSPI\_Type \*base, *lpspi\_which\_pcs\_t* select)

Configures the peripheral chip select used for the transfer.

#### Parameters

- base – LPSPI peripheral address.
- select – LPSPI Peripheral Chip Select (PCS) configuration.

static inline void LPSPI\_SetPCSContinuous(LPSPI\_Type \*base, bool IsContinuous)

Set the PCS signal to continuous or uncontinuous mode.

---

**Note:** In master mode, continuous transfer will keep the PCS asserted at the end of the frame size, until a command word is received that starts a new frame. So PCS must be set back to uncontinuous when transfer finishes. In slave mode, when continuous transfer is enabled, the LPSPI will only transmit the first frame size bits, after that the LPSPI will transmit received data back (assuming a 32-bit shift register).

---

#### Parameters

- base – LPSPI peripheral address.
- IsContinuous – True to set the transfer PCS to continuous mode, false to set to uncontinuous mode.

static inline bool LPSPI\_IsMaster(LPSPI\_Type \*base)

Returns whether the LPSPI module is in master mode.

#### Parameters

- base – LPSPI peripheral address.

#### Returns

Returns true if the module is in master mode or false if the module is in slave mode.

static inline void LPSPI\_FlushFifo(LPSPI\_Type \*base, bool flushTxFifo, bool flushRxFifo)

Flushes the LPSPI FIFOs.

#### Parameters

- base – LPSPI peripheral address.
- flushTxFifo – Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO.
- flushRxFifo – Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO.

static inline void LPSPI\_SetFifoWatermarks(LPSPI\_Type \*base, uint32\_t txWater, uint32\_t rxWater)

Sets the transmit and receive FIFO watermark values.

This function allows the user to set the receive and transmit FIFO watermarks. The function does not compare the watermark settings to the FIFO size. The FIFO watermark should not be equal to or greater than the FIFO size. It is up to the higher level driver to make this check.

#### Parameters

- base – LPSPI peripheral address.
- txWater – The TX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.

- rxWater – The RX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPSPI\_SetAllPcsPolarity(LPSPI\_Type \*base, uint32\_t mask)

Configures all LPSPI peripheral chip select polarities simultaneously.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx\_CR\_MEN = 0).

This is an example: PCS0 and PCS1 set to active low and other PCSs set to active high. Note that the number of PCS is device-specific.

```
LPSPI_SetAllPcsPolarity(base, kLPSPI_Pcs0ActiveLow | kLPSPI_Pcs1ActiveLow);
```

### Parameters

- base – LPSPI peripheral address.
- mask – The PCS polarity mask; Use the enum `_lpspi_pcs_polarity`.

static inline void LPSPI\_SetFrameSize(LPSPI\_Type \*base, uint32\_t frameSize)

Configures the frame size.

The minimum frame size is 8-bits and the maximum frame size is 4096-bits. If the frame size is less than or equal to 32-bits, the word size and frame size are identical. If the frame size is greater than 32-bits, the word size is 32-bits for each word except the last (the last word contains the remainder bits if the frame size is not divisible by 32). The minimum word size is 2-bits. A frame size of 33-bits (or similar) is not supported.

Note 1: The transmit command register should be initialized before enabling the LPSPI in slave mode, although the command register does not update until after the LPSPI is enabled. After it is enabled, the transmit command register should only be changed if the LPSPI is idle.

Note 2: The transmit and command FIFO is a combined FIFO that includes both transmit data and command words. That means the TCR register should be written to when the Tx FIFO is not full.

### Parameters

- base – LPSPI peripheral address.
- frameSize – The frame size in number of bits.

uint32\_t LPSPI\_MasterSetBaudRate(LPSPI\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz, uint32\_t \*tcrPrescaleValue)

Sets the LPSPI baud rate in bits per second.

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate and returns the calculated baud rate in bits-per-second. It requires the caller to provide the frequency of the module source clock (in Hertz). Note that the baud rate does not go into effect until the Transmit Control Register (TCR) is programmed with the prescale value. Hence, this function returns the prescale `tcrPrescaleValue` parameter for later programming in the TCR. The higher level peripheral driver should alert the user of an out of range baud rate input.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

### Parameters

- base – LPSPI peripheral address.
- baudRate\_Bps – The desired baud rate in bits per second.
- srcClock\_Hz – Module source input clock in Hertz.

- `tcrPrescaleValue` – The TCR prescale value needed to program the TCR.

**Returns**

The actual calculated baud rate. This function may also return a “0” if the LPSPI is not configured for master mode or if the LPSPI module is not disabled.

```
void LPSPI_MasterSetDelayScaler(LPSPI_Type *base, uint32_t scaler, lpspi_delay_type_t
                               whichDelay)
```

Manually configures a specific LPSPI delay parameter (module must be disabled to change the delay values).

This function configures the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type `lpspi_delay_type_t`.

The user passes the desired delay along with the delay value. This allows the user to directly set the delay values if they have pre-calculated them or if they simply wish to manually increment the value.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

**Parameters**

- `base` – LPSPI peripheral address.
- `scaler` – The 8-bit delay value 0x00 to 0xFF (255).
- `whichDelay` – The desired delay to configure, must be of type `lpspi_delay_type_t`.

```
uint32_t LPSPI_MasterSetDelayTimes(LPSPI_Type *base, uint32_t delayTimeInNanoSec,
                                   lpspi_delay_type_t whichDelay, uint32_t srcClock_Hz)
```

Calculates the delay based on the desired delay input in nanoseconds (module must be disabled to change the delay values).

This function calculates the values for the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type `lpspi_delay_type_t`.

The user passes the desired delay and the desired delay value in nano-seconds. The function calculates the value needed for the desired delay parameter and returns the actual calculated delay because 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. It is up to the higher level peripheral driver to alert the user of an out of range delay input.

Note that the LPSPI module must be configured for master mode before configuring this. And note that the `delayTime = LPSPI_clockSource / (PRESCALE * Delay_scaler)`.

**Parameters**

- `base` – LPSPI peripheral address.
- `delayTimeInNanoSec` – The desired delay value in nano-seconds.
- `whichDelay` – The desired delay to configuration, which must be of type `lpspi_delay_type_t`.
- `srcClock_Hz` – Module source input clock in Hertz.

**Returns**

actual Calculated delay value in nano-seconds.

```
static inline void LPSPI_WriteData(LPSPI_Type *base, uint32_t data)
```

Writes data into the transmit data buffer.

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user has to manage sending the data one 32-bit word at a time. Any writes to the TDR result in an immediate push to the transmit FIFO. This function can be used for either master or slave modes.

#### Parameters

- `base` – LPSPI peripheral address.
- `data` – The data word to be sent.

```
static inline uint32_t LPSPI_ReadData(LPSPI_Type *base)
```

Reads data from the data buffer.

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

#### Parameters

- `base` – LPSPI peripheral address.

#### Returns

The data read from the data buffer.

```
void LPSPI_SetDummyData(LPSPI_Type *base, uint8_t dummyData)
```

Set up the dummy data.

#### Parameters

- `base` – LPSPI peripheral address.
- `dummyData` – Data to be transferred when tx buffer is NULL. Note: This API has no effect when LPSPI in slave interrupt mode, because driver will set the TXMSK bit to 1 if txData is NULL, no data is loaded from transmit FIFO and output pin is tristated.

```
void LPSPI_MasterTransferCreateHandle(LPSPI_Type *base, lpspi_master_handle_t *handle,
                                     lpspi_master_transfer_callback_t callback, void
                                     *userData)
```

Initializes the LPSPI master handle.

This function initializes the LPSPI handle, which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

#### Parameters

- `base` – LPSPI peripheral address.
- `handle` – LPSPI handle pointer to `lpspi_master_handle_t`.
- `callback` – DSPI callback.
- `userData` – callback function parameter.

```
status_t LPSPI_MasterTransferBlocking(LPSPI_Type *base, lpspi_transfer_t *transfer)
```

LPSPI master transfer data using a polling method.

This function transfers data using a polling method. This is a blocking function, which does not return until all transfers have been completed.

Note: The transfer data size should be integer multiples of `bytesPerFrame` if `bytesPerFrame` is less than or equal to 4. For `bytesPerFrame` greater than 4: The transfer data size should be equal to `bytesPerFrame` if the `bytesPerFrame` is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of `bytesPerFrame`.



**Parameters**

- base – LPSPI peripheral address.
- transfer – pointer to `lpspi_transfer_t` structure.

**Returns**

status of `status_t`.

`status_t` LPSPI\_MasterTransferNonBlocking(LP\_SPI\_Type \*base, *lpspi\_master\_handle\_t* \*handle, *lpspi\_transfer\_t* \*transfer)

LPSPI master transfer data using an interrupt method.

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of `bytesPerFrame` if `bytesPerFrame` is less than or equal to 4. For `bytesPerFrame` greater than 4: The transfer data size should be equal to `bytesPerFrame` if the `bytesPerFrame` is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of `bytesPerFrame`.

**Parameters**

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_master_handle_t` structure which stores the transfer state.
- transfer – pointer to `lpspi_transfer_t` structure.

**Returns**

status of `status_t`.

`status_t` LPSPI\_MasterTransferGetCount(LP\_SPI\_Type \*base, *lpspi\_master\_handle\_t* \*handle, `size_t` \*count)

Gets the master transfer remaining bytes.

This function gets the master transfer remaining bytes.

**Parameters**

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_master_handle_t` structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

**Returns**

status of `status_t`.

`void` LPSPI\_MasterTransferAbort(LP\_SPI\_Type \*base, *lpspi\_master\_handle\_t* \*handle)

LPSPI master abort transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

**Parameters**

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_master_handle_t` structure which stores the transfer state.

`void` LPSPI\_MasterTransferHandleIRQ(LP\_SPI\_Type \*base, *lpspi\_master\_handle\_t* \*handle)

LPSPI Master IRQ handler function.

This function processes the LPSPI transmit and receive IRQ.

**Parameters**

- `base` – LPSPi peripheral address.
- `handle` – pointer to `lpspi_master_handle_t` structure which stores the transfer state.

```
void LPSPI_SlaveTransferCreateHandle(LPSPI_Type *base, lpspi_slave_handle_t *handle,
                                   lpspi_slave_transfer_callback_t callback, void *userData)
```

Initializes the LPSPi slave handle.

This function initializes the LPSPi handle, which can be used for other LPSPi transactional APIs. Usually, for a specified LPSPi instance, call this API once to get the initialized handle.

#### Parameters

- `base` – LPSPi peripheral address.
- `handle` – LPSPi handle pointer to `lpspi_slave_handle_t`.
- `callback` – DSPI callback.
- `userData` – callback function parameter.

```
status_t LPSPI_SlaveTransferNonBlocking(LPSPI_Type *base, lpspi_slave_handle_t *handle,
                                       lpspi_transfer_t *transfer)
```

LPSPi slave transfer data using an interrupt method.

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of `bytesPerFrame` if `bytesPerFrame` is less than or equal to 4. For `bytesPerFrame` greater than 4: The transfer data size should be equal to `bytesPerFrame` if the `bytesPerFrame` is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of `bytesPerFrame`.

#### Parameters

- `base` – LPSPi peripheral address.
- `handle` – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.
- `transfer` – pointer to `lpspi_transfer_t` structure.

#### Returns

status of `status_t`.

```
status_t LPSPI_SlaveTransferGetCount(LPSPI_Type *base, lpspi_slave_handle_t *handle, size_t
                                   *count)
```

Gets the slave transfer remaining bytes.

This function gets the slave transfer remaining bytes.

#### Parameters

- `base` – LPSPi peripheral address.
- `handle` – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

#### Returns

status of `status_t`.

```
void LPSPI_SlaveTransferAbort(LPSPI_Type *base, lpspi_slave_handle_t *handle)
```

LPSPi slave aborts a transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

#### Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.

`void LPSPI_SlaveTransferHandleIRQ(LPSPI_Type *base, lpspi_slave_handle_t *handle)`  
LPSPI Slave IRQ handler function.

This function processes the LPSPI transmit and receives an IRQ.

#### Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.

`bool LPSPI_WaitTxFifoEmpty(LPSPI_Type *base)`  
Wait for tx FIFO to be empty.

This function wait the tx fifo empty

#### Parameters

- base – LPSPI peripheral address.

#### Returns

true for the tx FIFO is ready, false is not.

`void LPSPI_DriverIRQHandler(uint32_t instance)`  
LPSPI driver IRQ handler common entry.

This function provides the common IRQ request entry for LPSPI.

#### Parameters

- instance – LPSPI instance.

`FSL_LPSPI_DRIVER_VERSION`  
LPSPI driver version.

Status for the LPSPI driver.

*Values:*

enumerator `kStatus_LPSPI_Busy`  
LPSPI transfer is busy.

enumerator `kStatus_LPSPI_Error`  
LPSPI driver error.

enumerator `kStatus_LPSPI_Idle`  
LPSPI is idle.

enumerator `kStatus_LPSPI_OutOfRange`  
LPSPI transfer out Of range.

enumerator `kStatus_LPSPI_Timeout`  
LPSPI timeout polling status flags.

`enum _lpspi_flags`  
LPSPI status flags in SPIx\_SR register.

*Values:*

enumerator `kLPSPI_TxDataRequestFlag`  
Transmit data flag

enumerator kLPSPI\_RxDataReadyFlag  
Receive data flag

enumerator kLPSPI\_WordCompleteFlag  
Word Complete flag

enumerator kLPSPI\_FrameCompleteFlag  
Frame Complete flag

enumerator kLPSPI\_TransferCompleteFlag  
Transfer Complete flag

enumerator kLPSPI\_TransmitErrorFlag  
Transmit Error flag (FIFO underrun)

enumerator kLPSPI\_ReceiveErrorFlag  
Receive Error flag (FIFO overrun)

enumerator kLPSPI\_DataMatchFlag  
Data Match flag

enumerator kLPSPI\_ModuleBusyFlag  
Module Busy flag

enumerator kLPSPI\_AllStatusFlag  
Used for clearing all w1c status flags

enum \_lpspi\_interrupt\_enable  
LPSPI interrupt source.

*Values:*

enumerator kLPSPI\_TxInterruptEnable  
Transmit data interrupt enable

enumerator kLPSPI\_RxInterruptEnable  
Receive data interrupt enable

enumerator kLPSPI\_WordCompleteInterruptEnable  
Word complete interrupt enable

enumerator kLPSPI\_FrameCompleteInterruptEnable  
Frame complete interrupt enable

enumerator kLPSPI\_TransferCompleteInterruptEnable  
Transfer complete interrupt enable

enumerator kLPSPI\_TransmitErrorInterruptEnable  
Transmit error interrupt enable(FIFO underrun)

enumerator kLPSPI\_ReceiveErrorInterruptEnable  
Receive Error interrupt enable (FIFO overrun)

enumerator kLPSPI\_DataMatchInterruptEnable  
Data Match interrupt enable

enumerator kLPSPI\_AllInterruptEnable  
All above interrupts enable.

enum \_lpspi\_dma\_enable  
LPSPI DMA source.

*Values:*

enumerator kLPSPi\_TxDmaEnable

Transmit data DMA enable

enumerator kLPSPi\_RxDmaEnable

Receive data DMA enable

enum \_lpspi\_master\_slave\_mode

LPSPi master or slave mode configuration.

*Values:*

enumerator kLPSPi\_Master

LPSPi peripheral operates in master mode.

enumerator kLPSPi\_Slave

LPSPi peripheral operates in slave mode.

enum \_lpspi\_which\_pcs\_config

LPSPi Peripheral Chip Select (PCS) configuration (which PCS to configure).

*Values:*

enumerator kLPSPi\_Pcs0

PCS[0]

enumerator kLPSPi\_Pcs1

PCS[1]

enumerator kLPSPi\_Pcs2

PCS[2]

enumerator kLPSPi\_Pcs3

PCS[3]

enum \_lpspi\_pcs\_polarity\_config

LPSPi Peripheral Chip Select (PCS) Polarity configuration.

*Values:*

enumerator kLPSPi\_PcsActiveHigh

PCS Active High (idles low)

enumerator kLPSPi\_PcsActiveLow

PCS Active Low (idles high)

enum \_lpspi\_pcs\_polarity

LPSPi Peripheral Chip Select (PCS) Polarity.

*Values:*

enumerator kLPSPi\_Pcs0ActiveLow

Pcs0 Active Low (idles high).

enumerator kLPSPi\_Pcs1ActiveLow

Pcs1 Active Low (idles high).

enumerator kLPSPi\_Pcs2ActiveLow

Pcs2 Active Low (idles high).

enumerator kLPSPi\_Pcs3ActiveLow

Pcs3 Active Low (idles high).

enumerator kLPSPi\_PcsAllActiveLow

Pcs0 to Pcs5 Active Low (idles high).

enum \_lpspi\_clock\_polarity

LPSPI clock polarity configuration.

*Values:*

enumerator kLPSPIClockPolarityActiveHigh

CPOL=0. Active-high LPSPI clock (idles low)

enumerator kLPSPIClockPolarityActiveLow

CPOL=1. Active-low LPSPI clock (idles high)

enum \_lpspi\_clock\_phase

LPSPI clock phase configuration.

*Values:*

enumerator kLPSPIClockPhaseFirstEdge

CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.

enumerator kLPSPIClockPhaseSecondEdge

CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

enum \_lpspi\_shift\_direction

LPSPI data shifter direction options.

*Values:*

enumerator kLPSPIMsbFirst

Data transfers start with most significant bit.

enumerator kLPSPILsbFirst

Data transfers start with least significant bit.

enum \_lpspi\_host\_request\_select

LPSPI Host Request select configuration.

*Values:*

enumerator kLPSPICHostReqExtPin

Host Request is an ext pin.

enumerator kLPSPICHostReqInternalTrigger

Host Request is an internal trigger.

enum \_lpspi\_match\_config

LPSPI Match configuration options.

*Values:*

enumerator kLPSPIMatchDisabled

LPSPI Match Disabled.

enumerator kLPSPIMatch1stWordEqualsM0orM1

LPSPI Match Enabled.

enumerator kLPSPIMatchAnyWordEqualsM0orM1

LPSPI Match Enabled.

enumerator kLPSPIMatch1stWordEqualsM0and2ndWordEqualsM1

LPSPI Match Enabled.

enumerator kLPSPIMatchAnyWordEqualsM0andNxtWordEqualsM1

LPSPI Match Enabled.

enumerator kLPSI\_1stWordAndM1EqualsM0andM1  
LPSPi Match Enabled.

enumerator kLPSI\_AnyWordAndM1EqualsM0andM1  
LPSPi Match Enabled.

enum \_lpspi\_pin\_config  
LPSPi pin (SDO and SDI) configuration.

*Values:*

enumerator kLPSPI\_SdiInSdoOut  
LPSPi SDI input, SDO output.

enumerator kLPSPI\_SdiInSdiOut  
LPSPi SDI input, SDI output.

enumerator kLPSPI\_SdoInSdoOut  
LPSPi SDO input, SDO output.

enumerator kLPSPI\_SdoInSdiOut  
LPSPi SDO input, SDI output.

enum \_lpspi\_data\_out\_config  
LPSPi data output configuration.

*Values:*

enumerator kLpspiDataOutRetained  
Data out retains last value when chip select is de-asserted

enumerator kLpspiDataOutTristate  
Data out is tristated when chip select is de-asserted

enum \_lpspi\_transfer\_width  
LPSPi transfer width configuration.

*Values:*

enumerator kLPSPI\_SingleBitXfer  
1-bit shift at a time, data out on SDO, in on SDI (normal mode)

enumerator kLPSPI\_TwoBitXfer  
2-bits shift out on SDO/SDI and in on SDO/SDI

enumerator kLPSPI\_FourBitXfer  
4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

enum \_lpspi\_delay\_type  
LPSPi delay type selection.

*Values:*

enumerator kLPSPI\_PcsToSck  
PCS-to-SCK delay.

enumerator kLPSPI\_LastSckToPcs  
Last SCK edge to PCS delay.

enumerator kLPSPI\_BetweenTransfer  
Delay between transfers.

enum \_lpspi\_transfer\_config\_flag\_for\_master

Use this enumeration for LPSPi master transfer configFlags.

*Values:*

enumerator kLPSPi\_MasterPcs0

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS0 signal

enumerator kLPSPi\_MasterPcs1

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS1 signal

enumerator kLPSPi\_MasterPcs2

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS2 signal

enumerator kLPSPi\_MasterPcs3

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS3 signal

enumerator kLPSPi\_MasterPcsContinuous

Is PCS signal continuous

enumerator kLPSPi\_MasterByteSwap

Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi\_shift\_direction\_t to MSB).

- i. If you set bitPerFrame = 8 , no matter the kLPSPi\_MasterByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
- ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPi\_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi\_MasterByteSwap flag.
- iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPi\_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi\_MasterByteSwap flag.

enum \_lpspi\_transfer\_config\_flag\_for\_slave

Use this enumeration for LPSPi slave transfer configFlags.

*Values:*

enumerator kLPSPi\_SlavePcs0

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS0 signal

enumerator kLPSPi\_SlavePcs1

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS1 signal

enumerator kLPSPi\_SlavePcs2

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS2 signal

enumerator kLPSPi\_SlavePcs3

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS3 signal

enumerator kLPSPi\_SlaveByteSwap

Is slave swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi\_shift\_direction\_t to MSB).

- i. If you set bitPerFrame = 8 , no matter the kLPSPi\_SlaveByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
- ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPi\_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi\_SlaveByteSwap flag.
- iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPi\_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi\_SlaveByteSwap flag.



enum `_lpspi_transfer_state`

LPSPI transfer state, which is used for LPSPI transactional API state machine.

*Values:*

enumerator `kLPSPI_Idle`

Nothing in the transmitter/receiver.

enumerator `kLPSPI_Busy`

Transfer queue is not finished.

enumerator `kLPSPI_Error`

Transfer error.

typedef enum `_lpspi_master_slave_mode` `lpspi_master_slave_mode_t`

LPSPI master or slave mode configuration.

typedef enum `_lpspi_which_pcs_config` `lpspi_which_pcs_t`

LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

typedef enum `_lpspi_pcs_polarity_config` `lpspi_pcs_polarity_config_t`

LPSPI Peripheral Chip Select (PCS) Polarity configuration.

typedef enum `_lpspi_clock_polarity` `lpspi_clock_polarity_t`

LPSPI clock polarity configuration.

typedef enum `_lpspi_clock_phase` `lpspi_clock_phase_t`

LPSPI clock phase configuration.

typedef enum `_lpspi_shift_direction` `lpspi_shift_direction_t`

LPSPI data shifter direction options.

typedef enum `_lpspi_host_request_select` `lpspi_host_request_select_t`

LPSPI Host Request select configuration.

typedef enum `_lpspi_match_config` `lpspi_match_config_t`

LPSPI Match configuration options.

typedef enum `_lpspi_pin_config` `lpspi_pin_config_t`

LPSPI pin (SDO and SDI) configuration.

typedef enum `_lpspi_data_out_config` `lpspi_data_out_config_t`

LPSPI data output configuration.

typedef enum `_lpspi_transfer_width` `lpspi_transfer_width_t`

LPSPI transfer width configuration.

typedef enum `_lpspi_delay_type` `lpspi_delay_type_t`

LPSPI delay type selection.

typedef struct `_lpspi_master_config` `lpspi_master_config_t`

LPSPI master configuration structure.

typedef struct `_lpspi_slave_config` `lpspi_slave_config_t`

LPSPI slave configuration structure.

typedef struct `_lpspi_master_handle` `lpspi_master_handle_t`

Forward declaration of the `_lpspi_master_handle` typedefs.

typedef struct `_lpspi_slave_handle` `lpspi_slave_handle_t`

Forward declaration of the `_lpspi_slave_handle` typedefs.

```
typedef void (*lpspi_master_transfer_callback_t)(LPSPI_Type *base, lpspi_master_handle_t *handle, status_t status, void *userData)
```

Master completion callback function pointer type.

**Param base**

LPSPI peripheral address.

**Param handle**

Pointer to the handle for the LPSPI master.

**Param status**

Success or error code describing whether the transfer is completed.

**Param userData**

Arbitrary pointer-dataSized value passed from the application.

```
typedef void (*lpspi_slave_transfer_callback_t)(LPSPI_Type *base, lpspi_slave_handle_t *handle, status_t status, void *userData)
```

Slave completion callback function pointer type.

**Param base**

LPSPI peripheral address.

**Param handle**

Pointer to the handle for the LPSPI slave.

**Param status**

Success or error code describing whether the transfer is completed.

**Param userData**

Arbitrary pointer-dataSized value passed from the application.

```
typedef struct _lpspi_transfer lpspi_transfer_t
```

LPSPI master/slave transfer structure.

```
volatile uint8_t g_lpspiDummyData[]
```

Global variable for dummy data value setting.

```
LPSPI_DUMMY_DATA
```

LPSPI dummy data if no Tx data.

Dummy data used for tx if there is not txData.

```
SPI_RETRY_TIMES
```

Retry times for waiting flag.

```
LPSPI_MASTER_PCS_SHIFT
```

LPSPI master PCS shift macro , internal used.

```
LPSPI_MASTER_PCS_MASK
```

LPSPI master PCS shift macro , internal used.

```
LPSPI_SLAVE_PCS_SHIFT
```

LPSPI slave PCS shift macro , internal used.

```
LPSPI_SLAVE_PCS_MASK
```

LPSPI slave PCS shift macro , internal used.

```
struct _lpspi_master_config
```

*#include <fsl\_lpspi.h>* LPSPI master configuration structure.

### Public Members

uint32\_t baudRate

Baud Rate for LPSPI.

uint32\_t bitsPerFrame

Bits per frame, minimum 8, maximum 4096.

*lpspi\_clock\_polarity\_t* cpol

Clock polarity.

*lpspi\_clock\_phase\_t* cpha

Clock phase.

*lpspi\_shift\_direction\_t* direction

MSB or LSB data shift direction.

uint32\_t pcsToSckDelayInNanoSec

PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32\_t lastSckToPcsDelayInNanoSec

Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32\_t betweenTransferDelayInNanoSec

After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

*lpspi\_which\_pcs\_t* whichPcs

Desired Peripheral Chip Select (PCS).

*lpspi\_pcs\_polarity\_config\_t* pcsActiveHighOrLow

Desired PCS active high or low

*lpspi\_pin\_config\_t* pinCfg

Configures which pins are used for input and output data during single bit transfers.

*lpspi\_data\_out\_config\_t* dataOutConfig

Configures if the output data is tristated between accesses (LPSPI\_PCS is negated).

bool enableInputDelay

Enable master to sample the input data on a delayed SCK. This can help improve slave setup time. Refer to device data sheet for specific time length.

struct \_\_lpspi\_slave\_config

#include <fsl\_lpspi.h> LPSPI slave configuration structure.

### Public Members

uint32\_t bitsPerFrame

Bits per frame, minimum 8, maximum 4096.

*lpspi\_clock\_polarity\_t* cpol

Clock polarity.

*lpspi\_clock\_phase\_t* cpha

Clock phase.

*lpspi\_shift\_direction\_t* direction

MSB or LSB data shift direction.

*lpspi\_which\_pcs\_t* whichPcs

Desired Peripheral Chip Select (pcs)

*lpspi\_pcs\_polarity\_config\_t* pcsActiveHighOrLow

Desired PCS active high or low

*lpspi\_pin\_config\_t* pinCfg

Configures which pins are used for input and output data during single bit transfers.

*lpspi\_data\_out\_config\_t* dataOutConfig

Configures if the output data is tristated between accesses (LPSPI\_PCS is negated).

struct *\_lpspi\_transfer*

*#include <fsl\_lpspi.h>* LPSPI master/slave transfer structure.

### Public Members

const uint8\_t \*txData

Send buffer.

uint8\_t \*rxData

Receive buffer.

volatile size\_t dataSize

Transfer bytes.

uint32\_t configFlags

Transfer transfer configuration flags. Set from *\_lpspi\_transfer\_config\_flag\_for\_master* if the transfer is used for master or *\_lpspi\_transfer\_config\_flag\_for\_slave* enumeration if the transfer is used for slave.

struct *\_lpspi\_master\_handle*

*#include <fsl\_lpspi.h>* LPSPI master transfer handle structure used for transactional API.

### Public Members

volatile bool isPcsContinuous

Is PCS continuous in transfer.

volatile bool writeTcrInIsr

A flag that whether should write TCR in ISR.

volatile bool isByteSwap

A flag that whether should byte swap.

volatile bool isTxMask

A flag that whether TCR[TXMSK] is set.

volatile uint16\_t bytesPerFrame

Number of bytes in each frame

volatile uint16\_t frameSize

Backup of TCR[FRAMESZ]

volatile uint8\_t fifoSize

FIFO dataSize.

volatile uint8\_t rxWatermark

Rx watermark.

`volatile uint8_t bytesEachWrite`  
Bytes for each write TDR.

`volatile uint8_t bytesEachRead`  
Bytes for each read RDR.

`const uint8_t *volatile txData`  
Send buffer.

`uint8_t *volatile rxData`  
Receive buffer.

`volatile size_t txRemainingByteCount`  
Number of bytes remaining to send.

`volatile size_t rxRemainingByteCount`  
Number of bytes remaining to receive.

`volatile uint32_t writeRegRemainingTimes`  
Write TDR register remaining times.

`volatile uint32_t readRegRemainingTimes`  
Read RDR register remaining times.

`uint32_t totalByteCount`  
Number of transfer bytes

`uint32_t txBuffIfNull`  
Used if the txData is NULL.

`volatile uint8_t state`  
LPSPI transfer state , `_lpspi_transfer_state`.

`lpspi_master_transfer_callback_t callback`  
Completion callback.

`void *userData`  
Callback user data.

`struct _lpspi_slave_handle`  
*#include <fsl\_lpspi.h>* LPSPI slave transfer handle structure used for transactional API.

### Public Members

`volatile bool isByteSwap`  
A flag that whether should byte swap.

`volatile uint8_t fifoSize`  
FIFO dataSize.

`volatile uint8_t rxWatermark`  
Rx watermark.

`volatile uint8_t bytesEachWrite`  
Bytes for each write TDR.

`volatile uint8_t bytesEachRead`  
Bytes for each read RDR.

`const uint8_t *volatile txData`  
Send buffer.

`uint8_t *volatile rxData`  
Receive buffer.

`volatile size_t txRemainingByteCount`  
Number of bytes remaining to send.

`volatile size_t rxRemainingByteCount`  
Number of bytes remaining to receive.

`volatile uint32_t writeRegRemainingTimes`  
Write TDR register remaining times.

`volatile uint32_t readRegRemainingTimes`  
Read RDR register remaining times.

`uint32_t totalByteCount`  
Number of transfer bytes

`volatile uint8_t state`  
LPSPI transfer state , `_lpspi_transfer_state`.

`volatile uint32_t errorCount`  
Error count for slave transfer.

`lpspi_slave_transfer_callback_t callback`  
Completion callback.

`void *userData`  
Callback user data.

## 2.36 LPSPI eDMA Driver

`FSL_LPSPI_EDMA_DRIVER_VERSION`

LPSPI EDMA driver version.

`DMA_MAX_TRANSFER_COUNT`

DMA max transfer size.

`typedef struct _lpspi_master_edma_handle lpspi_master_edma_handle_t`

Forward declaration of the `_lpspi_master_edma_handle` typedefs.

`typedef struct _lpspi_slave_edma_handle lpspi_slave_edma_handle_t`

Forward declaration of the `_lpspi_slave_edma_handle` typedefs.

`typedef void (*lpspi_master_edma_transfer_callback_t)(LPSPI_Type *base, lpspi_master_edma_handle_t *handle, status_t status, void *userData)`

Completion callback function pointer type.

**Param base**

LPSPI peripheral base address.

**Param handle**

Pointer to the handle for the LPSPI master.

**Param status**

Success or error code describing whether the transfer completed.

**Param userData**

Arbitrary pointer-dataSized value passed from the application.

```
typedef void (*lpspi_slave_edma_transfer_callback_t)(LPSPi_Type *base,  
lpspi_slave_edma_handle_t *handle, status_t status, void *userData)
```

Completion callback function pointer type.

**Param base**

LPSPi peripheral base address.

**Param handle**

Pointer to the handle for the LPSPi slave.

**Param status**

Success or error code describing whether the transfer completed.

**Param userData**

Arbitrary pointer-dataSized value passed from the application.

```
void LPSPi_MasterTransferCreateHandleEDMA(LPSPi_Type *base, lpspi_master_edma_handle_t  
*handle, lpspi_master_edma_transfer_callback_t  
callback, void *userData, edma_handle_t  
*edmaRxRegToRxDataHandle, edma_handle_t  
*edmaTxDataToTxRegHandle)
```

Initializes the LPSPi master eDMA handle.

This function initializes the LPSPi eDMA handle which can be used for other LPSPi transactional APIs. Usually, for a specified LPSPi instance, call this API once to get the initialized handle.

Note that the LPSPi eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx are the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Tx DMAMUX source for edmaRxRegToRxDataHandle.

**Parameters**

- base – LPSPi peripheral base address.
- handle – LPSPi handle pointer to lpspi\_master\_edma\_handle\_t.
- callback – LPSPi callback.
- userData – callback function parameter.
- edmaRxRegToRxDataHandle – edmaRxRegToRxDataHandle pointer to edma\_handle\_t.
- edmaTxDataToTxRegHandle – edmaTxDataToTxRegHandle pointer to edma\_handle\_t.

```
status_t LPSPi_MasterTransferEDMA(LPSPi_Type *base, lpspi_master_edma_handle_t *handle,  
lpspi_transfer_t *transfer)
```

LPSPi master transfer data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

**Parameters**

- base – LPSPi peripheral base address.
- handle – pointer to lpspi\_master\_edma\_handle\_t structure which stores the transfer state.

- transfer – pointer to `lpspi_transfer_t` structure.

**Returns**

status of `status_t`.

`status_t` LPSPI\_MasterTransferPrepareEDMALite(LPSPI\_Type \*base, *lpspi\_master\_edma\_handle\_t* \*handle, uint32\_t configFlags)

LPSPI master config transfer parameter while using eDMA.

This function is preparing to transfer data using eDMA, work with LPSPI\_MasterTransferEDMALite.

**Parameters**

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.
- configFlags – transfer configuration flags. `_lpspi_transfer_config_flag_for_master`.

**Return values**

- `kStatus_Success` – Execution successfully.
- `kStatus_LPSPI_Busy` – The LPSPI device is busy.

**Returns**

Indicates whether LPSPI master transfer was successful or not.

`status_t` LPSPI\_MasterTransferEDMALite(LPSPI\_Type \*base, *lpspi\_master\_edma\_handle\_t* \*handle, *lpspi\_transfer\_t* \*transfer)

LPSPI master transfer data using eDMA without configs.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: This API is only for transfer through DMA without configuration. Before calling this API, you must call LPSPI\_MasterTransferPrepareEDMALite to configure it once. The transfer data size should be an integer multiple of `bytesPerFrame` if `bytesPerFrame` is less than or equal to 4. For `bytesPerFrame` greater than 4: The transfer data size should be equal to `bytesPerFrame` if the `bytesPerFrame` is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of `bytesPerFrame`.

**Parameters**

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.
- transfer – pointer to `lpspi_transfer_t` structure, config field is not used.

**Return values**

- `kStatus_Success` – Execution successfully.
- `kStatus_LPSPI_Busy` – The LPSPI device is busy.
- `kStatus_InvalidArgument` – The transfer structure is invalid.

**Returns**

Indicates whether LPSPI master transfer was successful or not.

`void` LPSPI\_MasterTransferAbortEDMA(LPSPI\_Type \*base, *lpspi\_master\_edma\_handle\_t* \*handle)

LPSPI master aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.



**Parameters**

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.

*status\_t* LPSPI\_MasterTransferGetCountEDMA(LPSPI\_Type \*base, *lpspi\_master\_edma\_handle\_t* \*handle, *size\_t* \*count)

Gets the master eDMA transfer remaining bytes.

This function gets the master eDMA transfer remaining bytes.

**Parameters**

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.
- count – Number of bytes transferred so far by the EDMA transaction.

**Returns**

status of *status\_t*.

*void* LPSPI\_SlaveTransferCreateHandleEDMA(LPSPI\_Type \*base, *lpspi\_slave\_edma\_handle\_t* \*handle, *lpspi\_slave\_edma\_transfer\_callback\_t* callback, *void* \*userData, *edma\_handle\_t* \*edmaRxRegToRxDataHandle, *edma\_handle\_t* \*edmaTxDataToTxRegHandle)

Initializes the LPSPI slave eDMA handle.

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx as the same source) DMA request source.

(1) For a separated DMA request source, enable and set the Rx DMAMUX source for `edmaRxRegToRxDataHandle` and Tx DMAMUX source for `edmaTxDataToTxRegHandle`. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for `edmaRxRegToRxDataHandle`.

**Parameters**

- base – LPSPI peripheral base address.
- handle – LPSPI handle pointer to `lpspi_slave_edma_handle_t`.
- callback – LPSPI callback.
- userData – callback function parameter.
- `edmaRxRegToRxDataHandle` – `edmaRxRegToRxDataHandle` pointer to `edma_handle_t`.
- `edmaTxDataToTxRegHandle` – `edmaTxDataToTxRegHandle` pointer to `edma_handle_t`.

*status\_t* LPSPI\_SlaveTransferEDMA(LPSPI\_Type \*base, *lpspi\_slave\_edma\_handle\_t* \*handle, *lpspi\_transfer\_t* \*transfer)

LPSPI slave transfers data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which return right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of `bytesPerFrame` if `bytesPerFrame` is less than or equal to 4. For `bytesPerFrame` greater than 4: The transfer data size

should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

#### Parameters

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_slave_edma_handle_t` structure which stores the transfer state.
- transfer – pointer to `lpspi_transfer_t` structure.

#### Returns

status of `status_t`.

`void LPSPI_SlaveTransferAbortEDMA(LPSPI_Type *base, lpspi_slave_edma_handle_t *handle)`  
LPSPI slave aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

#### Parameters

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_slave_edma_handle_t` structure which stores the transfer state.

`status_t LPSPI_SlaveTransferGetCountEDMA(LPSPI_Type *base, lpspi_slave_edma_handle_t *handle, size_t *count)`

Gets the slave eDMA transfer remaining bytes.

This function gets the slave eDMA transfer remaining bytes.

#### Parameters

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_slave_edma_handle_t` structure which stores the transfer state.
- count – Number of bytes transferred so far by the eDMA transaction.

#### Returns

status of `status_t`.

`struct _lpspi_master_edma_handle`

*#include <fsl\_lpspi\_edma.h>* LPSPI master eDMA transfer handle structure used for transactional API.

#### Public Members

`volatile bool isPcsContinuous`

Is PCS continuous in transfer.

`volatile bool isByteSwap`

A flag that whether should byte swap.

`volatile uint8_t fifoSize`

FIFO dataSize.

`volatile uint8_t rxWatermark`

Rx watermark.

`volatile uint8_t bytesEachWrite`

Bytes for each write TDR.

`volatile uint8_t bytesEachRead`  
Bytes for each read RDR.

`volatile uint8_t bytesLastRead`  
Bytes for last read RDR.

`volatile bool isThereExtraRxBytes`  
Is there extra RX byte.

`const uint8_t *volatile txData`  
Send buffer.

`uint8_t *volatile rxData`  
Receive buffer.

`volatile size_t txRemainingByteCount`  
Number of bytes remaining to send.

`volatile size_t rxRemainingByteCount`  
Number of bytes remaining to receive.

`volatile uint32_t writeRegRemainingTimes`  
Write TDR register remaining times.

`volatile uint32_t readRegRemainingTimes`  
Read RDR register remaining times.

`uint32_t totalByteCount`  
Number of transfer bytes

`edma_tcd_t *lastTimeTCD`  
Pointer to the lastTime TCD

`bool isMultiDMATransmit`  
Is there multi DMA transmit

`volatile uint8_t dmaTransmitTime`  
DMA Transfer times.

`uint32_t lastTimeDataBytes`  
DMA transmit last Time data Bytes

`uint32_t dataBytesEveryTime`  
Bytes in a time for DMA transfer, default is DMA\_MAX\_TRANSFER\_COUNT

`edma_transfer_config_t transferConfigRx`  
Config of DMA rx channel.

`edma_transfer_config_t transferConfigTx`  
Config of DMA tx channel.

`uint32_t txBuffIfNull`  
Used if there is not txData for DMA purpose.

`uint32_t rxBuffIfNull`  
Used if there is not rxData for DMA purpose.

`uint32_t transmitCommand`  
Used to write TCR for DMA purpose.

`volatile uint8_t state`  
LPSPI transfer state , `_lpspi_transfer_state`.

```

uint8_t nbytes
    eDMA minor byte transfer count initially configured.

lpspi_master_edma_transfer_callback_t callback
    Completion callback.

void *userData
    Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
    edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToTxRegHandle
    edma_handle_t handle point used for TxData to TxReg buff

edma_tcd_t lpspiSoftwareTCD[3]
    SoftwareTCD, internal used

```

struct `_lpspi_slave_edma_handle`  
*#include <fsl\_lpspi\_edma.h>* LPSPI slave eDMA transfer handle structure used for transactional API.

### Public Members

```

volatile bool isByteSwap
    A flag that whether should byte swap.

volatile uint8_t fifoSize
    FIFO dataSize.

volatile uint8_t rxWatermark
    Rx watermark.

volatile uint8_t bytesEachWrite
    Bytes for each write TDR.

volatile uint8_t bytesEachRead
    Bytes for each read RDR.

volatile uint8_t bytesLastRead
    Bytes for last read RDR.

volatile bool isThereExtraRxBytes
    Is there extra RX byte.

uint8_t nbytes
    eDMA minor byte transfer count initially configured.

const uint8_t *volatile txData
    Send buffer.

uint8_t *volatile rxData
    Receive buffer.

volatile size_t txRemainingByteCount
    Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
    Number of bytes remaining to receive.

```

`volatile uint32_t writeRegRemainingTimes`  
Write TDR register remaining times.

`volatile uint32_t readRegRemainingTimes`  
Read RDR register remaining times.

`uint32_t totalByteCount`  
Number of transfer bytes

`uint32_t txBuffIfNull`  
Used if there is not txData for DMA purpose.

`uint32_t rxBuffIfNull`  
Used if there is not rxData for DMA purpose.

`volatile uint8_t state`  
LPSPI transfer state.

`uint32_t errorCount`  
Error count for slave transfer.

`lpspi_slave_edma_transfer_callback_t callback`  
Completion callback.

`void *userData`  
Callback user data.

`edma_handle_t *edmaRxRegToRxDataHandle`  
edma\_handle\_t handle point used for RxReg to RxData buff

`edma_handle_t *edmaTxDataToTxRegHandle`  
edma\_handle\_t handle point used for TxData to TxReg

`edma_tcd_t lpspiSoftwareTCD[2]`  
SoftwareTCD, internal used

## 2.37 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:*

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.

### 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

## 2.38 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

### 2.39 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, false 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 receives 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 be 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 example 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 data 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 RX 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 remainBytes 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_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\_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 enableTxCTS

TX CTS enable

*lpuart\_transmit\_cts\_source\_t* txCtsSource

TX CTS source

*lpuart\_transmit\_cts\_config\_t* txCtsConfig

TX CTS configure

*lpuart\_idle\_type\_select\_t* rxIdleType

RX IDLE type.

*lpuart\_idle\_config\_t* rxIdleConfig

RX IDLE configuration.

bool enableTx

Enable TX

bool enableRx

Enable RX

struct `_lpuart_transfer`

*#include <fsl\_lpuart.h>* LPUART transfer structure.

### Public Members

size\_t dataSize

The byte count to be transfer.

struct `_lpuart_handle`  
*#include <fsl\_lpuart.h>* LPUART handle structure.

### Public Members

`volatile size_t txDataSize`  
Size of the remaining data to send.

`size_t txDataSizeAll`  
Size of the data to send out.

`volatile size_t rxDataSize`  
Size of the remaining data to receive.

`size_t rxDataSizeAll`  
Size of the data to receive.

`size_t rxRingBufferSize`  
Size of the ring buffer.

`volatile uint16_t rxRingBufferHead`  
Index for the driver to store received data into ring buffer.

`volatile uint16_t rxRingBufferTail`  
Index for the user to get data from the ring buffer.

*lpuart\_transfer\_callback\_t* callback  
Callback function.

`void *userData`  
LPUART callback function parameter.

`volatile uint8_t txState`  
TX transfer state.

`volatile uint8_t rxState`  
RX transfer state.

`bool isSevenDataBits`  
Seven data bits flag.

`bool is16bitData`  
16bit data bits flag, only used for 9bit or 10bit data

`union __unnamed25__`

### 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 __unnamed27__
```

### 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 __unnamed29__
```

### Public Members

```
uint8_t *volatile rxData
    Address of remaining data to receive.
uint16_t *volatile rxData16
    Address of remaining data to receive.
union __unnamed31__
```

### Public Members

```
uint8_t *rxRingBuffer
    Start address of the receiver ring buffer.
uint16_t *rxRingBuffer16
    Start address of the receiver ring buffer.
```

## 2.40 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
```

## 2.41 MCX\_CMC: Core Mode Controller Driver

```
enum _cmc_power_mode_protection
    CMC power mode Protection enumeration.

    Values:

    enumerator kCMC_AllowDeepSleepMode
        Allow Deep Sleep mode.

    enumerator kCMC_AllowPowerDownMode
        Allow Power Down mode.

    enumerator kCMC_AllowDeepPowerDownMode
        Allow Deep Power Down mode.

    enumerator kCMC_AllowAllLowPowerModes
        Allow Deep Sleep, Power Down, Deep Power Down modes.

enum _cmc_wakeup_sources
    Wake up sources from the previous low power mode entry.
```

---

**Note:** kCMC\_WakeupFromUsbFs, kCMC\_WakeupFromITRC, kCMC\_WakeupFromCpu1 are not supported in MCXA family.

---

*Values:*



enumerator kCMC\_WakeupFromResetInterruptOrPowerDown

Wakeup source is reset interrupt, or wake up from Deep Power Down.

enumerator kCMC\_WakeupFromDebugReuquest

Wakeup source is debug request.

enumerator kCMC\_WakeupFromInterrupt

Wakeup source is interrupt.

enumerator kCMC\_WakeupFromDMAWakeup

Wakeup source is DMA Wakeup.

enumerator kCMC\_WakeupFromWUURquest

Wakeup source is WUU request.

enumerator kCMC\_WakeupFromUsbFs

Wakeup source is USBFS(USB0).

enumerator kCMC\_WakeupFromITRC

Wakeup source is ITRC.

enumerator kCMC\_WakeupFromCpu1

Wakeup source is CPU1.

enum \_cmc\_system\_reset\_interrupt\_enable

System Reset Interrupt enable enumeration.

*Values:*

enumerator kCMC\_PinResetInterruptEnable

Pin Reset interrupt enable.

enumerator kCMC\_DAPResetInterruptEnable

DAP Reset interrupt enable.

enumerator kCMC\_LowPowerAcknowledgeTimeoutResetInterruptEnable

Low Power Acknowledge Timeout Reset interrupt enable.

enumerator kCMC\_SoftwareResetInterruptEnable

Software Reset interrupt enable.

enumerator kCMC\_LockupResetInterruptEnable

Lockup Reset interrupt enable.

enum \_cmc\_system\_reset\_interrupt\_flag

CMC System Reset Interrupt Status flag.

*Values:*

enumerator kCMC\_PinResetInterruptFlag

Pin Reset interrupt flag.

enumerator kCMC\_DAPResetInterruptFlag

DAP Reset interrupt flag.

enumerator kCMC\_LowPowerAcknowledgeTimeoutResetFlag

Low Power Acknowledge Timeout Reset interrupt flag.

enumerator kCMC\_SoftwareResetInterruptFlag

Software Reset interrupt flag.

enumerator kCMC\_LockupResetInterruptFlag

Lock up Reset interrupt flag.

enum \_cmc\_system\_sram\_arrays

CMC System SRAM arrays low power mode enable enumeration.

*Values:*

enumerator kCMC\_RAMX0

Used to control RAMX0.

enumerator kCMC\_RAMX1

Used to control RAMX1.

enumerator kCMC\_RAMX2

Used to control RAMX2.

enumerator kCMC\_RAMB

Used to control RAMB.

enumerator kCMC\_RAMC0

Used to control RAMC0.

enumerator kCMC\_RAMC1

Used to control RAMC1.

enumerator kCMC\_RAMD0

Used to control RAMD0.

enumerator kCMC\_RAMD1

Used to control RAMD1.

enumerator kCMC\_RAME0

Used to control RAME0.

enumerator kCMC\_RAME1

Used to control RAME1.

enumerator kCMC\_RAMF0

Used to control RAMF0.

enumerator kCMC\_RAMF1

Used to control RAMF1.

enumerator kCMC\_RAMG0\_RAMG1

Used to control RAMG0 and RAMG1.

enumerator kCMC\_RAMG2\_RAMG3

Used to control RAMG2 and RAMG3.

enumerator kCMC\_RAMH0\_RAMH1

Used to control RAMH0 and RAMH1.

enumerator kCMC\_LPCAC

Used to control LPCAC.

enumerator kCMC\_DMA0\_DMA1\_PKC

Used to control DMA0, DMA1 and PKC.

enumerator kCMC\_USB0

Used to control USB0.

enumerator kCMC\_PQ

Used to control PQ.

enumerator kCMC\_CAN0\_CAN1\_ENET\_USB1

Used to control CAN0, CAN1, ENET, USB1.

enumerator kCMC\_FlexSPI

Used to control FlexSPI.

enumerator kCMC\_AllSramArrays

Mask of all System SRAM arrays.

enum \_cmc\_system\_reset\_sources

System reset sources enumeration.

*Values:*

enumerator kCMC\_WakeUpReset

The reset caused by a wakeup from Power Down or Deep Power Down mode.

enumerator kCMC\_PORReset

The reset caused by power on reset detection logic.

enumerator kCMC\_WarmReset

The last reset source is a warm reset source.

enumerator kCMC\_FatalReset

The last reset source is a fatal reset source.

enumerator kCMC\_PinReset

The reset caused by the RESET\_b pin.

enumerator kCMC\_DAPReset

The reset caused by a reset request from the Debug Access port.

enumerator kCMC\_ResetTimeout

The reset caused by a timeout or other error condition in the system reset generation.

enumerator kCMC\_LowPowerAcknowledgeTimeoutReset

The reset caused by a timeout in low power mode entry logic.

enumerator kCMC\_SCGReset

The reset caused by a loss of clock or loss of lock event in the SCG.

enumerator kCMC\_SoftwareReset

The reset caused by a software reset request.

enumerator kCMC\_LockUoReset

The reset caused by the ARM core indication of a LOCKUP event.

enumerator kCMC\_JTAGSystemReset

The reset caused by a JTAG system reset request.

enum \_cmc\_core\_clock\_gate\_status

Indicate the core clock was gated.

*Values:*

enumerator kCMC\_CoreClockNotGated

Core clock not gated.

enumerator kCMC\_CoreClockGated

Core clock was gated due to low power mode entry.

enum \_cmc\_clock\_mode

CMC clock mode enumeration.

*Values:*

enumerator kCMC\_GateNoneClock

No clock gating.

enumerator kCMC\_GateCoreClock

Gate Core clock.

enumerator kCMC\_GateCorePlatformClock

Gate Core clock and platform clock.

enumerator kCMC\_GateAllSystemClocks

Gate all System clocks, without getting core entering into low power mode.

enumerator kCMC\_GateAllSystemClocksEnterLowPowerMode

Gate all System clocks, with core entering into low power mode.

enum \_cmc\_low\_power\_mode

CMC power mode enumeration.

*Values:*

enumerator kCMC\_ActiveOrSleepMode

Select Active/Sleep mode.

enumerator kCMC\_DeepSleepMode

Select Deep Sleep mode when a core executes WFI or WFE instruction.

enumerator kCMC\_PowerDownMode

Select Power Down mode when a core executes WFI or WFE instruction.

enumerator kCMC\_DeepPowerDown

Select Deep Power Down mode when a core executes WFI or WFE instruction.

typedef enum \_cmc\_core\_clock\_gate\_status cmc\_core\_clock\_gate\_status\_t

Indicate the core clock was gated.

typedef enum \_cmc\_clock\_mode cmc\_clock\_mode\_t

CMC clock mode enumeration.

typedef enum \_cmc\_low\_power\_mode cmc\_low\_power\_mode\_t

CMC power mode enumeration.

typedef struct \_cmc\_reset\_pin\_config cmc\_reset\_pin\_config\_t

CMC reset pin configuration.

typedef struct \_cmc\_power\_domain\_config cmc\_power\_domain\_config\_t

power mode configuration for each power domain.

FSL\_CMC\_DRIVER\_VERSION

CMC driver version 2.5.0.

void CMC\_SetClockMode(CMC\_Type \*base, cmc\_clock\_mode\_t mode)

Sets clock mode.

This function configfs the amount of clock gating when the core asserts Sleeping due to WFI, WFE or SLEEPONEXIT.

#### Parameters

- base – CMC peripheral base address.
- mode – System clock mode.

```
static inline void CMC_LockClockModeSetting(CMC_Type *base)
```

Locks the clock mode setting.

After invoking this function, any clock mode setting will be blocked.

#### Parameters

- base – CMC peripheral base address.

```
static inline cmc_core_clock_gate_status_t CMC_GetCoreClockGatedStatus(CMC_Type *base)
```

Gets the core clock gated status.

This function get the status to indicate whether the core clock is gated. The core clock gated status can be cleared by software.

#### Parameters

- base – CMC peripheral base address.

#### Returns

The status to indicate whether the core clock is gated.

```
static inline void CMC_ClearCoreClockGatedStatus(CMC_Type *base)
```

Clears the core clock gated status.

This function clear clock status flag by software.

#### Parameters

- base – CMC peripheral base address.

```
static inline uint8_t CMC_GetWakeupSource(CMC_Type *base)
```

Gets the Wakeup Source.

This function gets the Wakeup sources from the previous low power mode entry.

#### Parameters

- base – CMC peripheral base address.

#### Returns

The Wakeup sources from the previous low power mode entry. See `_cmc_wakeup_sources` for details.

```
static inline cmc_clock_mode_t CMC_GetClockMode(CMC_Type *base)
```

Gets the Clock mode.

This function gets the clock mode of the previous low power mode entry.

#### Parameters

- base – CMC peripheral base address.

#### Returns

The Low Power status.

```
static inline uint32_t CMC_GetSystemResetStatus(CMC_Type *base)
```

Gets the System reset status.

This function returns the system reset status. Those status updates on every MAIN Warm Reset to indicate the type/source of the most recent reset.

#### Parameters

- base – CMC peripheral base address.

#### Returns

The most recent system reset status. See `_cmc_system_reset_sources` for details.

```
static inline uint32_t CMC_GetStickySystemResetStatus(CMC_Type *base)
```

Gets the sticky system reset status since the last WAKE Cold Reset.

This function gets all source of system reset that have generated a system reset since the last WAKE Cold Reset, and that have not been cleared by software.

**Parameters**

- base – CMC peripheral base address.

**Returns**

System reset status that have not been cleared by software. See `_cmc_system_reset_sources` for details.

```
static inline void CMC_ClearStickySystemResetStatus(CMC_Type *base, uint32_t mask)
```

Clears the sticky system reset status flags.

**Parameters**

- base – CMC peripheral base address.
- mask – Bitmap of the sticky system reset status to be cleared.

```
static inline uint8_t CMC_GetResetCount(CMC_Type *base)
```

Gets the number of reset sequences completed since the last Cold Reset.

**Parameters**

- base – CMC peripheral base address.

**Returns**

The number of reset sequences.

```
void CMC_SetPowerModeProtection(CMC_Type *base, uint32_t allowedModes)
```

Configures all power mode protection settings.

This function configures the power mode protection settings for supported power modes. This should be done before set the lowPower mode for each power doamin.

The allowed lowpower modes are passed as bit map. For example, to allow Sleep and DeepSleep, use `CMC_SetPowerModeProtection(CMC_base, kCMC_AllowSleepMode | kCMC_AllowDeepSleepMode)`. To allow all low power modes, use `CMC_SetPowerModeProtection(CMC_base, kCMC_AllowAllLowPowerModes)`.

**Parameters**

- base – CMC peripheral base address.
- allowedModes – Bitmaps of the allowed power modes. See `_cmc_power_mode_protection` for details.

```
static inline void CMC_LockPowerModeProtectionSetting(CMC_Type *base)
```

Locks the power mode protection.

This function locks the power mode protection. After invoking this function, any power mode protection setting will be ignored.

**Parameters**

- base – CMC peripheral base address.

```
static inline void CMC_SetGlobalPowerMode(CMC_Type *base, cmc_low_power_mode_t  
lowPowerMode)
```

Config the same lowPower mode for all power domain.

This function configures the same low power mode for MAIN power domian and WAKE power domain.

**Parameters**

- `base` – CMC peripheral base address.
- `lowPowerMode` – The desired lowPower mode. See `cmc_low_power_mode_t` for details.

```
static inline void CMC_SetMAINPowerMode(CMC_Type *base, cmc_low_power_mode_t
                                         lowPowerMode)
```

Configures entry into low power mode for the MAIN Power domain.

This function configures the low power mode for the MAIN power domain, when the core executes WFI/WFE instruction. The available lowPower modes are defined in the `cmc_low_power_mode_t`.

#### Parameters

- `base` – CMC peripheral base address.
- `lowPowerMode` – The desired lowPower mode. See `cmc_low_power_mode_t` for details.

```
static inline cmc_low_power_mode_t CMC_GetMAINPowerMode(CMC_Type *base)
```

Gets the power mode of the MAIN Power domain.

#### Parameters

- `base` – CMC peripheral base address.

#### Returns

The power mode of MAIN Power domain. See `cmc_low_power_mode_t` for details.

```
void CMC_ConfigResetPin(CMC_Type *base, const cmc_reset_pin_config_t *config)
```

Configure reset pin.

This function configures reset pin. When enabled, the low power filter is enabled in both Active and Low power modes, the reset filter is only enabled in Active mode. When both filters are enabled, they operate in series.

#### Parameters

- `base` – CMC peripheral base address.
- `config` – Pointer to the reset pin config structure.

```
static inline void CMC_EnableSystemResetInterrupt(CMC_Type *base, uint32_t mask)
```

Enable system reset interrupts.

This function enables the system reset interrupts. The assertion of non-fatal warm reset can be delayed for 258 cycles of the 32K\_CLK clock while an enabled interrupt is generated. Then Software can perform a graceful shutdown or abort the non-fatal warm reset provided the pending reset source is cleared by resetting the reset source and then clearing the pending flag.

#### Parameters

- `base` – CMC peripheral base address.
- `mask` – System reset interrupts. See `_cmc_system_reset_interrupt_enable` for details.

```
static inline void CMC_DisableSystemResetInterrupt(CMC_Type *base, uint32_t mask)
```

Disable system reset interrupts.

This function disables the system reset interrupts.

#### Parameters

- `base` – CMC peripheral base address.

- mask – System reset interrupts. See `_cmc_system_reset_interrupt_enable` for details.

static inline uint32\_t CMC\_GetSystemResetInterruptFlags(CMC\_Type \*base)

Gets System Reset interrupt flags.

This function returns the System reset interrupt flags.

#### Parameters

- base – CMC peripheral base address.

#### Returns

System reset interrupt flags. See `_cmc_system_reset_interrupt_flag` for details.

static inline void CMC\_ClearSystemResetInterruptFlags(CMC\_Type \*base, uint32\_t mask)

Clears System Reset interrupt flags.

This function clears system reset interrupt flags. The pending reset source can be cleared by resetting the source of the reset and then clearing the pending flags.

#### Parameters

- base – CMC peripheral base address.
- mask – System Reset interrupt flags. See `_cmc_system_reset_interrupt_flag` for details.

static inline void CMC\_EnableNonMaskablePinInterrupt(CMC\_Type \*base, bool enable)

Enable/Disable Non maskable Pin interrupt.

#### Parameters

- base – CMC peripheral base address.
- enable – Enable or disable Non maskable pin interrupt. true - enable Non-maskable pin interrupt. false - disable Non-maskable pin interrupt.

static inline uint8\_t CMC\_GetISPMODEPinLogic(CMC\_Type \*base)

Gets the logic state of the ISPMODE\_n pin.

This function returns the logic state of the ISPMODE\_n pin on the last negation of RESET\_b pin.

#### Parameters

- base – CMC peripheral base address.

#### Returns

The logic state of the ISPMODE\_n pin on the last negation of RESET\_b pin.

static inline void CMC\_ClearISPMODEPinLogic(CMC\_Type \*base)

Clears ISPMODE\_n pin state.

#### Parameters

- base – CMC peripheral base address.

static inline void CMC\_ForceBootConfiguration(CMC\_Type \*base, bool assert)

Set the logic state of the BOOT\_CONFIGn pin.

This function force the logic state of the Boot\_Confign pin to assert on next system reset.

#### Parameters

- base – CMC peripheral base address.
- assert – Assert the corresponding pin or not. true - Assert corresponding pin on next system reset. false - No effect.



```
static inline uint32_t CMC_GetBootRomStatus(CMC_Type *base)
```

Gets the status information written by the BootROM.

**Parameters**

- base – CMC peripheral base address.

**Returns**

The status information written by the BootROM.

```
static inline void CMC_SetBootRomStatus(CMC_Type *base, uint32_t statValue)
```

Sets the bootROM status value.

---

**Note:** This function is useful when result of CMC\_CheckBootRomRegisterWrittable() is true.

---

**Parameters**

- base – CMC peripheral base address.
- stat – The state value to set.

```
static inline bool CMC_CheckBootRomRegisterWrittable(CMC_Type *base)
```

Check if BootROM status and lock registers is writable.

**Parameters**

- base – CMC peripheral base address.

**Returns**

The result of whether BootROM status and lock register is writable.

- **true** BootROM status and lock registers are writable;
- **false** BootROM status and lock registers are not writable.

```
static inline void CMC_LockBootRomStatusWritten(CMC_Type *base)
```

After invoking this function, BootROM status and lock registers cannot be written.

**Parameters**

- base – CMC peripheral base address.

```
static inline void CMC_UnlockBootRomStatusWritten(CMC_Type *base)
```

After invoking this function, BootROM status and lock register can be written.s.

**Parameters**

- base –

```
void CMC_PowerOffSRAMAllMode(CMC_Type *base, uint32_t mask)
```

Power off the selected system SRAM always.

---

**Note:** This function power off the selected system SRAM always. The SRAM arrays should not be accessed while they are shut down. SRAM array contents are not retained if they are powered off.

---

---

**Note:** Once invoked, the previous settings will be overwritten.

---

**Parameters**

- base – CMC peripheral base address.

- mask – Bitmap of the SRAM arrays to be powered off all modes. See `_cmc_system_sram_arrays` for details. Check Reference Manual for the SRAM region and mask bit relationship.

static inline void CMC\_PowerOnSRAMAllMode(CMC\_Type \*base, uint32\_t mask)

Power on SRAM during all mode.

---

**Note:** Once invoked, the previous settings will be overwritten.

---

#### Parameters

- base – CMC peripheral base address.
- mask – Bitmap of the SRAM arrays to be powered on all modes. See `_cmc_system_sram_arrays` for details. Check Reference Manual for the SRAM region and mask bit relationship.

void CMC\_PowerOffSRAMLowPowerOnly(CMC\_Type \*base, uint32\_t mask)

Power off the selected system SRAM during low power modes only.

This function power off the selected system SRAM only during low power mode. SRAM array contents are not retained if they are power off.

#### Parameters

- base – CMC peripheral base address.
- mask – Bitmap of the SRAM arrays to be power off during low power mode only. See `_cmc_system_sram_arrays` for details. Check Reference Manual for the SRAM region and mask bit relationship.

static inline void CMC\_PowerOnSRAMLowPowerOnly(CMC\_Type \*base, uint32\_t mask)

Power on the selected system SRAM during low power modes only.

This function power on the selected system SRAM. The SRAM array contents are retained in low power modes.

#### Parameters

- base – CMC peripheral base address.
- mask – Bitmap of the SRAM arrays to be power on during low power mode only. See `_cmc_system_sram_arrays` for details. Check Reference Manual for the SRAM region and mask bit relationship.

void CMC\_ConfigFlashMode(CMC\_Type \*base, bool wake, bool doze, bool disable)

Configs the low power mode of the on-chip flash memory.

This function configs the low power mode of the on-chip flash memory.

#### Parameters

- base – CMC peripheral base address.
- wake – true: Flash will exit low power state during the flash memory accesses. false: No effect.
- doze – true: Flash is disabled while core is sleeping false: No effect.
- disable – true: Flash memory is placed in low power state. false: No effect.

static inline void CMC\_EnableDebugOperation(CMC\_Type \*base, bool enable)

Enables/Disables debug Operation when the core sleep.

This function configs what happens to debug when core sleeps.

#### Parameters

- base – CMC peripheral base address.
- enable – Enable or disable Debug when Core is sleeping. true - Debug remains enabled when the core is sleeping. false - Debug is disabled when the core is sleeping.

`void CMC_PreEnterLowPowerMode(void)`

Prepares to enter low power modes.

This function should be called before entering low power modes.

`void CMC_PostExitLowPowerMode(void)`

Recovers after wake up from low power modes.

This function should be called after wake up from low power modes. This function should be used with `CMC_PreEnterLowPowerMode()`

`void CMC_GlobalEnterLowPowerMode(CMC_Type *base, cmc_low_power_mode_t lowPowerMode)`

Configures the entry into the same low power mode for each power domains.

This function provides the feature to entry into the same low power mode for each power domains. Before invoking this function, please ensure the selected power mode have been allowed.

#### Parameters

- base – CMC peripheral base address.
- lowPowerMode – The low power mode to be entered. See `cmc_low_power_mode_t` for the details.

`void CMC_EnterLowPowerMode(CMC_Type *base, const cmc_power_domain_config_t *config)`

Configures the entry into different low power modes for each power domains.

This function provides the feature to entry into different low power modes for each power domains. Before invoking this function please ensure the selected modes are allowed.

#### Parameters

- base – CMC peripheral base address.
- config – Pointer to the `cmc_power_domain_config_t` structure.

`bool lowpowerFilterEnable`

Low Power Filter enable.

`bool resetFilterEnable`

Reset Filter enable.

`uint8_t resetFilterWidth`

Width of the Reset Filter.

`cmc_clock_mode_t clock_mode`

Clock mode for each power domain.

`cmc_low_power_mode_t main_domain`

The low power mode of the MAIN power domain.

`struct _cmc_reset_pin_config`

`#include <fsl_cmc.h>` CMC reset pin configuration.

`struct _cmc_power_domain_config`

`#include <fsl_cmc.h>` power mode configuration for each power domain.

## 2.42 MCX\_SPC: System Power Control driver

uint8\_t SPC\_GetPeriphIOIsolationStatus(SPC\_Type \*base)

Gets Isolation status for each power domains.

This function gets the status which indicates whether certain peripheral and the IO pads are in a latched state as a result of having been in POWERDOWN mode.

### Parameters

- base – SPC peripheral base address.

### Returns

Current isolation status for each power domains. See `_spc_power_domains` for details.

static inline void SPC\_ClearPeriphIOIsolationFlag(SPC\_Type \*base)

Clears peripherals and I/O pads isolation flags for each power domains.

This function clears peripherals and I/O pads isolation flags for each power domains. After recovering from the POWERDOWN mode, user must invoke this function to release the I/O pads and certain peripherals to their normal run mode state. Before invoking this function, user must restore chip configuration in particular pin configuration for enabled WUU wakeup pins.

### Parameters

- base – SPC peripheral base address.

static inline bool SPC\_GetBusyStatusFlag(SPC\_Type \*base)

Gets SPC busy status flag.

This function gets SPC busy status flag. When SPC executing any type of power mode transition in ACTIVE mode or any of the SOC low power mode, the SPC busy status flag is set and this function returns true. When changing CORE LDO voltage level and DCDC voltage level in ACTIVE mode, the SPC busy status flag is set and this function return true.

### Parameters

- base – SPC peripheral base address.

### Returns

Ack busy flag. true - SPC is busy. false - SPC is not busy.

static inline bool SPC\_CheckLowPowerRequest(SPC\_Type \*base)

Checks system low power request.

---

**Note:** Only when all power domains request low power mode entry, the result of this function is true. That means when all power domains request low power mode entry, the SPC regulators will be controlled by LP\_CFG register.

---

### Parameters

- base – SPC peripheral base address.

### Returns

The system low power request check result.

- **true** All power domains have requested low power mode and SPC has entered a low power state and power mode configuration are based on the LP\_CFG configuration register.
- **false** SPC in active mode and ACTIVE\_CFG register control system power supply.

```
static inline void SPC_ClearLowPowerRequest(SPC_Type *base)
```

Clears system low power request, set SPC in active mode.

#### Parameters

- `base` – SPC peripheral base address.

```
static inline spc_power_domain_low_power_mode_t SPC_GetRequestedLowPowerMode(SPC_Type  
                                                                           *base)
```

Check the last low-power mode that the power domain requested.

#### Parameters

- `base` – SPC peripheral base address.

#### Returns

The last low-power mode that the power domain requested.

```
static inline bool SPC_CheckSwitchState(SPC_Type *base)
```

Checks whether the power switch is on.

#### Parameters

- `base` – SPC peripheral base address.

#### Return values

- `true` – The power switch is on.
- `false` – The power switch is off.

```
spc_power_domain_low_power_mode_t SPC_GetPowerDomainLowPowerMode(SPC_Type *base,  
                                                                    spc_power_domain_id_t  
                                                                    powerDomainId)
```

Gets selected power domain's requested low power mode.

#### Parameters

- `base` – SPC peripheral base address.
- `powerDomainId` – Power Domain Id, please refer to `spc_power_domain_id_t`.

#### Returns

The selected power domain's requested low power mode, please refer to `spc_power_domain_low_power_mode_t`.

```
static inline bool SPC_CheckPowerDomainLowPowerRequest(SPC_Type *base,  
                                                         spc_power_domain_id_t  
                                                         powerDomainId)
```

Checks power domain's low power request.

#### Parameters

- `base` – SPC peripheral base address.
- `powerDomainId` – Power Domain Id, please refer to `spc_power_domain_id_t`.

#### Returns

The result of power domain's low power request.

- **true** The selected power domain requests low power mode entry.
- **false** The selected power domain does not request low power mode entry.

```
static inline void SPC_ClearPowerDomainLowPowerRequestFlag(SPC_Type *base,  
                                                           spc_power_domain_id_t  
                                                           powerDomainId)
```

Clears selected power domain's low power request flag.

#### Parameters

- base – SPC peripheral base address.
- powerDomainId – Power Domain Id, please refer to `spc_power_domain_id_t`.

```
static inline void SPC_TrimSRAMLdoRefVoltage(SPC_Type *base, uint8_t trimValue)  
Trims SRAM retention regulator reference voltage, trim step is 12 mV, range is around 0.48V  
to 0.85V.
```

#### Parameters

- base – SPC peripheral base address.
- trimValue – Reference voltage trim value.

```
static inline void SPC_EnableSRAMLdo(SPC_Type *base, bool enable)  
Enables/disables SRAM retention LDO.
```

#### Parameters

- base – SPC peripheral base address.
- enable – Used to enable/disable SRAM LDO :
  - **true** Enable SRAM LDO;
  - **false** Disable SRAM LDO.

```
static inline void SPC_RetainSRAMArray(SPC_Type *base, uint8_t mask)
```

#### Parameters

- base – SPC peripheral base address.
- mask – The OR'ed value of SRAM Array.

```
static inline void SPC_UnRetainSRAMArray(SPC_Type *base, uint8_t mask)  
Unretain SRAM array.
```

#### Parameters

- base – SPC peripheral base address.
- mask – The OR'ed value of SRAM Array.

```
void SPC_SetLowPowerRequestConfig(SPC_Type *base, const spc_lowpower_request_config_t  
                                *config)
```

Configs Low power request output pin.

This function config the low power request output pin

#### Parameters

- base – SPC peripheral base address.
- config – Pointer the `spc_lowpower_request_config_t` structure.

```
static inline void SPC_EnableIntegratedPowerSwitchManually(SPC_Type *base, bool enable)  
Enables/disables the integrated power switch manually.
```

#### Parameters

- base – SPC peripheral base address.

- **enable** – Used to enable/disable the integrated power switch:
  - **true** Enable the integrated power switch;
  - **false** Disable the integrated power switch.

```
static inline void SPC__EnableIntegratedPowerSwitchAutomatically(SPC_Type *base, bool
                                                                sleepGate, bool
                                                                wakeupUngate)
```

Enables/disables the integrated power switch automatically.

To gate the integrated power switch when chip enter low power modes, and ungate the switch after wake-up from low power modes:

```
SPC__EnableIntegratedPowerSwitchAutomatically(SPC, true, true);
```

#### Parameters

- **base** – SPC peripheral base address.
- **sleepGate** – Enable the integrated power switch when chip enter low power modes:
  - **true** SPC asserts an output pin at low-power entry to power-gate the switch;
  - **false** SPC does not assert an output pin at low-power entry to power-gate the switch.
- **wakeupUngate** – Enables the switch after wake-up from low power modes:
  - **true** SPC asserts an output pin at low-power exit to power-ungate the switch;
  - **false** SPC does not assert an output pin at low-power exit to power-ungate the switch.

```
void SPC__SetSRAMOperateVoltage(SPC_Type *base, const spc_sram_voltage_config_t *config)
```

Set SRAM operate voltage.

#### Parameters

- **base** – SPC peripheral base address.
- **config** – The pointer to *spc\_sram\_voltage\_config\_t*, specifies the configuration of sram voltage.

```
static inline spc_bandgap_mode_t SPC__GetActiveModeBandgapMode(SPC_Type *base)
```

Gets the Bandgap mode in Active mode.

#### Parameters

- **base** – SPC peripheral base address.

#### Returns

Bandgap mode in the type of *spc\_bandgap\_mode\_t* enumeration.

```
static inline uint32_t SPC__GetActiveModeVoltageDetectStatus(SPC_Type *base)
```

Gets all voltage detectors status in Active mode.

#### Parameters

- **base** – SPC peripheral base address.

#### Returns

All voltage detectors status in Active mode.

*status\_t* SPC\_SetActiveModeBandgapModeConfig(SPC\_Type \*base, *spc\_bandgap\_mode\_t* mode)  
Config Bandgap mode in Active mode.

---

**Note:** To disable bandgap in Active mode:

- a. Disable all LVD's and HVD's in active mode;
  - b. Disable Glitch detect;
  - c. Configure LDO's and DCDC to low drive strength in active mode;
  - d. Invoke this function to disable bandgap in active mode; otherwise the error status will be reported.
- 

---

**Note:** Some other system resources(such as PLL, CMP) require bandgap to be enabled, to disable bandgap please take care of other system resources.

---

#### Parameters

- base – SPC peripheral base address.
- mode – The Bandgap mode be selected.

#### Return values

- kStatus\_SPC\_BandgapModeWrong – The Bandgap can not be disabled in active mode.
- kStatus\_Success – Config Bandgap mode in Active power mode successful.

static inline void SPC\_EnableActiveModeCMPBandgapBuffer(SPC\_Type \*base, bool enable)  
Enables/Disable the CMP Bandgap Buffer in Active mode.

#### Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable CMP Bandgap buffer. true - Enable Buffer Stored Reference voltage to CMP. false - Disable Buffer Stored Reference voltage to CMP.

static inline void SPC\_SetActiveModeVoltageTrimDelay(SPC\_Type \*base, uint16\_t delay)  
Sets the delay when the regulators change voltage level in Active mode.

#### Parameters

- base – SPC peripheral base address.
- delay – The number of SPC timer clock cycles.

*status\_t* SPC\_SetActiveModeRegulatorsConfig(SPC\_Type \*base, const  
*spc\_active\_mode\_regulators\_config\_t* \*config)  
Config all settings of regulators in Active mode at a time.

---

**Note:** This function is used to overwrite all settings of regulators(including bandgap mode, regulators' drive strength and voltage level) in active mode at a time.

---

---

**Note:** Enable/disable LVDs/HVDs before invoking this function.

---



---

**Note:** This function will check input parameters based on hardware restrictions before setting registers, if input parameters do not satisfy hardware restrictions the specific error will be reported.

---

---

**Note:** Some hardware restrictions not covered, application should be aware of this and follow this hardware restrictions otherwise some unknown issue may occur:

- a. If Core LDO's drive strength are set to same value in both Active mode and low power mode, the voltage level should also set to same value.
  - b. When switching Core LDO's drive strength from low to normal, ensure the LDO\_CORE high voltage level is set to same level that was set prior to switching to the LDO\_CORE drive strength. Otherwise, if the LVDs are enabled, an unexpected LVD can occur.
- 

---

**Note:** If this function can not satisfy some tricky settings, please invoke other APIs in low-level function group.

---

### Parameters

- base – SPC peripheral base address.
- config – Pointer to `spc_active_mode_regulators_config_t` structure.

### Return values

- `kStatus_Success` – Config regulators in Active power mode successful.
- `kStatus_SPC_BandgapModeWrong` – Based on input setting, bandgap can not be disabled.
- `kStatus_SPC_Busy` – The SPC instance is busy to execute any type of power mode transition.
- `kStatus_SPC_CORELDOLowDriveStrengthIgnore` – Any of LVDs/HVDs kept enabled before invoking this function.
- `kStatus_SPC_SYSLDOOverDriveVoltageFail` – Fail to regulator to Over Drive Voltage due to System VDD HVD is not disabled.
- `kStatus_SPC_SYSLDOLowDriveStrengthIgnore` – Any of LVDs/HVDs kept enabled before invoking this function.
- `kStatus_SPC_CORELDOLowVoltageWrong` – Core LDO and System LDO do not have same voltage level.

`static inline void SPC_EnableActiveModeAnalogModules(SPC_Type *base, uint32_t maskValue)`  
Enables analog modules in active mode.

### Parameters

- base – SPC peripheral base address.
- maskValue – The mask of analog modules to enable in active mode, should be the OR'ed value of `spc_analog_module_control`.

`static inline void SPC_DisableActiveModeAnalogModules(SPC_Type *base, uint32_t maskValue)`  
Disables analog modules in active mode.

### Parameters

- base – SPC peripheral base address.

- `maskValue` – The mask of analog modules to disable in active mode, should be the OR'ed value of `spc_analog_module_control`.

static inline uint32\_t SPC\_GetActiveModeEnabledAnalogModules(SPC\_Type \*base)

Gets enabled analog modules that enabled in active mode.

#### Parameters

- `base` – SPC peripheral base address.

#### Returns

The mask of enabled analog modules that enabled in active mode.

static inline *spc\_bandgap\_mode\_t* SPC\_GetLowPowerModeBandgapMode(SPC\_Type \*base)

Gets the Bandgap mode in Low Power mode.

#### Parameters

- `base` – SPC peripheral base address.

#### Returns

Bandgap mode in the type of *spc\_bandgap\_mode\_t* enumeration.

static inline uint32\_t SPC\_GetLowPowerModeVoltageDetectStatus(SPC\_Type \*base)

Gets the status of all voltage detectors in Low Power mode.

#### Parameters

- `base` – SPC peripheral base address.

#### Returns

The status of all voltage detectors in low power mode.

static inline void SPC\_EnableLowPowerModeLowPowerIREF(SPC\_Type \*base, bool enable)

Enables/Disables Low Power IREF in low power modes.

This function enables/disables Low Power IREF. Low Power IREF can only get disabled in Deep power down mode. In other low power modes, the Low Power IREF is always enabled.

#### Parameters

- `base` – SPC peripheral base address.
- `enable` – Enable/Disable Low Power IREF. true - Enable Low Power IREF for Low Power modes. false - Disable Low Power IREF for Deep Power Down mode.

*status\_t* SPC\_SetLowPowerModeBandgapmodeConfig(SPC\_Type \*base, *spc\_bandgap\_mode\_t* mode)

Configs Bandgap mode in Low Power mode.

---

**Note:** To disable Bandgap in Low-power mode:

- Disable all LVD's and HVD's in low power mode;
  - Disable Glitch detect in low power mode;
  - Configure LDO's and DCDC to low drive strength in low power mode;
  - Disable bandgap in low power mode; Otherwise, the error status will be reported.
- 

---

**Note:** Some other system resources(such as PLL, CMP) require bandgap to be enabled, to disable bandgap please take care of other system resources.

---

#### Parameters

- base – SPC peripheral base address.
- mode – The Bandgap mode be selected.

**Return values**

- kStatus\_SPC\_BandgapModeWrong – The bandgap mode setting in Low Power mode is wrong.
- kStatus\_Success – Config Bandgap mode in Low Power power mode successful.

static inline void SPC\_EnableSRAMLdOLowPowerModeIREF(SPC\_Type \*base, bool enable)  
Enables/disables SRAM\_LDO deep power low power IREF.

**Parameters**

- base – SPC peripheral base address.
- enable – Used to enable/disable low power IREF :
  - **true:** Low Power IREF is enabled ;
  - **false:** Low Power IREF is disabled for power saving.

static inline void SPC\_EnableLowPowerModeCMPBandgapBufferMode(SPC\_Type \*base, bool enable)

Enables/Disables CMP Bandgap Buffer.

This function gates CMP bandgap buffer. CMP bandgap buffer is automatically disabled and turned off in Deep Power Down mode.

*Deprecated:*

No longer used, please use SPC\_EnableLowPowerModeCMPBandgapBuffer as instead.

**Parameters**

- base – SPC peripheral base address.
- enable – Enable/Disable CMP Bandgap buffer. true - Enable Buffer Stored Reference Voltage to CMP. false - Disable Buffer Stored Reference Voltage to CMP.

static inline void SPC\_EnableLowPowerModeCMPBandgapBuffer(SPC\_Type \*base, bool enable)  
Enables/Disables CMP Bandgap Buffer.

This function gates CMP bandgap buffer. CMP bandgap buffer is automatically disabled and turned off in Deep Power Down mode.

*Deprecated:*

No longer used.

**Parameters**

- base – SPC peripheral base address.
- enable – Enable/Disable CMP Bandgap buffer. true - Enable Buffer Stored Reference Voltage to CMP. false - Disable Buffer Stored Reference Voltage to CMP.

static inline void SPC\_EnableLowPowerModeCoreVDDInternalVoltageScaling(SPC\_Type \*base, bool enable)

Enables/Disables CORE VDD IVS(Internal Voltage Scaling) in power down modes.

This function gates CORE VDD IVS. When enabled, the IVS regulator will scale the external input CORE VDD to a lower voltage level to reduce internal leakage. IVS is invalid in Sleep or Deep power down mode.

#### Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IVS. true - enable CORE VDD IVS in Power Down mode. false - disable CORE VDD IVS in Power Down mode.

```
static inline void SPC_SetLowPowerWakeUpDelay(SPC_Type *base, uint16_t delay)
```

Sets the delay when exit the low power modes.

#### Parameters

- base – SPC peripheral base address.
- delay – The number of SPC timer clock cycles that the SPC waits on exit from low power modes.

```
status_t SPC_SetLowPowerModeRegulatorsConfig(SPC_Type *base, const
                                              spc_lowpower_mode_regulators_config_t
                                              *config)
```

Configs all settings of regulators in Low power mode at a time.

---

**Note:** This function is used to overwrite all settings of regulators(including bandgap mode, regulators' drive strength and voltage level) in low power mode at a time.

---

---

**Note:** Enable/disable LVDs/HVDs before invoking this function.

---

---

**Note:** This function will check input parameters based on hardware restrictions before setting registers, if input parameters do not satisfy hardware restrictions the specific error will be reported.

---

---

**Note:** Some hardware restrictions not covered, application should be aware of this and follow this hardware restrictions otherwise some unknown issue may occur:

- a. If Core LDO's drive strength are set to same value in both Active mode and low power mode, the voltage level should also set to same value.
  - b. When switching Core LDO's drive strength from low to normal, ensure the LDO\_CORE high voltage level is set to same level that was set prior to switching to the LDO\_CORE drive strength. Otherwise, if the LVDs are enabled, an unexpected LVD can occur.
- 

---

**Note:** If this function can not satisfy some tricky settings, please invoke other APIs in low-level function group.

---

#### Parameters

- base – SPC peripheral base address.
- config – Pointer to spc\_lowpower\_mode\_regulators\_config\_t structure.

**Return values**

- `kStatus_Success` – Config regulators in Low power mode successful.
- `kStatus_SPC_BandgapModeWrong` – The bandgap should not be disabled based on input settings.
- `kStatus_SPC_CORELDOLowDriveStrengthIgnore` – Set driver strength to low will be ignored.
- `kStatus_SPC_SYSLDOLowDriveStrengthIgnore` – Set driver strength to low will be ignored.
- `kStatus_SPC_CORELDOLowVoltageWrong` – Core LDO and System LDO do not have same voltage level.

```
static inline void SPC_EnableLowPowerModeAnalogModules(SPC_Type *base, uint32_t  
                                                       maskValue)
```

Enables analog modules in low power modes.

**Parameters**

- `base` – SPC peripheral base address.
- `maskValue` – The mask of analog modules to enable in low power modes, should be OR'ed value of `spc_analog_module_control`.

```
static inline void SPC_DisableLowPowerModeAnalogModules(SPC_Type *base, uint32_t  
                                                       maskValue)
```

Disables analog modules in low power modes.

**Parameters**

- `base` – SPC peripheral base address.
- `maskValue` – The mask of analog modules to disable in low power modes, should be OR'ed value of `spc_analog_module_control`.

```
static inline uint32_t SPC_GetLowPowerModeEnabledAnalogModules(SPC_Type *base)
```

Gets enabled analog modules that enabled in low power modes.

**Parameters**

- `base` – SPC peripheral base address.

**Returns**

The mask of enabled analog modules that enabled in low power modes.

```
static inline uint32_t SPC_GetVoltageDetectStatusFlag(SPC_Type *base)
```

Get Voltage Detect Status Flags.

**Parameters**

- `base` – SPC peripheral base address.

**Returns**

Voltage Detect Status Flags. See `_spc_voltage_detect_flags` for details.

```
static inline void SPC_ClearVoltageDetectStatusFlag(SPC_Type *base, uint8_t mask)
```

Clear Voltage Detect Status Flags.

**Parameters**

- `base` – SPC peripheral base address.
- `mask` – The mask of the voltage detect status flags. See `_spc_voltage_detect_flags` for details.

```
void SPC_SetCoreVoltageDetectConfig(SPC_Type *base, const spc_core_voltage_detect_config_t
                                   *config)
```

Configs CORE voltage detect options.

---

**Note:** : Setting both the voltage detect interrupt and reset enable will cause interrupt to be generated on exit from reset. If those conditioned is not desired, interrupt/reset so only one is enabled.

---

#### Parameters

- base – SPC peripheral base address.
- config – Pointer to *spc\_core\_voltage\_detect\_config\_t* structure.

```
static inline void SPC_LockCoreVoltageDetectResetSetting(SPC_Type *base)
```

Locks Core voltage detect reset setting.

This function locks core voltage detect reset setting. After invoking this function any configuration of Core voltage detect reset will be ignored.

#### Parameters

- base – SPC peripheral base address.

```
static inline void SPC_UnlockCoreVoltageDetectResetSetting(SPC_Type *base)
```

Unlocks Core voltage detect reset setting.

This function unlocks core voltage detect reset setting. If locks the Core voltage detect reset setting, invoking this function to unlock.

#### Parameters

- base – SPC peripheral base address.

```
status_t SPC_EnableActiveModeCoreLowVoltageDetect(SPC_Type *base, bool enable)
```

Enables/Disables the Core Low Voltage Detector in Active mode.

---

**Note:** If the CORE\_LDO low voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low.

---

#### Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Core LVD. true - Enable Core Low voltage detector in active mode. false - Disable Core Low voltage detector in active mode.

#### Return values

kStatus\_Success – Enable/Disable Core Low Voltage Detect successfully.

```
status_t SPC_EnableLowPowerModeCoreLowVoltageDetect(SPC_Type *base, bool enable)
```

Enables/Disables the Core Low Voltage Detector in Low Power mode.

This function enables/disables the Core Low Voltage Detector. If enabled the Core Low Voltage detector. The Bandgap mode in low power mode must be programmed so that Bandgap is enabled.

---

**Note:** If the CORE\_LDO low voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

---

**Parameters**

- base – SPC peripheral base address.
- enable – Enable/Disable Core HVD. true - Enable Core Low voltage detector in low power mode. false - Disable Core Low voltage detector in low power mode.

**Return values**

kStatus\_Success – Enable/Disable Core Low Voltage Detect in low power mode successfully.

*status\_t* SPC\_EnableActiveModeCoreHighVoltageDetect(SPC\_Type \*base, bool enable)

Enables/Disables the Core High Voltage Detector in Active mode.

---

**Note:** If the CORE\_LDO high voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low.

---

**Parameters**

- base – SPC peripheral base address.
- enable – Enable/Disable Core HVD. true - Enable Core High voltage detector in active mode. false - Disable Core High voltage detector in active mode.

**Return values**

kStatus\_Success – Enable/Disable Core High Voltage Detect successfully.

*status\_t* SPC\_EnableLowPowerModeCoreHighVoltageDetect(SPC\_Type \*base, bool enable)

Enables/Disables the Core High Voltage Detector in Low Power mode.

This function enables/disables the Core High Voltage Detector. If enabled the Core High Voltage detector. The Bandgap mode in low power mode must be programmed so that Bandgap is enabled.

---

**Note:** If the CORE\_LDO high voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in low power mode.

---

**Parameters**

- base – SPC peripheral base address.
- enable – Enable/Disable Core HVD. true - Enable Core High voltage detector in low power mode. false - Disable Core High voltage detector in low power mode.

**Return values**

kStatus\_Success – Enable/Disable Core High Voltage Detect in low power mode successfully.

void SPC\_SetSystemVDDLowVoltageLevel(SPC\_Type \*base, *spc\_low\_voltage\_level\_select\_t* level)

Set system VDD Low-voltage level selection.

This function selects the system VDD low-voltage level. Changing system VDD low-voltage level must be done after disabling the System VDD low voltage reset and interrupt.

**Deprecated:**

In latest RM, reserved for all devices, will removed in next release.

**Parameters**

- base – SPC peripheral base address.
- level – System VDD Low-Voltage level selection.

```
void SPC__SetSystemVoltageDetectConfig(SPC_Type *base, const  
                                     spc_system_voltage_detect_config_t *config)
```

Configs SYS voltage detect options.

This function config SYS voltage detect options.

---

**Note:** : Setting both the voltage detect interrupt and reset enable will cause interrupt to be generated on exit from reset. If those conditioned is not desired, interrupt/reset so only one is enabled.

---

**Parameters**

- base – SPC peripheral base address.
- config – Pointer to `spc_system_voltage_detect_config_t` structure.

```
static inline void SPC__LockSystemVoltageDetectResetSetting(SPC_Type *base)
```

Lock System voltage detect reset setting.

This function locks system voltage detect reset setting. After invoking this function any configuration of System Voltage detect reset will be ignored.

**Parameters**

- base – SPC peripheral base address.

```
static inline void SPC__UnlockSystemVoltageDetectResetSetting(SPC_Type *base)
```

Unlock System voltage detect reset setting.

This function unlocks system voltage detect reset setting. If locks the System voltage detect reset setting, invoking this function to unlock.

**Parameters**

- base – SPC peripheral base address.

```
status_t SPC__EnableActiveModeSystemHighVoltageDetect(SPC_Type *base, bool enable)
```

Enables/Disables the System High Voltage Detector in Active mode.

---

**Note:** If the System\_LDO high voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Active mode.

---

**Parameters**

- base – SPC peripheral base address.
- enable – Enable/Disable System HVD. true - Enable System High voltage detector in active mode. false - Disable System High voltage detector in active mode.

**Return values**

kStatus\_\_Success – Enable/Disable System High Voltage Detect successfully.



*status\_t* SPC\_EnableActiveModeSystemLowVoltageDetect(SPC\_Type \*base, bool enable)

Enables/Disable the System Low Voltage Detector in Active mode.

---

**Note:** If the System\_LDO low voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Active mode.

---

#### Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable System LVD. true - Enable System Low voltage detector in active mode. false - Disable System Low voltage detector in active mode.

#### Return values

kStatus\_Success – Enable/Disable the System Low Voltage Detect successfully.

*status\_t* SPC\_EnableLowPowerModeSystemHighVoltageDetect(SPC\_Type \*base, bool enable)

Enables/Disables the System High Voltage Detector in Low Power mode.

---

**Note:** If the System\_LDO high voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

---

#### Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable System HVD. true - Enable System High voltage detector in low power mode. false - Disable System High voltage detector in low power mode.

#### Return values

kStatus\_Success – Enable/Disable System High Voltage Detect in low power mode successfully.

*status\_t* SPC\_EnableLowPowerModeSystemLowVoltageDetect(SPC\_Type \*base, bool enable)

Enables/Disables the System Low Voltage Detector in Low Power mode.

---

**Note:** If the System\_LDO low voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

---

#### Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable System HVD. true - Enable System Low voltage detector in low power mode. false - Disable System Low voltage detector in low power mode.

#### Return values

kStatus\_Success – Enables System Low Voltage Detect in low power mode successfully.

```
void SPC_SetIOVDDLowVoltageLevel(SPC_Type *base, spc_low_voltage_level_select_t level)
```

Set IO VDD Low-Voltage level selection.

This function selects the IO VDD Low-voltage level. Changing IO VDD low-voltage level must be done after disabling the IO VDD low voltage reset and interrupt.

#### Parameters

- base – SPC peripheral base address.
- level – IO VDD Low-voltage level selection.

```
void SPC_SetIOVoltageDetectConfig(SPC_Type *base, const spc_io_voltage_detect_config_t *config)
```

Configs IO voltage detect options.

This function config IO voltage detect options.

---

**Note:** : Setting both the voltage detect interrupt and reset enable will cause interrupt to be generated on exit from reset. If those conditioned is not desired, interrupt/reset so only one is enabled.

---

#### Parameters

- base – SPC peripheral base address.
- config – Pointer to `spc_voltage_detect_config_t` structure.

```
static inline void SPC_LockIOVoltageDetectResetSetting(SPC_Type *base)
```

Lock IO Voltage detect reset setting.

This function locks IO voltage detect reset setting. After invoking this function any configuration of system voltage detect reset will be ignored.

#### Parameters

- base – SPC peripheral base address.

```
static inline void SPC_UnlockIOVoltageDetectResetSetting(SPC_Type *base)
```

Unlock IO voltage detect reset setting.

This function unlocks IO voltage detect reset setting. If locks the IO voltage detect reset setting, invoking this function to unlock.

#### Parameters

- base – SPC peripheral base address.

```
status_t SPC_EnableActiveModeIOHighVoltageDetect(SPC_Type *base, bool enable)
```

Enables/Disables the IO High Voltage Detector in Active mode.

---

**Note:** If the IO high voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Active mode.

---

#### Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO HVD. true - Enable IO High voltage detector in active mode. false - Disable IO High voltage detector in active mode.

#### Return values

kStatus\_\_Success – Enable/Disable IO High Voltage Detect successfully.

*status\_t* SPC\_EnableActiveModeIOLowVoltageDetect(SPC\_Type \*base, bool enable)

Enables/Disables the IO Low Voltage Detector in Active mode.

---

**Note:** If the IO low voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Active mode.

---

#### Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO LVD. true - Enable IO Low voltage detector in active mode. false - Disable IO Low voltage detector in active mode.

#### Return values

kStatus\_Success – Enable IO Low Voltage Detect successfully.

*status\_t* SPC\_EnableLowPowerModeIOHighVoltageDetect(SPC\_Type \*base, bool enable)

Enables/Disables the IO High Voltage Detector in Low Power mode.

---

**Note:** If the IO high voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

---

#### Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO HVD. true - Enable IO High voltage detector in low power mode. false - Disable IO High voltage detector in low power mode.

#### Return values

kStatus\_Success – Enable IO High Voltage Detect in low power mode successfully.

*status\_t* SPC\_EnableLowPowerModeIOLowVoltageDetect(SPC\_Type \*base, bool enable)

Enables/Disables the IO Low Voltage Detector in Low Power mode.

---

**Note:** If the IO low voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

---

#### Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO LVD. true - Enable IO Low voltage detector in low power mode. false - Disable IO Low voltage detector in low power mode.

#### Return values

kStatus\_Success – Enable/Disable IO Low Voltage Detect in low power mode successfully.

void SPC\_SetExternalVoltageDomainsConfig(SPC\_Type \*base, uint8\_t lowPowerIsoMask, uint8\_t IsoMask)

Configs external voltage domains.

This function configs external voltage domains isolation.

**Parameters**

- base – SPC peripheral base address.
- lowPowerIsoMask – The mask of external domains isolate enable during low power mode. Please read the Reference Manual for the Bitmap.
- IsoMask – The mask of external domains isolate. Please read the Reference Manual for the Bitmap.

```
static inline uint8_t SPC_GetExternalDomainsStatus(SPC_Type *base)
```

Gets External Domains status.

**Parameters**

- base – SPC peripheral base address.

**Returns**

The status of each external domain.

```
static inline void SPC_EnableCoreLDORegulator(SPC_Type *base, bool enable)
```

Enable/Disable Core LDO regulator.

---

**Note:** The CORE LDO enable bit is write-once.

---

**Parameters**

- base – SPC peripheral base address.
- enable – Enable/Disable CORE LDO Regulator. true - Enable CORE LDO Regulator. false - Disable CORE LDO Regulator.

```
static inline void SPC_PullDownCoreLDORegulator(SPC_Type *base, bool pulldown)
```

Enable/Disable the CORE LDO Regulator pull down in Deep Power Down.

---

**Note:** This function only useful when enabled the CORE LDO Regulator.

---

**Parameters**

- base – SPC peripheral base address.
- pulldown – Enable/Disable CORE LDO pulldown in Deep Power Down mode. true - CORE LDO Regulator will discharge in Deep Power Down mode. false - CORE LDO Regulator will not discharge in Deep Power Down mode.

```
status_t SPC_SetActiveModeCoreLDORegulatorConfig(SPC_Type *base, const  
                                                  spc_active_mode_core_ldo_option_t  
                                                  *option)
```

Configs Core LDO Regulator in Active mode.

---

**Note:** The bandgap must be enabled before invoking this function.

---

---

**Note:** To set Core LDO as low drive strength, all HVDs/LVDs must be disabled previously.

---

**Parameters**

- base – SPC peripheral base address.
- option – Pointer to the spc\_active\_mode\_core\_ldo\_option\_t structure.

**Return values**

- `kStatus_Success` – Config Core LDO regulator in Active power mode successful.
- `kStatus_SPC_Busy` – The SPC instance is busy to execute any type of power mode transition.
- `kStatus_SPC_BandgapModeWrong` – Bandgap should be enabled before invoking this function.
- `kStatus_SPC_CORELDOLowDriveStrengthIgnore` – To set Core LDO as low drive strength, all LVDs/HVDs must be disabled before invoking this function.

```
status_t SPC_SetActiveModeCoreLDORegulatorVoltageLevel(SPC_Type *base,  
                                                         spc_core_ldo_voltage_level_t  
                                                         voltageLevel)
```

Set Core LDO Regulator Voltage level in Active mode.

---

**Note:** In active mode, the Core LDO voltage level should only be changed when the Core LDO is in normal drive strength.

---

---

**Note:** Update Core LDO voltage level will set Busy flag, this function return only when busy flag is cleared by hardware

---

**Parameters**

- `base` – SPC peripheral base address.
- `voltageLevel` – Specify the voltage level of CORE LDO Regulator in Active mode, please refer to `spc_core_ldo_voltage_level_t`.

**Return values**

- `kStatus_SPC_CORELDONotNormal` – The drive strength of Core LDO is not normal.
- `kStatus_Success` – Set Core LDO regulator voltage level in Active power mode successful.

```
static inline spc_core_ldo_voltage_level_t SPC_GetActiveModeCoreLDONormalVoltageLevel(SPC_Type  
                                                                                       *base)
```

Gets CORE LDO Regulator Voltage level.

This function returns the voltage level of CORE LDO Regulator in Active mode.

**Parameters**

- `base` – SPC peripheral base address.

**Returns**

Voltage level of CORE LDO in type of `spc_core_ldo_voltage_level_t` enumeration.

```
status_t SPC_SetActiveModeCoreLDORegulatorDriveStrength(SPC_Type *base,  
                                                         spc_core_ldo_drive_strength_t  
                                                         driveStrength)
```

Set Core LDO VDD Regulator Drive Strength in Active mode.

**Parameters**

- `base` – SPC peripheral base address.

- `driveStrength` – Specify the drive strength of CORE LDO Regulator in Active mode, please refer to `spc_core_ldo_drive_strength_t`.

#### Return values

- `kStatus_Success` – Set Core LDO regulator drive strength in Active power mode successful.
- `kStatus_SPC_CORELDOLowDriveStrengthIgnore` – If any voltage detect enabled, `core_ldo`'s drive strength can not set to low.
- `kStatus_SPC_BandgapModeWrong` – The selected bandgap mode is not allowed.

```
static inline spc_core_ldo_drive_strength_t SPC_GetActiveModeCoreLDODriveStrength(SPC_Type
                                                                                   *base)
```

Gets CORE LDO VDD Regulator Drive Strength in Active mode.

#### Parameters

- `base` – SPC peripheral base address.

#### Returns

Drive Strength of CORE LDO regulator in Active mode, please refer to `spc_core_ldo_drive_strength_t`.

```
status_t SPC_SetLowPowerModeCoreLDORegulatorConfig(SPC_Type *base, const
                                                    spc_lowpower_mode_core_ldo_option_t
                                                    *option)
```

Configs CORE LDO Regulator in low power mode.

This function configs CORE LDO Regulator in Low Power mode. If CORE LDO VDD Drive Strength is set to Normal, the CORE LDO VDD regulator voltage level in Active mode must be equal to the voltage level in Low power mode. And the Bandgap must be programmed to select bandgap enabled. Core VDD voltage levels for the Core LDO low power regulator can only be changed when the CORE LDO Drive Strength set as Normal.

#### Parameters

- `base` – SPC peripheral base address.
- `option` – Pointer to the `spc_lowpower_mode_core_ldo_option_t` structure.

#### Return values

- `kStatus_Success` – Config Core LDO regulator in power mode successfully.
- `kStatus_SPC_Busy` – The SPC instance is busy to execute any type of power mode transition.
- `kStatus_SPC_CORELDOLowDriveStrengthIgnore` – Set driver strength to low will be ignored.
- `#kStatus_SPC_CORELDIVoltageSetFail` – Fail to change Core LDO voltage level.

```
status_t SPC_SetLowPowerModeCoreLDORegulatorVoltageLevel(SPC_Type *base,
                                                           spc_core_ldo_voltage_level_t
                                                           voltageLevel)
```

Set Core LDO VDD Regulator Voltage level in Low power mode.

---

**Note:** If CORE LDO's drive strength is set to Normal, the CORE LDO VDD regulator voltage in active mode and low power mode must be same.

---

---

**Note:** Voltage level for the CORE LDO in low power mode can only be changed when the CORE LDO Drive Strength set as Normal.

---

### Parameters

- base – SPC peripheral base address.
- voltageLevel – Voltage level of CORE LDO Regulator in Low power mode, please refer to `spc_core_ldo_voltage_level_t`.

### Return values

- `kStatus_SPC_CORELDOVoltageWrong` – Voltage level in active mode and low power mode is not same.
- `kStatus_Success` – Set Core LDO regulator voltage level in Low power mode successful.
- `kStatus_SPC_CORELDOVoltageSetFail` – Fail to update voltage level because drive strength is incorrect.

```
static inline spc_core_ldo_voltage_level_t SPC_GetLowPowerCoreLDOVDDVoltageLevel(SPC_Type  
                                                                                   *base)
```

Gets the CORE LDO VDD Regulator Voltage Level for Low Power modes.

### Parameters

- base – SPC peripheral base address.

### Returns

The CORE LDO VDD Regulator's voltage level.

```
status_t SPC_SetLowPowerModeCoreLDORegulatorDriveStrength(SPC_Type *base,  
                                                           spc_core_ldo_drive_strength_t  
                                                           driveStrength)
```

Set Core LDO VDD Regulator Drive Strength in Low power mode.

### Parameters

- base – SPC peripheral base address.
- driveStrength – Specify drive strength of CORE LDO in low power mode.

### Return values

- `kStatus_SPC_CORELDOLowDriveStrengthIgnore` – Some voltage detect enabled, CORE LDO's drive strength can not set as low.
- `kStatus_Success` – Set Core LDO regulator drive strength in Low power mode successful.
- `kStatus_SPC_BandgapModeWrong` – Bandgap is disabled when attempt to set CORE LDO work as normal drive strength.

```
static inline spc_core_ldo_drive_strength_t SPC_GetLowPowerCoreLDOVDDDriveStrength(SPC_Type  
                                                                                   *base)
```

Gets CORE LDO VDD Drive Strength for Low Power modes.

### Parameters

- base – SPC peripheral base address.

### Returns

The CORE LDO's VDD Drive Strength.

---

```
static inline void SPC_EnableSystemLDORegulator(SPC_Type *base, bool enable)
```

Enable/Disable System LDO regulator.

---

**Note:** The SYSTEM LDO enable bit is write-once.

---

#### Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable System LDO Regulator. true - Enable System LDO Regulator. false - Disable System LDO Regulator.

```
static inline void SPC_EnableSystemLDOSinkFeature(SPC_Type *base, bool sink)
```

Enable/Disable current sink feature of System LDO Regulator.

#### Parameters

- base – SPC peripheral base address.
- sink – Enable/Disable current sink feature. true - Enable current sink feature of System LDO Regulator. false - Disable current sink feature of System LDO Regulator.

```
status_t SPC_SetActiveModeSystemLDORegulatorConfig(SPC_Type *base, const
                                                    spc_active_mode_sys_ldo_option_t
                                                    *option)
```

Configs System LDO VDD Regulator in Active mode.

---

**Note:** If System LDO VDD Drive Strength is set to Normal, the Bandgap mode in Active mode must be programmed to a value that enables the bandgap.

---



---

**Note:** If any voltage detects are kept enabled, configuration to set System LDO VDD drive strength to low will be ignored.

---



---

**Note:** If select System LDO VDD Regulator voltage level to Over Drive Voltage, the Drive Strength of System LDO VDD Regulator must be set to Normal otherwise the regulator Drive Strength will be forced to Normal.

---



---

**Note:** If select System LDO VDD Regulator voltage level to Over Drive Voltage, the High voltage detect must be disabled. Otherwise it will be fail to regulator to Over Drive Voltage.

---

#### Parameters

- base – SPC peripheral base address.
- option – Pointer to the `spc_active_mode_sys_ldo_option_t` structure.

#### Return values

- `kStatus_Success` – Config System LDO regulator in Active power mode successful.
- `kStatus_SPC_Busy` – The SPC instance is busy to execute any type of power mode transition.



- `kStatus_SPC_BandgapModeWrong` – The bandgap is not enabled before invoking this function.
- `kStatus_SPC_SYSLDOOverDriveVoltageFail` – HVD of System VDD is not disable before setting to Over Drive voltage.
- `kStatus_SPC_SYSLDOLowDriveStrengthIgnore` – Set System LDO VDD regulator's driver strength to Low will be ignored.

```
status_t SPC_SetActiveModeSystemLDORegulatorVoltageLevel(SPC_Type *base,  
                                                         spc_sys_ldo_voltage_level_t  
                                                         voltageLevel)
```

Set System LDO Regulator voltage level in Active mode.

---

**Note:** The system LDO regulator can only operate at the overdrive voltage level for a limited amount of time for the life of chip.

---

#### Parameters

- `base` – SPC peripheral base address.
- `voltageLevel` – Specify the voltage level of System LDO Regulator in Active mode.

#### Return values

- `kStatus_Success` – Set System LDO Regulator voltage level in Active mode successfully.
- `kStatus_SPC_SYSLDOOverDriveVoltageFail` – Must disable system LDO high voltage detector before specifying overdrive voltage.

```
static inline spc_sys_ldo_voltage_level_t SPC_GetActiveModeSystemLDORegulatorVoltageLevel(SPC_Type  
                                                                                       *base)
```

Get System LDO Regulator voltage level in Active mode.

#### Parameters

- `base` – SPC peripheral base address.

#### Returns

System LDO Regulator voltage level in Active mode, please refer to `spc_sys_ldo_voltage_level_t`.

```
status_t SPC_SetActiveModeSystemLDORegulatorDriveStrength(SPC_Type *base,  
                                                         spc_sys_ldo_drive_strength_t  
                                                         driveStrength)
```

Set System LDO Regulator Drive Strength in Active mode.

#### Parameters

- `base` – SPC peripheral base address.
- `driveStrength` – Specify the drive strength of System LDO Regulator in Active mode.

#### Return values

- `kStatus_Success` – Set System LDO Regulator drive strength in Active mode successfully.
- `kStatus_SPC_SYSLDOLowDriveStrengthIgnore` – Attempt to specify low drive strength is ignored due to any voltage detect feature is enabled in active mode.

- `kStatus_SPC_BandgapModeWrong` – Bandgap mode in Active mode must be programmed to a value that enables the bandgap if attempt to specify normal drive strength.

static inline `spc_sys_ldo_drive_strength_t` SPC\_GetActiveModeSystemLDORegulatorDriveStrength(`SPC_Type *base`)

Get System LDO Regulator Drive Strength in Active mode.

#### Parameters

- `base` – SPC peripheral base address.

#### Returns

System LDO regulator drive strength in Active mode, please refer to `spc_sys_ldo_drive_strength_t`.

`status_t` SPC\_SetLowPowerModeSystemLDORegulatorConfig(`SPC_Type *base`, const `spc_lowpower_mode_sys_ldo_option_t *option`)

Configs System LDO regulator in low power modes.

This function configs System LDO regulator in low power modes. If System LDO VDD Regulator Drive strength is set to normal, bandgap mode in low power mode must be programmed to a value that enables the Bandgap. If any High voltage detectors or Low Voltage detectors are kept enabled, configuration to set System LDO Regulator drive strength as Low will be ignored.

#### Parameters

- `base` – SPC peripheral base address.
- `option` – Pointer to `spc_lowpower_mode_sys_ldo_option_t` structure.

#### Return values

- `kStatus_Success` – Config System LDO regulator in Low Power Mode successfully.
- `kStatus_SPC_Busy` – The SPC instance is busy to execute any type of power mode transition.
- `kStatus_SPC_SYSLDOLowDriveStrengthIgnore` – Set driver strength to low will be ignored.

`status_t` SPC\_SetLowPowerModeSystemLDORegulatorDriveStrength(`SPC_Type *base`, `spc_sys_ldo_drive_strength_t driveStrength`)

Set System LDO Regulator drive strength in Low Power Mode.

#### Parameters

- `base` – SPC peripheral base address.
- `driveStrength` – Specify the drive strength of System LDO Regulator in Low Power Mode.

#### Return values

- `kStatus_Success` – Set System LDO Regulator drive strength in Low Power Mode successfully.
- `kStatus_SPC_SYSLDOLowDriveStrengthIgnore` – Attempt to specify low drive strength is ignored due to any voltage detect feature is enabled in low power mode.
- `kStatus_SPC_BandgapModeWrong` – Bandgap mode in low power mode must be programmed to a value that enables the bandgap if attempt to specify normal drive strength.

```
static inline spc_sys_ldo_drive_strength_t SPC_GetLowPowerModeSystemLDORegulatorDriveStrength(SPC_Type  
*base)
```

Get System LDO Regulator drive strength in Low Power Mode.

#### Parameters

- base – SPC peripheral base address.

#### Returns

System LDO regulator drive strength in Low Power Mode, please refer to *spc\_sys\_ldo\_drive\_strength\_t*.

```
static inline void SPC_EnableDCDCRegulator(SPC_Type *base, bool enable)  
Enable/Disable DCDC Regulator.
```

---

**Note:** The DCDC enable bit is write-once, settings only reset after a POR, LVD, or HVD event.

---

#### Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable DCDC Regulator. true - Enable DCDC Regulator.  
false - Disable DCDC Regulator.

```
void SPC_SetDCDCBurstConfig(SPC_Type *base, spc_dcdc_burst_config_t *config)  
Config DCDC Burst options.
```

#### Parameters

- base – SPC peripheral base address.
- config – Pointer to *spc\_dcdc\_burst\_config\_t* structure.

```
static inline void SPC_TriggerDCDCBurstRequest(SPC_Type *base)  
Trigger a software burst request to DCDC.
```

#### Parameters

- base – SPC peripheral base address.

```
static inline bool SPC_CheckDCDCBurstAck(SPC_Type *base)  
Check if burst acknowledge flag is asserted.
```

#### Parameters

- base – SPC peripheral base address.

#### Return values

- false – DCDC burst not complete.
- true – DCDC burst complete.

```
static inline void SPC_ClearDCDCBurstAckFlag(SPC_Type *base)  
Clear DCDC burst acknowledge flag.
```

#### Parameters

- base – SPC peripheral base address.

```
void SPC_SetDCDCRefreshCount(SPC_Type *base, uint16_t count)  
Set the count value of the reference clock to configure the period of DCDC not active.
```

---

**Note:** This function is only useful when DCDC's drive strength is set as pulse refresh.

---



- base – SPC peripheral base address.

**Returns**

DCDC\_CORE Regulator voltage level, please refer to `spc_dcdc_voltage_level_t`.

*status\_t* SPC\_SetActiveModeDCDCRegulatorDriveStrength(SPC\_Type \*base,  
                                                          *spc\_dcdc\_drive\_strength\_t*  
                                                          driveStrength)

Set DCDC\_CORE Regulator drive strength in Active mode.

---

**Note:** To set DCDC drive strength as Normal, the bandgap must be enabled.

---

**Parameters**

- base – SPC peripheral base address.
- driveStrength – Specify the DCDC\_CORE regulator drive strength, please refer to `spc_dcdc_drive_strength_t`.

**Return values**

- `kStatus_Success` – Set DCDC\_CORE Regulator drive strength in Active mode successfully.
- `kStatus_SPC_BandgapModeWrong` – Set DCDC\_CORE Regulator drive strength to Normal, the Bandgap must be enabled.

static inline *spc\_dcdc\_drive\_strength\_t* SPC\_GetActiveModeDCDCRegulatorDriveStrength(SPC\_Type  
                                                                                                  \*base)

Get DCDC\_CORE Regulator drive strength in Active mode.

**Parameters**

- base – SPC peripheral base address.

**Returns**

DCDC\_CORE Regulator drive strength, please refer to  
`spc_dcdc_drive_strength_t`.

*status\_t* SPC\_SetLowPowerModeDCDCRegulatorConfig(SPC\_Type \*base, const  
                                                                                                  *spc\_lowpower\_mode\_dcdc\_option\_t*  
                                                                                                  \*option)

Configs DCDC\_CORE Regulator in Low power modes.

---

**Note:** If DCDC\_CORE Drive Strength is set to Normal, the Bandgap mode in Low Power mode must be programmed to a value that enables the Bandgap.

---

---

**Note:** In Deep Power Down mode, DCDC regulator is always turned off.

---

**Parameters**

- base – SPC peripheral base address.
- option – Pointer to the `spc_lowpower_mode_dcdc_option_t` structure.

**Return values**

- `kStatus_Success` – Config DCDC regulator in low power mode successfully.
- `kStatus_SPC_Busy` – The SPC instance is busy to execute any type of power mode transition.

- `kStatus_SPC_BandgapModeWrong` – The bandgap mode setting in Low Power mode is wrong.

```
status_t SPC_SetLowPowerModeDCDCRegulatorDriveStrength(SPC_Type *base,  
                                                         spc_dcdc_drive_strength_t  
                                                         driveStrength)
```

Set DCDC\_CORE Regulator drive strength in Low power mode.

---

**Note:** To set drive strength as normal, the bandgap must be enabled.

---

#### Parameters

- `base` – SPC peripheral base address.
- `driveStrength` – Specify the DCDC\_CORE Regulator drive strength, please refer to `spc_dcdc_drive_strength_t`.

#### Return values

- `kStatus_Success` – Set DCDC\_CORE Regulator drive strength in Low power mode successfully.
- `kStatus_SPC_BandgapModeWrong` – Set DCDC\_CORE Regulator drive strength to Normal, the Bandgap must be enabled.

```
static inline spc_dcdc_drive_strength_t SPC_GetLowPowerModeDCDCRegulatorDriveStrength(SPC_Type  
                                                                                       *base)
```

Get DCDC\_CORE Regulator drive strength in Low power mode.

#### Parameters

- `base` – SPC peripheral base address.

#### Returns

DCDC\_CORE Regulator drive strength, please refer to `spc_dcdc_drive_strength_t`.

```
static inline void SPC_SetLowPowerModeDCDCRegulatorVoltageLevel(SPC_Type *base,  
                                                                spc_dcdc_voltage_level_t  
                                                                voltageLevel)
```

Set DCDC\_CORE Regulator voltage level in Low power mode.

- a. Configure `ACTIVE_CFG[DCDC_VDD_LVL]` to same level programmed in #1.

---

**Note:** To change DCDC level in Low-Power mode:

- a. Configure `LP_CFG[DCDC_VDD_LVL]` to desired level;
  - b. Configure `LP_CFG[DCDC_VDD_DS]` to low driver strength;
- 

---

**Note:** After invoking this function, the voltage level in active mode(wakeup from low power modes) also changed, if it is necessary, please invoke `SPC_SetActiveModeDCDCRegulatorVoltageLevel()` to change to desired voltage level.

---

#### Parameters

- `base` – SPC peripheral base address.

- voltageLevel – Specify the DCDC\_CORE Regulator voltage level, please refer to spc\_dcdc\_voltage\_level\_t.

static inline spc\_dcdc\_voltage\_level\_t SPC\_GetLowPowerModeDCDCRegulatorVoltageLevel(SPC\_Type \*base)

Get DCDC\_CORE Regulator voltage level in Low power mode.

#### Parameters

- base – SPC peripheral base address.

#### Returns

DCDC\_CORE Regulator voltage level, please refer to spc\_dcdc\_voltage\_level\_t.

FSL\_SPC\_DRIVER\_VERSION

SPC driver version 2.10.0.

SPC status enumeration.

---

**Note:** Some device(such as MCXA family) do not equip DCDC or System LDO, please refer to the reference manual to check.

---

#### Values:

enumerator kStatus\_SPC\_Busy

The SPC instance is busy executing any type of power mode transition.

enumerator kStatus\_SPC\_DCDCLowDriveStrengthIgnore

DCDC Low drive strength setting be ignored for LVD/HVD enabled.

enumerator kStatus\_SPC\_DCDCPulseRefreshModeIgnore

DCDC Pulse Refresh Mode drive strength setting be ignored for LVD/HVD enabled.

enumerator kStatus\_SPC\_SYSLDOOverDriveVoltageFail

SYS LDO regulate to Over drive voltage failed for SYS LDO HVD must be disabled.

enumerator kStatus\_SPC\_SYSLDOLowDriveStrengthIgnore

SYS LDO Low driver strength setting be ignored for LDO LVD/HVD enabled.

enumerator kStatus\_SPC\_CORELDOLowDriveStrengthIgnore

CORE LDO Low driver strength setting be ignored for LDO LVD/HVD enabled.

enumerator kStatus\_SPC\_BandgapModeWrong

Selected Bandgap Mode wrong.

enumerator kStatus\_SPC\_CORELDOVoltageWrong

Core LDO voltage is wrong.

enumerator kStatus\_SPC\_CORELDOVoltageSetFail

Core LDO voltage set fail.

enumerator kStatus\_SPC\_CORELDOVoltageDetectWrong

Settings of CORE\_LDO voltage detection is not allowed.

enumerator kStatus\_SPC\_DCDCCoreLdoVoltageMisMatch

Target voltage level of DCDC not equal to CORE\_LDO.

enum \_spc\_voltage\_detect\_flags

Voltage Detect Status Flags.

#### Values:

enumerator kSPC\_IOVDDHighVoltageDetectFlag  
IO VDD High-Voltage detect flag.

enumerator kSPC\_IOVDDLowVoltageDetectFlag  
IO VDD Low-Voltage detect flag.

enumerator kSPC\_SystemVDDHighVoltageDetectFlag  
System VDD High-Voltage detect flag.

enumerator kSPC\_SystemVDDLowVoltageDetectFlag  
System VDD Low-Voltage detect flag.

enumerator kSPC\_CoreVDDHighVoltageDetectFlag  
Core VDD High-Voltage detect flag.

enumerator kSPC\_CoreVDDLowVoltageDetectFlag  
Core VDD Low-Voltage detect flag.

enum \_spc\_power\_domains  
SPC power domain isolation status.

---

**Note:** Some devices(such as MCXA family) do not contain WAKE Power Domain, please refer to the reference manual to check.

---

*Values:*

enumerator kSPC\_MAINPowerDomainRetain  
Peripherals and IO pads retain in MAIN Power Domain.

enumerator kSPC\_WAKEPowerDomainRetain  
Peripherals and IO pads retain in WAKE Power Domain.

enum \_spc\_analog\_module\_control  
The enumeration of all analog module that can be controlled by SPC in active or low-power modes.

---

**Note:** Enumerations may not suitable for all devices, please check the specific device's RM for supported analog modules.

---

*Values:*

enumerator kSPC\_controlVref  
Enable/disable VREF in active or low-power modes.

enumerator kSPC\_controlUsb3vDet  
Enable/disable USB3V\_Det in active or low-power modes.

enumerator kSPC\_controlDac0  
Enable/disable DAC0 in active or low-power modes.

enumerator kSPC\_controlDac1  
Enable/disable DAC1 in active or low-power modes.

enumerator kSPC\_controlDac2  
Enable/disable DAC2 in active or low-power modes.

enumerator kSPC\_controlOpamp0  
Enable/disable OPAMP0 in active or low-power modes.



enumerator kSPC\_controlOpamp1

Enable/disable OPAMP1 in active or low-power modes.

enumerator kSPC\_controlOpamp2

Enable/disable OPAMP2 in active or low-power modes.

enumerator kSPC\_controlOpamp3

Enable/disable OPAMP3 in active or low-power modes.

enumerator kSPC\_controlCmp0

Enable/disable CMP0 in active or low-power modes.

enumerator kSPC\_controlCmp1

Enable/disable CMP1 in active or low-power modes.

enumerator kSPC\_controlCmp2

Enable/disable CMP2 in active or low-power modes.

enumerator kSPC\_controlCmp0Dac

Enable/disable CMP0\_DAC in active or low-power modes.

enumerator kSPC\_controlCmp1Dac

Enable/disable CMP1\_DAC in active or low-power modes.

enumerator kSPC\_controlCmp2Dac

Enable/disable CMP2\_DAC in active or low-power modes.

enumerator kSPC\_controlAllModules

Enable/disable all modules in active or low-power modes.

enum \_spc\_power\_domain\_id

The enumeration of spc power domain, the connected power domain is chip specific, please refer to chip's RM for details.

*Values:*

enumerator kSPC\_PowerDomain0

Power domain0, the connected power domain is chip specific.

enumerator kSPC\_PowerDomain1

Power domain1, the connected power domain is chip specific.

enum \_spc\_power\_domain\_low\_power\_mode

The enumeration of Power domain's low power mode.

*Values:*

enumerator kSPC\_SleepWithSYSClockRunning

Power domain request SLEEP mode with SYS clock running.

enumerator kSPC\_DeepSleepWithSysClockOff

Power domain request deep sleep mode with system clock off.

enumerator kSPC\_PowerDownWithSysClockOff

Power domain request power down mode with system clock off.

enumerator kSPC\_DeepPowerDownWithSysClockOff

Power domain request deep power down mode with system clock off.

enum \_spc\_lowPower\_request\_pin\_polarity

SPC low power request output pin polarity.

*Values:*

enumerator kSPC\_HighTruePolarity

Control the High Polarity of the Low Power Request Pin.

enumerator kSPC\_LowTruePolarity

Control the Low Polarity of the Low Power Request Pin.

enum \_spc\_lowPower\_request\_output\_override

SPC low power request output override.

*Values:*

enumerator kSPC\_LowPowerRequestNotForced

Not Forced.

enumerator kSPC\_LowPowerRequestReserved

Reserved.

enumerator kSPC\_LowPowerRequestForcedLow

Forced Low (Ignore LowPower request output polarity setting.)

enumerator kSPC\_LowPowerRequestForcedHigh

Forced High (Ignore LowPower request output polarity setting.)

enum \_spc\_bandgap\_mode

SPC Bandgap mode enumeration in Active mode or Low Power mode.

*Values:*

enumerator kSPC\_BandgapDisabled

Bandgap disabled.

enumerator kSPC\_BandgapEnabledBufferDisabled

Bandgap enabled with Buffer disabled.

enumerator kSPC\_BandgapEnabledBufferEnabled

Bandgap enabled with Buffer enabled.

enumerator kSPC\_BandgapReserved

Reserved.

enum \_spc\_dcdc\_voltage\_level

DCDC regulator voltage level enumeration in Active mode or Low Power Mode.

---

**Note:** kSPC\_DCDC\_RetentionVoltage not supported for all power modes.

---

*Values:*

enumerator kSPC\_DCDC\_RetentionVoltage

DCDC\_CORE Regulator regulate to retention Voltage(Only supported in low power modes)

enumerator kSPC\_DCDC\_MidVoltage

DCDC\_CORE Regulator regulate to Mid Voltage(1.0V).

enumerator kSPC\_DCDC\_NormalVoltage

DCDC\_CORE Regulator regulate to Normal Voltage(1.1V).

enumerator kSPC\_DCDC\_OverdriveVoltage

DCDC\_CORE Regulator regulate to Safe-Mode Voltage(1.2V).

enum \_spc\_dcdc\_drive\_strength

DCDC regulator Drive Strength enumeration in Active mode or Low Power Mode.

---

**Note:** Different drive strength differ in these DCDC characteristics: Maximum load current  
Quiescent current Transient response.

---

*Values:*

enumerator kSPC\_DCDC\_PulseRefreshMode

DCDC\_CORE Regulator Drive Strength set to Pulse Refresh Mode, This enum member is only useful for Low Power Mode config, please note that pluse refresh mode is invalid in SLEEP mode.

enumerator kSPC\_DCDC\_LowDriveStrength

DCDC\_CORE regulator Drive Strength set to low.

enumerator kSPC\_DCDC\_NormalDriveStrength

DCDC\_CORE regulator Drive Strength set to Normal.

enum \_spc\_sys\_ldo\_voltage\_level

SYS LDO regulator voltage level enumeration in Active mode.

*Values:*

enumerator kSPC\_SysLDO\_NormalVoltage

SYS LDO VDD Regulator regulate to Normal Voltage(1.8V).

enumerator kSPC\_SysLDO\_OverDriveVoltage

SYS LDO VDD Regulator regulate to Over Drive Voltage(2.5V).

enum \_spc\_sys\_ldo\_drive\_strength

SYS LDO regulator Drive Strength enumeration in Active mode or Low Power mode.

*Values:*

enumerator kSPC\_SysLDO\_LowDriveStrength

SYS LDO VDD regulator Drive Strength set to low.

enumerator kSPC\_SysLDO\_NormalDriveStrength

SYS LDO VDD regulator Drive Strength set to Normal.

enum \_spc\_core\_ldo\_voltage\_level

Core LDO regulator voltage level enumeration in Active mode or Low Power mode.

*Values:*

enumerator kSPC\_CoreLDO\_UnderDriveVoltage

*Deprecated:*

, to align with description of latest RM, please use  
kSPC\_CoreLDO\_RetentionVoltage as instead.

enumerator kSPC\_CoreLDO\_RetentionVoltage

Core LDO VDD regulator regulate to retention voltage, please note that only useful in low power modes and not all devices support this options please refer to devices' RM for details.

enumerator kSPC\_CoreLDO\_MidDriveVoltage

Core LDO VDD regulator regulate to Mid Drive Voltage.

enumerator kSPC\_CoreLDO\_NormalVoltage

Core LDO VDD regulator regulate to Normal Voltage.

enumerator kSPC\_CoreLDO\_OverDriveVoltage  
Core LDO VDD regulator regulate to overdrive Voltage.

enum \_spc\_core\_ldo\_drive\_strength  
CORE LDO VDD regulator Drive Strength enumeration in Low Power mode.

*Values:*

enumerator kSPC\_CoreLDO\_LowDriveStrength  
Core LDO VDD regulator Drive Strength set to low.

enumerator kSPC\_CoreLDO\_NormalDriveStrength  
Core LDO VDD regulator Drive Strength set to Normal.

enum \_spc\_low\_voltage\_level\_select  
IO VDD Low-Voltage Level Select.

*Values:*

enumerator kSPC\_LowVoltageNormalLevel

*Deprecated:*

, please use kSPC\_LowVoltageHighRange as instead.

enumerator kSPC\_LowVoltageSafeLevel

*Deprecated:*

, please use kSPC\_LowVoltageLowRange as instead.

enumerator kSPC\_LowVoltageHighRange  
High range LVD threshold.

enumerator kSPC\_LowVoltageLowRange  
Low range LVD threshold.

enum \_spc\_sram\_operate\_voltage  
The list of the operating voltage for the SRAM's read/write timing margin.

*Values:*

enumerator kSPC\_sramOperateAt1P0V  
SRAM configured for 1.0V operation.

enumerator kSPC\_sramOperateAt1P1V  
SRAM configured for 1.1V operation.

enumerator kSPC\_sramOperateAt1P2V  
SRAM configured for 1.2V operation.

typedef enum \_spc\_power\_domain\_id spc\_power\_domain\_id\_t  
The enumeration of spc power domain, the connected power domain is chip specific, please refer to chip's RM for details.

typedef enum \_spc\_power\_domain\_low\_power\_mode spc\_power\_domain\_low\_power\_mode\_t  
The enumeration of Power domain's low power mode.

typedef enum \_spc\_lowPower\_request\_pin\_polarity spc\_lowpower\_request\_pin\_polarity\_t  
SPC low power request output pin polarity.

typedef enum \_spc\_lowPower\_request\_output\_override spc\_lowpower\_request\_output\_override\_t  
SPC low power request output override.

typedef enum \_spc\_bandgap\_mode spc\_bandgap\_mode\_t  
SPC Bandgap mode enumeration in Active mode or Low Power mode.

typedef enum *\_spc\_dcdc\_voltage\_level* spc\_dcdc\_voltage\_level\_t

DCDC regulator voltage level enumeration in Active mode or Low Power Mode.

---

**Note:** kSPC\_DCDC\_RetentionVoltage not supported for all power modes.

---

typedef enum *\_spc\_dcdc\_drive\_strength* spc\_dcdc\_drive\_strength\_t

DCDC regulator Drive Strength enumeration in Active mode or Low Power Mode.

---

**Note:** Different drive strength differ in these DCDC characteristics: Maximum load current  
Quiescent current Transient response.

---

typedef enum *\_spc\_sys\_ldo\_voltage\_level* spc\_sys\_ldo\_voltage\_level\_t

SYS LDO regulator voltage level enumeration in Active mode.

typedef enum *\_spc\_sys\_ldo\_drive\_strength* spc\_sys\_ldo\_drive\_strength\_t

SYS LDO regulator Drive Strength enumeration in Active mode or Low Power mode.

typedef enum *\_spc\_core\_ldo\_voltage\_level* spc\_core\_ldo\_voltage\_level\_t

Core LDO regulator voltage level enumeration in Active mode or Low Power mode.

typedef enum *\_spc\_core\_ldo\_drive\_strength* spc\_core\_ldo\_drive\_strength\_t

CORE LDO VDD regulator Drive Strength enumeration in Low Power mode.

typedef enum *\_spc\_low\_voltage\_level\_select* spc\_low\_voltage\_level\_select\_t

IO VDD Low-Voltage Level Select.

typedef enum *\_spc\_sram\_operate\_voltage* spc\_sram\_operate\_voltage\_t

The list of the operating voltage for the SRAM's read/write timing margin.

typedef struct *\_spc\_sram\_voltage\_config* spc\_sram\_voltage\_config\_t

typedef struct *\_spc\_lowpower\_request\_config* spc\_lowpower\_request\_config\_t

Low Power Request output pin configuration.

typedef struct *\_spc\_active\_mode\_core\_ldo\_option* spc\_active\_mode\_core\_ldo\_option\_t

Core LDO regulator options in Active mode.

typedef struct *\_spc\_active\_mode\_sys\_ldo\_option* spc\_active\_mode\_sys\_ldo\_option\_t

System LDO regulator options in Active mode.

typedef struct *\_spc\_active\_mode\_dcdc\_option* spc\_active\_mode\_dcdc\_option\_t

DCDC regulator options in Active mode.

typedef struct *\_spc\_lowpower\_mode\_core\_ldo\_option* spc\_lowpower\_mode\_core\_ldo\_option\_t

Core LDO regulator options in Low Power mode.

typedef struct *\_spc\_lowpower\_mode\_sys\_ldo\_option* spc\_lowpower\_mode\_sys\_ldo\_option\_t

System LDO regulator options in Low Power mode.

typedef struct *\_spc\_lowpower\_mode\_dcdc\_option* spc\_lowpower\_mode\_dcdc\_option\_t

DCDC regulator options in Low Power mode.

typedef struct *\_spc\_dcdc\_burst\_config* spc\_dcdc\_burst\_config\_t

DCDC Burst configuration.

*Deprecated:*

Do not recommend to use this structure.

```
typedef struct _spc_voltage_detect_option spc_voltage_detect_option_t
    CORE/SYS/IO VDD Voltage Detect options.

typedef struct _spc_core_voltage_detect_config spc_core_voltage_detect_config_t
    Core Voltage Detect configuration.

typedef struct _spc_system_voltage_detect_config spc_system_voltage_detect_config_t
    System Voltage Detect Configuration.

typedef struct _spc_io_voltage_detect_config spc_io_voltage_detect_config_t
    IO Voltage Detect Configuration.

typedef struct _spc_active_mode_regulators_config spc_active_mode_regulators_config_t
    Active mode configuration.

typedef struct _spc_lowpower_mode_regulators_config spc_lowpower_mode_regulators_config_t
    Low Power Mode configuration.

SPC_EVD_CFG_REG_EVDISO_SHIFT
SPC_EVD_CFG_REG_EVDLPISO_SHIFT
SPC_EVD_CFG_REG_EVDSTAT_SHIFT
SPC_EVD_CFG_REG_EVDISO(x)
SPC_EVD_CFG_REG_EVDLPISO(x)
SPC_EVD_CFG_REG_EVDSTAT(x)

struct _spc_sram_voltage_config
    #include <fsl_spc.h>
```

### Public Members

```
spc_sram_operate_voltage_t operateVoltage
    Specifies the operating voltage for the SRAM's read/write timing margin.

bool requestVoltageUpdate
    Used to control whether request an SRAM trim value change.

struct _spc_lowpower_request_config
    #include <fsl_spc.h> Low Power Request output pin configuration.
```

### Public Members

```
bool enable
    Low Power Request Output enable.

spc_lowpower_request_pin_polarity_t polarity
    Low Power Request Output pin polarity select.

spc_lowpower_request_output_override_t override
    Low Power Request Output Override.

struct _spc_active_mode_core_ldo_option
    #include <fsl_spc.h> Core LDO regulator options in Active mode.
```

### Public Members

*spc\_core\_ldo\_voltage\_level\_t* CoreLDOVoltage

Core LDO Regulator Voltage Level selection in Active mode.

*spc\_core\_ldo\_drive\_strength\_t* CoreLDODriveStrength

Core LDO Regulator Drive Strength selection in Active mode

struct *\_spc\_active\_mode\_sys\_ldo\_option*

*#include <fsl\_spc.h>* System LDO regulator options in Active mode.

### Public Members

*spc\_sys\_ldo\_voltage\_level\_t* SysLDOVoltage

System LDO Regulator Voltage Level selection in Active mode.

*spc\_sys\_ldo\_drive\_strength\_t* SysLDODriveStrength

System LDO Regulator Drive Strength selection in Active mode.

struct *\_spc\_active\_mode\_dcdc\_option*

*#include <fsl\_spc.h>* DCDC regulator options in Active mode.

### Public Members

*spc\_dcdc\_voltage\_level\_t* DCDCVoltage

DCDC Regulator Voltage Level selection in Active mode.

*spc\_dcdc\_drive\_strength\_t* DCDCDriveStrength

DCDC\_CORE Regulator Drive Strength selection in Active mode.

struct *\_spc\_lowpower\_mode\_core\_ldo\_option*

*#include <fsl\_spc.h>* Core LDO regulator options in Low Power mode.

### Public Members

*spc\_core\_ldo\_voltage\_level\_t* CoreLDOVoltage

Core LDO Regulator Voltage Level selection in Low Power mode.

*spc\_core\_ldo\_drive\_strength\_t* CoreLDODriveStrength

Core LDO Regulator Drive Strength selection in Low Power mode

struct *\_spc\_lowpower\_mode\_sys\_ldo\_option*

*#include <fsl\_spc.h>* System LDO regulator options in Low Power mode.

### Public Members

*spc\_sys\_ldo\_drive\_strength\_t* SysLDODriveStrength

System LDO Regulator Drive Strength selection in Low Power mode.

struct *\_spc\_lowpower\_mode\_dcdc\_option*

*#include <fsl\_spc.h>* DCDC regulator options in Low Power mode.

### Public Members

*spc\_dcdc\_voltage\_level\_t* DCDCVoltage

DCDC Regulator Voltage Level selection in Low Power mode.

*spc\_dcdc\_drive\_strength\_t* DCDCDriveStrength

DCDC\_CORE Regulator Drive Strength selection in Low Power mode.

struct *\_spc\_dcdc\_burst\_config*

*#include <fsl\_spc.h>* DCDC Burst configuration.

### Deprecated:

Do not recommend to use this structure.

### Public Members

bool softwareBurstRequest

Enable/Disable DCDC Software Burst Request.

bool externalBurstRequest

Enable/Disable DCDC External Burst Request.

bool stabilizeBurstFreq

Enable/Disable DCDC frequency stabilization.

uint8\_t freq

The frequency of the current burst.

struct *\_spc\_voltage\_detect\_option*

*#include <fsl\_spc.h>* CORE/SYS/IO VDD Voltage Detect options.

### Public Members

bool HVDInterruptEnable

CORE/SYS/IO VDD High Voltage Detect interrupt enable.

bool HVDRestEnable

CORE/SYS/IO VDD High Voltage Detect reset enable.

bool LVDInterruptEnable

CORE/SYS/IO VDD Low Voltage Detect interrupt enable.

bool LVDRestEnable

CORE/SYS/IO VDD Low Voltage Detect reset enable.

struct *\_spc\_core\_voltage\_detect\_config*

*#include <fsl\_spc.h>* Core Voltage Detect configuration.

### Public Members

*spc\_voltage\_detect\_option\_t* option

Core VDD Voltage Detect option.

struct *\_spc\_system\_voltage\_detect\_config*

*#include <fsl\_spc.h>* System Voltage Detect Configuration.



### Public Members

*spc\_voltage\_detect\_option\_t* option  
System VDD Voltage Detect option.

*spc\_low\_voltage\_level\_select\_t* level

*Deprecated:*

, reserved for all devices, will removed in next release.

struct *\_spc\_io\_voltage\_detect\_config*  
*#include <fsl\_spc.h>* IO Voltage Detect Configuration.

### Public Members

*spc\_voltage\_detect\_option\_t* option  
IO VDD Voltage Detect option.

*spc\_low\_voltage\_level\_select\_t* level  
IO VDD Low-voltage level selection.

struct *\_spc\_active\_mode\_regulators\_config*  
*#include <fsl\_spc.h>* Active mode configuration.

### Public Members

*spc\_bandgap\_mode\_t* bandgapMode  
Specify bandgap mode in active mode.

bool *lpBuff*  
Enable/disable CMP bandgap buffer.

*spc\_active\_mode\_dcdc\_option\_t* DCDCOption  
Specify DCDC configurations in active mode.

*spc\_active\_mode\_sys\_ldo\_option\_t* SysLDOOption  
Specify System LDO configurations in active mode.

*spc\_active\_mode\_core\_ldo\_option\_t* CoreLDOOption  
Specify Core LDO configurations in active mode.

struct *\_spc\_lowpower\_mode\_regulators\_config*  
*#include <fsl\_spc.h>* Low Power Mode configuration.

### Public Members

bool *lpIREF*  
Enable/disable low power IREF in low power modes.

*spc\_bandgap\_mode\_t* bandgapMode  
Specify bandgap mode in low power modes.

bool *lpBuff*  
Enable/disable CMP bandgap buffer in low power modes.

bool *CoreIVS*  
Enable/disable CORE VDD internal voltage scaling.

*spc\_lowpower\_mode\_dcdc\_option\_t* DCDCOption

Specify DCDC configurations in low power modes.

*spc\_lowpower\_mode\_sys\_ldo\_option\_t* SysLDOOption

Specify system LDO configurations in low power modes.

*spc\_lowpower\_mode\_core\_ldo\_option\_t* CoreLDOOption

Specify core LDO configurations in low power modes.

## 2.43 MCX\_VBAT: Smart Power Switch

The enumeration of VBAT module status.

*Values:*

enumerator kStatus\_VBAT\_Fro16kNotEnabled

Internal 16kHz free running oscillator not enabled.

enumerator kStatus\_VBAT\_BandgapNotEnabled

Bandgap not enabled.

enumerator kStatus\_VBAT\_WrongCapacitanceValue

Wrong capacitance for selected oscillator mode.

enumerator kStatus\_VBAT\_ClockMonitorLocked

Clock monitor locked.

enumerator kStatus\_VBAT\_OSC32KNotReady

OSC32K not ready.

enumerator kStatus\_VBAT\_LDONotReady

LDO not ready.

enumerator kStatus\_VBAT\_TamperLocked

Tamper locked.

enum \_vbat\_status\_flag

The enumeration of VBAT status flags.

*Values:*

enumerator kVBAT\_StatusFlagPORDetect

VBAT domain has been reset

enumerator kVBAT\_StatusFlagWakeupPin

A falling edge is detected on the wakeup pin.

enumerator kVBAT\_StatusFlagBandgapTimer0

Bandgap Timer0 period reached.

enumerator kVBAT\_StatusFlagBandgapTimer1

Bandgap Timer1 period reached.

enumerator kVBAT\_StatusFlagLdoReady

LDO is enabled and ready.

enumerator kVBAT\_StatusFlagOsc32kReady

OSC32k is enabled and clock is ready.

enumerator kVBAT\_\_StatusFlagInterrupt0Detect  
Interrupt 0 asserted.

enumerator kVBAT\_\_StatusFlagInterrupt1Detect  
Interrupt 1 asserted.

enumerator kVBAT\_\_StatusFlagInterrupt2Detect  
Interrupt 2 asserted.

enumerator kVBAT\_\_StatusFlagInterrupt3Detect  
Interrupt 2 asserted.

enum \_\_vbat\_\_interrupt\_\_enable

The enumeration of VBAT interrupt enable.

*Values:*

enumerator kVBAT\_\_InterruptEnablePORDetect  
Enable POR detect interrupt.

enumerator kVBAT\_\_InterruptEnableWakeupPin  
Enable the interrupt when a falling edge is detected on the wakeup pin.

enumerator kVBAT\_\_InterruptEnableBandgapTimer0  
Enable the interrupt if Bandgap Timer0 period reached.

enumerator kVBAT\_\_InterruptEnableBandgapTimer1  
Enable the interrupt if Bandgap Timer1 period reached.

enumerator kVBAT\_\_InterruptEnableLdoReady  
Enable LDO ready interrupt.

enumerator kVBAT\_\_InterruptEnableOsc32kReady  
Enable OSC32K ready interrupt.

enumerator kVBAT\_\_InterruptEnableInterrupt0  
Enable the interrupt0.

enumerator kVBAT\_\_InterruptEnableInterrupt1  
Enable the interrupt1.

enumerator kVBAT\_\_InterruptEnableInterrupt2  
Enable the interrupt2.

enumerator kVBAT\_\_InterruptEnableInterrupt3  
Enable the interrupt3.

enumerator kVBAT\_\_AllInterruptsEnable  
Enable all interrupts.

enum \_\_vbat\_\_wakeup\_\_enable

The enumeration of VBAT wakeup enable.

*Values:*

enumerator kVBAT\_\_WakeupEnablePORDetect  
Enable POR detect wakeup.

enumerator kVBAT\_\_WakeupEnableWakeupPin  
Enable wakeup feature when a falling edge is detected on the wakeup pin.

enumerator kVBAT\_\_WakeupEnableBandgapTimer0  
Enable wakeup feature when bandgap timer0 period reached.

enumerator kVBAT\_WakeupEnableBandgapTimer1

Enable wakeup feature when bandgap timer1 period reached.

enumerator kVBAT\_WakeupEnableLdoReady

Enable wakeup when LDO ready.

enumerator kVBAT\_WakeupEnableOsc32kReady

Enable wakeup when OSC32k ready.

enumerator kVBAT\_WakeupEnableInterrupt0

Enable wakeup when interrupt0 asserted.

enumerator kVBAT\_WakeupEnableInterrupt1

Enable wakeup when interrupt1 asserted.

enumerator kVBAT\_WakeupEnableInterrupt2

Enable wakeup when interrupt2 asserted.

enumerator kVBAT\_WakeupEnableInterrupt3

Enable wakeup when interrupt3 asserted.

enumerator kVBAT\_AllWakeupEnable

Enable all wakeup.

enum \_vbat\_tamper\_enable

The enumeration of VBAT tamper enable.

*Values:*

enumerator kVBAT\_TamperEnablePOR

Enable tamper if POR asserted in STATUS register.

enumerator kVBAT\_TamperEnableClockDetect

Enable tamper if clock monitor detect an error.

enumerator kVBAT\_TamperEnableConfigDetect

Enable tamper if configuration error detected.

enumerator kVBAT\_TamperEnableVoltageDetect

Enable tamper if voltage monitor detect an error.

enumerator kVBAT\_TamperEnableTemperatureDetect

Enable tamper if temperature monitor detect an error.

enumerator kVBAT\_TamperEnableSec0Detect

Enable tamper if security input 0 detect an error.

enum \_vbat\_bandgap\_timer\_id

The enumeration of bandgap timer id, VBAT support two bandgap timers.

*Values:*

enumerator kVBAT\_BandgapTimer0

Bandgap Timer0.

enumerator kVBAT\_BandgapTimer1

Bandgap Timer1.

enum \_vbat\_clock\_enable

The enumeration of connections for OSC32K/FRO32K output clock to other modules.

*Values:*

enumerator kVBAT\_EnableClockToDomain0

Enable clock to power domain0.

enumerator kVBAT\_EnableClockToDomain1

Enable clock to power domain1.

enumerator kVBAT\_EnableClockToDomain2

Enable clock to power domain2.

enumerator kVBAT\_EnableClockToDomain3

Enable clock to power domain3.

enum \_vbat\_ram\_array

The enumeration of SRAM arrays that controlled by VBAT. .

*Values:*

enumerator kVBAT\_SramArray0

Specify SRAM array0 that controlled by VBAT.

enumerator kVBAT\_SramArray1

Specify SRAM array1 that controlled by VBAT.

enumerator kVBAT\_SramArray2

Specify SRAM array2 that controlled by VBAT.

enumerator kVBAT\_SramArray3

Specify SRAM array3 that controlled by VBAT.

enum \_vbat\_bandgap\_refresh\_period

The enumeration of bandgap refresh period.

*Values:*

enumerator kVBAT\_BandgapRefresh7P8125ms

Bandgap refresh every 7.8125ms.

enumerator kVBAT\_BandgapRefresh15P625ms

Bandgap refresh every 15.625ms.

enumerator kVBAT\_BandgapRefresh31P25ms

Bandgap refresh every 31.25ms.

enumerator kVBAT\_BandgapRefresh62P5ms

Bandgap refresh every 62.5ms.

enum \_vbat\_bandgap\_timer0\_timeout\_period

The enumeration of bandgap timer0 timeout period.

*Values:*

enumerator kVBAT\_BangapTimer0Timeout1s

Bandgap timer0 timerout every 1s.

enumerator kVBAT\_BangapTimer0Timeout500ms

Bandgap timer0 timerout every 500ms.

enumerator kVBAT\_BangapTimer0Timeout250ms

Bandgap timer0 timerout every 250ms.

enumerator kVBAT\_BangapTimer0Timeout125ms

Bandgap timer0 timerout every 125ms.

enumerator kVBAT\_BangapTimer0Timeout62P5ms

Bandgap timer0 timerout every 62.5ms.

enumerator kVBAT\_BangapTimer0Timeout31P25ms

Bandgap timer0 timerout every 31.25ms.

enum \_\_vbat\_osc32k\_operate\_mode

The enumeration of osc32k operate mode, including Bypass mode, low power switched mode and so on.

*Values:*

enumerator kVBAT\_Osc32kEnabledToTransconductanceMode

Set to transconductance mode.

enumerator kVBAT\_Osc32kEnabledToLowPowerBackupMode

Set to low power backup mode.

enumerator kVBAT\_Osc32kEnabledToLowPowerSwitchedMode

Set to low power switched mode.

enum \_\_vbat\_osc32k\_load\_capacitance\_select

The enumeration of OSC32K load capacitance.

*Values:*

enumerator kVBAT\_Osc32kCrystalLoadCap0pF

Internal capacitance bank is enabled, set the internal capacitance to 0 pF.

enumerator kVBAT\_Osc32kCrystalLoadCap2pF

Internal capacitance bank is enabled, set the internal capacitance to 2 pF.

enumerator kVBAT\_Osc32kCrystalLoadCap4pF

Internal capacitance bank is enabled, set the internal capacitance to 4 pF.

enumerator kVBAT\_Osc32kCrystalLoadCap6pF

Internal capacitance bank is enabled, set the internal capacitance to 6 pF.

enumerator kVBAT\_Osc32kCrystalLoadCap8pF

Internal capacitance bank is enabled, set the internal capacitance to 8 pF.

enumerator kVBAT\_Osc32kCrystalLoadCap10pF

Internal capacitance bank is enabled, set the internal capacitance to 10 pF.

enumerator kVBAT\_Osc32kCrystalLoadCap12pF

Internal capacitance bank is enabled, set the internal capacitance to 12 pF.

enumerator kVBAT\_Osc32kCrystalLoadCap14pF

Internal capacitance bank is enabled, set the internal capacitance to 14 pF.

enumerator kVBAT\_Osc32kCrystalLoadCap16pF

Internal capacitance bank is enabled, set the internal capacitance to 16 pF.

enumerator kVBAT\_Osc32kCrystalLoadCap18pF

Internal capacitance bank is enabled, set the internal capacitance to 18 pF.

enumerator kVBAT\_Osc32kCrystalLoadCap20pF

Internal capacitance bank is enabled, set the internal capacitance to 20 pF.

enumerator kVBAT\_Osc32kCrystalLoadCap22pF

Internal capacitance bank is enabled, set the internal capacitance to 22 pF.

enumerator kVBAT\_Osc32kCrystalLoadCap24pF

Internal capacitance bank is enabled, set the internal capacitance to 24 pF.

enumerator kVBAT\_Osc32kCrystalLoadCap26pF

Internal capacitance bank is enabled, set the internal capacitance to 26 pF.

enumerator kVBAT\_Osc32kCrystalLoadCap28pF

Internal capacitance bank is enabled, set the internal capacitance to 28 pF.

enumerator kVBAT\_Osc32kCrystalLoadCap30pF

Internal capacitance bank is enabled, set the internal capacitance to 30 pF.

enumerator kVBAT\_Osc32kCrystalLoadCapBankDisabled

Internal capacitance bank is disabled.

enum \_\_vbat\_osc32k\_start\_up\_time

The enumeration of start-up time of the oscillator.

*Values:*

enumerator kVBAT\_Osc32kStartUpTime8Sec

Configure the start-up time as 8 seconds.

enumerator kVBAT\_Osc32kStartUpTime4Sec

Configure the start-up time as 4 seconds.

enumerator kVBAT\_Osc32kStartUpTime2Sec

Configure the start-up time as 2 seconds.

enumerator kVBAT\_Osc32kStartUpTime1Sec

Configure the start-up time as 1 seconds.

enumerator kVBAT\_Osc32kStartUpTime0P5Sec

Configure the start-up time as 0.5 seconds.

enumerator kVBAT\_Osc32kStartUpTime0P25Sec

Configure the start-up time as 0.25 seconds.

enumerator kVBAT\_Osc32kStartUpTime0P125Sec

Configure the start-up time as 0.125 seconds.

enumerator kVBAT\_Osc32kStartUpTime0P5MSec

Configure the start-up time as 0.5 milliseconds.

enum \_\_vbat\_internal\_module\_supply

The enumeration of VBAT module supplies.

*Values:*

enumerator kVBAT\_ModuleSuppliedByVddBat

VDD\_BAT supplies VBAT modules.

enumerator kVBAT\_ModuleSuppliedByVddSys

VDD\_SYS supplies VBAT modules.

enum \_\_vbat\_clock\_monitor\_divide\_trim

The enumeration of VBAT clock monitor divide trim value.

*Values:*

enumerator kVBAT\_ClockMonitorOperateAt1kHz

Clock monitor operates at 1 kHz.

enumerator kVBAT\_ClockMonitorOperateAt64Hz

Clock monitor operates at 64 Hz.

enum `_vbat_clock_monitor_freq_trim`

The enumeration of VBAT clock monitor frequency trim value used to adjust the clock monitor assert.

*Values:*

enumerator `kVBAT_ClockMonitorAssert2Cycle`

Clock monitor assert 2 cycles after expected edge.

enumerator `kVBAT_ClockMonitorAssert4Cycle`

Clock monitor assert 4 cycles after expected edge.

enumerator `kVBAT_ClockMonitorAssert6Cycle`

Clock monitor assert 8 cycles after expected edge.

enumerator `kVBAT_ClockMonitorAssert8Cycle`

Clock monitor assert 8 cycles after expected edge.

typedef enum `_vbat_bandgap_refresh_period` `vbat_bandgap_refresh_period_t`

The enumeration of bandgap refresh period.

typedef enum `_vbat_bandgap_timer0_timeout_period` `vbat_bandgap_timer0_timeout_period_t`

The enumeration of bandgap timer0 timeout period.

typedef enum `_vbat_osc32k_operate_mode` `vbat_osc32k_operate_mode_t`

The enumeration of osc32k operate mode, including Bypass mode, low power switched mode and so on.

typedef enum `_vbat_osc32k_load_capacitance_select` `vbat_osc32k_load_capacitance_select_t`

The enumeration of OSC32K load capacitance.

typedef enum `_vbat_osc32k_start_up_time` `vbat_osc32k_start_up_time_t`

The enumeration of start-up time of the oscillator.

typedef enum `_vbat_internal_module_supply` `vbat_internal_module_supply_t`

The enumeration of VBAT module supplies.

typedef enum `_vbat_clock_monitor_divide_trim` `vbat_clock_monitor_divide_trim_t`

The enumeration of VBAT clock monitor divide trim value.

typedef enum `_vbat_clock_monitor_freq_trim` `vbat_clock_monitor_freq_trim_t`

The enumeration of VBAT clock monitor frequency trim value used to adjust the clock monitor assert.

typedef struct `_vbat_fro16k_config` `vbat_fro16k_config_t`

The structure of internal 16kHz free running oscillator attributes.

typedef struct `_vbat_clock_monitor_config` `vbat_clock_monitor_config_t`

The structure of internal clock monitor, including divide trim and frequency trim.

typedef struct `_vbat_tamper_config` `vbat_tamper_config_t`

The structure of Tamper configuration.

FSL\_VBAT\_DRIVER\_VERSION

VBAT driver version 2.5.0.

VBAT\_LDORAMC\_RET\_MASK

VBAT\_LDORAMC\_RET\_SHIFT

VBAT\_LDORAMC\_RET(x)

kVBAT\_EnableClockToVddBat



kVBAT\_\_EnableClockToVddSys

kVBAT\_\_EnableClockToVddWake

kVBAT\_\_EnableClockToVddMain

void VBAT\_\_ConfigFRO16k(VBAT\_Type \*base, const *vbat\_fro16k\_config\_t* \*config)

Configure internal 16kHz free running oscillator, including enabel FRO16k, gate FRO16k output.

#### Parameters

- base – VBAT peripheral base address.
- config – Pointer to *vbat\_fro16k\_config\_t* structure.

static inline void VBAT\_\_EnableFRO16k(VBAT\_Type \*base, bool enable)

Enable/disable internal 16kHz free running oscillator.

#### Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable 16kHz FRO.
  - **true** Enable internal 16kHz free running oscillator.
  - **false** Disable internal 16kHz free running oscillator.

static inline bool VBAT\_\_CheckFRO16kEnabled(VBAT\_Type \*base)

Check if internal 16kHz free running oscillator is enabled.

#### Parameters

- base – VBAT peripheral base address.

#### Return values

- true – The internal 16kHz Free running oscillator is enabled.
- false – The internal 16kHz Free running oscillator is disabled.

static inline void VBAT\_\_UngateFRO16k(VBAT\_Type \*base, uint8\_t connectionsMask)

Enable FRO16kHz output clock to selected modules.

#### Parameters

- base – VBAT peripheral base address.
- connectionsMask – The mask of modules that FRO16k is connected, should be the OR'ed value of *vbat\_clock\_enable\_t*.

static inline void VBAT\_\_GateFRO16k(VBAT\_Type \*base, uint8\_t connectionsMask)

Disable FRO16kHz output clock to selected modules.

#### Parameters

- base – VBAT peripheral base address.
- connectionsMask – The OR'ed value of *vbat\_clock\_enable\_t*.

static inline void VBAT\_\_LockFRO16kSettings(VBAT\_Type \*base)

Lock settings of internal 16kHz free running oscillator, please note that if locked 16kHz FRO's settings can not be updated until the next POR.

---

**Note:** Please note that the operation to ungate/gate FRO 16kHz output clock can not be locked by this function.

---

**Parameters**

- base – VBAT peripheral base address.

static inline bool VBAT\_\_CheckFRO16kSettingsLocked(VBAT\_Type \*base)

Check if FRO16K settings are locked.

**Parameters**

- base – VBAT peripheral base address.

**Returns**

true in case of FRO16k settings are locked, false in case of FRO16k settings are not locked.

static inline void VBAT\_\_EnableCrystalOsc32k(VBAT\_Type \*base, bool enable)

Enable/disable 32K Crystal Oscillator.

**Parameters**

- base – VBAT peripheral base address.
- enable – Used to enable/disable 32k Crystal Oscillator:
  - **true** Enable crystal oscillator and polling status register to check clock is ready.
  - **false** Disable crystal oscillator.

static inline void VBAT\_\_BypassCrystalOsc32k(VBAT\_Type \*base, bool enableBypass)

Bypass 32k crystal oscillator, the clock is still output by oscillator but this clock is the same as clock provided on EXTAL pin.

---

**Note:** In bypass mode, oscillator must be enabled; To exit bypass mode, oscillator must be disabled.

---

**Parameters**

- base – VBAT peripheral base address.
- enableBypass – Used to enter/exit bypass mode:
  - **true** Enter into bypass mode;
  - **false** Exit bypass mode.

static inline void VBAT\_\_AdjustCrystalOsc32kAmplifierGain(VBAT\_Type \*base, uint8\_t coarse)

Adjust 32k crystal oscillator amplifier gain.

**Parameters**

- base – VBAT peripheral base address.
- coarse – Specify amplifier coarse trim value.

*status\_t* VBAT\_\_SetCrystalOsc32kModeAndLoadCapacitance(VBAT\_Type \*base,  
vbat\_osc32k\_operate\_mode\_t  
operateMode,  
vbat\_osc32k\_load\_capacitance\_select\_t  
xtalCap,  
vbat\_osc32k\_load\_capacitance\_select\_t  
extalCap)

Set 32k crystal oscillator mode and load capacitance for the XTAL/EXTAL pin.

**Parameters**

- base – VBAT peripheral base address.

- `operateMode` – Specify the crystal oscillator mode, please refer to `vbat_osc32k_operate_mode_t`.
- `xtalCap` – Specify the internal capacitance for the XTAL pin from the capacitor bank.
- `extalCap` – Specify the internal capacitance for the EXTAL pin from the capacitor bank.

#### Return values

- `kStatus_VBAT_WrongCapacitanceValue` – The load capacitance value to set is not align with operate mode's requirements.
- `kStatus_Success` – Success to set operate mode and load capacitance.

```
static inline void VBAT_TrimCrystalOsc32kStartupTime(VBAT_Type *base,  
                                                    vbat_osc32k_start_up_time_t  
                                                    startupTime)
```

Trim 32k crystal oscillator startup time.

#### Parameters

- `base` – VBAT peripheral base address.
- `startupTime` – Specify the startup time of the oscillator.

```
static inline void VBAT_SetOsc32kSwitchModeComparatorTrimValue(VBAT_Type *base, uint8_t  
                                                                comparatorTrimValue)
```

Set crystal oscillator comparator trim value when oscillator is set as low power switch mode.

#### Parameters

- `base` – VBAT peripheral base address.
- `comparatorTrimValue` – Comparator trim value, ranges from 0 to 7.

```
static inline void VBAT_SetOsc32kSwitchModeDelayTrimValue(VBAT_Type *base, uint8_t  
                                                         delayTrimValue)
```

Set crystal oscillator delay trim value when oscillator is set as low power switch mode.

#### Parameters

- `base` – VBAT peripheral base address.
- `delayTrimValue` – Delay trim value, ranges from 0 to 15.

```
static inline void VBAT_SetOsc32kSwitchModeCapacitorTrimValue(VBAT_Type *base, uint8_t  
                                                             capacitorTrimValue)
```

Set crystal oscillator capacitor trim value when oscillator is set as low power switch mode.

#### Parameters

- `base` – VBAT peripheral base address.
- `capacitorTrimValue` – Capacitor value to trim, ranges from 0 to 3.

```
static inline void VBAT_LockOsc32kSettings(VBAT_Type *base)
```

Lock Osc32k settings, after locked all writes to the Oscillator registers are blocked.

#### Parameters

- `base` – VBAT peripheral base address.

```
static inline void VBAT_UnlockOsc32kSettings(VBAT_Type *base)
```

Unlock Osc32k settings.

#### Parameters

- base – VBAT peripheral base address.

static inline bool VBAT\_\_CheckOsc32kSettingsLocked(VBAT\_Type \*base)

Check if osc32k settings are locked.

#### Parameters

- base – VBAT peripheral base address.

#### Returns

true in case of osc32k settings are locked, false in case of osc32k settings are not locked.

static inline void VBAT\_\_UngateOsc32k(VBAT\_Type \*base, uint8\_t connectionsMask)

Enable OSC32k output clock to selected modules.

#### Parameters

- base – VBAT peripheral base address.
- connectionsMask – The OR'ed value of vbat\_clock\_enable\_t.

static inline void VBAT\_\_GateOsc32k(VBAT\_Type \*base, uint8\_t connectionsMask)

Disable OSC32k output clock to selected modules.

#### Parameters

- base – VBAT peripheral base address.
- connectionsMask – The OR'ed value of vbat\_clock\_enable\_t.

status\_t VBAT\_\_EnableBandgap(VBAT\_Type \*base, bool enable)

Enable/disable Bandgap.

---

**Note:** The FRO16K must be enabled before enabling the bandgap.

---

---

**Note:** This setting can be locked by VBAT\_\_LockRamLdoSettings() function.

---

#### Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable bandgap.
  - **true** Enable the bandgap.
  - **false** Disable the bandgap.

#### Return values

- kStatus\_\_Success – Success to enable/disable the bandgap.
- kStatus\_VBAT\_\_Fro16kNotEnabled – Fail to enable the bandgap due to FRO16k is not enabled previously.

static inline bool VBAT\_\_CheckBandgapEnabled(VBAT\_Type \*base)

Check if bandgap is enabled.

#### Parameters

- base – VBAT peripheral base address.

#### Return values

- true – The bandgap is enabled.
- false – The bandgap is disabled.

```
static inline void VBAT_EnableBandgapRefreshMode(VBAT_Type *base, bool
                                                enableRefreshMode)
```

Enable/disable bandgap low power refresh mode.

---

**Note:** For lowest power consumption, refresh mode must be enabled.

---

---

**Note:** This setting can be locked by VBAT\_LockRamLdoSettings() function.

---

#### Parameters

- base – VBAT peripheral base address.
- enableRefreshMode – Used to enable/disable bandgap low power refresh mode.
  - **true** Enable bandgap low power refresh mode.
  - **false** Disable bandgap low power refresh mode.

```
status_t VBAT_EnableBackupSRAMRegulator(VBAT_Type *base, bool enable)
```

Enable/disable Backup RAM Regulator(RAM\_LDO).

---

**Note:** This setting can be locked by VBAT\_LockRamLdoSettings() function.

---

#### Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable RAM\_LDO.
  - **true** Enable backup SRAM regulator.
  - **false** Disable backup SRAM regulator.

#### Return values

- kStatusSuccess – Success to enable/disable backup SRAM regulator.
- kStatus\_VBAT\_Fro16kNotEnabled – Fail to enable backup SRAM regulator due to FRO16k is not enabled previously.
- kStatus\_VBAT\_BandgapNotEnabled – Fail to enable backup SRAM regulator due to the bandgap is not enabled previously.

```
static inline void VBAT_LockRamLdoSettings(VBAT_Type *base)
```

Lock settings of RAM\_LDO, please note that if locked then RAM\_LDO's settings can not be updated until the next POR.

#### Parameters

- base – VBAT peripheral base address.

```
static inline bool VBAT_CheckRamLdoSettingsLocked(VBAT_Type *base)
```

Check if RAM\_LDO settings is locked.

#### Parameters

- base – VBAT peripheral base address.

#### Returns

true in case of RAM\_LDO settings are locked, false in case of RAM\_LDO settings are unlocked.

`status_t VBAT_SwitchSRAMPowerByLDOSRAM(VBAT_Type *base)`

Switch the SRAM to be powered by LDO\_RAM.

---

**Note:** This function can be used to switch the SRAM to the VBAT retention supply at any time, but please note that the SRAM must not be accessed during this time.

---

---

**Note:** Invoke this function to switch power supply before switching off external power.

---

---

**Note:** RAM\_LDO must be enabled before invoking this function.

---

---

**Note:** To access the SRAM arrays retained by the LDO\_RAM, please invoke `VBAT_SwitchSRAMPowerBySocSupply()`, after external power is switched back on.

---

#### Parameters

- `base` – VBAT peripheral base address.

#### Return values

- `kStatusSuccess` – Success to Switch SRAM powered by VBAT.
- `kStatus_VBAT_Fro16kNotEnabled` – Fail to switch SRAM powered by VBAT due to FRO16K not enabled previously.

`static inline void VBAT_SwitchSRAMPowerBySocSupply(VBAT_Type *base)`

Switch the RAM to be powered by Soc Supply in software mode.

#### Parameters

- `base` – VBAT peripheral base address.

`static inline void VBAT_PowerOffSRAMsInLowPowerModes(VBAT_Type *base, uint8_t sramMask)`

Power off selected SRAM array in low power modes.

#### Parameters

- `base` – VBAT peripheral base address.
- `sramMask` – The mask of SRAM array to power off, should be the OR'ed value of `vbat_ram_array_t`.

`static inline void VBAT_RetainSRAMsInLowPowerModes(VBAT_Type *base, uint8_t sramMask)`

Retain selected SRAM array in low power modes.

#### Parameters

- `base` – VBAT peripheral base address.
- `sramMask` – The mask of SRAM array to retain, should be the OR'ed value of `vbat_ram_array_t`.

`static inline void VBAT_EnableSRAMIsolation(VBAT_Type *base, bool enable)`

Enable/disable SRAM isolation.

#### Parameters

- `base` – VBAT peripheral base address.
- `enable` – Used to enable/disable SRAM violation.

- **true** SRAM will be isolated.
- **false** SRAM state follows the SoC power modes.

*status\_t* VBAT\_EnableBandgapTimer(VBAT\_Type \*base, bool enable, uint8\_t timerIdMask)  
Enable/disable Bandgap timer.

---

**Note:** The bandgap timer is available when the bandgap is enabled and are clocked by the FRO16k.

---

#### Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable bandgap timer.
- timerIdMask – The mask of bandgap timer Id, should be the OR'ed value of vbat\_bandgap\_timer\_id\_t.

#### Return values

- kStatus\_Success – Success to enable/disable selected bandgap timer.
- kStatus\_VBAT\_Fro16kNotEnabled – Fail to enable/disable selected bandgap timer due to FRO16k not enabled previously.
- kStatus\_VBAT\_BandgapNotEnabled – Fail to enable/disable selected bandgap timer due to bandgap not enabled previously.

void VBAT\_SetBandgapTimer0TimeoutValue(VBAT\_Type \*base,  
vbat\_bandgap\_timer0\_timeout\_period\_t  
timeoutPeriod)

Set bandgap timer0 timeout value.

---

**Note:** The timeout value can only be changed when the timer is disabled.

---

#### Parameters

- base – VBAT peripheral base address.
- timeoutPeriod – Bandgap timer timeout value, please refer to vbat\_bandgap\_timer0\_timeout\_period\_t.

void VBAT\_SetBandgapTimer1TimeoutValue(VBAT\_Type \*base, uint32\_t timeoutPeriod)  
Set bandgap timer1 timeout value.

---

**Note:** The timeout value can only be changed when the timer is disabled.

---

#### Parameters

- base – VBAT peripheral base address.
- timeoutPeriod – The bandgap timerout 1 period, in number of seconds, ranging from 0 to 65535s.

static inline void VBAT\_SwitchVBATModuleSupplyActiveMode(VBAT\_Type \*base,  
vbat\_internal\_module\_supply\_t  
supply)

Control the VBAT internal switch in active mode, VBAT modules can be supplied by VDD\_BAT and VDD\_SYS.

**Parameters**

- base – VBAT peripheral base address.
- supply – Used to control the VBAT internal switch.

static inline *vbat\_internal\_module\_supply\_t* VBAT\_GetVBATModuleSupply(VBAT\_Type \*base)  
Get VBAT module supply in active mode.

**Parameters**

- base – VBAT peripheral base address.

**Returns**

VDD\_SYS supplies VBAT modules or VDD\_BAT supplies VBAT modules, in type of *vbat\_internal\_module\_supply\_t*.

static inline void VBAT\_SwitchVBATModuleSupplyLowPowerMode(VBAT\_Type \*base,  
*vbat\_internal\_module\_supply\_t*  
supply)

Control the VBAT internal switch in low power modes.

---

**Note:** If VBAT modules are supplied by VDD\_SYS in low power modes, VBAT module will also supplied by VDD\_SYS in active mode.

---

**Parameters**

- base – VBAT peripheral base address.
- supply – Used to specify which voltage input supply VBAT modules in low power mode.

static inline void VBAT\_LockSwitchControl(VBAT\_Type \*base)  
Lock switch control, if locked all writes to the switch registers will be blocked.

**Parameters**

- base – VBAT peripheral base address.

static inline void VBAT\_UnlockSwitchControl(VBAT\_Type \*base)  
Unlock switch control.

**Parameters**

- base – VBAT peripheral base address.

static inline bool VBAT\_CheckSwitchControlLocked(VBAT\_Type \*base)  
Check if switch control is locked.

**Parameters**

- base – VBAT peripheral base address.

**Return values**

- false – switch control is not locked.
- true – switch control is locked, any writes to related registers are blocked.

*status\_t* VBAT\_InitClockMonitor(VBAT\_Type \*base, const *vbat\_clock\_monitor\_config\_t* \*config)  
Initialize the VBAT clock monitor, enable clock monitor and set the clock monitor configuration.

---

**Note:** Both FRO16K and OSC32K should be enabled and stable before invoking this function.

---



**Parameters**

- base – VBAT peripheral base address.
- config – Pointer to `vbat_clock_monitor_config_t` structure.

**Return values**

- `kStatus_Success` – Clock monitor is initialized successfully.
- `kStatus_VBAT_Fro16kNotEnabled` – FRO16K is not enabled.
- `kStatus_VBAT_Osc32kNotReady` – OSC32K is not ready.
- `kStatus_VBAT_ClockMonitorLocked` – Clock monitor is locked.

`status_t` VBAT\_DeinitMonitor(VBAT\_Type \*base)

Deinitialize the VBAT clock monitor.

**Parameters**

- base – VBAT peripheral base address.

**Return values**

- `kStatus_Success` – Clock monitor is de-initialized successfully.
- `kStatus_VBAT_ClockMonitorLocked` – Control of Clock monitor is locked.

`static inline void` VBAT\_EnableClockMonitor(VBAT\_Type \*base, bool enable)

Enable/disable clock monitor.

- false: disable clock monitor.

**Parameters**

- base – VBAT peripheral base address.
- enable – Switcher to enable/disable clock monitor:
  - true: enable clock monitor;

`static inline void` VBAT\_SetClockMonitorDivideTrim(VBAT\_Type \*base,  
vbat\_clock\_monitor\_divide\_trim\_t  
divideTrim)

Set clock monitor's divide trim, available value is `kVBAT_ClockMonitorOperateAt1kHz` and `kVBAT_ClockMonitorOperateAt64Hz`.

**Parameters**

- base – VBAT peripheral base address.
- divideTrim – Specify divide trim value, please refer to `vbat_clock_monitor_divide_trim_t`.

`static inline void` VBAT\_SetClockMonitorFrequencyTrim(VBAT\_Type \*base,  
vbat\_clock\_monitor\_freq\_trim\_t  
freqTrim)

Set clock monitor's frequency trim, available value is `kVBAT_ClockMonitorAssert2Cycle`, `kVBAT_ClockMonitorAssert4Cycle`, `kVBAT_ClockMonitorAssert6Cycle` and `kVBAT_ClockMonitorAssert8Cycle`.

**Parameters**

- base – VBAT peripheral base address.
- freqTrim – Specify frequency trim value, please refer to `vbat_clock_monitor_freq_trim_t`.

static inline void VBAT\_LockClockMonitorControl(VBAT\_Type \*base)

Lock clock monitor enable/disable control.

---

**Note:** If locked, it is not allowed to change clock monitor enable/disable control.

---

#### Parameters

- base – VBAT peripheral base address.

static inline void VBAT\_UnlockClockMonitorControl(VBAT\_Type \*base)

Unlock clock monitor enable/disable control.

#### Parameters

- base – VBAT peripheral base address.

static inline bool VBAT\_CheckClockMonitorControlLocked(VBAT\_Type \*base)

Check if clock monitor enable/disable control is locked.

---

**Note:** If locked, it is not allowed to change clock monitor enable/disable control.

---

#### Parameters

- base – VBAT peripheral base address.

#### Return values

- false – clock monitor enable/disable control is not locked.
- true – clock monitor enable/disable control is locked, any writes to related registers are blocked.

*status\_t* VBAT\_InitTamper(VBAT\_Type \*base, const *vbat\_tamper\_config\_t* \*config)

Initialize tamper control.

---

**Note:** Both FRO16K and bandgap should be enabled before calling this function.

---

#### Parameters

- base – VBAT peripheral base address.
- config – Pointer to *vbat\_tamper\_config\_t* structure.

#### Return values

- *kStatus\_Success* – Tamper is initialized successfully.
- *kStatus\_VBAT\_TamperLocked* – Tamper control is locked.
- *kStatus\_VBAT\_BandgapNotEnabled* – Bandgap is not enabled.
- *kStatus\_VBAT\_Fro16kNotEnabled* – FRO 16K is not enabled.

*status\_t* VBAT\_DeinitTamper(VBAT\_Type \*base)

De-initialize tamper control.

#### Parameters

- base – VBAT peripheral base address.

#### Return values

- *kStatus\_Success* – Tamper is de-initialized successfully.

- kStatus\_VBAT\_TamperLocked – Tamper control is locked.

static inline void VBAT\_EnableTamper(VBAT\_Type \*base, uint32\_t tamperEnableMask)

Enable tamper for VBAT.

#### Parameters

- base – VBAT peripheral base address.
- tamperEnableMask – Mask of tamper to be enabled, should be the OR'ed value of \_vbat\_tamper\_enable.

static inline void VBAT\_DisableTamper(VBAT\_Type \*base, uint32\_t tamperEnableMask)

Disable tamper for VBAT.

#### Parameters

- base – VBAT peripheral base address.
- tamperEnableMask – Mask of tamper to be disabled, should be the OR'ed value of \_vbat\_tamper\_enable.

static inline uint32\_t VBAT\_GetTamperEnableInfo(VBAT\_Type \*base)

Get tamper enable information.

#### Parameters

- base – VBAT peripheral base address.

#### Returns

Mask of tamper enable information, should be the OR'ed value of \_vbat\_tamper\_enable.

static inline void VBAT\_LockTamperControl(VBAT\_Type \*base)

Lock tamper control, if locked, it is not allowed to change tamper control.

#### Parameters

- base – VBAT peripheral base address.

static inline void VBAT\_UnlockTamperControl(VBAT\_Type \*base)

Unlock tamper control.

#### Parameters

- base – VBAT peripheral base address.

static inline bool VBAT\_CheckTamperControlLocked(VBAT\_Type \*base)

Check if tamper control is locked.

#### Parameters

- base – VBAT peripheral base address.

#### Return values

- false – Tamper control is not locked.
- true – Tamper control is locked, any writes to related registers are blocked.

static inline uint32\_t VBAT\_GetStatusFlags(VBAT\_Type \*base)

Get VBAT status flags.

#### Parameters

- base – VBAT peripheral base address.

#### Returns

The asserted status flags, should be the OR'ed value of vbat\_status\_flag\_t.

static inline void VBAT\_ClearStatusFlags(VBAT\_Type \*base, uint32\_t mask)

Clear VBAT status flags.

**Parameters**

- base – VBAT peripheral base address.
- mask – The mask of status flags to be cleared, should be the OR'ed value of vbat\_status\_flag\_t except kVBAT\_StatusFlagLdoReady, kVBAT\_StatusFlagOsc32kReady, kVBAT\_StatusFlagInterrupt0Detect, kVBAT\_StatusFlagInterrupt1Detect, kVBAT\_StatusFlagInterrupt2Detect, kVBAT\_StatusFlagInterrupt3Detect.

static inline void VBAT\_EnableInterrupts(VBAT\_Type \*base, uint32\_t mask)

Enable interrupts for the VBAT module, such as POR detect interrupt, Wakeup Pin interrupt and so on.

**Parameters**

- base – VBAT peripheral base address.
- mask – The mask of interrupts to be enabled, should be the OR'ed value of vbat\_interrupt\_enable\_t.

static inline void VBAT\_DisableInterrupts(VBAT\_Type \*base, uint32\_t mask)

Disable interrupts for the VBAT module, such as POR detect interrupt, wakeup pin interrupt and so on.

**Parameters**

- base – VBAT peripheral base address.
- mask – The mask of interrupts to be disabled, should be the OR'ed value of vbat\_interrupt\_enable\_t.

static inline void VBAT\_EnableWakeup(VBAT\_Type \*base, uint32\_t mask)

Enable wakeup for the VBAT module, such as POR detect wakeup, wakeup pin wakeup and so on.

**Parameters**

- base – VBAT peripheral base address.
- mask – The mask of enumerators in vbat\_wakeup\_enable\_t.

static inline void VBAT\_DisableWakeup(VBAT\_Type \*base, uint32\_t mask)

Disable wakeup for VBAT module, such as POR detect wakeup, wakeup pin wakeup and so on.

**Parameters**

- base – VBAT peripheral base address.
- mask – The mask of enumerators in vbat\_wakeup\_enable\_t.

static inline void VBAT\_LockInterruptWakeupSettings(VBAT\_Type \*base)

Lock VBAT interrupt and wakeup settings, please note that if locked the interrupt and wakeup settings can not be updated until the next POR.

**Parameters**

- base – VBAT peripheral base address.

static inline void VBAT\_SetWakeupPinDefaultState(VBAT\_Type \*base, bool assert)

Set the default state of the WAKEUP\_b pin output when no enabled wakeup source is asserted.

**Parameters**

- `base` – VBAT peripheral base address.
- `assert` – Used to set default state of the WAKEUP\_b pin output:
  - **true** WAKEUP\_b output state is logic one;
  - **false** WAKEUP\_b output state is logic zero.

`struct __vbat_fro16k_config`

*#include <fsl\_vbat.h>* The structure of internal 16kHz free running oscillator attributes.

### Public Members

`bool enableFRO16k`

Enable/disable internal 16kHz free running oscillator.

`uint8_t enabledConnectionsMask`

The mask of connected modules to enable FRO16k clock output.

`struct __vbat_clock_monitor_config`

*#include <fsl\_vbat.h>* The structure of internal clock monitor, including divide trim and frequency trim.

### Public Members

`vbat_clock_monitor_freq_trim_t freqTrim`

Frequency trim value used to adjust the clock monitor assert, please refer to `vbat_clock_monitor_freq_trim_t`.

`bool lock`

Lock the clock monitor control after enabled.

`struct __vbat_tamper_config`

*#include <fsl\_vbat.h>* The structure of Tamper configuration.

### Public Members

`bool enableVoltageDetect`

Enable/disable voltage detection.

`bool enableTemperatureDetect`

Enable/disable temperature detection.

`bool lock`

Lock the tamper control after enabled.

## 2.44 OSTIMER: OS Event Timer Driver

`void OSTIMER_Init(OSTIMER_Type *base)`

Initializes an OSTIMER by turning its bus clock on.

`void OSTIMER_Deinit(OSTIMER_Type *base)`

Deinitializes a OSTIMER instance.

This function shuts down OSTIMER bus clock

### Parameters

- base – OSTIMER peripheral base address.

uint64\_t OSTIMER\_GrayToDecimal(uint64\_t gray)

Translate the value from gray-code to decimal.

**Parameters**

- gray – The gray value input.

**Returns**

The decimal value.

static inline uint64\_t OSTIMER\_DecimalToGray(uint64\_t dec)

Translate the value from decimal to gray-code.

**Parameters**

- dec – The decimal value.

**Returns**

The gray code of the input value.

uint32\_t OSTIMER\_GetStatusFlags(OSTIMER\_Type \*base)

Get OSTIMER status Flags.

This returns the status flag. Currently, only match interrupt flag can be got.

**Parameters**

- base – OSTIMER peripheral base address.

**Returns**

status register value

void OSTIMER\_ClearStatusFlags(OSTIMER\_Type \*base, uint32\_t mask)

Clear Status Interrupt Flags.

This clears intrrupt status flag. Currently, only match interrupt flag can be cleared.

**Parameters**

- base – OSTIMER peripheral base address.
- mask – Clear bit mask.

**Returns**

none

status\_t OSTIMER\_SetMatchRawValue(OSTIMER\_Type \*base, uint64\_t count, *ostimer\_callback\_t* cb)

Set the match raw value for OSTIMER.

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central EVTIMER. Please note that, the data format may be gray-code, if so, please using OSTIMER\_SetMatchValue().

**Parameters**

- base – OSTIMER peripheral base address.
- count – OSTIMER timer match value.(Value may be gray-code format)
- cb – OSTIMER callback (can be left as NULL if none, otherwise should be a void func(void)).

**Return values**

kStatus\_Success – - Set match raw value and enable interrupt Successfully.

*status\_t* OSTIMER\_SetMatchValue(OSTIMER\_Type \*base, uint64\_t count, *ostimer\_callback\_t* cb)

Set the match value for OSTIMER.

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central OS TIMER.

#### Parameters

- base – OSTIMER peripheral base address.
- count – OSTIMER timer match value.(Value is decimal format, and this value will be translate to Gray code in API if the IP counter is gray encoded.)
- cb – OSTIMER callback (can be left as NULL if none, otherwise should be a void func(void)).

#### Return values

kStatus\_Success -- Set match raw value and enable interrupt Successfully.

static inline void OSTIMER\_SetMatchRegister(OSTIMER\_Type \*base, uint64\_t value)

Set value to OSTIMER MATCH register directly.

This function writes the input value to OSTIMER MATCH register directly, it does not touch any other registers. Note that, the data format is gray-code. The function OSTIMER\_DecimalToGray could convert decimal value to gray code.

#### Parameters

- base – OSTIMER peripheral base address.
- value – OSTIMER timer match value (Value is gray-code format).

static inline uint64\_t OSTIMER\_GetMatchRegister(OSTIMER\_Type \*base)

Get the match value from OSTIMER.

This function will get the match value from OSTIMER. The value of timer match is gray code format.

#### Parameters

- base – OSTIMER peripheral base address.

#### Returns

Value of match register, data format is gray code.

static inline uint64\_t OSTIMER\_GetMatchValue(OSTIMER\_Type \*base)

Get the match value from OSTIMER.

This function will get a match value from OSTIMER.

#### Parameters

- base – OSTIMER peripheral base address.

#### Returns

Value of match register.

static inline void OSTIMER\_EnableMatchInterrupt(OSTIMER\_Type \*base)

Enable the OSTIMER counter match interrupt.

Enable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

#### Parameters

- base – OSTIMER peripheral base address.

static inline void OSTIMER\_DisableMatchInterrupt(OSTIMER\_Type \*base)

Disable the OSTIMER counter match interrupt.

Disable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

**Parameters**

- base – OSTIMER peripheral base address.

static inline uint64\_t OSTIMER\_GetCurrentTimerRawValue(OSTIMER\_Type \*base)

Get current timer raw count value from OSTIMER.

This function will get the timer count value from OS timer register. The raw value of timer count may be gray code format.

**Parameters**

- base – OSTIMER peripheral base address.

**Returns**

Raw value of OSTIMER, may be gray code format.

uint64\_t OSTIMER\_GetCurrentTimerValue(OSTIMER\_Type \*base)

Get current timer count value from OSTIMER.

This function will get a decimal timer count value. If the RAW value of timer count is gray code format, it will be translated to decimal data internally.

**Parameters**

- base – OSTIMER peripheral base address.

**Returns**

Value of OSTIMER which will be formatted to decimal value.

static inline uint64\_t OSTIMER\_GetCaptureRawValue(OSTIMER\_Type \*base)

Get the capture value from OSTIMER.

This function will get a captured value from OSTIMER. The Raw value of timer capture may be gray code format.

**Parameters**

- base – OSTIMER peripheral base address.

**Returns**

Raw value of capture register, data format may be gray code.

uint64\_t OSTIMER\_GetCaptureValue(OSTIMER\_Type \*base)

Get the capture value from OSTIMER.

This function will get a capture decimal-value from OSTIMER. If the RAW value of timer count is gray code format, it will be translated to decimal data internally.

**Parameters**

- base – OSTIMER peripheral base address.

**Returns**

Value of capture register, data format is decimal.

void OSTIMER\_HandleIRQ(OSTIMER\_Type \*base, *ostimer\_callback\_t* cb)

OS timer interrupt Service Handler.

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in OSTIMER\_SetMatchValue()). if no user callback is scheduled, the interrupt will simply be cleared.

**Parameters**



- base – OS timer peripheral base address.
- cb – callback scheduled for this instance of OS timer

**Returns**

none

FSL\_OSTIMER\_DRIVER\_VERSION

OSTIMER driver version.

enum \_\_ostimer\_flags

OSTIMER status flags.

*Values:*

enumerator kOSTIMER\_MatchInterruptFlag

Match interrupt flag bit, sets if the match value was reached.

typedef void (\*ostimer\_callback\_t)(void)

ostimer callback function.

## 2.45 PORT: Port Control and Interrupts

static inline void PORT\_GetVersionInfo(PORT\_Type \*base, port\_version\_info\_t \*info)

Get PORT version information.

**Parameters**

- base – PORT peripheral base pointer
- info – PORT version information

static inline void PORT\_SecletPortVoltageRange(PORT\_Type \*base, port\_voltage\_range\_t range)

Get PORT version information.

---

**Note:** : PORTA\_CONFIG[RANGE] controls the voltage ranges of Port A, B, and C. Read or write PORTB\_CONFIG[RANGE] and PORTC\_CONFIG[RANGE] does not take effect.

---

**Parameters**

- base – PORT peripheral base pointer
- range – port voltage range

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\_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 void PORT__EnablePinDoubleDriveStrength(PORT_Type *base, uint32_t pin, bool enable)
```

Enables the port pin double drive strength.

#### Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- enable – PORT pin drive strength configuration.

```
static inline void PORT__SetPinPullValue(PORT_Type *base, uint32_t pin, uint8_t value)
```

Configures the port pin pull value.

#### Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- value – PORT pin pull value
  - kPORT\_LowPullResistor = 0U - Low internal pull resistor value is selected.
  - kPORT\_HighPullResistor = 1U - High internal pull resistor value is selected.

```
static inline uint32_t PORT__GetEFTDetectFlags(PORT_Type *base)
```

Get EFT detect flags.

#### Parameters

- base – PORT peripheral base pointer

#### Returns

EFT detect flags

```
static inline void PORT__EnableEFTDetectInterrupts(PORT_Type *base, uint32_t interrupt)
```

Enable EFT detect interrupts.

#### Parameters

- base – PORT peripheral base pointer
- interrupt – EFT detect interrupt

```
static inline void PORT__DisableEFTDetectInterrupts(PORT_Type *base, uint32_t interrupt)
```

Disable EFT detect interrupts.

#### Parameters

- base – PORT peripheral base pointer
- interrupt – EFT detect interrupt

```
static inline void PORT__ClearAllLowEFTDetectors(PORT_Type *base)
```

Clear all low EFT detector.

---

**Note:** : Port B and Port C pins share the same EFT detector clear control from PORTC\_EDCR register. Any write to the PORTB\_EDCR does not take effect.

---

**Parameters**

- base – PORT peripheral base pointer
- interrupt – EFT detect interrupt

static inline void PORT\_ClearAllHighEFTDetectors(PORT\_Type \*base)

Clear all high EFT detector.

**Parameters**

- base – PORT peripheral base pointer
- interrupt – EFT detect interrupt

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\_pull\_value

Internal resistor pull value selection.

*Values:*

enumerator kPORT\_LowPullResistor

Low internal pull resistor value is selected.

enumerator kPORT\_HighPullResistor

High internal pull resistor value is selected.

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_drive_strength1`  
Configures the drive strength1.  
*Values:*  
enumerator `kPORT_NormalDriveStrength`  
Normal drive strength  
enumerator `kPORT_DoubleDriveStrength`  
Double drive strength

enum `_port_input_buffer`  
input buffer disable/enable.  
*Values:*  
enumerator `kPORT_InputBufferDisable`  
Digital input is disabled  
enumerator `kPORT_InputBufferEnable`  
Digital input is enabled

enum `_port_invet_input`  
Digital input is not inverted or it is inverted.  
*Values:*  
enumerator `kPORT_InputNormal`  
Digital input is not inverted  
enumerator `kPORT_InputInvert`  
Digital input is inverted

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\_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.

enum \_\_port\_voltage\_range

PORT voltage range.

*Values:*

enumerator kPORT\_VoltageRange1Dot71V\_3Dot6V

Port voltage range is 1.71 V - 3.6 V.

enumerator kPORT\_VoltageRange2Dot70V\_3Dot6V

Port voltage range is 2.70 V - 3.6 V.

typedef enum \_\_port\_mux port\_mux\_t

Pin mux selection.

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.

typedef struct \_\_port\_version\_info port\_version\_info\_t

PORT version information.

typedef enum \_\_port\_voltage\_range port\_voltage\_range\_t

PORT voltage range.

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 pullValueSelect

Pull value 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 driveStrength1
    Normal/Double drive strength enable/disable

uint16_t inputBuffer
    Input Buffer Configure

uint16_t invertInput
    Invert Input Configure

uint16_t lockRegister
    Lock/unlock the PCR field[15:0]

struct __port__version__info
    #include <fsl_port.h> PORT version information.
```

### Public Members

```
uint16_t feature
    Feature Specification Number.

uint8_t minor
    Minor Version Number.

uint8_t major
    Major Version Number.
```

## 2.46 PWM: Pulse Width Modulator

```
status_t PWM_Init(PWM_Type *base, pwm_submodule_t subModule, const pwm_config_t
    *config)
```

Ungates the PWM submodule clock and configures the peripheral for basic operation.

This API should be called at the beginning of the application using the PWM driver. When user select PWMX, user must choose edge aligned output, because there are some limitations on center aligned PWMX output. When output PWMX in center aligned mode, VAL1 register controls both PWM period and PWMX duty cycle, PWMA and PWMB output will be corrupted. But edge aligned PWMX output do not have such limit. In master reload counter initialization mode, PWM period is depended by period of set LDOK in submodule 0 because this operation will reload register. Submodule 0 counter initialization cannot be master sync or master reload.

### Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- config – Pointer to user's PWM config structure.

### Returns

kStatus\_Success means success; else failed.

```
void PWM_Deinit(PWM_Type *base, pwm_submodule_t subModule)
```

Gate the PWM submodule clock.

#### Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to deinitialize

```
void PWM_GetDefaultConfig(pwm_config_t *config)
```

Fill in the PWM config struct with the default settings.

The default values are:

```
config->enableDebugMode = false;
config->enableWait = false;
config->reloadSelect = kPWM_LocalReload;
config->clockSource = kPWM_BusClock;
config->prescale = kPWM_Prescale_Divide_1;
config->initializationControl = kPWM_Initialize_LocalSync;
config->forceTrigger = kPWM_Force_Local;
config->reloadFrequency = kPWM_LoadEveryOpportunity;
config->reloadLogic = kPWM_ReloadImmediate;
config->pairOperation = kPWM_Independent;
```

#### Parameters

- config – Pointer to user's PWM config structure.

```
status_t PWM_SetupPwm(PWM_Type *base, pwm_submodule_t subModule, const
    pwm_signal_param_t *chnlParams, uint8_t numOfChnls,
    pwm_mode_t mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz)
```

Sets up the PWM signals for a PWM submodule.

The function initializes the submodule according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements. If the dead time insertion logic is enabled, the pulse period is reduced by the dead time period specified by the user. When user select PWMX, user must choose edge aligned output, because there are some limitation on center aligned PWMX output. Due to edge aligned PWMX is negative true signal, need to configure PWMX active low true level to get correct duty cycle. The half cycle point will not be exactly in the middle of the PWM cycle when PWMX enabled.

#### Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- 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. Array size should not be more than 3 as each submodule has 3 pins to output PWM.
- mode – PWM operation mode, options available in enumeration `pwm_mode_t`
- pwmFreq\_Hz – PWM signal frequency in Hz
- srcClock\_Hz – PWM source clock of correspond submodule in Hz. If source clock of submodule1,2,3 is from submodule0 AUX\_CLK, its source clock is submodule0 source clock divided with submodule0 prescaler value instead of submodule0 source clock.

**Returns**

Returns `kStatus_Fail` if there was error setting up the signal; `kStatus_Success` otherwise

```
status_t PWM_SetupPwmPhaseShift(PWM_Type *base, pwm_submodule_t subModule,  
                                pwm_channels_t pwmChannel, uint32_t pwmFreq_Hz,  
                                uint32_t srcClock_Hz, uint8_t shiftvalue, bool doSync)
```

Set PWM phase shift for PWM channel running on channel PWM\_A, PWM\_B which with 50% duty cycle.

**Parameters**

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- pwmFreq\_Hz – PWM signal frequency in Hz
- srcClock\_Hz – PWM main counter clock in Hz.
- shiftvalue – Phase shift value, range in 0 ~ 50
- doSync – true: Set LDOK bit for the submodule list; false: LDOK bit don't set, need to call PWM\_SetPwmLdok to sync update.

**Returns**

Returns `kStatus_Fail` if there was error setting up the signal; `kStatus_Success` otherwise

```
void PWM_UpdatePwmDutycycle(PWM_Type *base, pwm_submodule_t subModule,  
                            pwm_channels_t pwmSignal, pwm_mode_t currPwmMode,  
                            uint8_t dutyCyclePercent)
```

Updates the PWM signal's dutycycle.

The function updates the PWM dutycycle to the new value that is passed in. If the dead time insertion logic is enabled then the pulse period is reduced by the dead time period specified by the user.

**Parameters**

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmSignal – Signal (PWM A, PWM B, PWM X) to update
- currPwmMode – The current PWM mode set during PWM setup
- dutyCyclePercent – New PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle)

```
void PWM_UpdatePwmDutycycleHighAccuracy(PWM_Type *base, pwm_submodule_t subModule,  
                                         pwm_channels_t pwmSignal, pwm_mode_t  
                                         currPwmMode, uint16_t dutyCycle)
```

Updates the PWM signal's dutycycle with 16-bit accuracy.

The function updates the PWM dutycycle to the new value that is passed in. If the dead time insertion logic is enabled then the pulse period is reduced by the dead time period specified by the user.

**Parameters**

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmSignal – Signal (PWM A, PWM B, PWM X) to update

- `currPwmMode` – The current PWM mode set during PWM setup
- `dutyCycle` – New PWM pulse width, value should be between 0 to 65535  
0=inactive signal(0% duty cycle)... 65535=active signal (100% duty cycle)

```
void PWM_UpdatePwmPeriodAndDutycycle(PWM_Type *base, pwm_submodule_t subModule,  
                                     pwm_channels_t pwmSignal, pwm_mode_t  
                                     currPwmMode, uint16_t pulseCnt, uint16_t  
                                     dutyCycle)
```

Update the PWM signal's period and dutycycle for a PWM submodule.

The function updates PWM signal period generated by a specific submodule according to the parameters passed in by the user. This function can also set dutycycle whether you want to keep original dutycycle or update new dutycycle. Call this function in local sync control mode because PWM period is depended by

INIT and VAL1 register of each submodule. In master sync initialization control mode, call this function to update INIT and VAL1 register of all submodule because PWM period is depended by INIT and VAL1 register in submodule0. If the dead time insertion logic is enabled, the pulse period is reduced by the dead time period specified by the user. PWM signal will not be generated if its period is less than dead time duration.

#### Parameters

- `base` – PWM peripheral base address
- `subModule` – PWM submodule to configure
- `pwmSignal` – Signal (PWM A or PWM B) to update
- `currPwmMode` – The current PWM mode set during PWM setup, options available in enumeration `pwm_mode_t`
- `pulseCnt` – New PWM period, value should be between 0 to 65535 0=minimum PWM period... 65535=maximum PWM period
- `dutyCycle` – New PWM pulse width of channel, value should be between 0 to 65535 0=inactive signal(0% duty cycle)... 65535=active signal (100% duty cycle) You can keep original duty cycle or update new duty cycle

```
static inline void PWM_EnableInterrupts(PWM_Type *base, pwm_submodule_t subModule,  
                                       uint32_t mask)
```

Enables the selected PWM interrupts.

#### Parameters

- `base` – PWM peripheral base address
- `subModule` – PWM submodule to configure
- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `pwm_interrupt_enable_t`

```
static inline void PWM_DisableInterrupts(PWM_Type *base, pwm_submodule_t subModule,  
                                       uint32_t mask)
```

Disables the selected PWM interrupts.

#### Parameters

- `base` – PWM peripheral base address
- `subModule` – PWM submodule to configure
- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `pwm_interrupt_enable_t`

```
static inline uint32_t PWM_GetEnabledInterrupts(PWM_Type *base, pwm_submodule_t
subModule)
```

Gets the enabled PWM interrupts.

**Parameters**

- base – PWM peripheral base address
- subModule – PWM submodule to configure

**Returns**

The enabled interrupts. This is the logical OR of members of the enumeration `pwm_interrupt_enable_t`

```
static inline void PWM_DMAFIFOWatermarkControl(PWM_Type *base, pwm_submodule_t
subModule, pwm_watermark_control_t
pwm_watermark_control)
```

Capture DMA Enable Source Select.

**Parameters**

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwm\_watermark\_control – PWM FIFO watermark and control

```
static inline void PWM_DMACaptureSourceSelect(PWM_Type *base, pwm_submodule_t
subModule, pwm_dma_source_select_t
pwm_dma_source_select)
```

Capture DMA Enable Source Select.

**Parameters**

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwm\_dma\_source\_select – PWM capture DMA enable source select

```
static inline void PWM_EnableDMACapture(PWM_Type *base, pwm_submodule_t subModule,
uint16_t mask, bool activate)
```

Enables or disables the selected PWM DMA Capture read request.

**Parameters**

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- mask – The DMA to enable or disable. This is a logical OR of members of the enumeration `pwm_dma_enable_t`
- activate – true: Enable DMA read request; false: Disable DMA read request

```
static inline void PWM_EnableDMAWrite(PWM_Type *base, pwm_submodule_t subModule, bool
activate)
```

Enables or disables the PWM DMA write request.

**Parameters**

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- activate – true: Enable DMA write request; false: Disable DMA write request

```
static inline uint32_t PWM_GetStatusFlags(PWM_Type *base, pwm_submodule_t subModule)
```

Gets the PWM status flags.

**Parameters**

- base – PWM peripheral base address
- subModule – PWM submodule to configure

**Returns**

The status flags. This is the logical OR of members of the enumeration `pwm_status_flags_t`

```
static inline void PWM_ClearStatusFlags(PWM_Type *base, pwm_submodule_t subModule,  
                                       uint32_t mask)
```

Clears the PWM status flags.

**Parameters**

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- mask – The status flags to clear. This is a logical OR of members of the enumeration `pwm_status_flags_t`

```
static inline void PWM_StartTimer(PWM_Type *base, uint8_t subModulesToStart)
```

Starts the PWM counter for a single or multiple submodules.

Sets the Run bit which enables the clocks to the PWM submodule. This function can start multiple submodules at the same time.

**Parameters**

- base – PWM peripheral base address
- subModulesToStart – PWM submodules to start. This is a logical OR of members of the enumeration `pwm_module_control_t`

```
static inline void PWM_StopTimer(PWM_Type *base, uint8_t subModulesToStop)
```

Stops the PWM counter for a single or multiple submodules.

Clears the Run bit which resets the submodule's counter. This function can stop multiple submodules at the same time.

**Parameters**

- base – PWM peripheral base address
- subModulesToStop – PWM submodules to stop. This is a logical OR of members of the enumeration `pwm_module_control_t`

FSL\_PWM\_DRIVER\_VERSION

Version 2.9.1

```
enum _pwm_submodule
```

List of PWM submodules.

*Values:*

enumerator `kPWM_Module_0`

Submodule 0

enumerator `kPWM_Module_1`

Submodule 1

enumerator `kPWM_Module_2`

Submodule 2

enum `_pwm_channels`

List of PWM channels in each module.

*Values:*

enumerator `kPWM_PwmB`

enumerator `kPWM_PwmA`

enumerator `kPWM_PwmX`

enum `_pwm_value_register`

List of PWM value registers.

*Values:*

enumerator `kPWM_ValueRegister_0`

PWM Value0 register

enumerator `kPWM_ValueRegister_1`

PWM Value1 register

enumerator `kPWM_ValueRegister_2`

PWM Value2 register

enumerator `kPWM_ValueRegister_3`

PWM Value3 register

enumerator `kPWM_ValueRegister_4`

PWM Value4 register

enumerator `kPWM_ValueRegister_5`

PWM Value5 register

enum `_pwm_value_register_mask`

List of PWM value registers mask.

*Values:*

enumerator `kPWM_ValueRegisterMask_0`

PWM Value0 register mask

enumerator `kPWM_ValueRegisterMask_1`

PWM Value1 register mask

enumerator `kPWM_ValueRegisterMask_2`

PWM Value2 register mask

enumerator `kPWM_ValueRegisterMask_3`

PWM Value3 register mask

enumerator `kPWM_ValueRegisterMask_4`

PWM Value4 register mask

enumerator `kPWM_ValueRegisterMask_5`

PWM Value5 register mask

enum `_pwm_clock_source`

PWM clock source selection.

*Values:*

enumerator `kPWM_BusClock`

Device specific IPBus clock, refer reference manual for frequency

enumerator kPWM\_ExternalClock

EXT\_CLK is used as the clock

enumerator kPWM\_Submodule0Clock

Clock of the submodule 0 (AUX\_CLK) is used as the source clock

enum \_\_pwm\_clock\_prescale

PWM prescaler factor selection for clock source.

*Values:*

enumerator kPWM\_Prescale\_Divide\_1

PWM clock frequency = fclk/1

enumerator kPWM\_Prescale\_Divide\_2

PWM clock frequency = fclk/2

enumerator kPWM\_Prescale\_Divide\_4

PWM clock frequency = fclk/4

enumerator kPWM\_Prescale\_Divide\_8

PWM clock frequency = fclk/8

enumerator kPWM\_Prescale\_Divide\_16

PWM clock frequency = fclk/16

enumerator kPWM\_Prescale\_Divide\_32

PWM clock frequency = fclk/32

enumerator kPWM\_Prescale\_Divide\_64

PWM clock frequency = fclk/64

enumerator kPWM\_Prescale\_Divide\_128

PWM clock frequency = fclk/128

enum \_\_pwm\_force\_output\_trigger

Options that can trigger a PWM FORCE\_OUT.

*Values:*

enumerator kPWM\_Force\_Local

The local force signal, CTRL2[FORCE], from the submodule is used to force updates

enumerator kPWM\_Force\_Master

The master force signal from submodule 0 is used to force updates

enumerator kPWM\_Force\_LocalReload

The local reload signal from this submodule is used to force updates without regard to the state of LDOK

enumerator kPWM\_Force\_MasterReload

The master reload signal from submodule 0 is used to force updates if LDOK is set

enumerator kPWM\_Force\_LocalSync

The local sync signal from this submodule is used to force updates

enumerator kPWM\_Force\_MasterSync

The master sync signal from submodule0 is used to force updates

enumerator kPWM\_Force\_External

The external force signal, EXT\_FORCE, from outside the PWM module causes updates

enumerator kPWM\_Force\_ExternalSync

The external sync signal, EXT\_SYNC, from outside the PWM module causes updates



enum `_pwm_output_state`

PWM channel output status.

*Values:*

enumerator `kPWM_HighState`

The output state of PWM channel is high

enumerator `kPWM_LowState`

The output state of PWM channel is low

enumerator `kPWM_NormalState`

The output state of PWM channel is normal

enumerator `kPWM_InvertState`

The output state of PWM channel is invert

enumerator `kPWM_MaskState`

The output state of PWM channel is mask

enum `_pwm_init_source`

PWM counter initialization options.

*Values:*

enumerator `kPWM_Initialize_LocalSync`

Local sync causes initialization

enumerator `kPWM_Initialize_MasterReload`

Master reload from submodule 0 causes initialization

enumerator `kPWM_Initialize_MasterSync`

Master sync from submodule 0 causes initialization

enumerator `kPWM_Initialize_ExtSync`

EXT\_SYNC causes initialization

enum `_pwm_load_frequency`

PWM load frequency selection.

*Values:*

enumerator `kPWM_LoadEveryOportunity`

Every PWM opportunity

enumerator `kPWM_LoadEvery2Oportunity`

Every 2 PWM opportunities

enumerator `kPWM_LoadEvery3Oportunity`

Every 3 PWM opportunities

enumerator `kPWM_LoadEvery4Oportunity`

Every 4 PWM opportunities

enumerator `kPWM_LoadEvery5Oportunity`

Every 5 PWM opportunities

enumerator `kPWM_LoadEvery6Oportunity`

Every 6 PWM opportunities

enumerator `kPWM_LoadEvery7Oportunity`

Every 7 PWM opportunities

enumerator kPWM\_LoadEvery8Opportunity  
Every 8 PWM opportunities

enumerator kPWM\_LoadEvery9Opportunity  
Every 9 PWM opportunities

enumerator kPWM\_LoadEvery10Opportunity  
Every 10 PWM opportunities

enumerator kPWM\_LoadEvery11Opportunity  
Every 11 PWM opportunities

enumerator kPWM\_LoadEvery12Opportunity  
Every 12 PWM opportunities

enumerator kPWM\_LoadEvery13Opportunity  
Every 13 PWM opportunities

enumerator kPWM\_LoadEvery14Opportunity  
Every 14 PWM opportunities

enumerator kPWM\_LoadEvery15Opportunity  
Every 15 PWM opportunities

enumerator kPWM\_LoadEvery16Opportunity  
Every 16 PWM opportunities

enum \_pwm\_fault\_input  
List of PWM fault selections.

*Values:*

enumerator kPWM\_Fault\_0  
Fault 0 input pin

enumerator kPWM\_Fault\_1  
Fault 1 input pin

enumerator kPWM\_Fault\_2  
Fault 2 input pin

enumerator kPWM\_Fault\_3  
Fault 3 input pin

enum \_pwm\_fault\_disable  
List of PWM fault disable mapping selections.

*Values:*

enumerator kPWM\_FaultDisable\_0  
Fault 0 disable mapping

enumerator kPWM\_FaultDisable\_1  
Fault 1 disable mapping

enumerator kPWM\_FaultDisable\_2  
Fault 2 disable mapping

enumerator kPWM\_FaultDisable\_3  
Fault 3 disable mapping

enum \_pwm\_fault\_channels  
List of PWM fault channels.

*Values:*

enumerator kPWM\_faultchannel\_0

enum \_\_pwm\_input\_capture\_edge

PWM capture edge select.

*Values:*

enumerator kPWM\_Disable

Disabled

enumerator kPWM\_FallingEdge

Capture on falling edge only

enumerator kPWM\_RisingEdge

Capture on rising edge only

enumerator kPWM\_RiseAndFallEdge

Capture on rising or falling edge

enum \_\_pwm\_force\_signal

PWM output options when a FORCE\_OUT signal is asserted.

*Values:*

enumerator kPWM\_UsePwm

Generated PWM signal is used by the deadtime logic.

enumerator kPWM\_InvertedPwm

Inverted PWM signal is used by the deadtime logic.

enumerator kPWM\_SoftwareControl

Software controlled value is used by the deadtime logic.

enumerator kPWM\_UseExternal

PWM\_EXT\_A signal is used by the deadtime logic.

enum \_\_pwm\_chnl\_pair\_operation

Options available for the PWM A & B pair operation.

*Values:*

enumerator kPWM\_Independent

PWM A & PWM B operate as 2 independent channels

enumerator kPWM\_ComplementaryPwmA

PWM A & PWM B are complementary channels, PWM A generates the signal

enumerator kPWM\_ComplementaryPwmB

PWM A & PWM B are complementary channels, PWM B generates the signal

enum \_\_pwm\_register\_reload

Options available on how to load the buffered-registers with new values.

*Values:*

enumerator kPWM\_ReloadImmediate

Buffered-registers get loaded with new values as soon as LDOK bit is set

enumerator kPWM\_ReloadPwmHalfCycle

Registers loaded on a PWM half cycle

enumerator kPWM\_ReloadPwmFullCycle

Registers loaded on a PWM full cycle

enumerator kPWM\_ReloadPwmHalfAndFullCycle  
Registers loaded on a PWM half & full cycle

enum \_\_pwm\_fault\_recovery\_mode

Options available on how to re-enable the PWM output when recovering from a fault.

*Values:*

enumerator kPWM\_NoRecovery

PWM output will stay inactive

enumerator kPWM\_RecoverHalfCycle

PWM output re-enabled at the first half cycle

enumerator kPWM\_RecoverFullCycle

PWM output re-enabled at the first full cycle

enumerator kPWM\_RecoverHalfAndFullCycle

PWM output re-enabled at the first half or full cycle

enum \_\_pwm\_interrupt\_enable

List of PWM interrupt options.

*Values:*

enumerator kPWM\_CompareVal0InterruptEnable

PWM VAL0 compare interrupt

enumerator kPWM\_CompareVal1InterruptEnable

PWM VAL1 compare interrupt

enumerator kPWM\_CompareVal2InterruptEnable

PWM VAL2 compare interrupt

enumerator kPWM\_CompareVal3InterruptEnable

PWM VAL3 compare interrupt

enumerator kPWM\_CompareVal4InterruptEnable

PWM VAL4 compare interrupt

enumerator kPWM\_CompareVal5InterruptEnable

PWM VAL5 compare interrupt

enumerator kPWM\_CaptureX0InterruptEnable

PWM capture X0 interrupt

enumerator kPWM\_CaptureX1InterruptEnable

PWM capture X1 interrupt

enumerator kPWM\_CaptureB0InterruptEnable

PWM capture B0 interrupt

enumerator kPWM\_CaptureB1InterruptEnable

PWM capture B1 interrupt

enumerator kPWM\_CaptureA0InterruptEnable

PWM capture A0 interrupt

enumerator kPWM\_CaptureA1InterruptEnable

PWM capture A1 interrupt

enumerator kPWM\_ReloadInterruptEnable

PWM reload interrupt

enumerator kPWM\_\_ReloadErrorInterruptEnable

PWM reload error interrupt

enumerator kPWM\_\_Fault0InterruptEnable

PWM fault 0 interrupt

enumerator kPWM\_\_Fault1InterruptEnable

PWM fault 1 interrupt

enumerator kPWM\_\_Fault2InterruptEnable

PWM fault 2 interrupt

enumerator kPWM\_\_Fault3InterruptEnable

PWM fault 3 interrupt

enum \_pwm\_status\_flags

List of PWM status flags.

*Values:*

enumerator kPWM\_\_CompareVal0Flag

PWM VAL0 compare flag

enumerator kPWM\_\_CompareVal1Flag

PWM VAL1 compare flag

enumerator kPWM\_\_CompareVal2Flag

PWM VAL2 compare flag

enumerator kPWM\_\_CompareVal3Flag

PWM VAL3 compare flag

enumerator kPWM\_\_CompareVal4Flag

PWM VAL4 compare flag

enumerator kPWM\_\_CompareVal5Flag

PWM VAL5 compare flag

enumerator kPWM\_\_CaptureX0Flag

PWM capture X0 flag

enumerator kPWM\_\_CaptureX1Flag

PWM capture X1 flag

enumerator kPWM\_\_CaptureB0Flag

PWM capture B0 flag

enumerator kPWM\_\_CaptureB1Flag

PWM capture B1 flag

enumerator kPWM\_\_CaptureA0Flag

PWM capture A0 flag

enumerator kPWM\_\_CaptureA1Flag

PWM capture A1 flag

enumerator kPWM\_\_ReloadFlag

PWM reload flag

enumerator kPWM\_\_ReloadErrorFlag

PWM reload error flag

enumerator kPWM\_RegUpdatedFlag

PWM registers updated flag

enumerator kPWM\_Fault0Flag

PWM fault 0 flag

enumerator kPWM\_Fault1Flag

PWM fault 1 flag

enumerator kPWM\_Fault2Flag

PWM fault 2 flag

enumerator kPWM\_Fault3Flag

PWM fault 3 flag

enum \_pwm\_dma\_enable

List of PWM DMA options.

*Values:*

enumerator kPWM\_CaptureX0DMAEnable

PWM capture X0 DMA

enumerator kPWM\_CaptureX1DMAEnable

PWM capture X1 DMA

enumerator kPWM\_CaptureB0DMAEnable

PWM capture B0 DMA

enumerator kPWM\_CaptureB1DMAEnable

PWM capture B1 DMA

enumerator kPWM\_CaptureA0DMAEnable

PWM capture A0 DMA

enumerator kPWM\_CaptureA1DMAEnable

PWM capture A1 DMA

enum \_pwm\_dma\_source\_select

List of PWM capture DMA enable source select.

*Values:*

enumerator kPWM\_DMAResourceDisable

Read DMA requests disabled

enumerator kPWM\_DMAWatermarksEnable

Exceeding a FIFO watermark sets the DMA read request

enumerator kPWM\_DMALocalSync

A local sync (VAL1 matches counter) sets the read DMA request

enumerator kPWM\_DMALocalReload

A local reload (STS[RF] being set) sets the read DMA request

enum \_pwm\_watermark\_control

PWM FIFO Watermark AND Control.

*Values:*

enumerator kPWM\_FIFOWatermarksOR

Selected FIFO watermarks are OR'ed together

enumerator kPWM\_FIFOWatermarksAND  
Selected FIFO watermarks are AND'ed together

enum \_\_pwm\_mode

PWM operation mode.

*Values:*

enumerator kPWM\_SignedCenterAligned  
Signed center-aligned

enumerator kPWM\_CenterAligned  
Unsigned cente-aligned

enumerator kPWM\_SignedEdgeAligned  
Signed edge-aligned

enumerator kPWM\_EdgeAligned  
Unsigned edge-aligned

enum \_\_pwm\_level\_select

PWM output pulse mode, high-true or low-true.

*Values:*

enumerator kPWM\_HighTrue  
High level represents “on” or “active” state

enumerator kPWM\_LowTrue  
Low level represents “on” or “active” state

enum \_\_pwm\_fault\_state

PWM output fault status.

*Values:*

enumerator kPWM\_PwmFaultState0  
Output is forced to logic 0 state prior to consideration of output polarity control.

enumerator kPWM\_PwmFaultState1  
Output is forced to logic 1 state prior to consideration of output polarity control.

enumerator kPWM\_PwmFaultState2  
Output is tristated.

enumerator kPWM\_PwmFaultState3  
Output is tristated.

enum \_\_pwm\_reload\_source\_select

PWM reload source select.

*Values:*

enumerator kPWM\_LocalReload  
The local reload signal is used to reload registers

enumerator kPWM\_MasterReload  
The master reload signal (from submodule 0) is used to reload

enum \_\_pwm\_fault\_clear

PWM fault clearing options.

*Values:*

enumerator kPWM\_\_Automatic  
Automatic fault clearing

enumerator kPWM\_\_ManualNormal  
Manual fault clearing with no fault safety mode

enumerator kPWM\_\_ManualSafety  
Manual fault clearing with fault safety mode

enum \_\_pwm\_\_module\_\_control  
Options for submodule master control operation.  
*Values:*

enumerator kPWM\_\_Control\_\_Module\_\_0  
Control submodule 0's start/stop,buffer reload operation

enumerator kPWM\_\_Control\_\_Module\_\_1  
Control submodule 1's start/stop,buffer reload operation

enumerator kPWM\_\_Control\_\_Module\_\_2  
Control submodule 2's start/stop,buffer reload operation

enumerator kPWM\_\_Control\_\_Module\_\_3  
Control submodule 3's start/stop,buffer reload operation

typedef enum \_\_pwm\_\_submodule pwm\_submodule\_t  
List of PWM submodules.

typedef enum \_\_pwm\_\_channels pwm\_channels\_t  
List of PWM channels in each module.

typedef enum \_\_pwm\_\_value\_\_register pwm\_value\_register\_t  
List of PWM value registers.

typedef enum \_\_pwm\_\_clock\_\_source pwm\_clock\_source\_t  
PWM clock source selection.

typedef enum \_\_pwm\_\_clock\_\_prescale pwm\_clock\_prescale\_t  
PWM prescaler factor selection for clock source.

typedef enum \_\_pwm\_\_force\_\_output\_\_trigger pwm\_force\_output\_trigger\_t  
Options that can trigger a PWM FORCE\_OUT.

typedef enum \_\_pwm\_\_output\_\_state pwm\_output\_state\_t  
PWM channel output status.

typedef enum \_\_pwm\_\_init\_\_source pwm\_init\_source\_t  
PWM counter initialization options.

typedef enum \_\_pwm\_\_load\_\_frequency pwm\_load\_frequency\_t  
PWM load frequency selection.

typedef enum \_\_pwm\_\_fault\_\_input pwm\_fault\_input\_t  
List of PWM fault selections.

typedef enum \_\_pwm\_\_fault\_\_disable pwm\_fault\_disable\_t  
List of PWM fault disable mapping selections.

typedef enum \_\_pwm\_\_fault\_\_channels pwm\_fault\_channels\_t  
List of PWM fault channels.



typedef enum *\_pwm\_input\_capture\_edge* pwm\_input\_capture\_edge\_t  
PWM capture edge select.

typedef enum *\_pwm\_force\_signal* pwm\_force\_signal\_t  
PWM output options when a FORCE\_OUT signal is asserted.

typedef enum *\_pwm\_chnl\_pair\_operation* pwm\_chnl\_pair\_operation\_t  
Options available for the PWM A & B pair operation.

typedef enum *\_pwm\_register\_reload* pwm\_register\_reload\_t  
Options available on how to load the buffered-registers with new values.

typedef enum *\_pwm\_fault\_recovery\_mode* pwm\_fault\_recovery\_mode\_t  
Options available on how to re-enable the PWM output when recovering from a fault.

typedef enum *\_pwm\_interrupt\_enable* pwm\_interrupt\_enable\_t  
List of PWM interrupt options.

typedef enum *\_pwm\_status\_flags* pwm\_status\_flags\_t  
List of PWM status flags.

typedef enum *\_pwm\_dma\_enable* pwm\_dma\_enable\_t  
List of PWM DMA options.

typedef enum *\_pwm\_dma\_source\_select* pwm\_dma\_source\_select\_t  
List of PWM capture DMA enable source select.

typedef enum *\_pwm\_watermark\_control* pwm\_watermark\_control\_t  
PWM FIFO Watermark AND Control.

typedef enum *\_pwm\_mode* pwm\_mode\_t  
PWM operation mode.

typedef enum *\_pwm\_level\_select* pwm\_level\_select\_t  
PWM output pulse mode, high-true or low-true.

typedef enum *\_pwm\_fault\_state* pwm\_fault\_state\_t  
PWM output fault status.

typedef enum *\_pwm\_reload\_source\_select* pwm\_reload\_source\_select\_t  
PWM reload source select.

typedef enum *\_pwm\_fault\_clear* pwm\_fault\_clear\_t  
PWM fault clearing options.

typedef enum *\_pwm\_module\_control* pwm\_module\_control\_t  
Options for submodule master control operation.

typedef struct *\_pwm\_signal\_param* pwm\_signal\_param\_t  
Structure for the user to define the PWM signal characteristics.

typedef struct *\_pwm\_config* pwm\_config\_t  
PWM config structure.

This structure holds the configuration settings for the PWM peripheral. To initialize this structure to reasonable defaults, call the PWM\_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

typedef struct *\_pwm\_fault\_input\_filter\_param* pwm\_fault\_input\_filter\_param\_t  
Structure for the user to configure the fault input filter.

```
typedef struct _pwm_fault_param pwm_fault_param_t
```

Structure is used to hold the parameters to configure a PWM fault.

```
typedef struct _pwm_input_capture_param pwm_input_capture_param_t
```

Structure is used to hold parameters to configure the capture capability of a signal pin.

```
void PWM_SetupInputCapture(PWM_Type *base, pwm_submodule_t subModule,
                          pwm_channels_t pwmChannel, const
                          pwm_input_capture_param_t *inputCaptureParams)
```

Sets up the PWM input capture.

Each PWM submodule has 3 pins that can be configured for use as input capture pins. This function sets up the capture parameters for each pin and enables the pin for input capture operation.

#### Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – Channel in the submodule to setup
- inputCaptureParams – Parameters passed in to set up the input pin

```
void PWM_SetupFaultInputFilter(PWM_Type *base, const pwm_fault_input_filter_param_t
                              *faultInputFilterParams)
```

Sets up the PWM fault channel 0 input filter.

#### Parameters

- base – PWM peripheral base address
- faultInputFilterParams – Parameters passed in to set up the fault input filter.

```
void PWM_SetupFaultInputFilterExt(PWM_Type *base, pwm_fault_channels_t faultChannel,
                                 const pwm_fault_input_filter_param_t
                                 *faultInputFilterParams)
```

Sets up the PWM fault input filter.

#### Parameters

- base – PWM peripheral base address
- faultChannel – PWM fault channel to configure.
- faultInputFilterParams – Parameters passed in to set up the fault input filter.

```
void PWM_SetupFaults(PWM_Type *base, pwm_fault_input_t faultNum, const
                    pwm_fault_param_t *faultParams)
```

Sets up the PWM fault channel 0 protection.

#### Parameters

- base – PWM peripheral base address
- faultNum – PWM fault to configure.
- faultParams – Pointer to the PWM fault config structure

```
void PWM_SetupFaultsExt(PWM_Type *base, pwm_fault_channels_t faultChannel,
                       pwm_fault_input_t faultNum, const pwm_fault_param_t
                       *faultParams)
```

Sets up the PWM fault protection.

#### Parameters

- base – PWM peripheral base address

- `faultChannel` – PWM fault channel to configure.
- `faultNum` – PWM fault to configure.
- `faultParams` – Pointer to the PWM fault config structure

`void PWM_FaultDefaultConfig(pwm_fault_param_t *config)`

Fill in the PWM fault config struct with the default settings.

The default values are:

```
config->faultClearingMode = kPWM_Automatic;
config->faultLevel = false;
config->enableCombinationalPath = true;
config->recoverMode = kPWM_NoRecovery;
```

### Parameters

- `config` – Pointer to user's PWM fault config structure.

`void PWM_SetupForceSignal(PWM_Type *base, pwm_submodule_t subModule, pwm_channels_t pwmChannel, pwm_force_signal_t mode)`

Selects the signal to output on a PWM pin when a FORCE\_OUT signal is asserted.

The user specifies which channel to configure by supplying the submodule number and whether to modify PWM A or PWM B within that submodule.

### Parameters

- `base` – PWM peripheral base address
- `subModule` – PWM submodule to configure
- `pwmChannel` – Channel to configure
- `mode` – Signal to output when a FORCE\_OUT is triggered

`static inline void PWM_SetVALxValue(PWM_Type *base, pwm_submodule_t subModule, pwm_value_register_t valueRegister, uint16_t value)`

Set the PWM VALx registers.

This function allows the user to write value into VAL registers directly. And it will destroying the PWM clock period set by the PWM\_SetupPwm()/PWM\_SetupPwmPhaseShift() functions. Due to VALx registers are buffered, the new value will not active unless call PWM\_SetPwmLdok() and the reload point is reached.

### Parameters

- `base` – PWM peripheral base address
- `subModule` – PWM submodule to configure
- `valueRegister` – VALx register that will be written new value
- `value` – Value that will be written into VALx register

`static inline uint16_t PWM_GetVALxValue(PWM_Type *base, pwm_submodule_t subModule, pwm_value_register_t valueRegister)`

Get the PWM VALx registers.

### Parameters

- `base` – PWM peripheral base address
- `subModule` – PWM submodule to configure
- `valueRegister` – VALx register that will be read value

### Returns

The VALx register value

```
static inline void PWM_OutputTriggerEnable(PWM_Type *base, pwm_submodule_t subModule,  
                                           pwm_value_register_t valueRegister, bool activate)
```

Enables or disables the PWM output trigger.

This function allows the user to enable or disable the PWM trigger. The PWM has 2 triggers. Trigger 0 is activated when the counter matches VAL 0, VAL 2, or VAL 4 register. Trigger 1 is activated when the counter matches VAL 1, VAL 3, or VAL 5 register.

#### Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- valueRegister – Value register that will activate the trigger
- activate – true: Enable the trigger; false: Disable the trigger

```
static inline void PWM_ActivateOutputTrigger(PWM_Type *base, pwm_submodule_t subModule,  
                                             uint16_t valueRegisterMask)
```

Enables the PWM output trigger.

This function allows the user to enable one or more (VAL0-5) PWM trigger.

#### Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- valueRegisterMask – Value register mask that will activate one or more (VAL0-5) trigger enumeration `_pwm_value_register_mask`

```
static inline void PWM_DeactivateOutputTrigger(PWM_Type *base, pwm_submodule_t  
                                              subModule, uint16_t valueRegisterMask)
```

Disables the PWM output trigger.

This function allows the user to disables one or more (VAL0-5) PWM trigger.

#### Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- valueRegisterMask – Value register mask that will Deactivate one or more (VAL0-5) trigger enumeration `_pwm_value_register_mask`

```
static inline void PWM_SetupSwCtrlOut(PWM_Type *base, pwm_submodule_t subModule,  
                                       pwm_channels_t pwmChannel, bool value)
```

Sets the software control output for a pin to high or low.

The user specifies which channel to modify by supplying the submodule number and whether to modify PWM A or PWM B within that submodule.

#### Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – Channel to configure
- value – true: Supply a logic 1, false: Supply a logic 0.

```
static inline void PWM_SetPwmLdok(PWM_Type *base, uint8_t subModulesToUpdate, bool  
                                  value)
```

Sets or clears the PWM LDOK bit on a single or multiple submodules.

Set LDOK bit to load buffered values into CTRL[PRSC] and the INIT, FRACVAL and VAL registers. The values are loaded immediately if kPWM\_ReloadImmediate option was chosen during config. Else the values are loaded at the next PWM reload point. This function can issue the load command to multiple submodules at the same time.

#### Parameters

- base – PWM peripheral base address
- subModulesToUpdate – PWM submodules to update with buffered values. This is a logical OR of members of the enumeration `pwm_module_control_t`
- value – true: Set LDOK bit for the submodule list; false: Clear LDOK bit

```
static inline void PWM_SetPwmFaultState(PWM_Type *base, pwm_submodule_t subModule,  
                                         pwm_channels_t pwmChannel, pwm_fault_state_t  
                                         faultState)
```

Set PWM output fault status.

These bits determine the fault state for the PWM\_A output in fault conditions and STOP mode. It may also define the output state in WAIT and DEBUG modes depending on the settings of CTRL2[WAITEN] and CTRL2[DBGEN]. This function can update PWM output fault status.

#### Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – Channel to configure
- faultState – PWM output fault status

```
static inline void PWM_SetupFaultDisableMap(PWM_Type *base, pwm_submodule_t subModule,  
                                             pwm_channels_t pwmChannel,  
                                             pwm_fault_channels_t pwm_fault_channels,  
                                             uint16_t value)
```

Set PWM fault disable mapping.

Each of the four bits of this read/write field is one-to-one associated with the four FAULTx inputs of fault channel 0/1. The PWM output will be turned off if there is a logic 1 on an FAULTx input and a 1 in the corresponding bit of this field. A reset sets all bits in this field.

#### Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- `pwm_fault_channels` – PWM fault channel to configure
- value – Fault disable mapping mask value enumeration `pwm_fault_disable_t`

```
static inline void PWM_OutputEnable(PWM_Type *base, pwm_channels_t pwmChannel,  
                                     pwm_submodule_t subModule)
```

Set PWM output enable.

This feature allows the user to enable the PWM Output. Recommend to invoke this API after PWM and fault configuration. But invoke this API before configure MCTRL register is okay, such as set LDOK or start timer.

#### Parameters

- base – PWM peripheral base address

- pwmChannel – PWM channel to configure
- subModule – PWM submodule to configure

```
static inline void PWM_OutputDisable(PWM_Type *base, pwm_channels_t pwmChannel,  
                                     pwm_submodule_t subModule)
```

Set PWM output disable.

This feature allows the user to disable the PWM output. Recommend to invoke this API after PWM and fault configuration. But invoke this API before configure MCTRL register is okay, such as set LDOK or start timer.

#### Parameters

- base – PWM peripheral base address
- pwmChannel – PWM channel to configure
- subModule – PWM submodule to configure

```
uint8_t PWM_GetPwmChannelState(PWM_Type *base, pwm_submodule_t subModule,  
                               pwm_channels_t pwmChannel)
```

Get the dutycycle value.

#### Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure

#### Returns

Current channel dutycycle value.

```
status_t PWM_SetOutputToIdle(PWM_Type *base, pwm_channels_t pwmChannel,  
                             pwm_submodule_t subModule, bool idleStatus)
```

Set PWM output in idle status (high or low).

---

**Note:** This API should call after PWM\_SetupPwm() APIs, and PWMX submodule is not supported.

---

#### Parameters

- base – PWM peripheral base address
- pwmChannel – PWM channel to configure
- subModule – PWM submodule to configure
- idleStatus – True: PWM output is high in idle status; false: PWM output is low in idle status.

#### Returns

kStatus\_Fail if there was error setting up the signal; kStatus\_Success if set output idle success

```
void PWM_SetClockMode(PWM_Type *base, pwm_submodule_t subModule,  
                     pwm_clock_prescale_t prescaler)
```

Set the pwm submodule prescaler.

#### Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure

- prescaler – Set prescaler value

```
void PWM_SetPwmForceOutputToZero(PWM_Type *base, pwm_submodule_t subModule,  
                                pwm_channels_t pwmChannel, bool forcetozero)
```

This function enables-disables the forcing of the output of a given eFlexPwm channel to logic 0.

#### Parameters

- base – PWM peripheral base address
- pwmChannel – PWM channel to configure
- subModule – PWM submodule to configure
- forcetozero – True: Enable the pwm force output to zero; False: Disable the pwm output resumes normal function.

```
void PWM_SetChannelOutput(PWM_Type *base, pwm_submodule_t subModule,  
                          pwm_channels_t pwmChannel, pwm_output_state_t outputstate)
```

This function set the output state of the PWM pin as requested for the current cycle.

#### Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- outputstate – Set pwm output state, see *pwm\_output\_state\_t*.

```
status_t PWM_SetPhaseDelay(PWM_Type *base, pwm_channels_t pwmChannel,  
                           pwm_submodule_t subModule, uint16_t delayCycles)
```

This function set the phase delay from the master sync signal of submodule 0.

#### Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- delayCycles – Number of cycles delayed from submodule 0.

#### Returns

kStatus\_Fail if the number of delay cycles is set larger than the period defined in submodule 0; kStatus\_Success if set phase delay success

```
static inline void PWM_SetFilterSampleCount(PWM_Type *base, pwm_channels_t pwmChannel,  
                                           pwm_submodule_t subModule, uint8_t  
                                           filterSampleCount)
```

This function set the number of consecutive samples that must agree prior to the input filter.

#### Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- filterSampleCount – Number of consecutive samples.

```
static inline void PWM_SetFilterSamplePeriod(PWM_Type *base, pwm_channels_t pwmChannel,  
                                           pwm_submodule_t subModule, uint8_t  
                                           filterSamplePeriod)
```

This function set the sampling period of the fault pin input filter.

**Parameters**

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- filterSamplePeriod – Sampling period of input filter.

PWM\_SUBMODULE\_SWCONTROL\_WIDTH

Number of bits per submodule for software output control

PWM\_SUBMODULE\_CHANNEL

Submodule channels include PWMA, PWMB, PWMX.

struct \_\_pwm\_signal\_param

*#include <fsl\_pwm.h>* Structure for the user to define the PWM signal characteristics.

**Public Members**

*pwm\_channels\_t* pwmChannel

PWM channel being configured; PWM A or PWM B

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)

*pwm\_level\_select\_t* level

PWM output active level select

uint16\_t deadtimeValue

The deadtime value; only used if channel pair is operating in complementary mode

*pwm\_fault\_state\_t* faultState

PWM output fault status

bool pwmchannelenable

Enable PWM output

struct \_\_pwm\_config

*#include <fsl\_pwm.h>* PWM config structure.

This structure holds the configuration settings for the PWM peripheral. To initialize this structure to reasonable defaults, call the PWM\_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 enableDebugMode

true: PWM continues to run in debug mode; false: PWM is paused in debug mode

*pwm\_init\_source\_t* initializationControl

Option to initialize the counter

*pwm\_clock\_source\_t* clockSource

Clock source for the counter

*pwm\_clock\_prescale\_t* prescale

Pre-scaler to divide down the clock



*pwm\_chnl\_pair\_operation\_t* pairOperation

Channel pair in independent or complementary mode

*pwm\_register\_reload\_t* reloadLogic

PWM Reload logic setup

*pwm\_reload\_source\_select\_t* reloadSelect

Reload source select

*pwm\_load\_frequency\_t* reloadFrequency

Specifies when to reload, used when user's choice is not immediate reload

*pwm\_force\_output\_trigger\_t* forceTrigger

Specify which signal will trigger a FORCE\_OUT

struct *\_pwm\_fault\_input\_filter\_param*

*#include <fsl\_pwm.h>* Structure for the user to configure the fault input filter.

### Public Members

uint8\_t faultFilterCount

Fault filter count

uint8\_t faultFilterPeriod

Fault filter period; value of 0 will bypass the filter

bool faultGlitchStretch

Fault Glitch Stretch Enable: A logic 1 means that input fault signals will be stretched to at least 2 IPBus clock cycles

struct *\_pwm\_fault\_param*

*#include <fsl\_pwm.h>* Structure is used to hold the parameters to configure a PWM fault.

### Public Members

*pwm\_fault\_clear\_t* faultClearingMode

Fault clearing mode to use

bool faultLevel

true: Logic 1 indicates fault; false: Logic 0 indicates fault

bool enableCombinationalPath

true: Combinational Path from fault input is enabled; false: No combination path is available

*pwm\_fault\_recovery\_mode\_t* recoverMode

Specify when to re-enable the PWM output

struct *\_pwm\_input\_capture\_param*

*#include <fsl\_pwm.h>* Structure is used to hold parameters to configure the capture capability of a signal pin.

### Public Members

bool captureInputSel

true: Use the edge counter signal as source false: Use the raw input signal from the pin as source

uint8\_t edgeCompareValue

Compare value, used only if edge counter is used as source

pwm\_input\_capture\_edge\_t edge0

Specify which edge causes a capture for input circuitry 0

pwm\_input\_capture\_edge\_t edge1

Specify which edge causes a capture for input circuitry 1

bool enableOneShotCapture

true: Use one-shot capture mode; false: Use free-running capture mode

uint8\_t fifoWatermark

Watermark level for capture FIFO. The capture flags in the status register will set if the word count in the FIFO is greater than this watermark level

## 2.47 Reset Driver

enum \_SYSCON\_RSTn

Enumeration for peripheral reset control bits.

Defines the enumeration for peripheral reset control bits in PRESETCTRL/ASYNCPRESETCTRL registers

*Values:*

enumerator kINPUTMUX0\_RST\_SHIFT\_RSTn

INPUTMUX0 reset control

enumerator kI3C0\_RST\_SHIFT\_RSTn

I3C0 reset control

enumerator kCTIMER0\_RST\_SHIFT\_RSTn

CTIMER0 reset control

enumerator kCTIMER1\_RST\_SHIFT\_RSTn

CTIMER1 reset control

enumerator kCTIMER2\_RST\_SHIFT\_RSTn

CTIMER2 reset control

enumerator kFREQME\_RST\_SHIFT\_RSTn

FREQME reset control

enumerator kUTICK0\_RST\_SHIFT\_RSTn

UTICK0 reset control

enumerator kDMA\_RST\_SHIFT\_RSTn

DMA reset control

enumerator kAOI0\_RST\_SHIFT\_RSTn

AOI0 reset control

enumerator kCRC\_RST\_SHIFT\_RSTn

CRC reset control

enumerator kEIM\_RST\_SHIFT\_RSTn

EIM reset control

enumerator kERM\_RST\_SHIFT\_RSTn

ERM reset control

enumerator kLPI2C0\_RST\_SHIFT\_RSTn  
LPI2C0 reset control

enumerator kLPSPi0\_RST\_SHIFT\_RSTn  
LPSPi0 reset control

enumerator kLPSPi1\_RST\_SHIFT\_RSTn  
LPSPi1 reset control

enumerator kLPUART0\_RST\_SHIFT\_RSTn  
LPUART0 reset control

enumerator kLPUART1\_RST\_SHIFT\_RSTn  
LPUART1 reset control

enumerator kLPUART2\_RST\_SHIFT\_RSTn  
LPUART2 reset control

enumerator kUSB0\_RST\_SHIFT\_RSTn  
USB0 reset control

enumerator kQDC0\_RST\_SHIFT\_RSTn  
QDC0 reset control

enumerator kFLEXPWM0\_RST\_SHIFT\_RSTn  
FLEXPWM0 reset control

enumerator kOSTIMER0\_RST\_SHIFT\_RSTn  
OSTIMER0 reset control

enumerator kADC0\_RST\_SHIFT\_RSTn  
ADC0 reset control

enumerator kCMP1\_RST\_SHIFT\_RSTn  
CMP1 reset control

enumerator kPORT0\_RST\_SHIFT\_RSTn  
PORT0 reset control

enumerator kPORT1\_RST\_SHIFT\_RSTn  
PORT1 reset control

enumerator kPORT2\_RST\_SHIFT\_RSTn  
PORT2 reset control

enumerator kPORT3\_RST\_SHIFT\_RSTn  
PORT3 reset control

enumerator kATX0\_RST\_SHIFT\_RSTn  
ATX0 reset control

enumerator kGPIO0\_RST\_SHIFT\_RSTn  
GPIO0 reset control

enumerator kGPIO1\_RST\_SHIFT\_RSTn  
GPIO1 reset control

enumerator kGPIO2\_RST\_SHIFT\_RSTn  
GPIO2 reset control

enumerator kGPIO3\_RST\_SHIFT\_RSTn  
GPIO3 reset control

enumerator NotAvail\_RSTn

No reset control

typedef enum \_SYSCON\_RSTn SYSCON\_RSTn\_t

Enumeration for peripheral reset control bits.

Defines the enumeration for peripheral reset control bits in PRESETCTRL/ASYNCPRESETCTRL registers

typedef SYSCON\_RSTn\_t reset\_ip\_name\_t

void RESET\_SetPeripheralReset(*reset\_ip\_name\_t* peripheral)

Assert reset to peripheral.

Asserts reset signal to specified peripheral module.

#### Parameters

- peripheral – Assert reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

void RESET\_ClearPeripheralReset(*reset\_ip\_name\_t* peripheral)

Clear reset to peripheral.

Clears reset signal to specified peripheral module, allows it to operate.

#### Parameters

- peripheral – Clear reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

void RESET\_PeripheralReset(*reset\_ip\_name\_t* peripheral)

Reset peripheral module.

Reset peripheral module.

#### Parameters

- peripheral – Peripheral to reset. The enum argument contains encoding of reset register and reset bit position in the reset register.

static inline void RESET\_ReleasePeripheralReset(*reset\_ip\_name\_t* peripheral)

Release peripheral module.

Release peripheral module.

#### Parameters

- peripheral – Peripheral to release. The enum argument contains encoding of reset register and reset bit position in the reset register.

FSL\_RESET\_DRIVER\_VERSION

reset driver version 2.4.0

AOI\_RSTS

Array initializers with peripheral reset bits

ADC\_RSTS

CRC\_RSTS

CTIMER\_RSTS

DMA\_RSTS\_N

FLEXPWM\_RSTS\_N

FREQME\_RSTS\_N  
GPIO\_RSTS\_N  
I3C\_RSTS  
INPUTMUX\_RSTS  
LPUART\_RSTS  
LPSPI\_RSTS  
LPI2C\_RSTS  
LPCMP\_RSTS  
OSTIMER\_RSTS  
PORT\_RSTS\_N  
EQDC\_RSTS  
UTICK\_RSTS  
EIM\_RSTS\_N  
ERM\_RSTS\_N

## 2.48 ROMAPI Driver

Flash driver status codes.

*Values:*

enumerator kStatus\_FLASH\_Success

API is executed successfully

enumerator kStatus\_FLASH\_InvalidArgument

Invalid argument

enumerator kStatus\_FLASH\_SizeError

Error size

enumerator kStatus\_FLASH\_AlignmentError

Parameter is not aligned with the specified baseline

enumerator kStatus\_FLASH\_AddressError

Address is out of range

enumerator kStatus\_FLASH\_AccessError

Invalid instruction codes and out-of bound addresses

enumerator kStatus\_FLASH\_ProtectionViolation

The program/erase operation is requested to execute on protected areas

enumerator kStatus\_FLASH\_CommandFailure

Run-time error during command execution.

enumerator kStatus\_FLASH\_UnknownProperty

Unknown property.

enumerator kStatus\_FLASH\_EraseKeyError

API erase key is invalid.

enum \_flash\_property\_tag

Enumeration for various flash properties.

*Values:*

enumerator kFLASH\_PropertyPflashSectorSize

Pflash sector size property.

enumerator kFLASH\_PropertyPflashTotalSize

Pflash total size property.

enumerator kFLASH\_PropertyPflashBlockSize

Pflash block size property.

enumerator kFLASH\_PropertyPflashBlockCount

Pflash block count property.

enumerator kFLASH\_PropertyPflashBlockBaseAddr

Pflash block base address property.

enumerator kFLASH\_PropertyPflashPageSize

Pflash page size property.

enumerator kFLASH\_PropertyPflashSystemFreq

System Frequency property.

enumerator kFLASH\_PropertyFfrSectorSize

FFR sector size property.

enumerator kFLASH\_PropertyFfrTotalSize

FFR total size property.

enumerator kFLASH\_PropertyFfrBlockBaseAddr

FFR block base address property.

enumerator kFLASH\_PropertyFfrPageSize

FFR page size property.

enum \_flash\_driver\_api\_keys

Enumeration for Flash 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 kFLASH\_ApiEraseKey

Key value used to validate all flash erase APIs.

typedef enum \_flash\_property\_tag flash\_property\_tag\_t

Enumeration for various flash properties.

typedef struct \_flash\_ffr\_config flash\_ffr\_config\_t

Flash controller paramter config.

`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.

`typedef struct _flash_driver_interface flash_driver_interface_t`

Interface for the flash driver.

`typedef struct _bootloader_tree bootloader_tree_t`

Root of the bootloader API tree.

An instance of this struct resides in read-only memory in the bootloader. It provides a user application access to APIs exported by the bootloader.

`kStatusGroupGeneric`

Flash driver status group.

`kStatusGroupFlashDriver`

`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.

`ROM_API_BASE`

ROM API base address

`ROM_API`

ROM API base pointer

`FLASH_API`

FLASH API base pointer

`static inline 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.

`static inline status_t FLASH_EraseSector(flash_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. NOTE: The start address need to be 4 Bytes-aligned.
- `lengthInBytes` – The length, given in bytes need be 4 Bytes-aligned.
- `key` – The value used to validate all flash erase APIs.

`static inline status_t FLASH_ProgramPhrase(flash_config_t *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)`

Programs flash phrases 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.

```
static inline status_t FLASH_ProgramPage(flash_config_t *config, uint32_t start, uint8_t *src,  
                                         uint32_t lengthInBytes)
```

Programs flash page 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.

```
static inline status_t FLASH_VerifyProgram(flash_config_t *config, uint32_t start, uint32_t  
                                           lengthInBytes, const uint8_t *expectedData,  
                                           uint32_t *failedAddress, uint32_t *failedData)
```

Verifies programming of the desired flash area.

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.
- `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.



```
static inline status_t FLASH_VerifyErasePhrase(flash_config_t *config, uint32_t start, uint32_t  
lengthInBytes)
```

Verify that the flash phrases are erased.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased

#### 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.

```
static inline status_t FLASH_VerifyErasePage(flash_config_t *config, uint32_t start, uint32_t  
lengthInBytes)
```

Verify that the flash pages are erased.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased

#### 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.

```
static inline status_t FLASH_VerifyEraseSector(flash_config_t *config, uint32_t start, uint32_t  
lengthInBytes)
```

Verify that the flash sectors are erased.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased

#### 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.

```
static inline 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.

```
static inline status_t IFR_VerifyErasePhrase(flash_config_t *config, uint32_t start, uint32_t  
lengthInBytes)
```

Verify that the IFR0 phrases are erased.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased

**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.

```
static inline status_t IFR_VerifyErasePage(flash_config_t *config, uint32_t start, uint32_t  
lengthInBytes)
```

Verify that the IFR0 pages are erased.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased

**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.

```
static inline status_t IFR_VerifyEraseSector(flash_config_t *config, uint32_t start, uint32_t  
lengthInBytes)
```

Verify that the IFR0 sectors are erased.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased

**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.

```
static inline status_t FLASH_Read(flash_config_t *config, uint32_t start, uint8_t *dest, uint32_t  
lengthInBytes)
```

Reads flash at locations passed in through parameters.

This function read the flash memory from 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 read.
- `dest` – A pointer to the dest buffer of data that is to be read from the flash.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be read.

```
static inline uint32_t ROMAPI_GetVersion(void)
```

Get ROM API version.

This function read the ROM API version.

static inline void ROMAPI\_RunBootloader(void \*arg)

Run the Bootloader API to force into the ISP mode base on the user arg.

**Parameters**

- arg – Indicates API prototype fields definition. Refer to the above user\_app\_boot\_invoke\_option\_t structure

static inline void ROMAPI\_GetUUID(uint8\_t \*uuid)

Get the UUID.

**Parameters**

- uuid – UUID data array

FSL\_ROMAPI\_DRIVER\_VERSION

romapi driver version 2.0.1.

uint32\_t ffrBlockBase

uint32\_t ffrTotalSize

uint32\_t ffrPageSize

uint32\_t sectorSize

uint32\_t cfpaPageVersion

uint32\_t cfpaPageOffset

uint32\_t PFlashBlockBase

A base address of the first PFlash block

uint32\_t PFlashTotalSize

The size of the combined PFlash block.

uint32\_t PFlashBlockCount

A number of PFlash blocks.

uint32\_t PFlashPageSize

The size in bytes of a page of PFlash.

uint32\_t PFlashSectorSize

The size in bytes of a sector of PFlash.

*flash\_ffr\_config\_t* ffrConfig

*status\_t* (\*flash\_init)(*flash\_config\_t* \*config)

*status\_t* (\*flash\_erase\_sector)(*flash\_config\_t* \*config, uint32\_t start, uint32\_t lengthInBytes, uint32\_t key)

*status\_t* (\*flash\_program\_phrase)(*flash\_config\_t* \*config, uint32\_t start, uint8\_t \*src, uint32\_t lengthInBytes)

*status\_t* (\*flash\_program\_page)(*flash\_config\_t* \*config, uint32\_t start, uint8\_t \*src, uint32\_t lengthInBytes)

*status\_t* (\*flash\_verify\_program)(*flash\_config\_t* \*config, uint32\_t start, uint32\_t lengthInBytes, const uint8\_t \*expectedData, uint32\_t \*failedAddress, uint32\_t \*failedData)

*status\_t* (\*flash\_verify\_erase\_phrase)(*flash\_config\_t* \*config, uint32\_t start, uint32\_t lengthInBytes)

```

status_t (*flash_verify_erase_page)(flash_config_t *config, uint32_t start, uint32_t lengthInBytes)
status_t (*flash_verify_erase_sector)(flash_config_t *config, uint32_t start, uint32_t lengthInBytes)
status_t (*flash_get_property)(flash_config_t *config, flash_property_tag_t whichProperty,
uint32_t *value)
status_t (*ifr_verify_erase_phrase)(flash_config_t *config, uint32_t start, uint32_t lengthInBytes)
status_t (*ifr_verify_erase_page)(flash_config_t *config, uint32_t start, uint32_t lengthInBytes)
status_t (*ifr_verify_erase_sector)(flash_config_t *config, uint32_t start, uint32_t lengthInBytes)
status_t (*flash_read)(flash_config_t *config, uint32_t start, uint8_t *dest, uint32_t lengthInBytes)
uint32_t version
uint32_t reserved
uint32_t boot_image_index
uint32_t instance
uint32_t boot_interface
uint32_t mode
uint32_t tag
struct user_app_boot_invoke_option_t B
uint32_t U
union user_app_boot_invoke_option_t option
void (*run_bootloader)(void *arg)
    Function to start the bootloader executing.
const flash_driver_interface_t *flash_driver
    Internal Flash driver API.
void (*jump)(void *arg)
FSL_COMPONENT_ID
struct _flash_ffr_config
    #include <fsl_romapi.h> Flash controller paramter config.
struct _flash_config
    #include <fsl_romapi.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.
struct _flash_driver_interface
    #include <fsl_romapi.h> Interface for the flash driver.
struct user_app_boot_invoke_option_t
    #include <fsl_romapi.h>
struct _bootloader_tree
    #include <fsl_romapi.h> Root of the bootloader API tree.
    An instance of this struct resides in read-only memory in the bootloader. It provides a user
    application access to APIs exported by the bootloader.
union option

```

**Public Members**

struct *user\_app\_boot\_invoke\_option\_t* B

uint32\_t U

struct B

## 2.49 TRDC: Trusted Resource Domain Controller

void TRDC\_Init(TRDC\_Type \*base)

Initializes the TRDC module.

This function enables the TRDC clock.

**Parameters**

- base – TRDC peripheral base address.

void TRDC\_Deinit(TRDC\_Type \*base)

De-initializes the TRDC module.

This function disables the TRDC clock.

**Parameters**

- base – TRDC peripheral base address.

static inline uint8\_t TRDC\_GetCurrentMasterDomainId(TRDC\_Type \*base)

Gets the domain ID of the current bus master.

**Parameters**

- base – TRDC peripheral base address.

**Returns**

Domain ID of current bus master.

void TRDC\_GetHardwareConfig(TRDC\_Type \*base, *trdc\_hardware\_config\_t* \*config)

Gets the TRDC hardware configuration.

This function gets the TRDC hardware configurations, including number of bus masters, number of domains, number of MRCs and number of PACs.

**Parameters**

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

static inline void TRDC\_SetDacGlobalValid(TRDC\_Type \*base)

Sets the TRDC DAC(Domain Assignment Controllers) global valid.

Once enabled, it will remain enabled until next reset.

**Parameters**

- base – TRDC peripheral base address.

static inline void TRDC\_LockMasterDomainAssignment(TRDC\_Type \*base, uint8\_t master, uint8\_t regNum)

Locks the bus master domain assignment register.

This function locks the master domain assignment. After it is locked, the register can't be changed until next reset.

**Parameters**

- base – TRDC peripheral base address.
- master – Which master to configure, refer to `trdcx_master_t` in processor header file, x is trdc instance.
- regNum – Which register to configure, processor master can have more than one register for the MDAC configuration.
- assignIndex – Which assignment register to lock.

```
static inline void TRDC_SetMasterDomainAssignmentValid(TRDC_Type *base, uint8_t master,
                                                       uint8_t regNum, bool valid)
```

Sets the master domain assignment as valid or invalid.

This function sets the master domain assignment as valid or invalid.

#### Parameters

- base – TRDC peripheral base address.
- master – Which master to configure.
- regNum – Which register to configure, processor master can have more than one register for the MDAC configuration.
- assignIndex – Index for the domain assignment register.
- valid – True to set valid, false to set invalid.

```
void TRDC_GetDefaultProcessorDomainAssignment(trdc_processor_domain_assignment_t
                                              *domainAssignment)
```

Gets the default master domain assignment for the processor bus master.

This function gets the default master domain assignment for the processor bus master. It should only be used for the processor bus masters, such as CORE0. This function sets the assignment as follows:

```
assignment->domainId      = 0U;
assignment->domainIdSelect = kTRDC_DidMda;
assignment->lock           = 0U;
```

#### Parameters

- domainAssignment – Pointer to the assignment structure.

```
void TRDC_GetDefaultNonProcessorDomainAssignment(trdc_non_processor_domain_assignment_t
                                                  *domainAssignment)
```

Gets the default master domain assignment for non-processor bus master.

This function gets the default master domain assignment for non-processor bus master. It should only be used for the non-processor bus masters, such as DMA. This function sets the assignment as follows:

```
assignment->domainId      = 0U;
assignment->privilegeAttr  = kTRDC_ForceUser;
assignment->secureAttr    = kTRDC_ForceSecure;
assignment->bypassDomainId = 0U;
assignment->lock           = 0U;
```

#### Parameters

- domainAssignment – Pointer to the assignment structure.

```
void TRDC_SetProcessorDomainAssignment(TRDC_Type *base, uint8_t master, uint8_t regNum,
                                       const trdc_processor_domain_assignment_t
                                       *domainAssignment)
```

Sets the processor bus master domain assignment.

This function sets the processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter `assignIndex` specifies which assignment register to set.

Example: Set domain assignment for core 0.

```
trdc_processor_domain_assignment_t processorAssignment;

TRDC_GetDefaultProcessorDomainAssignment(&processorAssignment);

processorAssignment.domainId = 0;
processorAssignment.xxx      = xxx;
TRDC_SetMasterDomainAssignment(TRDC, &processorAssignment);
```

### Parameters

- `base` – TRDC peripheral base address.
- `master` – Which master to configure, refer to `trdc_master_t` in processor header file.
- `regNum` – Which register to configure, processor master can have more than one register for the MDAC configuration.
- `domainAssignment` – Pointer to the assignment structure.

```
void TRDC_SetNonProcessorDomainAssignment(TRDC_Type *base, uint8_t master, const
                                         trdc_non_processor_domain_assignment_t
                                         *domainAssignment)
```

Sets the non-processor bus master domain assignment.

This function sets the non-processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter `assignIndex` specifies which assignment register to set.

Example: Set domain assignment for DMA0.

```
trdc_non_processor_domain_assignment_t nonProcessorAssignment;

TRDC_GetDefaultNonProcessorDomainAssignment(&nonProcessorAssignment);
nonProcessorAssignment.domainId = 1;
nonProcessorAssignment.xxx      = xxx;

TRDC_SetMasterDomainAssignment(TRDC, kTrdcMasterDma0, 0U, &nonProcessorAssignment);
```

### Parameters

- `base` – TRDC peripheral base address.
- `master` – Which master to configure, refer to `trdc_master_t` in processor header file.
- `domainAssignment` – Pointer to the assignment structure.

```
static inline uint64_t TRDC_GetActiveMasterPidMap(TRDC_Type *base)
```

Gets the bit map of the bus master(s) that is(are) sourcing a PID register.

This function sets the non-processor master domain assignment as valid.

### Parameters

- `base` – TRDC peripheral base address.

**Returns**

the bit map of the master(s). Bit 1 sets indicates bus master 1.

```
void TRDC_SetPid(TRDC_Type *base, uint8_t master, const trdc_pid_config_t *pidConfig)
```

Sets the current Process identifier(PID) for processor core.

Each processor has a corresponding process identifier (PID) which can be used to group tasks into different domains. Secure privileged software saves and restores the PID as part of any context switch. This data structure defines an array of 32-bit values, one per MDA module, that define the PID. Since this register resource is only applicable to processor cores, the data structure is typically sparsely populated. The HWCFG[2-3] registers provide a bitmap of the implemented PIDn registers. This data structure is indexed using the corresponding MDA instance number. Depending on the operating clock domain of each DAC instance, there may be optional information stored in the corresponding PIDm register to properly implement the LK2 = 2 functionality.

**Parameters**

- base – TRDC peripheral base address.
- master – Which processor master to configure, refer to *trdc\_master\_t* in processor header file.
- pidConfig – Pointer to the configuration structure.

```
void TRDC_GetDefaultIDAUConfig(trdc_idau_config_t *idauConfiguration)
```

Gets the default IDAU(Implementation-Defined Attribution Unit) configuration.

```
config->lockSecureVTOR    = false;
config->lockNonsecureVTOR = false;
config->lockSecureMPU     = false;
config->lockNonsecureMPU  = false;
config->lockSAU           = false;
```

**Parameters**

- domainAssignment – Pointer to the configuration structure.

```
void TRDC_SetIDAU(TRDC_Type *base, const trdc_idau_config_t *idauConfiguration)
```

Sets the IDAU(Implementation-Defined Attribution Unit) control configuration.

Example: Lock the secure and non-secure MPU registers.

```
trdc_idau_config_t idauConfiguration;

TRDC_GetDefaultIDAUConfig(&idauConfiguration);

idauConfiguration.lockSecureMPU = true;
idauConfiguration.lockNonsecureMPU = true;
TRDC_SetIDAU(TRDC, &idauConfiguration);
```

**Parameters**

- base – TRDC peripheral base address.
- domainAssignment – Pointer to the configuration structure.

```
static inline void TRDC_EnableFlashLogicalWindow(TRDC_Type *base, bool enable)
```

Enables/disables the FLW(flash logical window) function.

**Parameters**

- base – TRDC peripheral base address.
- enable – True to enable, false to disable.



static inline void TRDC\_LockFlashLogicalWindow(TRDC\_Type \*base)

Locks FLW registers. Once locked the registers can not be updated until next reset.

#### Parameters

- base – TRDC peripheral base address.

static inline uint32\_t TRDC\_GetFlashLogicalWindowPbase(TRDC\_Type \*base)

Gets the FLW physical base address.

#### Parameters

- base – TRDC peripheral base address.

#### Returns

Physical address of the FLW function.

static inline void TRDC\_GetSetFlashLogicalWindowSize(TRDC\_Type \*base, uint16\_t size)

Sets the FLW size.

#### Parameters

- base – TRDC peripheral base address.
- size – Size of the FLW in unit of 32k bytes.

void TRDC\_GetDefaultFlashLogicalWindowConfig(*trdc\_flw\_config\_t* \*flwConfiguration)

Gets the default FLW(Flash Logical Window) configuration.

```
config->blockCount = false;
config->arrayBaseAddr = false;
config->lock = false;
config->enable = false;
```

#### Parameters

- flwConfiguration – Pointer to the configuration structure.

void TRDC\_SetFlashLogicalWindow(TRDC\_Type \*base, const *trdc\_flw\_config\_t* \*flwConfiguration)

Sets the FLW function's configuration.

```
trdc_flw_config_t flwConfiguration;

TRDC_GetDefaultIDAUConfig(&flwConfiguration);

flwConfiguration.blockCount = 32U;
flwConfiguration.arrayBaseAddr = 0xFFFFFFFF;
TRDC_SetIDAU(TRDC, &flwConfiguration);
```

#### Parameters

- base – TRDC peripheral base address.
- flwConfiguration – Pointer to the configuration structure.

*status\_t* TRDC\_GetAndClearFirstDomainError(TRDC\_Type \*base, *trdc\_domain\_error\_t* \*error)

Gets and clears the first domain error of the current domain.

This function gets the first access violation information for the current domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

#### Parameters

- base – TRDC peripheral base address.

- error – Pointer to the error information.

### Returns

If the access violation is captured, this function returns the `kStatus_Success`. The error information can be obtained from the parameter `error`. If no access violation is captured, this function returns the `kStatus_NoData`.

```
status_t TRDC_GetAndClearFirstSpecificDomainError(TRDC_Type *base, trdc_domain_error_t
                                                  *error, uint8_t domainId)
```

Gets and clears the first domain error of the specific domain.

This function gets the first access violation information for the specific domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

### Parameters

- base – TRDC peripheral base address.
- error – Pointer to the error information.
- domainId – The error of which domain to get and clear.

### Returns

If the access violation is captured, this function returns the `kStatus_Success`. The error information can be obtained from the parameter `error`. If no access violation is captured, this function returns the `kStatus_NoData`.

```
static inline void TRDC_SetMrcGlobalValid(TRDC_Type *base)
```

Sets the TRDC MRC(Memory Region Checkers) global valid.

Once enabled, it will remain enabled until next reset.

### Parameters

- base – TRDC peripheral base address.

```
static inline uint8_t TRDC_GetMrcRegionNumber(TRDC_Type *base, uint8_t mrcIdx)
```

Gets the TRDC MRC(Memory Region Checkers) region number valid.

### Parameters

- base – TRDC peripheral base address.

### Returns

the region number of the given MRC instance

```
void TRDC_MrcSetMemoryAccessConfig(TRDC_Type *base, const
                                   trdc_memory_access_control_config_t *config, uint8_t
                                   mrcIdx, uint8_t regIdx)
```

Sets the memory access configuration for one of the access control register of one MRC.

Example: Enable the secure operations and lock the configuration for MRC0 region 1.

```
trdc_memory_access_control_config_t config;

config.securePrivX = true;
config.securePrivW = true;
config.securePrivR = true;
config.lock = true;
TRDC_SetMrcMemoryAccess(TRDC, &config, 0, 1);
```

### Parameters

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

- mrcIdx – MRC index.
- regIdx – Register number.

```
void TRDC__MrcEnableDomainNseUpdate(TRDC_Type *base, uint8_t mrcIdx, uint16_t  
                                   domianMask, bool enable)
```

Enables the update of the selected domians.

After the domians' update are enabled, their regions' NSE bits can be set or clear.

#### Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- domianMask – Bit mask of the domains to be enabled.
- enable – True to enable, false to disable.

```
void TRDC__MrcRegionNseSet(TRDC_Type *base, uint8_t mrcIdx, uint16_t regionMask)
```

Sets the NSE bits of the selected regions for domains.

This function sets the NSE bits for the selected regions for the domains whose update are enabled.

#### Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- regionMask – Bit mask of the regions whose NSE bits to set.

```
void TRDC__MrcRegionNseClear(TRDC_Type *base, uint8_t mrcIdx, uint16_t regionMask)
```

Clears the NSE bits of the selected regions for domains.

This function clears the NSE bits for the selected regions for the domains whose update are enabled.

#### Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- regionMask – Bit mask of the regions whose NSE bits to clear.

```
void TRDC__MrcDomainNseClear(TRDC_Type *base, uint8_t mrcIdx, uint16_t domainMask)
```

Clears the NSE bits for all the regions of the selected domains.

This function clears the NSE bits for all regions of selected domains whose update are enabled.

#### Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- domainMask – Bit mask of the domians whose NSE bits to clear.

```
void TRDC__MrcSetRegionDescriptorConfig(TRDC_Type *base, const  
                                       trdc_mrc_region_descriptor_config_t *config)
```

Sets the configuration for one of the region descriptor per domain per MRC instnce.

This function sets the configuration for one of the region descriptor, including the start and end address of the region, memory access control policy and valid.

#### Parameters

- base – TRDC peripheral base address.

- config – Pointer to region descriptor configuration structure.

static inline void TRDC\_SetMbcGlobalValid(TRDC\_Type \*base)

Sets the TRDC MBC(Memory Block Checkers) global valid.

Once enabled, it will remain enabled until next reset.

#### Parameters

- base – TRDC peripheral base address.

void TRDC\_GetMbcHardwareConfig(TRDC\_Type \*base, *trdc\_slave\_memory\_hardware\_config\_t* \*config, uint8\_t mbcIdx, uint8\_t slvIdx)

Gets the hardware configuration of the one of two slave memories within each MBC(memory block checker).

#### Parameters

- base – TRDC peripheral base address.
- config – Pointer to the structure to get the configuration.
- mbcIdx – MBC number.
- slvIdx – Slave number.

void TRDC\_MbcSetNseUpdateConfig(TRDC\_Type \*base, const *trdc\_mbc\_nse\_update\_config\_t* \*config, uint8\_t mbcIdx)

Sets the NSR update configuration for one of the MBC instance.

After set the NSE configuration, the configured memory area can be updateby NSE set/clear.

#### Parameters

- base – TRDC peripheral base address.
- config – Pointer to NSE update configuration structure.
- mbcIdx – MBC index.

void TRDC\_MbcWordNseSet(TRDC\_Type \*base, uint8\_t mbcIdx, uint32\_t bitMask)

Sets the NSE bits of the selected configuration words according to NSE update configuration.

This function sets the NSE bits of the word for the configured regio, memory.

#### Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- bitMask – Mask of the bits whose NSE bits to set.

void TRDC\_MbcWordNseClear(TRDC\_Type \*base, uint8\_t mbcIdx, uint32\_t bitMask)

Clears the NSE bits of the selected configuration words according to NSE update configuration.

This function sets the NSE bits of the word for the configured regio, memory.

#### Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- bitMask – Mask of the bits whose NSE bits to clear.

void TRDC\_MbcNseClearAll(TRDC\_Type \*base, uint8\_t mbcIdx, uint16\_t domainMask, uint8\_t slave)

Clears all configuration words' NSE bits of the selected domain and memory.

### Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- domainMask – Mask of the domains whose NSE bits to clear, 0b110 means clear domain 1&2.
- slaveMask – Mask of the slaves whose NSE bits to clear, 0x11 means clear all slave 0&1's NSE bits.

```
void TRDC_MbcSetMemoryAccessConfig(TRDC_Type *base, const
                                   trdc_memory_access_control_config_t *config, uint8_t
                                   mbcIdx, uint8_t rgdIdx)
```

Sets the memory access configuration for one of the region descriptor of one MBC.

Example: Enable the secure operations and lock the configuration for MRC0 region 1.

```
trdc_memory_access_control_config_t config;

config.securePrivX = true;
config.securePrivW = true;
config.securePrivR = true;
config.lock = true;
TRDC_SetMbcMemoryAccess(TRDC, &config, 0, 1);
```

### Parameters

- base – TRDC peripheral base address.
- config – Pointer to the configuration structure.
- mbcIdx – MBC index.
- rgdIdx – Region descriptor number.

```
void TRDC_MbcSetMemoryBlockConfig(TRDC_Type *base, const
                                   trdc_mbc_memory_block_config_t *config)
```

Sets the configuration for one of the memory block per domain per MBC instnce.

This function sets the configuration for one of the memory block, including the memory access control policy and nse enable.

### Parameters

- base – TRDC peripheral base address.
- config – Pointer to memory block configuration structure.

```
enum _trdc_did_sel
```

TRDC domain ID select method, the register bit TRDC\_MDA\_W0\_0\_DFMT0[DIDS], used for domain hit evaluation.

*Values:*

enumerator kTRDC\_DidMda

Use MDAn[2:0] as DID.

enumerator kTRDC\_DidInput

Use the input DID (DID\_in) as DID.

enumerator kTRDC\_DidMdaAndInput

Use MDAn[2] concatenated with DID\_in[1:0] as DID.

enumerator kTRDC\_DidReserved

Reserved.

enum \_trdc\_secure\_attr

TRDC secure attribute, the register bit TRDC\_MDA\_W0\_0\_DFMT0[SA], used for bus master domain assignment.

*Values:*

enumerator kTRDC\_ForceSecure

Force the bus attribute for this master to secure.

enumerator kTRDC\_ForceNonSecure

Force the bus attribute for this master to non-secure.

enumerator kTRDC\_MasterSecure

Use the bus master's secure/nonsecure attribute directly.

enumerator kTRDC\_MasterSecure1

Use the bus master's secure/nonsecure attribute directly.

enum \_trdc\_pid\_domain\_hit\_config

The configuration of domain hit evaluation of PID.

*Values:*

enumerator kTRDC\_pidDomainHitNone0

No PID is included in the domain hit evaluation.

enumerator kTRDC\_pidDomainHitNone1

No PID is included in the domain hit evaluation.

enumerator kTRDC\_pidDomainHitInclusive

The PID is included in the domain hit evaluation when (PID & ~PIDM).

enumerator kTRDC\_pidDomainHitExclusive

The PID is included in the domain hit evaluation when ~(PID & ~PIDM).

enum \_trdc\_privilege\_attr

TRDC privileged attribute, the register bit TRDC\_MDA\_W0\_x\_DFMT1[PA], used for non-processor bus master domain assignment.

*Values:*

enumerator kTRDC\_ForceUser

Force the bus attribute for this master to user.

enumerator kTRDC\_ForcePrivilege

Force the bus attribute for this master to privileged.

enumerator kTRDC\_MasterPrivilege

Use the bus master's attribute directly.

enumerator kTRDC\_MasterPrivilege1

Use the bus master's attribute directly.

enum \_trdc\_pid\_lock

PID lock configuration.

*Values:*

enumerator kTRDC\_PidUnlocked0

The PID value can be updated by any secure privileged write.

enumerator kTRDC\_PidUnlocked1

The PID value can be updated by any secure privileged write.

enumerator kTRDC\_PidUnlocked2

The PID value can be updated by any secure privileged write from the bus master that first configured this register.

enumerator kTRDC\_PidLocked

The PID value is locked until next reset.

enum \_trdc\_controller

TRDC controller definition for domain error check. Each TRDC instance may have different MRC or MBC count, call TRDC\_GetHardwareConfig to get the actual count.

*Values:*

enumerator kTRDC\_MemBlockController0

Memory block checker 0.

enumerator kTRDC\_MemBlockController1

Memory block checker 1.

enumerator kTRDC\_MemBlockController2

Memory block checker 2.

enumerator kTRDC\_MemBlockController3

Memory block checker 3.

enumerator kTRDC\_MemRegionChecker0

Memory region checker 0.

enumerator kTRDC\_MemRegionChecker1

Memory region checker 1.

enumerator kTRDC\_MemRegionChecker2

Memory region checker 2.

enumerator kTRDC\_MemRegionChecker3

Memory region checker 3.

enumerator kTRDC\_MemRegionChecker4

Memory region checker 4.

enumerator kTRDC\_MemRegionChecker5

Memory region checker 5.

enumerator kTRDC\_MemRegionChecker6

Memory region checker 6.

enum \_trdc\_error\_state

|                                    |                        |    |
|------------------------------------|------------------------|----|
| TRDC domain error state definition | TRDC_MBCn_DERR_W1[EST] | or |
| TRDC_MRCn_DERR_W1[EST].            |                        |    |

*Values:*

enumerator kTRDC\_ErrorStateNone

No access violation detected.

enumerator kTRDC\_ErrorStateNone1

No access violation detected.

enumerator kTRDC\_ErrorStateSingle

Single access violation detected.

enumerator kTRDC\_ErrorStateMulti

Multiple access violation detected.

enum \_trdc\_error\_attr

TRDC domain error attribute definition TRDC\_MBCn\_DERR\_W1[EATR] or TRDC\_MRCn\_DERR\_W1[EATR].

*Values:*

enumerator kTRDC\_ErrorSecureUserInst  
Secure user mode, instruction fetch access.

enumerator kTRDC\_ErrorSecureUserData  
Secure user mode, data access.

enumerator kTRDC\_ErrorSecurePrivilegeInst  
Secure privileged mode, instruction fetch access.

enumerator kTRDC\_ErrorSecurePrivilegeData  
Secure privileged mode, data access.

enumerator kTRDC\_ErrorNonSecureUserInst  
NonSecure user mode, instruction fetch access.

enumerator kTRDC\_ErrorNonSecureUserData  
NonSecure user mode, data access.

enumerator kTRDC\_ErrorNonSecurePrivilegeInst  
NonSecure privileged mode, instruction fetch access.

enumerator kTRDC\_ErrorNonSecurePrivilegeData  
NonSecure privileged mode, data access.

enum \_trdc\_error\_type

TRDC domain error access type definition TRDC\_DERR\_W1\_n[ERW].

*Values:*

enumerator kTRDC\_ErrorTypeRead  
Error occurs on read reference.

enumerator kTRDC\_ErrorTypeWrite  
Error occurs on write reference.

enum \_trdc\_region\_descriptor

The region descriptor enumeration, used to form a mask to set/clear the NSE bits for one or several regions.

*Values:*

enumerator kTRDC\_RegionDescriptor0  
Region descriptor 0.

enumerator kTRDC\_RegionDescriptor1  
Region descriptor 1.

enumerator kTRDC\_RegionDescriptor2  
Region descriptor 2.

enumerator kTRDC\_RegionDescriptor3  
Region descriptor 3.

enumerator kTRDC\_RegionDescriptor4  
Region descriptor 4.

enumerator kTRDC\_RegionDescriptor5  
Region descriptor 5.



enumerator kTRDC\_RegionDescriptor6

Region descriptor 6.

enumerator kTRDC\_RegionDescriptor7

Region descriptor 7.

enumerator kTRDC\_RegionDescriptor8

Region descriptor 8.

enumerator kTRDC\_RegionDescriptor9

Region descriptor 9.

enumerator kTRDC\_RegionDescriptor10

Region descriptor 10.

enumerator kTRDC\_RegionDescriptor11

Region descriptor 11.

enumerator kTRDC\_RegionDescriptor12

Region descriptor 12.

enumerator kTRDC\_RegionDescriptor13

Region descriptor 13.

enumerator kTRDC\_RegionDescriptor14

Region descriptor 14.

enumerator kTRDC\_RegionDescriptor15

Region descriptor 15.

enum \_trdc\_MRC\_domain

The MRC domain enumeration, used to form a mask to enable/disable the update or clear all NSE bits of one or several domains.

*Values:*

enumerator kTRDC\_MrcDomain0

Domain 0.

enumerator kTRDC\_MrcDomain1

Domain 1.

enumerator kTRDC\_MrcDomain2

Domain 2.

enumerator kTRDC\_MrcDomain3

Domain 3.

enumerator kTRDC\_MrcDomain4

Domain 4.

enumerator kTRDC\_MrcDomain5

Domain 5.

enumerator kTRDC\_MrcDomain6

Domain 6.

enumerator kTRDC\_MrcDomain7

Domain 7.

enumerator kTRDC\_MrcDomain8

Domain 8.

enumerator kTRDC\_MrcDomain9  
Domain 9.

enumerator kTRDC\_MrcDomain10  
Domain 10.

enumerator kTRDC\_MrcDomain11  
Domain 11.

enumerator kTRDC\_MrcDomain12  
Domain 12.

enumerator kTRDC\_MrcDomain13  
Domain 13.

enumerator kTRDC\_MrcDomain14  
Domain 14.

enumerator kTRDC\_MrcDomain15  
Domain 15.

enum \_trdc\_MBC\_domain

The MBC domain enumeration, used to form a mask to enable/disable the update or clear NSE bits of one or several domains.

*Values:*

enumerator kTRDC\_MbcDomain0  
Domain 0.

enumerator kTRDC\_MbcDomain1  
Domain 1.

enumerator kTRDC\_MbcDomain2  
Domain 2.

enumerator kTRDC\_MbcDomain3  
Domain 3.

enumerator kTRDC\_MbcDomain4  
Domain 4.

enumerator kTRDC\_MbcDomain5  
Domain 5.

enumerator kTRDC\_MbcDomain6  
Domain 6.

enumerator kTRDC\_MbcDomain7  
Domain 7.

enum \_trdc\_MBC\_memory

The MBC slave memory enumeration, used to form a mask to enable/disable the update or clear NSE bits of one or several memory block.

*Values:*

enumerator kTRDC\_MbcSlaveMemory0  
Memory 0.

enumerator kTRDC\_MbcSlaveMemory1  
Memory 1.

enumerator kTRDC\_MbcSlaveMemory2

Memory 2.

enumerator kTRDC\_MbcSlaveMemory3

Memory 3.

enum \_trdc\_MBC\_bit

The MBC bit enumeration, used to form a mask to set/clear configured words' NSE.

*Values:*

enumerator kTRDC\_MbcBit0

Bit 0.

enumerator kTRDC\_MbcBit1

Bit 1.

enumerator kTRDC\_MbcBit2

Bit 2.

enumerator kTRDC\_MbcBit3

Bit 3.

enumerator kTRDC\_MbcBit4

Bit 4.

enumerator kTRDC\_MbcBit5

Bit 5.

enumerator kTRDC\_MbcBit6

Bit 6.

enumerator kTRDC\_MbcBit7

Bit 7.

enumerator kTRDC\_MbcBit8

Bit 8.

enumerator kTRDC\_MbcBit9

Bit 9.

enumerator kTRDC\_MbcBit10

Bit 10.

enumerator kTRDC\_MbcBit11

Bit 11.

enumerator kTRDC\_MbcBit12

Bit 12.

enumerator kTRDC\_MbcBit13

Bit 13.

enumerator kTRDC\_MbcBit14

Bit 14.

enumerator kTRDC\_MbcBit15

Bit 15.

enumerator kTRDC\_MbcBit16

Bit 16.

enumerator kTRDC\_MbcBit17

Bit 17.

enumerator kTRDC\_MbcBit18

Bit 18.

enumerator kTRDC\_MbcBit19

Bit 19.

enumerator kTRDC\_MbcBit20

Bit 20.

enumerator kTRDC\_MbcBit21

Bit 21.

enumerator kTRDC\_MbcBit22

Bit 22.

enumerator kTRDC\_MbcBit23

Bit 23.

enumerator kTRDC\_MbcBit24

Bit 24.

enumerator kTRDC\_MbcBit25

Bit 25.

enumerator kTRDC\_MbcBit26

Bit 26.

enumerator kTRDC\_MbcBit27

Bit 27.

enumerator kTRDC\_MbcBit28

Bit 28.

enumerator kTRDC\_MbcBit29

Bit 29.

enumerator kTRDC\_MbcBit30

Bit 30.

enumerator kTRDC\_MbcBit31

Bit 31.

typedef struct *\_trdc\_hardware\_config* trdc\_hardware\_config\_t

TRDC hardware configuration.

typedef struct *\_trdc\_slave\_memory\_hardware\_config* trdc\_slave\_memory\_hardware\_config\_t

Hardware configuration of the two slave memories within each MBC(memory block checker).

typedef enum *\_trdc\_did\_sel* trdc\_did\_sel\_t

TRDC domain ID select method, the register bit TRDC\_MDA\_W0\_0\_DFMT0[DIDS], used for domain hit evaluation.

typedef enum *\_trdc\_secure\_attr* trdc\_secure\_attr\_t

TRDC secure attribute, the register bit TRDC\_MDA\_W0\_0\_DFMT0[SA], used for bus master domain assignment.

typedef enum *\_trdc\_pid\_domain\_hit\_config* trdc\_pid\_domain\_hit\_config\_t

The configuration of domain hit evaluation of PID.

```
typedef struct _trdc_processor_domain_assignment trdc_processor_domain_assignment_t
    Domain assignment for the processor bus master.

typedef enum _trdc_privilege_attr trdc_privilege_attr_t
    TRDC privileged attribute, the register bit TRDC_MDA_W0_x_DFMT1[PA], used for non-
    processor bus master domain assignment.

typedef struct _trdc_non_processor_domain_assignment
trdc_non_processor_domain_assignment_t
    Domain assignment for the non-processor bus master.

typedef enum _trdc_pid_lock trdc_pid_lock_t
    PID lock configuration.

typedef struct _trdc_pid_config trdc_pid_config_t
    Process identifier(PID) configuration for processor cores.

typedef struct _trdc_idau_config trdc_idau_config_t
    IDAU(Implementation-Defined Attribution Unit) configuration for TZ-M function control.

typedef struct _trdc_flw_config trdc_flw_config_t
    FLW(Flash Logical Window) configuration.

typedef enum _trdc_controller trdc_controller_t
    TRDC controller definition for domain error check. Each TRDC instance may have different
    MRC or MBC count, call TRDC_GetHardwareConfig to get the actual count.

typedef enum _trdc_error_state trdc_error_state_t
    TRDC domain error state definition TRDC_MBCn_DERR_W1[EST] or
    TRDC_MRCn_DERR_W1[EST].

typedef enum _trdc_error_attr trdc_error_attr_t
    TRDC domain error attribute definition TRDC_MBCn_DERR_W1[EATR] or
    TRDC_MRCn_DERR_W1[EATR].

typedef enum _trdc_error_type trdc_error_type_t
    TRDC domain error access type definition TRDC_DERR_W1_n[ERW].

typedef struct _trdc_domain_error trdc_domain_error_t
    TRDC domain error definition.

typedef struct _trdc_memory_access_control_config trdc_memory_access_control_config_t
    Memory access control configuration for MBC/MRC.

typedef struct _trdc_mrc_region_descriptor_config trdc_mrc_region_descriptor_config_t
    The configuration of each region descriptor per domain per MRC instance.

typedef struct _trdc_mbc_nse_update_config trdc_mbc_nse_update_config_t
    The configuration of MBC NSE update.

typedef struct _trdc_mbc_memory_block_config trdc_mbc_memory_block_config_t
    The configuration of each memory block per domain per MBC instance.

FSL_TRDC_DRIVER_VERSION

struct _trdc_hardware_config
    #include <fsl_trdc.h> TRDC hardware configuration.
```

### Public Members

uint8\_t masterNumber  
Number of bus masters.

uint8\_t domainNumber  
Number of domains.

uint8\_t mbcNumber  
Number of MBCs.

uint8\_t mrcNumber  
Number of MRCs.

struct \_trdc\_slave\_memory\_hardware\_config  
*#include <fsl\_trdc.h>* Hardware configuration of the two slave memories within each MBC(memory block checker).

### Public Members

uint32\_t blockNum  
Number of blocks.

uint32\_t blockSize  
Block size.

struct \_trdc\_processor\_domain\_assignment  
*#include <fsl\_trdc.h>* Domain assignment for the processor bus master.

### Public Members

uint32\_t domainId  
Domain ID.

uint32\_t domainIdSelect  
Domain ID select method, see trdc\_did\_sel\_t.

uint32\_t pidDomainHitConfig  
The configuration of the domain hit evaluation for PID, see trdc\_pid\_domain\_hit\_config\_t.

uint32\_t pidMask  
The mask combined with PID, so multiple PID can be included as part of the domain hit determination. Set to 0 to disable.

uint32\_t secureAttr  
Secure attribute, see trdc\_secure\_attr\_t.

uint32\_t pid  
The process identifier, combined with pidMask to form the domain hit determination.

uint32\_t \_\_pad0\_\_  
Reserved.

uint32\_t lock  
Lock the register.

uint32\_t \_\_pad1\_\_  
Reserved.

struct \_trdc\_non\_processor\_domain\_assignment  
*#include <fsl\_trdc.h>* Domain assignment for the non-processor bus master.

### Public Members

uint32\_t domainId

Domain ID.

uint32\_t privilegeAttr

Privileged attribute, see trdc\_privilege\_attr\_t.

uint32\_t secureAttr

Secure attribute, see trdc\_secure\_attr\_t.

uint32\_t bypassDomainId

Bypass domain ID.

uint32\_t \_\_pad0\_\_

Reserved.

uint32\_t lock

Lock the register.

uint32\_t \_\_pad1\_\_

Reserved.

struct \_trdc\_pid\_config

*#include <fsl\_trdc.h>* Process identifier(PID) configuration for processor cores.

### Public Members

uint32\_t pid

The process identifier of the executing task. The highest bit can be used to define secure/nonsecure attribute of the task.

uint32\_t \_\_pad0\_\_

Reserved.

uint32\_t lock

How to lock the register, see trdc\_pid\_lock\_t.

uint32\_t \_\_pad1\_\_

Reserved.

struct \_trdc\_idau\_config

*#include <fsl\_trdc.h>* IDAU(Implementation-Defined Attribution Unit) configuration for TZ-M function control.

### Public Members

uint32\_t \_\_pad0\_\_

Reserved.

uint32\_t lockSecureVTOR

Disable writes to secure VTOR(Vector Table Offset Register).

uint32\_t lockNonsecureVTOR

Disable writes to non-secure VTOR, Application interrupt and Reset Control Registers.

uint32\_t lockSecureMPU

Disable writes to secure MPU(Memory Protection Unit) from software or from a debug agent connected to the processor in Secure state.

uint32\_t lockNonsecureMPU

Disable writes to non-secure MPU(Memory Protection Unit) from software or from a debug agent connected to the processor.

uint32\_t lockSAU

Disable writes to SAU(Security Attribution Unit) registers.

uint32\_t \_\_pad1\_\_

Reserved.

struct \_\_trdc\_flw\_config

#include <fsl\_trdc.h> FLW(Flash Logical Window) configuration.

### Public Members

uint16\_t blockCount

Block count of the Flash Logic Window in 32KByte blocks.

uint32\_t arrayBaseAddr

Flash array base address of the Flash Logical Window.

bool lock

Disable writes to FLW registers.

bool enable

Enable FLW function.

struct \_\_trdc\_domain\_error

#include <fsl\_trdc.h> TRDC domain error definition.

### Public Members

trdc\_controller\_t controller

Which controller captured access violation.

uint32\_t address

Access address that generated access violation.

trdc\_error\_state\_t errorState

Error state.

trdc\_error\_attr\_t errorAttr

Error attribute.

trdc\_error\_type\_t errorType

Error type.

uint8\_t errorPort

Error port.

uint8\_t domainId

Domain ID.

uint8\_t slaveMemoryIdx

The slave memory index. Only apply when violation in MBC.

struct \_\_trdc\_memory\_access\_control\_config

#include <fsl\_trdc.h> Memory access control configuration for MBC/MRC.



### Public Members

uint32\_t nonsecureUsrX

Allow nonsecure user execute access.

uint32\_t nonsecureUsrW

Allow nonsecure user write access.

uint32\_t nonsecureUsrR

Allow nonsecure user read access.

uint32\_t \_\_pad0\_\_

Reserved.

uint32\_t nonsecurePrivX

Allow nonsecure privilege execute access.

uint32\_t nonsecurePrivW

Allow nonsecure privilege write access.

uint32\_t nonsecurePrivR

Allow nonsecure privilege read access.

uint32\_t \_\_pad1\_\_

Reserved.

uint32\_t secureUsrX

Allow secure user execute access.

uint32\_t secureUsrW

Allow secure user write access.

uint32\_t secureUsrR

Allow secure user read access.

uint32\_t \_\_pad2\_\_

Reserved.

uint32\_t securePrivX

Allownonsecure privilege execute access.

uint32\_t securePrivW

Allownonsecure privilege write access.

uint32\_t securePrivR

Allownonsecure privilege read access.

uint32\_t \_\_pad3\_\_

Reserved.

uint32\_t lock

Lock the configuration until next reset, only apply to access control register 0.

struct \_trdc\_mrc\_region\_descriptor\_config

*#include <fsl\_trdc.h>* The configuration of each region descriptor per domain per MRC instance.

### Public Members

uint8\_t memoryAccessControlSelect

Select one of the 8 access control policies for this region, for access cotrol policies see trdc\_memory\_access\_control\_config\_t.

`uint32_t startAddr`  
Physical start address.

`bool valid`  
Lock the register.

`bool nseEnable`  
Enable non-secure accesses and disable secure accesses.

`uint32_t endAddr`  
Physical start address.

`uint8_t mrcIdx`  
The index of the MRC for this configuration to take effect.

`uint8_t domainIdx`  
The index of the domain for this configuration to take effect.

`uint8_t regionIdx`  
The index of the region for this configuration to take effect.

`struct _trdc_mbc_nse_update_config`  
*#include <fsl\_trdc.h>* The configuration of MBC NSE update.

### Public Members

`uint32_t __pad0__`  
Reserved.

`uint32_t wordIdx`  
MBC configuration word index to be updated.

`uint32_t __pad1__`  
Reserved.

`uint32_t memorySelect`  
Bit mask of the selected memory to be updated. `_trdc_MBC_memory`.

`uint32_t __pad2__`  
Reserved.

`uint32_t domainSelect`  
Bit mask of the selected domain to be updated. `_trdc_MBC_domain`.

`uint32_t __pad3__`  
Reserved.

`uint32_t autoIncrement`  
Whether to increment the word index after current word is updated using this configuration.

`struct _trdc_mbc_memory_block_config`  
*#include <fsl\_trdc.h>* The configuration of each memory block per domain per MBC instance.

### Public Members

`uint32_t memoryAccessControlSelect`  
Select one of the 8 access control policies for this memory block, for access control policies see `trdc_memory_access_control_config_t`.

uint32\_t nseEnable

Enable non-secure accesses and disable secure accesses.

uint32\_t mbcIdx

The index of the MBC for this configuration to take effect.

uint32\_t domainIdx

The index of the domain for this configuration to take effect.

uint32\_t slaveMemoryIdx

The index of the slave memory for this configuration to take effect.

uint32\_t memoryBlockIdx

The index of the memory block for this configuration to take effect.

## 2.50 Trdc\_core

typedef struct *\_TRDC\_General\_Type* TRDC\_General\_Type

TRDC general configuration register definition.

typedef struct *\_TRDC\_FLW\_Type* TRDC\_FLW\_Type

TRDC flash logical control register definition.

typedef struct *\_TRDC\_DomainError\_Type* TRDC\_DomainError\_Type

TRDC domain error register definition.

typedef struct *\_TRDC\_DomainAssignment\_Type* TRDC\_DomainAssignment\_Type

TRDC master domain assignment register definition.

typedef struct *\_TRDC\_MBC\_Type* TRDC\_MBC\_Type

TRDC MBC control register definition.

typedef struct *\_TRDC\_MRC\_Type* TRDC\_MRC\_Type

TRDC MRC control register definition. MRC\_DOM0\_RGD\_W[region][word].

TRDC\_GENERAL\_BASE(base)

TRDC base address convert macro.

TRDC\_FLW\_BASE(base)

TRDC\_DOMAIN\_ERROR\_BASE(base)

TRDC\_DOMAIN\_ASSIGNMENT\_BASE(base)

TRDC\_MBC\_BASE(base, instance)

TRDC\_MRC\_BASE(base, instance)

struct *\_TRDC\_General\_Type*

*#include <fsl\_trdc\_core.h>* TRDC general configuration register definition.

### Public Members

\_\_IO uint32\_t TRDC\_CR

TRDC Register, offset: 0x0

\_\_I uint32\_t TRDC\_HWCFG0

TRDC Hardware Configuration Register 0, offset: 0xF0

```

__IO uint32_t TRDC_HWCFG1
    TRDC Hardware Configuration Register 1, offset: 0xF4
__IO uint32_t TRDC_HWCFG2
    TRDC Hardware Configuration Register 2, offset: 0xF8
__IO uint32_t TRDC_HWCFG3
    TRDC Hardware Configuration Register 3, offset: 0xFC
__IO uint8_t DACFG [8]
    Domain Assignment Configuration Register, array offset: 0x100, array step: 0x1
__IO uint32_t TRDC_IDAU_CR
    TRDC IDAU Control Register, offset: 0x1C0

```

```
struct __TRDC_FLW_Type
```

```
#include <fsl_trdc_core.h> TRDC flash logical control register definition.
```

### Public Members

```

__IO uint32_t TRDC_FLW_CTL
    TRDC FLW Control, offset: 0x1E0
__IO uint32_t TRDC_FLW_PBASE
    TRDC FLW Physical Base, offset: 0x1E4
__IO uint32_t TRDC_FLW_ABASE
    TRDC FLW Array Base, offset: 0x1E8
__IO uint32_t TRDC_FLW_BCNT
    TRDC FLW Block Count, offset: 0x1EC

```

```
struct __TRDC_DomainError_Type
```

```
#include <fsl_trdc_core.h> TRDC domain error register definition.
```

### Public Members

```

__IO uint32_t TRDC_FDIS
    TRDC Fault Domain ID, offset: 0x1FC
__IO uint32_t TRDC_DERRLOC [16]
    TRDC Domain Error Location Register, array offset: 0x200, array step: 0x4

```

```
struct __TRDC_DomainAssignment_Type
```

```
#include <fsl_trdc_core.h> TRDC master domain assignment register definition.
```

### Public Members

```

__IO uint32_t PID [8]
    Process Identifier, array offset: 0x700, array step: 0x4

```

```
struct __TRDC_MBC_Type
```

```
#include <fsl_trdc_core.h> TRDC MBC control register definition.
```

## Public Members

- \_\_\_I uint32\_t MBC\_MEM\_GLBCFG [4]  
MBC Global Configuration Register, array offset: 0x10000, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_NSE\_BLK\_INDEX  
MBC NonSecure Enable Block Index, array offset: 0x10010, array step: 0x2000
- \_\_\_O uint32\_t MBC\_NSE\_BLK\_SET  
MBC NonSecure Enable Block Set, array offset: 0x10014, array step: 0x2000
- \_\_\_O uint32\_t MBC\_NSE\_BLK\_CLR  
MBC NonSecure Enable Block Clear, array offset: 0x10018, array step: 0x2000
- \_\_\_O uint32\_t MBC\_NSE\_BLK\_CLR\_ALL  
MBC NonSecure Enable Block Clear All, array offset: 0x1001C, array step: 0x2000
- \_\_\_IO uint32\_t MBC\_MEMN\_GLBAC [8]  
MBC Global Access Control, array offset: 0x10020, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM0\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x10040, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM0\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10140, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM0\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10180, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM0\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x101A0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM0\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x101A8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM0\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x101C8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM0\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x101D0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM0\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x101F0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM1\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x10240, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM1\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10340, array step: index\*0x2000, index2\*0x4

- \_\_\_IO uint32\_t MBC\_DOM1\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10380, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM1\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x103A0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM1\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x103A8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM1\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x103C8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM1\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x103D0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM1\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x103F0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM2\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x10440, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM2\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10540, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM2\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10580, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM2\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x105A0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM2\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x105A8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM2\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x105C8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM2\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x105D0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM2\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x105F0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM3\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x10640, array step: index\*0x2000, index2\*0x4

- \_\_\_IO uint32\_t MBC\_DOM3\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10740, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM3\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10780, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM3\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x107A0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM3\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x107A8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM3\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x107C8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM3\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x107D0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM3\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x107F0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM4\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x10840, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM4\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10940, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM4\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10980, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM4\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x109A0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM4\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x109A8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM4\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x109C8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM4\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x109D0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM4\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x109F0, array step: index\*0x2000, index2\*0x4

- \_\_\_IO uint32\_t MBC\_DOM5\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x10A40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM5\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10B40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM5\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10B80, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM5\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10BA0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM5\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10BA8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM5\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10BC8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM5\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10BD0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM5\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10BF0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM6\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x10C40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM6\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10D40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM6\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10D80, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM6\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10DA0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM6\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10DA8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM6\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10DC8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM6\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10DD0, array step: index\*0x2000, index2\*0x4



- \_\_\_IO uint32\_t MBC\_DOM6\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10DF0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM7\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x10E40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM7\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10F40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM7\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10F80, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM7\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10FA0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM7\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10FA8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM7\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10FC8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM7\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10FD0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM7\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10FF0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM8\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x11040, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM8\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11140, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM8\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11180, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM8\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x111A0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM8\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x111A8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM8\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x111C8, array step: index\*0x2000, index2\*0x4

- \_\_\_IO uint32\_t MBC\_DOM8\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x111D0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM8\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x111F0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM9\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x11240, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM9\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11340, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM9\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11380, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM9\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x113A0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM9\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x113A8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM9\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x113C8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM9\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x113D0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM9\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x113F0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM10\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x11440, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM10\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11540, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM10\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11580, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM10\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x115A0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM10\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x115A8, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM10\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x115C8, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM10\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x115D0, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM10\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x115F0, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM11\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x11640, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM11\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11740, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM11\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11780, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM11\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x117A0, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM11\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x117A8, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM11\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x117C8, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM11\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x117D0, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM11\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x117F0, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM12\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x11840, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM12\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11940, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM12\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11980, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM12\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x119A0, array step: index\*0x2000, index2\*0x4

- \_\_\_IO uint32\_t MBC\_DOM12\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x119A8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM12\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x119C8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM12\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x119D0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM12\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x119F0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM13\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x11A40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM13\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11B40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM13\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11B80, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM13\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11BA0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM13\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11BA8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM13\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11BC8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM13\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11BD0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM13\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11BF0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM14\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x11C40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM14\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11D40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM14\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11D80, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM14\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11DA0, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM14\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11DA8, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM14\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11DC8, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM14\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11DD0, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM14\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11DF0, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM15\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x11E40, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM15\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11F40, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM15\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11F80, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM15\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11FA0, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM15\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11FA8, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM15\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11FC8, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM15\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11FD0, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM15\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11FF0, array step: index\*0x2000, index2\*0x4

struct \_TRDC\_MRC\_Type

#include <fsl\_trdc\_core.h> TRDC MRC control register definition.  
MRC\_DOM0\_RGD\_W[region][word].

## Public Members

\_\_IO uint32\_t MRC\_GLBCFG  
MRC Global Configuration Register, array offset: 0x14000, array step: 0x1000

---

```

__IO uint32_t MRC_NSE_RGN_INDIRECT
    MRC NonSecure Enable Region Indirect, array offset: 0x14010, array step: 0x1000
__O uint32_t MRC_NSE_RGN_SET
    MRC NonSecure Enable Region Set, array offset: 0x14014, array step: 0x1000
__O uint32_t MRC_NSE_RGN_CLR
    MRC NonSecure Enable Region Clear, array offset: 0x14018, array step: 0x1000
__O uint32_t MRC_NSE_RGN_CLR_ALL
    MRC NonSecure Enable Region Clear All, array offset: 0x1401C, array step: 0x1000
__IO uint32_t MRC_GLBAC [8]
    MRC Global Access Control, array offset: 0x14020, array step: index*0x1000, index2*0x4
__IO uint32_t MRC_DOM0_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14040, array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM0_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x140C0, array step: 0x1000
__IO uint32_t MRC_DOM1_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14140, array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM1_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x141C0, array step: 0x1000
__IO uint32_t MRC_DOM2_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14240, array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM2_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x142C0, array step: 0x1000
__IO uint32_t MRC_DOM3_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14340, array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM3_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x143C0, array step: 0x1000
__IO uint32_t MRC_DOM4_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14440, array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM4_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x144C0, array step: 0x1000
__IO uint32_t MRC_DOM5_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14540, array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM5_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x145C0, array step: 0x1000
__IO uint32_t MRC_DOM6_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14640, array step: index*0x1000, index2*0x8, index3*0x4

```

\_\_IO uint32\_t MRC\_DOM6\_RGD\_NSE  
MRC Region Descriptor NonSecure Enable, array offset: 0x146C0, array step: 0x1000

\_\_IO uint32\_t MRC\_DOM7\_RGD\_W [16][2]  
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14740, array step: index\*0x1000, index2\*0x8, index3\*0x4

\_\_IO uint32\_t MRC\_DOM7\_RGD\_NSE  
MRC Region Descriptor NonSecure Enable, array offset: 0x147C0, array step: 0x1000

\_\_IO uint32\_t MRC\_DOM8\_RGD\_W [16][2]  
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14840, array step: index\*0x1000, index2\*0x8, index3\*0x4

\_\_IO uint32\_t MRC\_DOM8\_RGD\_NSE  
MRC Region Descriptor NonSecure Enable, array offset: 0x148C0, array step: 0x1000

\_\_IO uint32\_t MRC\_DOM9\_RGD\_W [16][2]  
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14940, array step: index\*0x1000, index2\*0x8, index3\*0x4

\_\_IO uint32\_t MRC\_DOM9\_RGD\_NSE  
MRC Region Descriptor NonSecure Enable, array offset: 0x149C0, array step: 0x1000

\_\_IO uint32\_t MRC\_DOM10\_RGD\_W [16][2]  
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14A40, array step: index\*0x1000, index2\*0x8, index3\*0x4

\_\_IO uint32\_t MRC\_DOM10\_RGD\_NSE  
MRC Region Descriptor NonSecure Enable, array offset: 0x14AC0, array step: 0x1000

\_\_IO uint32\_t MRC\_DOM11\_RGD\_W [16][2]  
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14B40, array step: index\*0x1000, index2\*0x8, index3\*0x4

\_\_IO uint32\_t MRC\_DOM11\_RGD\_NSE  
MRC Region Descriptor NonSecure Enable, array offset: 0x14BC0, array step: 0x1000

\_\_IO uint32\_t MRC\_DOM12\_RGD\_W [16][2]  
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14C40, array step: index\*0x1000, index2\*0x8, index3\*0x4

\_\_IO uint32\_t MRC\_DOM12\_RGD\_NSE  
MRC Region Descriptor NonSecure Enable, array offset: 0x14CC0, array step: 0x1000

\_\_IO uint32\_t MRC\_DOM13\_RGD\_W [16][2]  
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14D40, array step: index\*0x1000, index2\*0x8, index3\*0x4

\_\_IO uint32\_t MRC\_DOM13\_RGD\_NSE  
MRC Region Descriptor NonSecure Enable, array offset: 0x14DC0, array step: 0x1000

\_\_IO uint32\_t MRC\_DOM14\_RGD\_W [16][2]  
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14E40, array step: index\*0x1000, index2\*0x8, index3\*0x4

\_\_IO uint32\_t MRC\_DOM14\_RGD\_NSE  
MRC Region Descriptor NonSecure Enable, array offset: 0x14EC0, array step: 0x1000

\_\_IO uint32\_t MRC\_DOM15\_RGD\_W [16][2]  
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14F40, array step: index\*0x1000, index2\*0x8, index3\*0x4

\_\_IO uint32\_t MRC\_DOM15\_RGD\_NSE  
MRC Region Descriptor NonSecure Enable, array offset: 0x14FC0, array step: 0x1000  
struct MBC\_DERR

#### Public Members

\_\_I uint32\_t W0  
MBC Domain Error Word0 Register, array offset: 0x400, array step: 0x10  
\_\_I uint32\_t W1  
MBC Domain Error Word1 Register, array offset: 0x404, array step: 0x10  
\_\_O uint32\_t W3  
MBC Domain Error Word3 Register, array offset: 0x40C, array step: 0x10  
struct MRC\_DERR

#### Public Members

\_\_I uint32\_t W0  
MRC Domain Error Word0 Register, array offset: 0x480, array step: 0x10  
\_\_I uint32\_t W1  
MRC Domain Error Word1 Register, array offset: 0x484, array step: 0x10  
\_\_O uint32\_t W3  
MRC Domain Error Word3 Register, array offset: 0x48C, array step: 0x10  
union \_\_unnamed37\_\_

#### Public Members

struct *\_TRDC\_DomainAssignment\_Type* MDA\_DFMT0[8]  
struct *\_TRDC\_DomainAssignment\_Type* MDA\_DFMT1[8]  
struct MDA\_DFMT0

#### Public Members

\_\_IO uint32\_t MDA\_W\_DFMT0 [8]  
DAC Master Domain Assignment Register, array offset: 0x800, array step: index\*0x20,  
index2\*0x4  
struct MDA\_DFMT1

#### Public Members

\_\_IO uint32\_t MDA\_W\_DFMT1 [1]  
DAC Master Domain Assignment Register, array offset: 0x800, array step: index\*0x20,  
index2\*0x4



## 2.51 Trdc\_soc

FSL\_TRDC\_SOC\_DRIVER\_VERSION

Driver version 2.0.0.

TRDC\_MBC\_MEM\_GLBCFG\_NBLKS\_MASK

TRDC\_MBC\_MEM\_GLBCFG\_SIZE\_LOG2\_MASK

TRDC\_MBC\_MEM\_GLBCFG\_SIZE\_LOG2\_SHIFT

TRDC\_MBC\_NSE\_BLK\_CLR\_ALL\_MEMSEL(x)

TRDC\_MBC\_NSE\_BLK\_CLR\_ALL\_DID\_SEL(x)

FSL\_FEATURE\_TRDC\_DOMAIN\_COUNT

TRDC feature.

TRDC\_MBC\_COUNT

TRDC base address convert macro.

TRDC\_MBC\_OFFSET(x)

TRDC\_MBC\_ARRAY\_STEP

FSL\_COMPONENT\_ID

## 2.52 UTICK: Mictotick Timer Driver

void UTICK\_Init(UTICK\_Type \*base)

Initializes an UTICK by turning its bus clock on.

void UTICK\_Deinit(UTICK\_Type \*base)

Deinitializes a UTICK instance.

This function shuts down Utick bus clock

### Parameters

- base – UTICK peripheral base address.

uint32\_t UTICK\_GetStatusFlags(UTICK\_Type \*base)

Get Status Flags.

This returns the status flag

### Parameters

- base – UTICK peripheral base address.

### Returns

status register value

void UTICK\_ClearStatusFlags(UTICK\_Type \*base)

Clear Status Interrupt Flags.

This clears intr status flag

### Parameters

- base – UTICK peripheral base address.

### Returns

none

```
void UTICK_SetTick(UTICK_Type *base, utick_mode_t mode, uint32_t count, utick_callback_t cb)
```

Starts UTICK.

This function starts a repeat/onetime countdown with an optional callback

#### Parameters

- base – UTICK peripheral base address.
- mode – UTICK timer mode (ie kUTICK\_onetime or kUTICK\_repeat)
- count – UTICK timer mode (ie kUTICK\_onetime or kUTICK\_repeat)
- cb – UTICK callback (can be left as NULL if none, otherwise should be a void func(void))

#### Returns

none

```
void UTICK_HandleIRQ(UTICK_Type *base, utick_callback_t cb)
```

UTICK Interrupt Service Handler.

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in UTICK\_SetTick()). if no user callback is scheduled, the interrupt will simply be cleared.

#### Parameters

- base – UTICK peripheral base address.
- cb – callback scheduled for this instance of UTICK

#### Returns

none

```
FSL_UTICK_DRIVER_VERSION
```

UTICK driver version 2.0.5.

```
enum _utick_mode
```

UTICK timer operational mode.

*Values:*

```
enumerator kUTICK_Onetime
```

Trigger once

```
enumerator kUTICK_Repeat
```

Trigger repeatedly

```
typedef enum _utick_mode utick_mode_t
```

UTICK timer operational mode.

```
typedef void (*utick_callback_t)(void)
```

UTICK callback function.

## 2.53 WAKETIMER: WAKETIMER Driver

```
void WAKETIMER_Init(WAKETIMER_Type *base, const waketimer_config_t *config)
```

Initializes an WAKETIMER.

This function initializes the WAKETIMER.

#### Parameters

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

void WAKETIMER\_Deinit(WAKETIMER\_Type \*base)

Deinitializes a WAKETIMER instance.

This function deinitialize the WAKETIMER.

#### Parameters

- base – WAKETIMER peripheral base address.

void WAKETIMER\_GetDefaultConfig(*waketimer\_config\_t* \*config)

Fills in the WAKETIMER configuration structure with the default settings.

The default values are:

```
config->enableInterrupt = true;
config->enableOSCDivide = true;
config->callback         = NULL;
```

#### Parameters

- config – Pointer to the user configuration structure.

void WAKETIMER\_EnableInterrupts(WAKETIMER\_Type \*base, uint32\_t mask)

Enables the selected WAKETIMER interrupts.

#### Parameters

- base – WAKETIMER peripheral base address
- mask – Mask value for interrupt events. See to `_waketimer_interrupt_enable`

void WAKETIMER\_DisableInterrupts(WAKETIMER\_Type \*base, uint32\_t mask)

Enables the selected WAKETIMER interrupts.

#### Parameters

- base – WAKETIMER peripheral base address
- mask – Mask value for interrupt events. See to `_waketimer_interrupt_enable`

void WAKETIMER\_ClearStatusFlags(WAKETIMER\_Type \*base, uint32\_t mask)

Clear Status Interrupt Flag.

This clears intrrupt status flag. Currently, only match interrupt flag can be cleared.

#### Parameters

- base – WAKETIMER peripheral base address.
- mask – Mask value for flags to be cleared. See to `_waketimer_status_flags`.

#### Returns

none

void WAKETIMER\_SetCallback(WAKETIMER\_Type \*base, *waketimer\_callback\_t* callback)

Receive noticification when waketime countdown.

If the interrupt for the waketime countdown is enabled, then a callback can be registered which will be invoked when the event is triggered

#### Parameters

- base – WAKETIMER peripheral base address

- `callback` – Function to invoke when the event is triggered

static inline void WAKETIMER\_HaltTimer(WAKETIMER\_Type \*base)

Halt and clear timer counter.

This halt and clear timer counter.

#### Parameters

- `base` – WAKETIMER peripheral base address.

#### Returns

none

static inline void WAKETIMER\_StartTimer(WAKETIMER\_Type \*base, uint32\_t value)

Set timer counter.

This set the timer counter and start the timer countdown.

#### Parameters

- `base` – WAKETIMER peripheral base address.
- `value` – countdown value.

#### Returns

none

uint32\_t WAKETIMER\_GetCurrentTimerValue(WAKETIMER\_Type \*base)

Get current timer count value from WAKETIMER.

This function will get a decimal timer count value. The RAW value of timer count is gray code format, will be translated to decimal data internally.

#### Parameters

- `base` – WAKETIMER peripheral base address.

#### Returns

Value of WAKETIMER which will be formatted to decimal value.

FSL\_WAKETIMER\_DRIVER\_VERSION

WAKETIMER driver version.

enum \_waketimer\_status\_flags

WAKETIMER status flags.

*Values:*

enumerator kWAKETIMER\_WakeFlag

Wake Timer Status Flag, sets wake timer has timed out.

enum \_waketimer\_interrupt\_enable

Define interrupt switchers of the module.

*Values:*

enumerator kWAKETIMER\_WakeInterruptEnable

Generate interrupt requests when WAKE\_FLAG is asserted.

typedef void (\*waketimer\_callback\_t)(void)

waketimer callback function.

typedef struct \_waketimer\_config waketimer\_config\_t

WAKETIMER configuration structure.

This structure holds the configuration settings for the WAKETIMER peripheral. To initialize this structure to reasonable defaults, call the WAKETIMER\_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

`struct _waketimer_config`

*#include <fsl\_waketimer.h>* WAKETIMER configuration structure.

This structure holds the configuration settings for the WAKETIMER peripheral. To initialize this structure to reasonable defaults, call the `WAKETIMER_GetDefaultConfig()` function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

### Public Members

`bool enableOSCDivide`

true: Enable OSC Divide. false: Disable OSC Divide.

`bool enableInterrupt`

true: Enable interrupt. false: Disable interrupt.

*waketimer\_callback\_t* callback

timer countdown callback.

## 2.54 WUU: Wakeup Unit driver

`void WUU_SetExternalWakeUpPinsConfig(WUU_Type *base, uint8_t pinIndex, const  
wuu_external_wakeup_pin_config_t *config)`

Enables and Configs External WakeUp Pins.

This function enables/disables the external pin as wakeup input. What's more this function configs pins options, including edge detection wakeup event and operate mode.

### Parameters

- `base` – MUU peripheral base address.
- `pinIndex` – The index of the external input pin. See Reference Manual for the details.
- `config` – Pointer to `wuu_external_wakeup_pin_config_t` structure.

`void WUU_ClearExternalWakeupPinsConfig(WUU_Type *base, uint8_t pinIndex)`

Disable and clear external wakeup pin settings.

### Parameters

- `base` – MUU peripheral base address.
- `pinIndex` – The index of the external input pin.

`static inline uint32_t WUU_GetExternalWakeUpPinsFlag(WUU_Type *base)`

Gets External Wakeup pin flags.

This function return the external wakeup pin flags.

### Parameters

- `base` – WUU peripheral base address.

### Returns

Wakeup flags for all external wakeup pins.

```
static inline void WUU_ClearExternalWakeUpPinsFlag(WUU_Type *base, uint32_t mask)
```

Clears External WakeUp Pin flags.

This function clears external wakeup pins flags based on the mask.

#### Parameters

- base – WUU peripheral base address.
- mask – The mask of Wakeup pin index to be cleared.

```
void WUU_SetInternalWakeUpModulesConfig(WUU_Type *base, uint8_t moduleIndex,  
                                         wuu_internal_wakeup_module_event_t event)
```

Config Internal modules' event as the wake up sources.

This function configs the internal modules event as the wake up sources.

#### Parameters

- base – WUU peripheral base address.
- moduleIndex – The selected internal module. See the Reference Manual for the details.
- event – Select interrupt or DMA/Trigger of the internal module as the wake up source.

```
void WUU_ClearInternalWakeUpModulesConfig(WUU_Type *base, uint8_t moduleIndex,  
                                           wuu_internal_wakeup_module_event_t event)
```

Disable an on-chip internal modules' event as the wakeup sources.

#### Parameters

- base – WUU peripheral base address.
- moduleIndex – The selected internal module. See the Reference Manual for the details.
- event – The event(interrupt or DMA/trigger) of the internal module to disable.

```
void WUU_SetPinFilterConfig(WUU_Type *base, uint8_t filterIndex, const  
                           wuu_pin_filter_config_t *config)
```

Configs and Enables Pin filters.

This function configs Pin filter, including pin select, filter operate mode filter wakeup event and filter edge detection.

#### Parameters

- base – WUU peripheral base address.
- filterIndex – The index of the pin filter.
- config – Pointer to *wuu\_pin\_filter\_config\_t* structure.

```
bool WUU_GetPinFilterFlag(WUU_Type *base, uint8_t filterIndex)
```

Gets the pin filter configuration.

This function gets the pin filter flag.

#### Parameters

- base – WUU 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 WUU_ClearPinFilterFlag(WUU_Type *base, uint8_t filterIndex)`

Clears the pin filter configuration.

This function clears the pin filter flag.

**Parameters**

- `base` – WUU peripheral base address.
- `filterIndex` – A pin filter index to clear the flag, starting from 1.

`bool WUU_GetExternalWakeupPinFlag(WUU_Type *base, uint32_t pinIndex)`

brief 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.

param `base` WUU peripheral base address. param `pinIndex` A pin index, which starts from 0. return True if the specific pin is a wakeup source.

`void WUU_ClearExternalWakeupPinFlag(WUU_Type *base, uint32_t pinIndex)`

brief Clears the external wakeup source flag.

This function clears the external wakeup source flag for a specific pin.

param `base` WUU peripheral base address. param `pinIndex` A pin index, which starts from 0.

`FSL_WUU_DRIVER_VERSION`

Defines WUU driver version 2.4.1.

`enum _wuu_external_pin_edge_detection`

External WakeUp pin edge detection enumeration.

*Values:*

enumerator `kWUU_ExternalPinDisable`

External input Pin disabled as wake up input.

enumerator `kWUU_ExternalPinRisingEdge`

External input Pin enabled with the rising edge detection.

enumerator `kWUU_ExternalPinFallingEdge`

External input Pin enabled with the falling edge detection.

enumerator `kWUU_ExternalPinAnyEdge`

External input Pin enabled with any change detection.

`enum _wuu_external_wakeup_pin_event`

External input wake up pin event enumeration.

*Values:*

enumerator `kWUU_ExternalPinInterrupt`

External input Pin configured as interrupt.

enumerator `kWUU_ExternalPinDMARequest`

External input Pin configured as DMA request.

enumerator `kWUU_ExternalPinTriggerEvent`

External input Pin configured as Trigger event.

`enum _wuu_external_wakeup_pin_mode`

External input wake up pin mode enumeration.

*Values:*

enumerator kWUU\_ExternalPinActiveDSPD

External input Pin is active only during Deep Sleep/Power Down Mode. NOTE: This enumerations has been deprecated, please switch to kWUU\_ExternalPinActiveLowLeakage.

enumerator kWUU\_ExternalPinActiveLowLeakageMode

External input Pin is active only during low-leakage power modes.

enumerator kWUU\_ExternalPinActiveAlways

External input Pin is active during all power modes.

enum \_\_wu\_\_internal\_wakeup\_\_module\_\_event

Internal module wake up event enumeration.

*Values:*

enumerator kWUU\_InternalModuleInterrupt

Internal modules' interrupt as a wakeup source.

enumerator kWUU\_InternalModuleDMATrigger

Internal modules' DMA/Trigger as a wakeup source.

enum \_\_wu\_\_filter\_\_edge

Pin filter edge enumeration.

*Values:*

enumerator kWUU\_FilterDisabled

Filter disabled.

enumerator kWUU\_FilterPosedgeEnable

Filter posedge detect enabled.

enumerator kWUU\_FilterNegedgeEnable

Filter negedge detect enabled.

enumerator kWUU\_FilterAnyEdge

Filter any edge detect enabled.

enum \_\_wu\_\_filter\_\_event

Pin Filter event enumeration.

*Values:*

enumerator kWUU\_FilterInterrupt

Filter output configured as interrupt.

enumerator kWUU\_FilterDMARequest

Filter output configured as DMA request.

enumerator kWUU\_FilterTriggerEvent

Filter output configured as Trigger event.

enum \_\_wu\_\_filter\_\_mode

Pin filter mode enumeration.

*Values:*

enumerator kWUU\_FilterActiveDSPD

External input pin filter is active only during Deep Sleep/Power Down Mode. NOTE: This enumerations has been deprecated, please switch to kWUU\_FilterActiveLowLeakage.



enumerator kWUU\_FilterActiveLowLeakageMode

External input pin filter is active only during low-leakage power modes.

enumerator kWUU\_FilterActiveAlways

External input Pin filter is active during all power modes.

typedef enum *\_wuu\_external\_pin\_edge\_detection* wuu\_external\_pin\_edge\_detection\_t

External WakeUp pin edge detection enumeration.

typedef enum *\_wuu\_external\_wakeup\_pin\_event* wuu\_external\_wakeup\_pin\_event\_t

External input wake up pin event enumeration.

typedef enum *\_wuu\_external\_wakeup\_pin\_mode* wuu\_external\_wakeup\_pin\_mode\_t

External input wake up pin mode enumeration.

typedef enum *\_wuu\_internal\_wakeup\_module\_event* wuu\_internal\_wakeup\_module\_event\_t

Internal module wake up event enumeration.

typedef enum *\_wuu\_filter\_edge* wuu\_filter\_edge\_t

Pin filter edge enumeration.

typedef enum *\_wuu\_filter\_event* wuu\_filter\_event\_t

Pin Filter event enumeration.

typedef enum *\_wuu\_filter\_mode* wuu\_filter\_mode\_t

Pin filter mode enumeration.

typedef struct *\_wuu\_external\_wakeup\_pin\_config* wuu\_external\_wakeup\_pin\_config\_t

External WakeUp pin configuration.

typedef struct *\_wuu\_pin\_filter\_config* wuu\_pin\_filter\_config\_t

Pin Filter configuration.

struct *\_wuu\_external\_wakeup\_pin\_config*

*#include <fsl\_wuu.h>* External WakeUp pin configuration.

### Public Members

*wuu\_external\_pin\_edge\_detection\_t* edge

External Input pin edge detection.

*wuu\_external\_wakeup\_pin\_event\_t* event

External Input wakeup Pin event

*wuu\_external\_wakeup\_pin\_mode\_t* mode

External Input wakeup Pin operate mode.

struct *\_wuu\_pin\_filter\_config*

*#include <fsl\_wuu.h>* Pin Filter configuration.

### Public Members

uint32\_t pinIndex

The index of wakeup pin to be muxxed into filter.

*wuu\_filter\_edge\_t* edge

The edge of the pin digital filter.

*wuu\_filter\_event\_t* event

The event of the filter output.

*wwu\_filter\_mode\_t* mode

The mode of the filter operate.

## 2.55 WWDT: Windowed Watchdog Timer Driver

`void WWDT_GetDefaultConfig(wwdt_config_t *config)`

Initializes WWDT configure structure.

This function initializes the WWDT configure structure to default value. The default value are:

```
config->enableWwdt = true;
config->enableWatchdogReset = false;
config->enableWatchdogProtect = false;
config->enableLockOscillator = false;
config->windowValue = 0xFFFFFFFU;
config->timeoutValue = 0xFFFFFFFU;
config->warningValue = 0;
```

**See also:**

`wwdt_config_t`

### Parameters

- `config` – Pointer to WWDT config structure.

`void WWDT_Init(WWDT_Type *base, const wwdt_config_t *config)`

Initializes the WWDT.

This function initializes the WWDT. When called, the WWDT runs according to the configuration.

Example:

```
wwdt_config_t config;
WWDT_GetDefaultConfig(&config);
config.timeoutValue = 0x7ffU;
WWDT_Init(wwdt_base,&config);
```

### Parameters

- `base` – WWDT peripheral base address
- `config` – The configuration of WWDT

`void WWDT_Deinit(WWDT_Type *base)`

Shuts down the WWDT.

This function shuts down the WWDT.

### Parameters

- `base` – WWDT peripheral base address

`static inline void WWDT_Enable(WWDT_Type *base)`

Enables the WWDT module.

This function write value into WWDT\_MOD register to enable the WWDT, it is a write-once bit; once this bit is set to one and a watchdog feed is performed, the watchdog timer will run permanently.

### Parameters

- base – WWDT peripheral base address

static inline void WWDT\_Disable(WWDT\_Type \*base)

Disables the WWDT module.

### Deprecated:

Do not use this function. It will be deleted in next release version, for once the bit field of WDEN written with a 1, it can not be re-written with a 0.

This function write value into WWDT\_MOD register to disable the WWDT.

### Parameters

- base – WWDT peripheral base address

static inline uint32\_t WWDT\_GetStatusFlags(WWDT\_Type \*base)

Gets all WWDT status flags.

This function gets all status flags.

Example for getting Timeout Flag:

```
uint32_t status;  
status = WWDT_GetStatusFlags(wwdt_base) & kWWDT_TimeoutFlag;
```

### Parameters

- base – WWDT peripheral base address

### Returns

The status flags. This is the logical OR of members of the enumeration `_wwdt_status_flags_t`

void WWDT\_ClearStatusFlags(WWDT\_Type \*base, uint32\_t mask)

Clear WWDT flag.

This function clears WWDT status flag.

Example for clearing warning flag:

```
WWDT_ClearStatusFlags(wwdt_base, kWWDT_WarningFlag);
```

### Parameters

- base – WWDT peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration `_wwdt_status_flags_t`

static inline void WWDT\_SetWarningValue(WWDT\_Type \*base, uint32\_t warningValue)

Set the WWDT warning value.

The WDWARNINT register determines the watchdog timer counter value that will generate a watchdog interrupt. When the watchdog timer counter is no longer greater than the value defined by WARNINT, an interrupt will be generated after the subsequent WDCLK.

### Parameters

- base – WWDT peripheral base address
- warningValue – WWDT warning value.

```
static inline void WWDT_SetTimeoutValue(WWDT_Type *base, uint32_t timeoutCount)
```

Set the WWDT timeout value.

This function sets the timeout value. Every time a feed sequence occurs the value in the TC register is loaded into the Watchdog timer. Writing a value below 0xFF will cause 0xFF to be loaded into the TC register. Thus the minimum time-out interval is  $TWDCLK * 256 * 4$ . If enableWatchdogProtect flag is true in wwdt\_config\_t config structure, any attempt to change the timeout value before the watchdog counter is below the warning and window values will cause a watchdog reset and set the WDTOF flag.

#### Parameters

- base – WWDT peripheral base address
- timeoutCount – WWDT timeout value, count of WWDT clock tick.

```
static inline void WWDT_SetWindowValue(WWDT_Type *base, uint32_t windowValue)
```

Sets the WWDT window value.

The WINDOW register determines the highest TV value allowed when a watchdog feed is performed. If a feed sequence occurs when timer value is greater than the value in WINDOW, a watchdog event will occur. To disable windowing, set windowValue to 0xFFFF (maximum possible timer value) so windowing is not in effect.

#### Parameters

- base – WWDT peripheral base address
- windowValue – WWDT window value.

```
void WWDT_Refresh(WWDT_Type *base)
```

Refreshes the WWDT timer.

This function feeds the WWDT. This function should be called before WWDT timer is in timeout. Otherwise, a reset is asserted.

#### Parameters

- base – WWDT peripheral base address

```
FSL_WWDT_DRIVER_VERSION
```

Defines WWDT driver version.

```
WWDT_FIRST_WORD_OF_REFRESH
```

First word of refresh sequence

```
WWDT_SECOND_WORD_OF_REFRESH
```

Second word of refresh sequence

```
enum __wwdt_status_flags_t
```

WWDT status flags.

This structure contains the WWDT status flags for use in the WWDT functions.

*Values:*

```
enumerator kWWDT_TimeoutFlag
```

Time-out flag, set when the timer times out

```
enumerator kWWDT_WarningFlag
```

Warning interrupt flag, set when timer is below the value WDWARNINT

```
typedef struct __wwdt_config wwdt_config_t
```

Describes WWDT configuration structure.

```
struct __wwdt_config
```

#include <fsl\_wwdt.h> Describes WWDT configuration structure.

## Public Members

`bool enableWwdt`

Enables or disables WWDT

`bool enableWatchdogReset`

true: Watchdog timeout will cause a chip reset false: Watchdog timeout will not cause a chip reset

`bool enableWatchdogProtect`

true: Enable watchdog protect i.e timeout value can only be changed after counter is below warning & window values false: Disable watchdog protect; timeout value can be changed at any time

`bool enableLockOscillator`

true: Disabling or powering down the watchdog oscillator is prevented Once set, this bit can only be cleared by a reset false: Do not lock oscillator

`uint32_t windowValue`

Window value, set this to 0xFFFFFFFF if windowing is not in effect

`uint32_t timeoutValue`

Timeout value

`uint32_t warningValue`

Watchdog time counter value that will generate a warning interrupt. Set this to 0 for no warning

`uint32_t clockFreq_Hz`

Watchdog clock source frequency.

# Chapter 3

## Middleware

### 3.1 Boot

#### 3.1.1 MCUXpresso SDK : mcuxsdk-middleware-mcuboot\_opensource

##### Overview

This repository is a fork of MCUboot (<https://github.com/mcu-tools/mcuboot>) for MCUXpresso SDK delivery and it contains the components officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository (mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

##### Documentation

Overall details can be reviewed here: [MCUXpresso SDK Online Documentation](#)

Visit [MCUboot - Documentation](#) to review details on the contents in this sub-repo.

##### Setup

Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest [Getting Started with SDK - Detailed Installation Instructions](#)

##### Contribution

Contributions are not currently accepted. If the intended contribution is not related to NXP specific code, consider contributing directly to the upstream MCUboot project. Once this MCUboot fork is synchronized with the upstream project, such contributions will end up here as well. If the intended contribution is a bugfix or improvement for NXP porting layer or for code added or modified by NXP, please open an issue or contact NXP support.

## NXP Fork

This fork of MCUboot contains specific modifications and enhancements for NXP MCUXpresso SDK integration.

See *changelog* for details.

### 3.1.2 MCUboot



This is MCUboot version 2.2.0

MCUboot is a secure bootloader for 32-bits microcontrollers. It defines a common infrastructure for the bootloader and the system flash layout on microcontroller systems, and provides a secure bootloader that enables easy software upgrade.

MCUboot is not dependent on any specific operating system and hardware and relies on hardware porting layers from the operating system it works with. Currently, MCUboot works with the following operating systems and SoCs:

- [Zephyr](#)
- [Apache Mynewt](#)
- [Apache NuttX](#)
- [RIOT](#)
- [Mbed OS](#)
- [Espressif](#)
- [Cypress/Infineon](#)

RIOT is supported only as a boot target. We will accept any new port contributed by the community once it is good enough.

#### MCUboot How-tos

See the following pages for instructions on using MCUboot with different operating systems and SoCs:

- [Zephyr](#)
- [Apache Mynewt](#)
- [Apache NuttX](#)
- [RIOT](#)
- [Mbed OS](#)
- [Espressif](#)
- [Cypress/Infineon](#)

There are also instructions for the *Simulator*.

## Roadmap

The issues being planned and worked on are tracked using GitHub issues. To give your input, visit [MCUboot GitHub Issues](#).

## Source files

You can find additional documentation on the bootloader in the source files. For more information, use the following links:

- [boot/bootutil](#) - The core of the bootloader itself.
- [boot/boot\\_serial](#) - Support for serial upgrade within the bootloader itself.
- [boot/zephyr](#) - Port of the bootloader to Zephyr.
- [boot/mynewt](#) - Bootloader application for Apache Mynewt.
- [boot/nuttX](#) - Bootloader application and port of MCUboot interfaces for Apache NuttX.
- [boot/mbed](#) - Port of the bootloader to Mbed OS.
- [boot/espressif](#) - Bootloader application and MCUboot port for Espressif SoCs.
- [boot/cypress](#) - Bootloader application and MCUboot port for Cypress/Infineon SoCs.
- [imgtool](#) - A tool to securely sign firmware images for booting by MCUboot.
- [sim](#) - A bootloader simulator for testing and regression.

## Joining the project

Developers are welcome!

Use the following links to join or see more about the project:

- [Our developer mailing list](#)
- [Our Discord channel](#) [Get your invite](#)

## 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 applications** 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\_QUEUE\_SIZE

```
#define FMSTR_COMM_QUEUE_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\_QUEUE\_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\_QUEUE\_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-language 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 */
```

(continues on next page)



(continued from previous page)

```

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_UINTn                    | Unsigned integer type of size <i>n</i> bits (n=8,16,32,64)     |
| FMSTR_TSA_SINTn                    | Signed integer type of size <i>n</i> bits (n=8,16,32,64)       |
| FMSTR_TSA_FRACn                    | Fractional number of size <i>n</i> bits (n=16,32,64).          |
| FMSTR_TSA_FRAC_Q( <i>m,n</i> )     | Signed fractional number in general Q form (m+n+1 total bits)  |
| FMSTR_TSA_FRAC_UQ( <i>m,n</i> )    | 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)  |
| FM-STR_TSA_USERTYPE( <i>name</i> ) | 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                                                                                                                         |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>FM-STR_ADDR</i><br>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. | 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. |
| <i>FM-STR_SIZE</i><br>It is required that this type is unsigned and at least 16 bits wide integer.                                                                                                                        | Data type used to hold the memory block size.                                                                                       |
| <i>FM-STR_BOOL</i><br>This type is used only in zero/non-zero conditions in the driver code.                                                                                                                              | Data type used as a general boolean type.                                                                                           |
| <i>FM-STR_APPCM</i><br>Generally, this is an unsigned 8-bit value.                                                                                                                                                        | Data type used to hold the Application Command code.                                                                                |
| <i>FM-STR_APPCM</i><br>Generally, this is an unsigned 8-bit value.                                                                                                                                                        | Data type used to create the Application Command data buffer.                                                                       |
| <i>FM-STR_APPCM</i><br>Generally, this is an unsigned 8-bit value.                                                                                                                                                        | Data type used to hold the Application Command result code.                                                                         |

**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.

|                       |                                                                                                                                                               |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>FM-STR_TSA_TII</i> | Data type used to hold a descriptor index in the TSA table or a table index in the list of TSA tables.<br>By default, this is defined as <i>FM-STR_SIZE</i> . |
| <i>FM-STR_TSA_TS</i>  | Data type used to hold a memory block size, as used in the TSA descriptors.<br>By default, this is defined as <i>FM-STR_SIZE</i> .                            |

**Public Pipes types** The table describes the data types used by the FreeMASTER Pipes API:

|                       |                                                                                                                |
|-----------------------|----------------------------------------------------------------------------------------------------------------|
| <i>FM-STR_HPIPE</i>   | Pipe handle that identifies the open-pipe object.<br>Generally, this is a pointer to a void type.              |
| <i>FM-STR_PIPE_PC</i> | Integer type required to hold at least 7 bits of data.<br>Generally, this is an unsigned 8-bit or 16-bit type. |
| <i>FM-STR_PIPE_SI</i> | Integer type required to hold at least 16 bits of data.<br>This is used to store the data buffer sizes.        |
| <i>FM-STR_PPIPEF</i>  | Pointer to the pipe handler function.<br>See <a href="#">FM-STR_PipeOpen</a> 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.

|                                                                                                          |                                                                         |
|----------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|
| <i>FMSTR_U8</i>                                                                                          | 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.                               |                                                                         |
| <i>FM-STR_U16</i>                                                                                        | Unsigned 16-bit integer.                                                |
| <i>FM-STR_U32</i>                                                                                        | Unsigned 32-bit integer.                                                |
| <i>FMSTR_S8</i>                                                                                          | Signed 8-bit integer.                                                   |
| <i>FM-STR_S16</i>                                                                                        | Signed 16-bit integer.                                                  |
| <i>FM-STR_S32</i>                                                                                        | Signed 32-bit integer.                                                  |
| <i>FM-STR_FLOAT</i>                                                                                      | 4-byte standard IEEE floating-point type.                               |
| <i>FM-STR_FLAGS</i>                                                                                      | Data type forming a union with a structure of flag bit-fields.          |
| <i>FM-STR_SIZE8</i>                                                                                      | Data type holding a general size value, at least 8 bits wide.           |
| <i>FM-STR_INDEX</i>                                                                                      | General for-loop index. Must be signed, at least 16 bits wide.          |
| <i>FM-STR_BCHR</i>                                                                                       | 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. |                                                                         |
| <i>FM-STR_BPTR</i>                                                                                       | A pointer to the communication buffer (an array of <i>FMSTR_BCHR</i> ). |

## 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](http://www.nxp.com/freemaster)
- FreeMASTER community area: [community.nxp.com/community/freemaster](http://community.nxp.com/community/freemaster)
- FreeMASTER GitHub code repo: <https://github.com/nxp-mcuxpresso/mcux-freemaster>
- MCUXpresso SDK home: [www.nxp.com/mcuxpresso](http://www.nxp.com/mcuxpresso)
- MCUXpresso SDK builder: [mcuxpresso.nxp.com/en](http://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.                                                                                                                                                                                                                                 |

# Chapter 4

## RTOS

### 4.1 FreeRTOS

#### 4.1.1 FreeRTOS kernel

Open source RTOS kernel for small devices.

[FreeRTOS kernel for MCUXpresso SDK Readme](#)

[FreeRTOS kernel for MCUXpresso SDK ChangeLog](#)

[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](#)

#### 4.1.4 corehttp

C language HTTP client library designed for embedded platforms.

#### 4.1.5 corejson

JSON parser.

## **Readme**

### **4.1.6 coremqtt**

MQTT publish/subscribe messaging library.

### **4.1.7 corepkcs11**

PKCS #11 key management library.

## **Readme**

### **4.1.8 freertos-plus-tcp**

Open source RTOS FreeRTOS Plus TCP.

## **Readme**