



# MCUXpresso SDK Documentation

Release 25.06.00



NXP  
Jun 26, 2025



# Table of contents

|          |  |            |
|----------|--|------------|
| <b>1</b> | <b>FRDM-K32L2A4S</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                                  | 133        |
| 1.7      | Middleware Documentation                                     | 133        |
| 1.7.1    | FreeMASTER   | 133        |
| 1.7.2    | FreeRTOS   | 133        |
| 1.7.3    | File systemFatfs   | 133        |
| <b>2</b> | <b>K32L2A41A</b>   | <b>135</b> |
| 2.1      | ADC16: 16-bit SAR Analog-to-Digital Converter Driver         | 135        |
| 2.2      | Clock Driver   | 144        |
| 2.3      | CMP: Analog Comparator Driver                                | 166        |
| 2.4      | CRC: Cyclic Redundancy Check Driver                          | 170        |
| 2.5      | DAC: Digital-to-Analog Converter Driver                      | 173        |
| 2.6      | DMAMUX: Direct Memory Access Multiplexer Driver              | 178        |
| 2.7      | eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver | 179        |
| 2.8      | FGPIO Driver   | 199        |
| 2.9      | C90TFS Flash Driver  | 200        |
| 2.10     | FlexIO: FlexIO Driver  | 200        |
| 2.11     | FlexIO Driver  | 200        |
| 2.12     | FlexIO eDMA I2S Driver                                       | 215        |
| 2.13     | FlexIO eDMA SPI Driver                                       | 218        |
| 2.14     | FlexIO eDMA UART Driver                                      | 222        |
| 2.15     | FlexIO I2C Master Driver                                     | 224        |
| 2.16     | FlexIO I2S Driver  | 233        |
| 2.17     | FlexIO SPI Driver  | 243        |
| 2.18     | FlexIO UART Driver   | 256        |
| 2.19     | ftfx adapter   | 267        |
| 2.20     | Ftftx CACHE Driver   | 267        |
| 2.21     | ftfx controller  | 269        |
| 2.22     | ftfx feature   | 285        |
| 2.23     | Ftftx FLASH Driver   | 286        |
| 2.24     | Ftftx FLEXNVM Driver   | 299        |
| 2.25     | ftfx utilities   | 310        |
| 2.26     | GPIO: General-Purpose Input/Output Driver                    | 311        |
| 2.27     | GPIO Driver  | 312        |
| 2.28     | INTMUX: Interrupt Multiplexer Driver                         | 314        |
| 2.29     | Common Driver  | 316        |

|          |  |            |
|----------|--|------------|
| 2.30     | Lin_lpuart_driver  | 328        |
| 2.31     | LLWU: Low-Leakage Wakeup Unit Driver                                 | 335        |
| 2.32     | LPI2C: Low Power Inter-Integrated Circuit Driver                     | 339        |
| 2.33     | LPI2C Master Driver  | 340        |
| 2.34     | LPI2C Master DMA Driver  | 355        |
| 2.35     | LPI2C Slave Driver   | 357        |
| 2.36     | LPIT: Low-Power Interrupt Timer                                      | 367        |
| 2.37     | LPSPi: Low Power Serial Peripheral Interface                         | 374        |
| 2.38     | LPSPi Peripheral driver  | 374        |
| 2.39     | LPSPi eDMA Driver  | 396        |
| 2.40     | LPTMR: Low-Power Timer   | 403        |
| 2.41     | LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver | 409        |
| 2.42     | LPUART Driver  | 409        |
| 2.43     | LPUART eDMA Driver   | 428        |
| 2.44     | MCM: Miscellaneous Control Module                                    | 431        |
| 2.45     | MMDVSQ: Memory-Mapped Divide and Square Root                         | 435        |
| 2.46     | MSCM: Miscellaneous System Control                                   | 438        |
| 2.47     | PMC: Power Management Controller                                     | 438        |
| 2.48     | PORT: Port Control and Interrupts                                    | 444        |
| 2.49     | RCM: Reset Control Module Driver                                     | 451        |
| 2.50     | RTC: Real Time Clock   | 457        |
| 2.51     | SIM: System Integration Module Driver                                | 464        |
| 2.52     | Smart Card   | 465        |
| 2.53     | Smart Card EMVSIM Driver   | 473        |
| 2.54     | Smart Card PHY Driver  | 476        |
| 2.55     | Smart Card PHY EMVSIM Driver   | 478        |
| 2.56     | Smart Card PHY TDA8035 Driver  | 478        |
| 2.57     | SMC: System Mode Controller Driver                                   | 478        |
| 2.58     | TPM: Timer PWM Module  | 484        |
| 2.59     | TRGMUX: Trigger Mux Driver   | 500        |
| 2.60     | TRNG: True Random Number Generator                                   | 501        |
| 2.61     | TSI: Touch Sensing Input   | 505        |
| 2.62     | TSTMR: Timestamp Timer Driver  | 518        |
| 2.63     | VREF: Voltage Reference Driver                                       | 518        |
| 2.64     | WDOG32: 32-bit Watchdog Timer  | 522        |
| <b>3</b> | <b>Middleware</b>  | <b>529</b> |
| 3.1      | Motor Control  | 529        |
| 3.1.1    | FreeMASTER   | 529        |
| <b>4</b> | <b>RTOS</b>  | <b>567</b> |
| 4.1      | FreeRTOS   | 567        |
| 4.1.1    | FreeRTOS kernel  | 567        |
| 4.1.2    | FreeRTOS drivers   | 567        |
| 4.1.3    | backoffalgorithm   | 567        |
| 4.1.4    | corehttp   | 567        |
| 4.1.5    | corejson   | 567        |
| 4.1.6    | coremqtt   | 568        |
| 4.1.7    | coremqtt-agent   | 568        |
| 4.1.8    | corepkcs11   | 568        |
| 4.1.9    | freertos-plus-tcp  | 568        |

This documentation contains information specific to the frdmk32l2a4s board.



# Chapter 1

## FRDM-K32L2A4S

### 1.1 Overview

The FRDM-K32L2A4S Freedom development platform provides an ideal board for evaluation and development of the K32 L2A MCU family based on the Arm Cortex-M0+ architecture. This platform includes an onboard debug probe, accelerometer/magnetometer, a full-speed USB and easy access to K32 L2A's MCU I/O.

The FRDM-K32L2A4S board is fully supported by the MCUXpresso suite of tools, which provides device drivers, middleware and examples to allow rapid development, plus configuration and an optional free of charge IDE.



MCU device and part on board is shown below:

- Device: K32L2A41A
- PartNumber: K32L2A41VLL1A

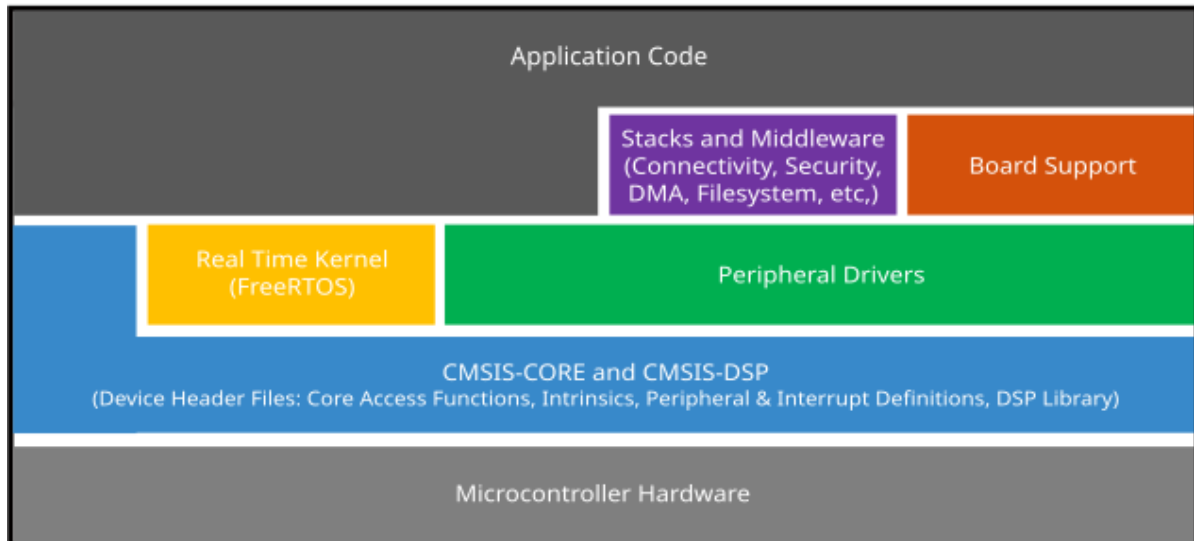
### 1.2 Getting Started with MCUXpresso SDK Package

#### 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.

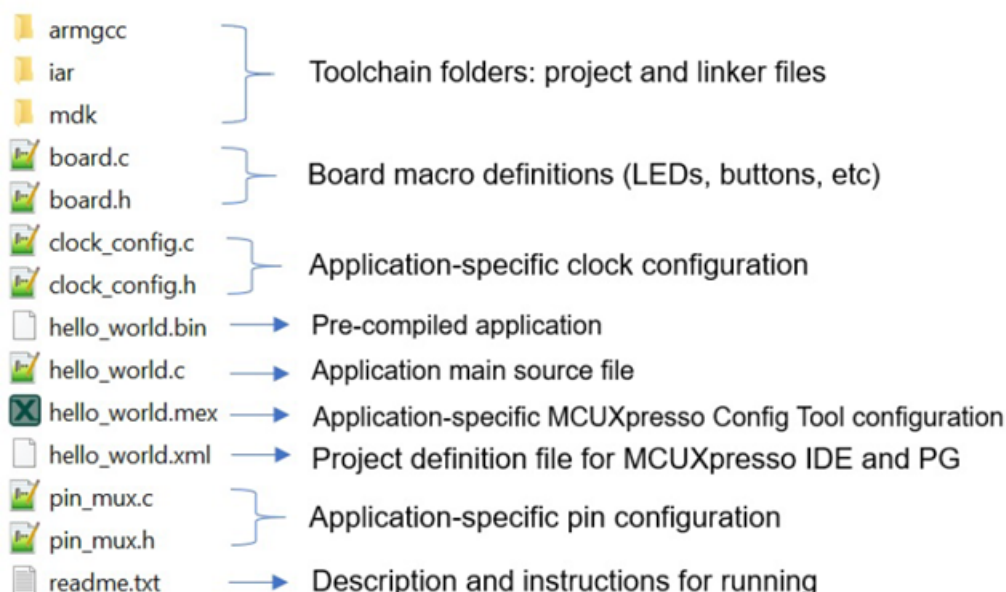
**Example application structure** This section describes how the various types of example applications interact with the other components in the MCUXpresso SDK. To get a comprehensive understanding of all MCUXpresso SDK components and folder structure, see *MCUXpresso SDK API Reference Manual*.

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` exam-



ple (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

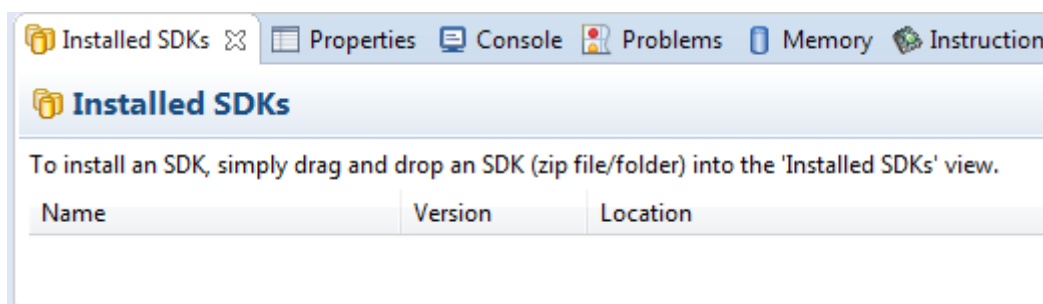
**Note:** Ensure that the MCUXpresso IDE toolchain is included when generating the MCUXpresso SDK package.

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.

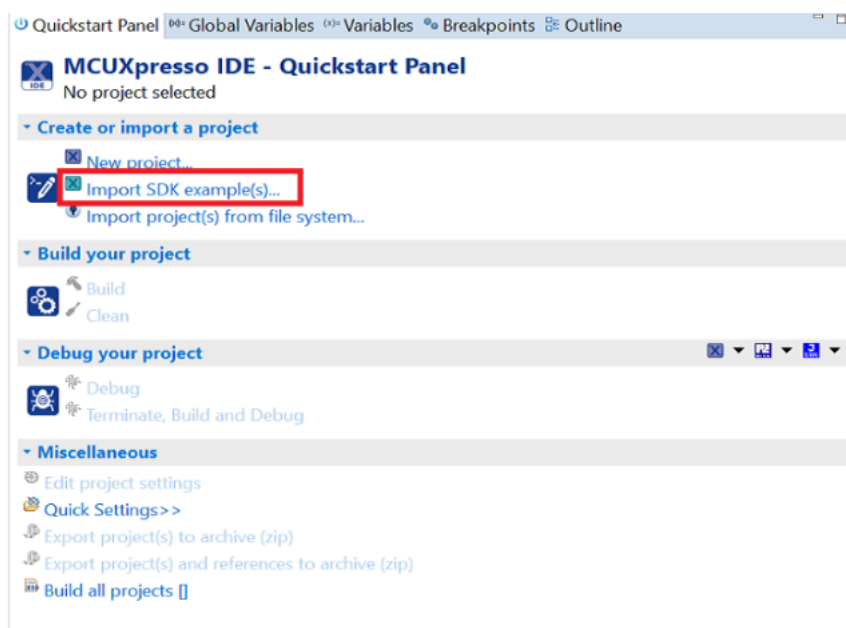
**Select the workspace location** Every time MCUXpresso IDE launches, it prompts the user to select a workspace location. MCUXpresso IDE is built on top of Eclipse which uses workspace to store information about its current configuration, and in some use cases, source files for the projects are in the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

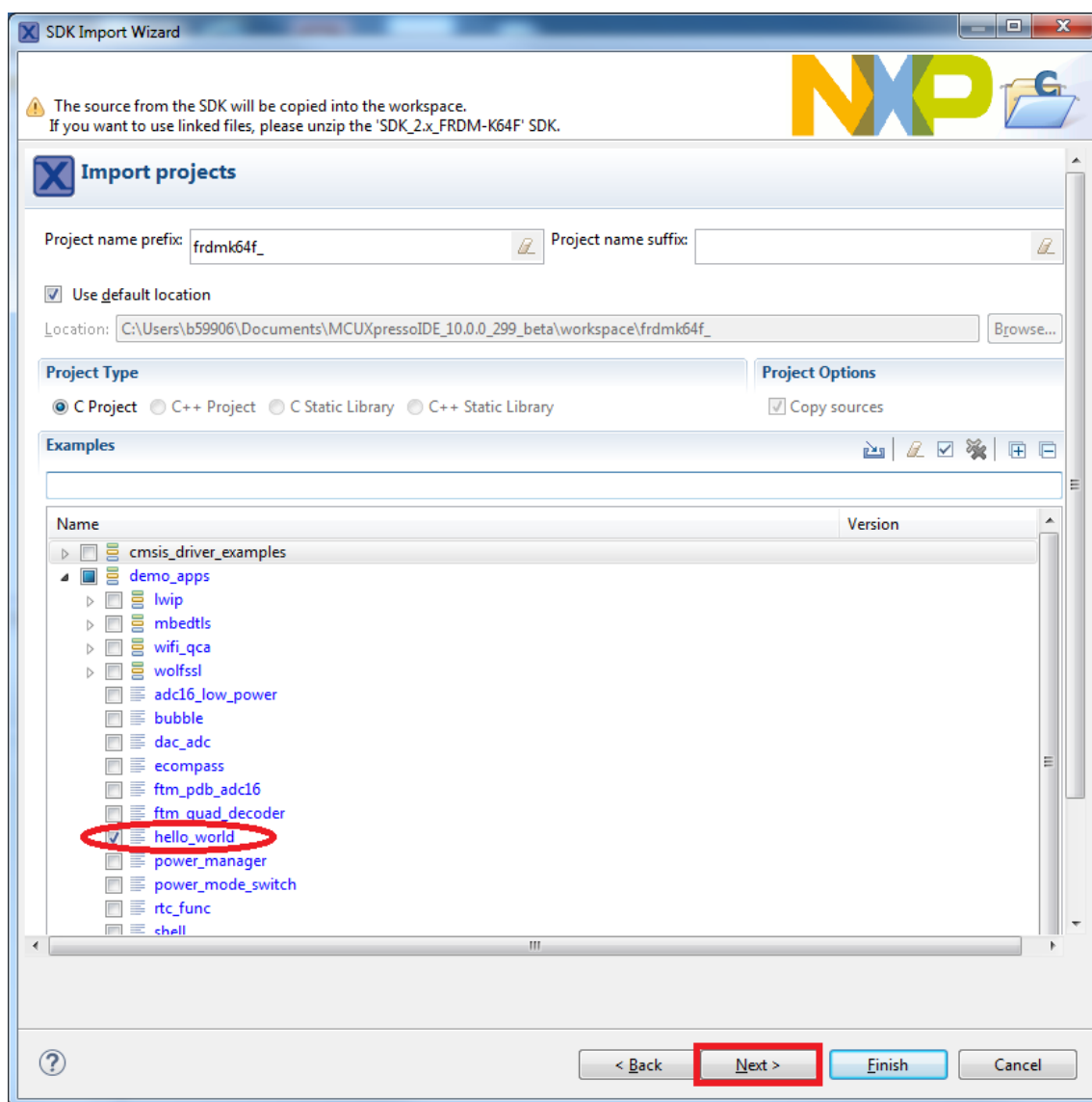
1. Drag and drop the SDK zip file into the **Installed SDKs** view to install an SDK. In the window that appears, click **OK** and wait until the import has finished.



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



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



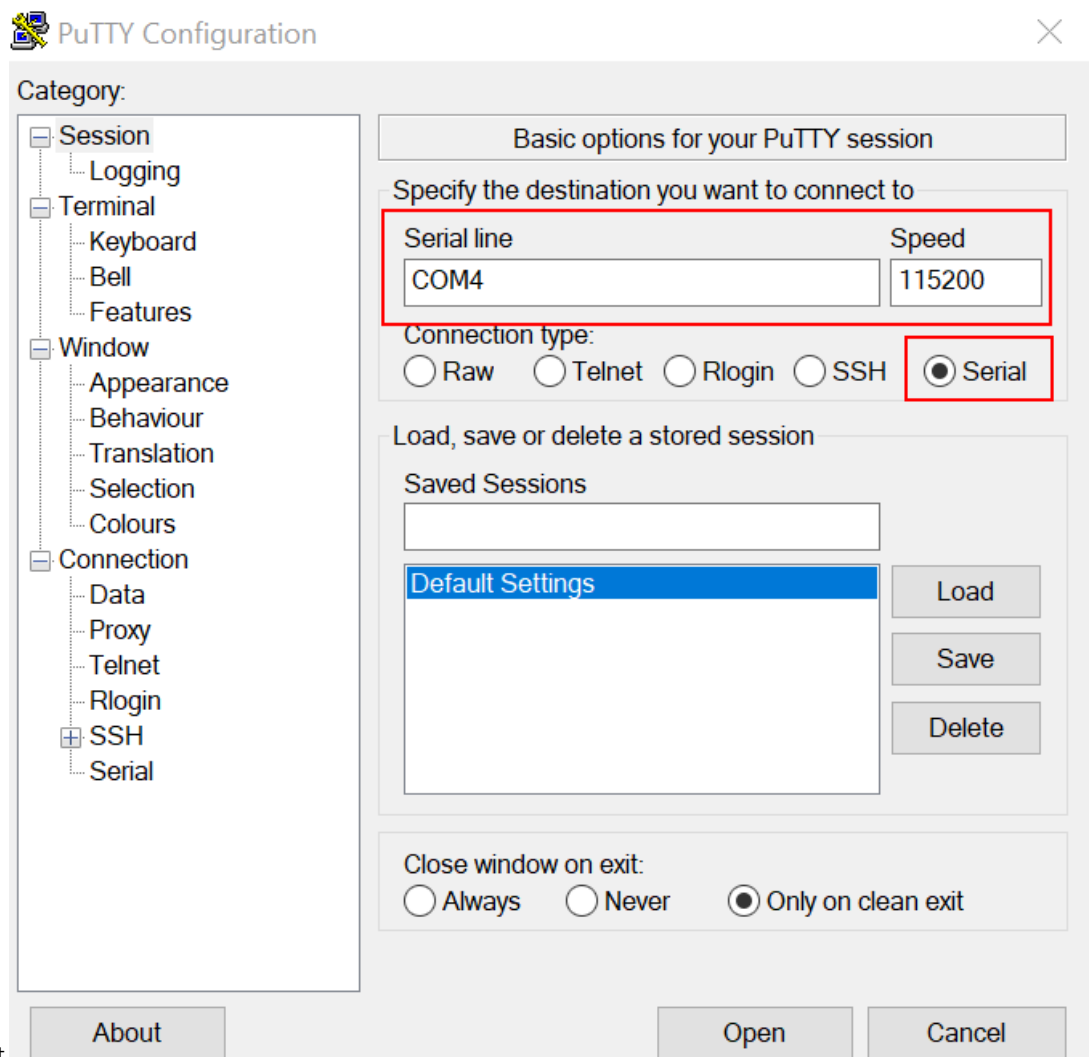
5. Ensure **Redlib: Use floating-point version of printf** is selected if the example prints floating-point numbers on the terminal for demo applications such as `adc_basic`, `adc_burst`, `adc_dma`, and `adc_interrupt`. Otherwise, it is not necessary to select this option. Then, click **Finish**.

**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:

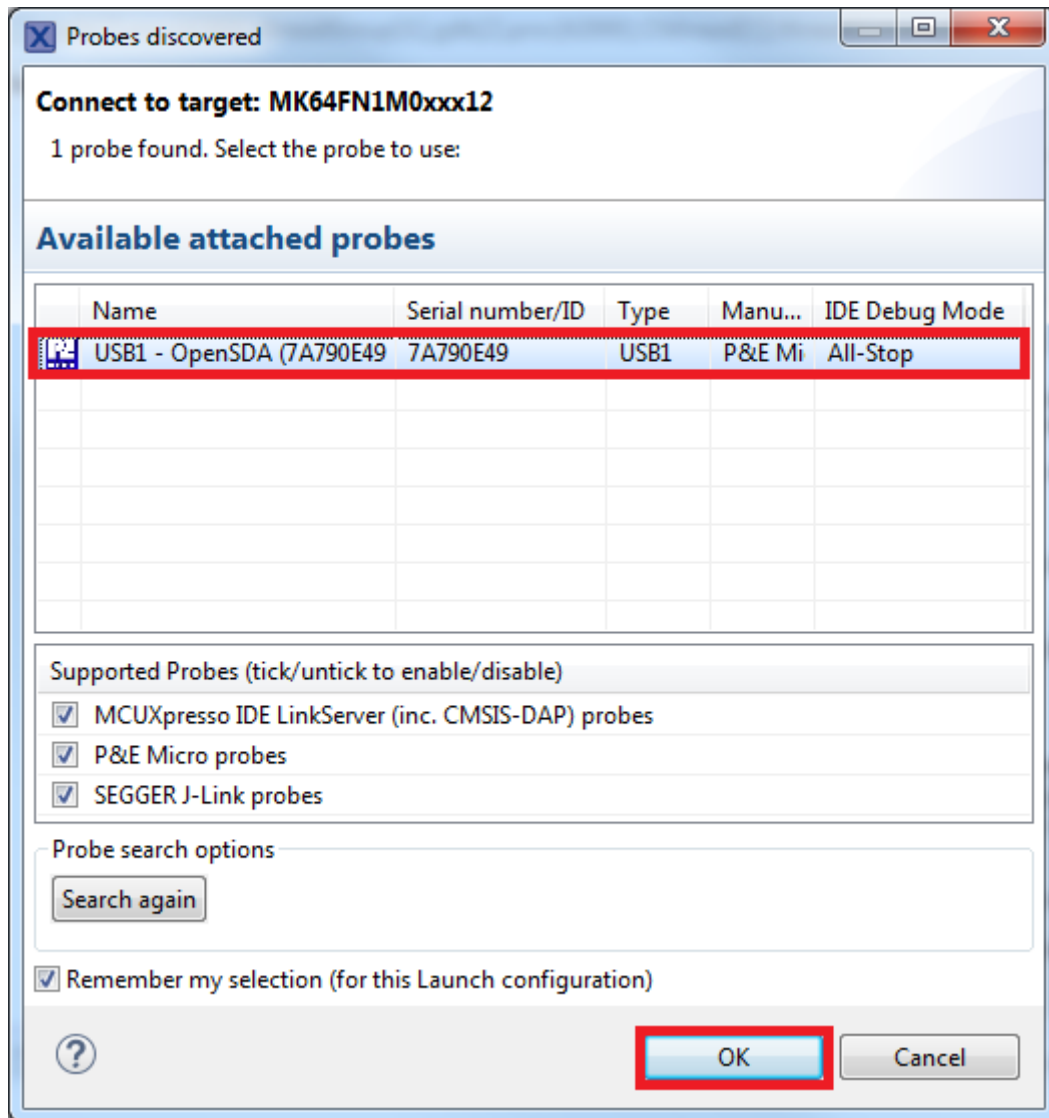
1. Ensure the host driver for the debugger firmware has been installed. See [On-board debugger](#).
2. Connect the development platform to your PC via a USB cable.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM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

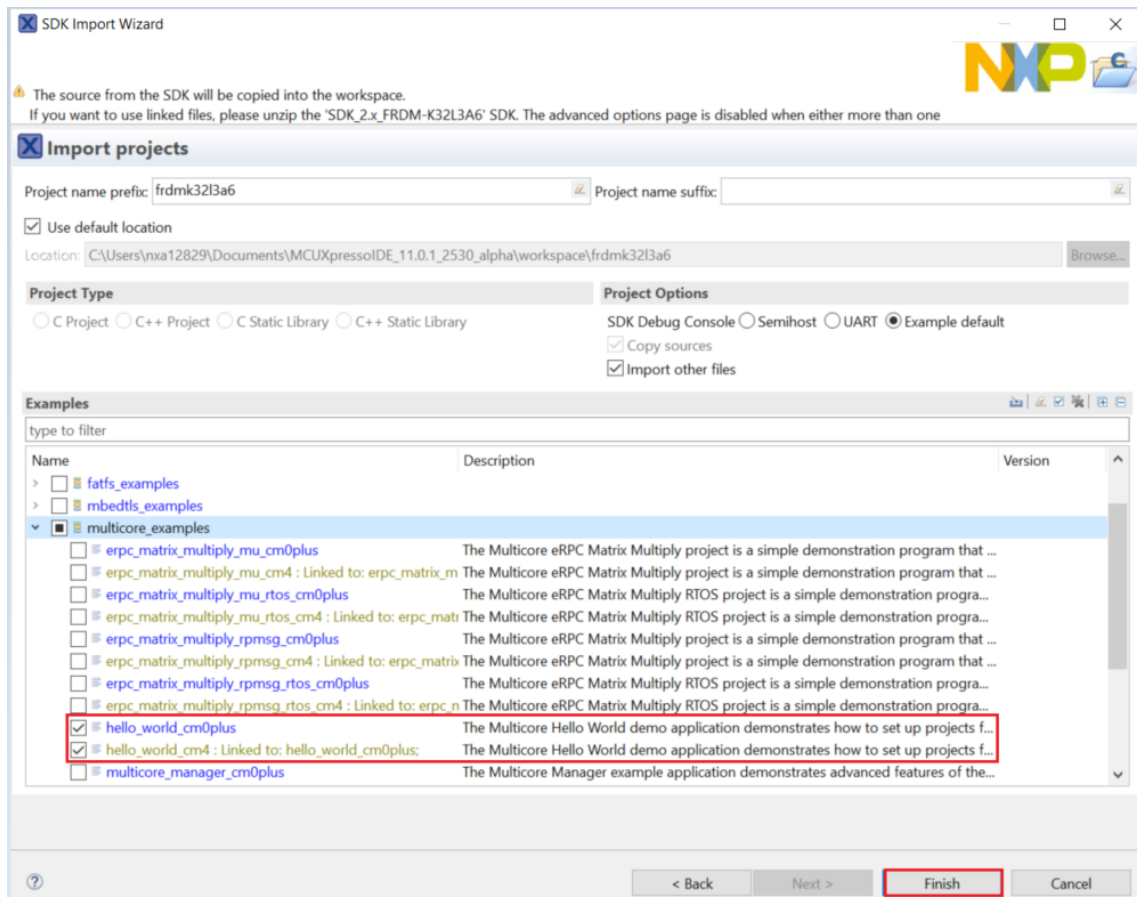


The `hello_world` application is now running and a banner is displayed on the terminal. If not, check your terminal settings and connections.

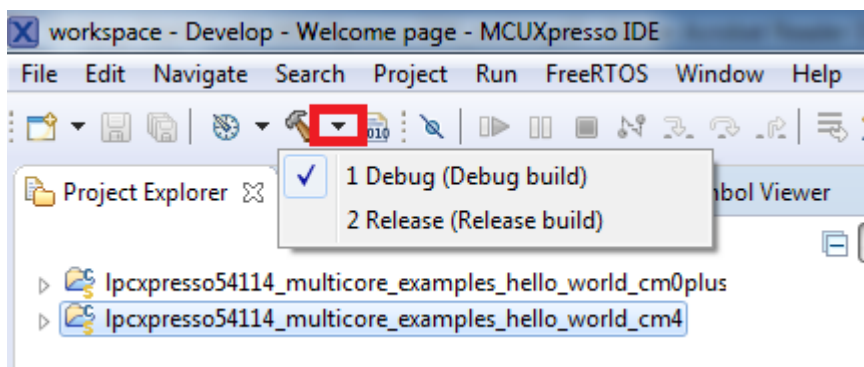


**Build a multicore example application** This section describes the steps required to configure MCUXpresso IDE to build, run, and debug multicore example applications. The following steps can be applied to any multicore example application in the MCUXpresso SDK. Here, the dual-core version of hello\_world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



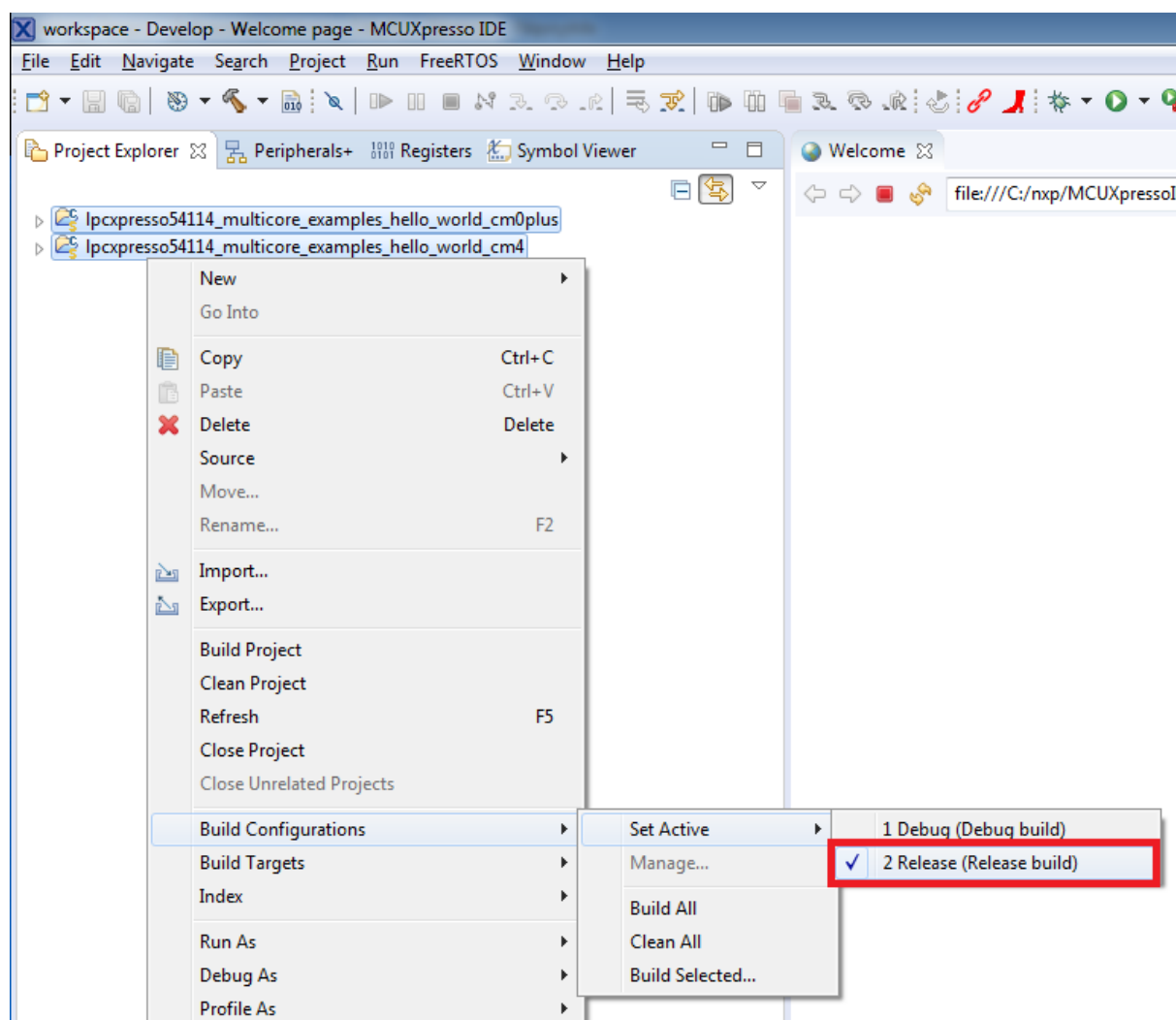
3. Now, two projects should be imported into the workspace. To start building the multicore application, highlight the `lpcxpresso54114_multicore_examples_hello_world_cm4` project (multicore master project) in the Project Explorer. Then choose the appropriate build target, **Debug** or **Release**, by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



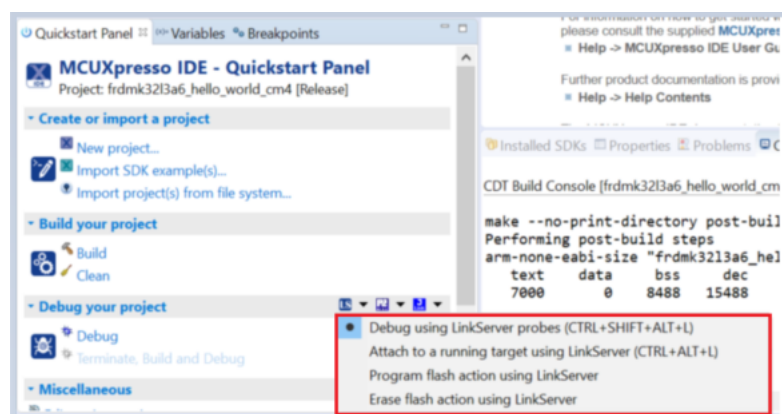
The project starts building after the build target is selected. Because of the project reference settings in multicore projects, triggering the build of the primary core application (cm4) also causes the referenced auxiliary core application (cm0plus) to build.

**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.

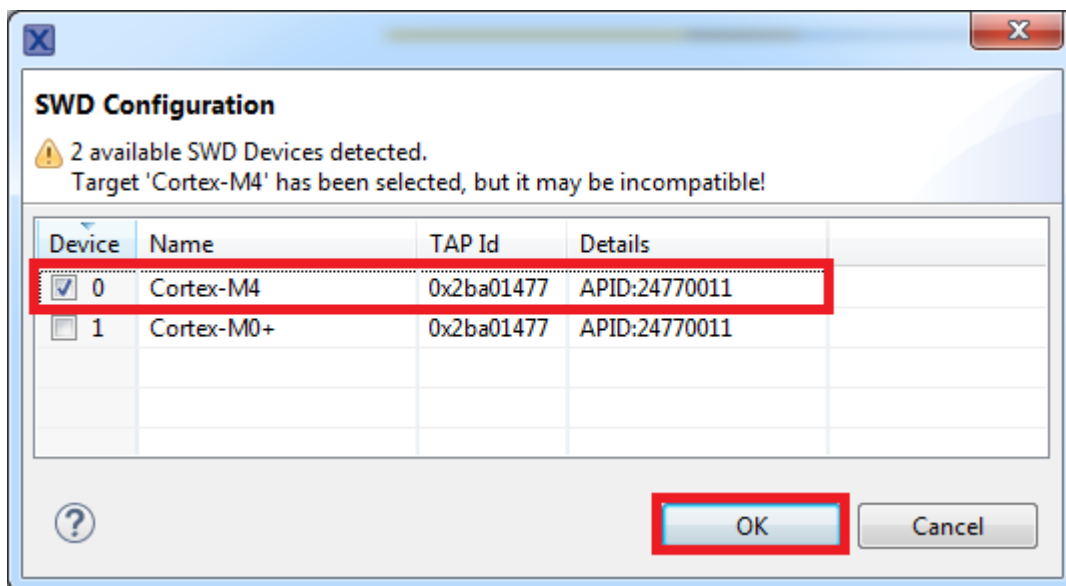
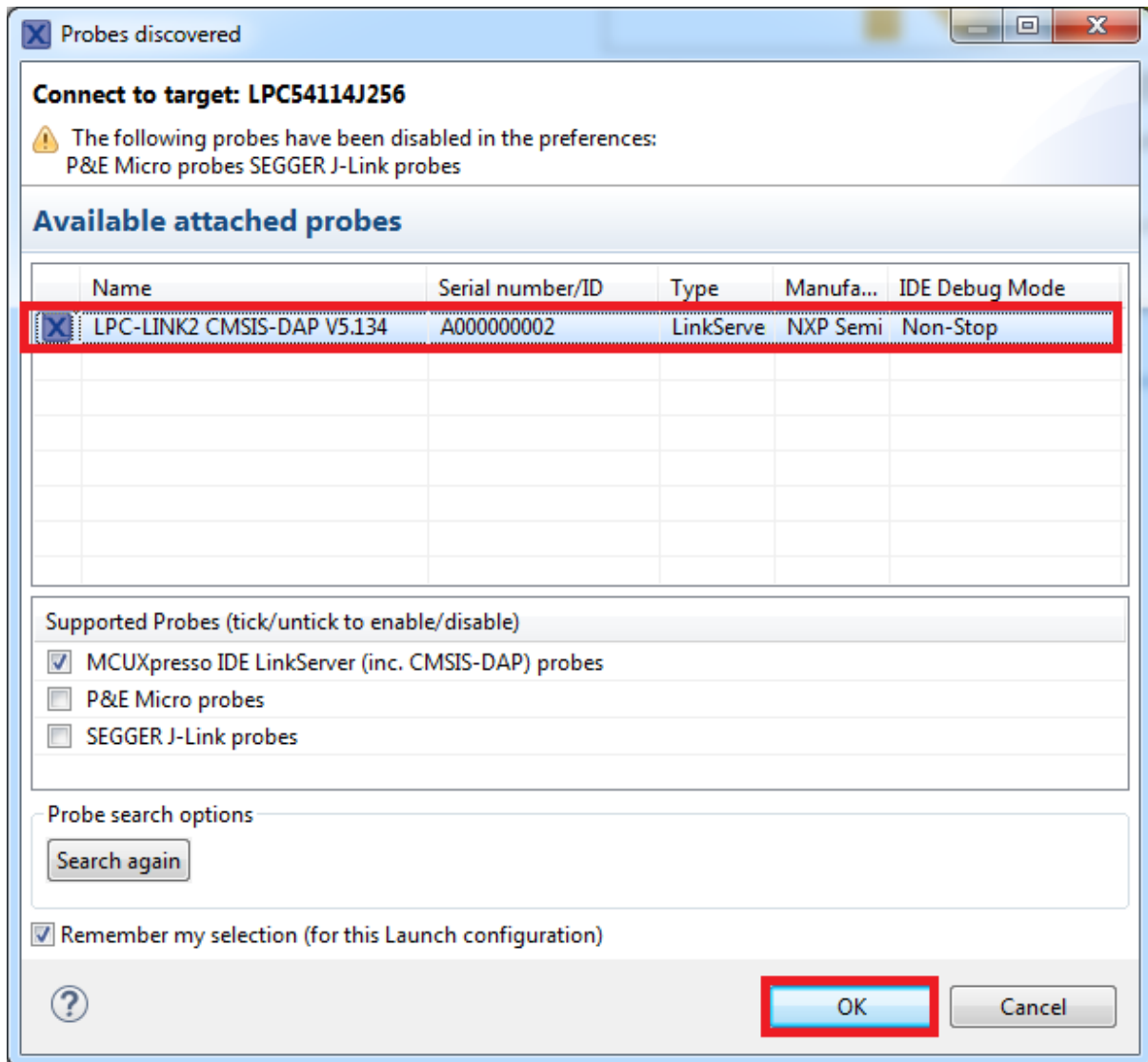


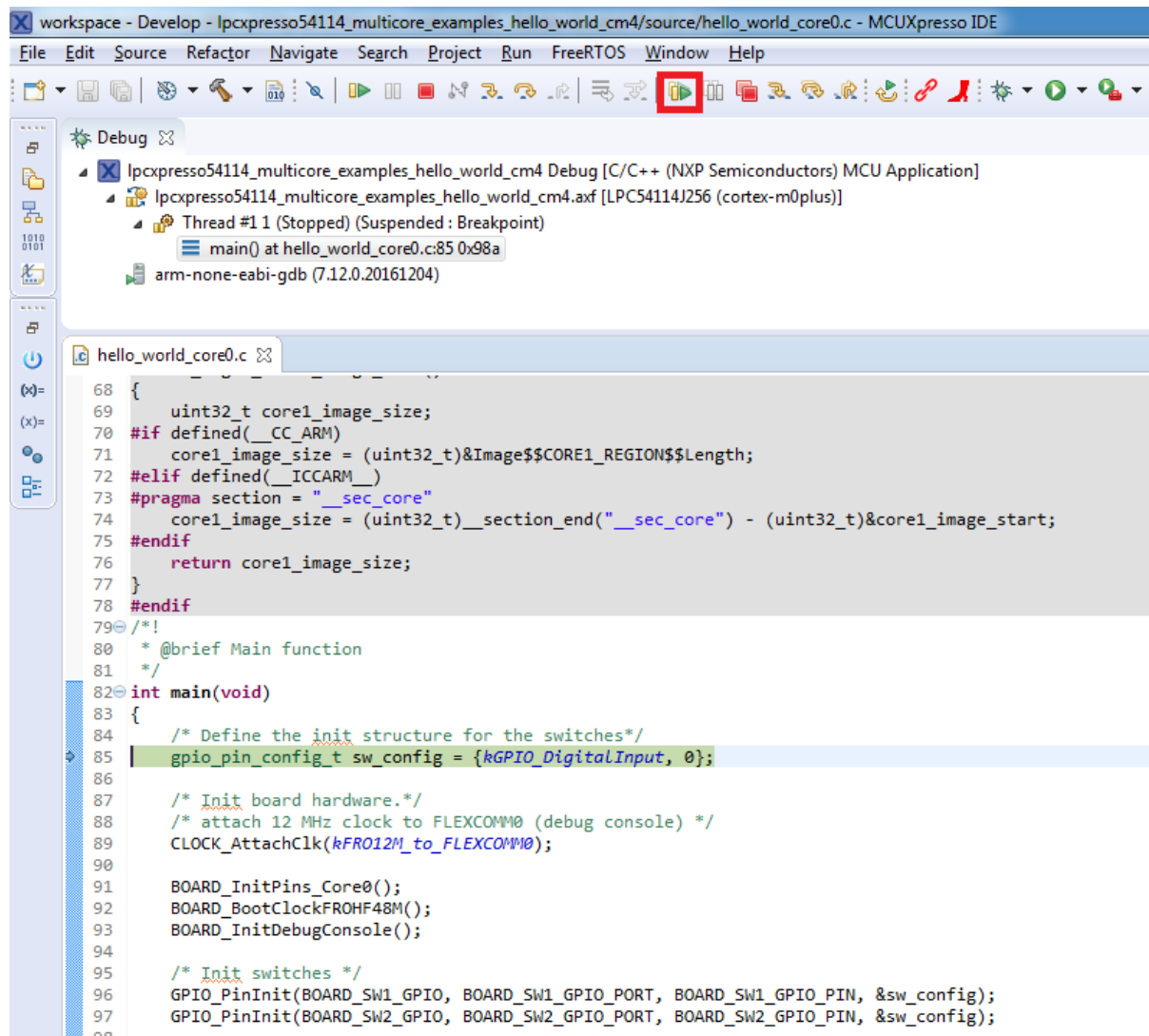


**Run a multicore example application** The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform all steps as described in **Run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

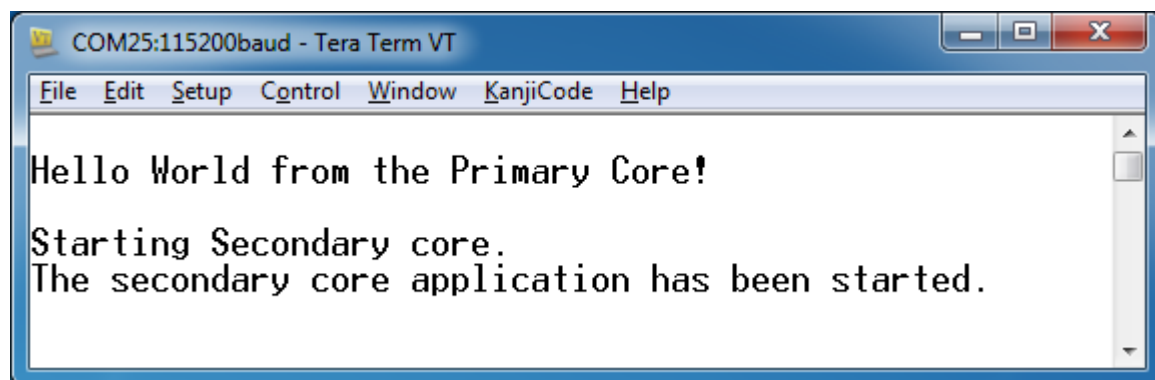






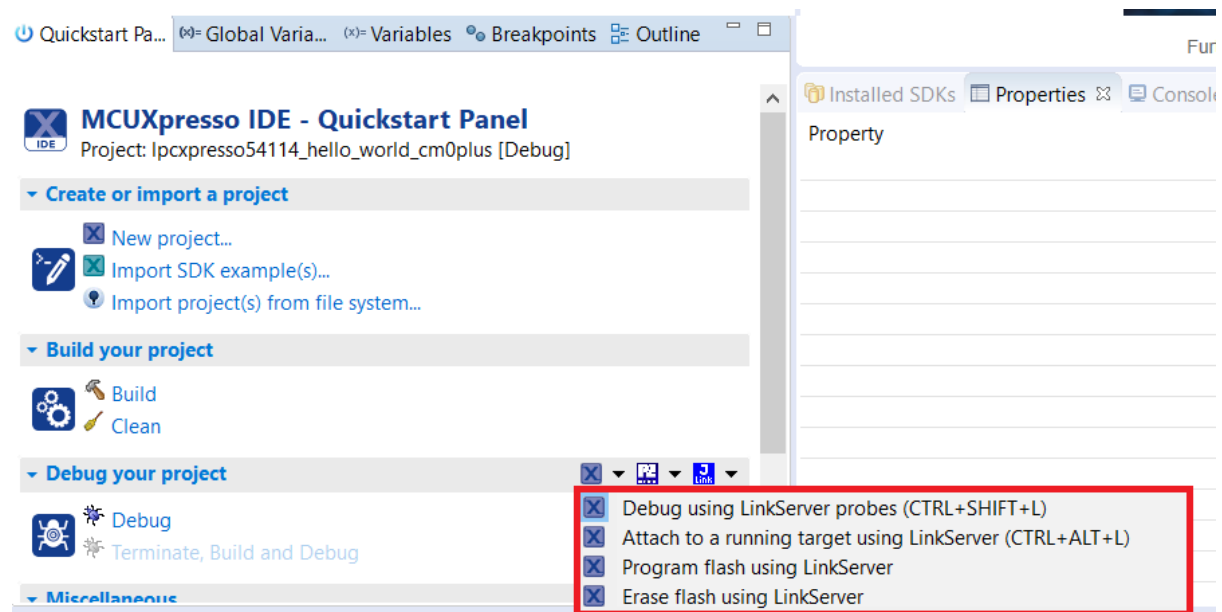


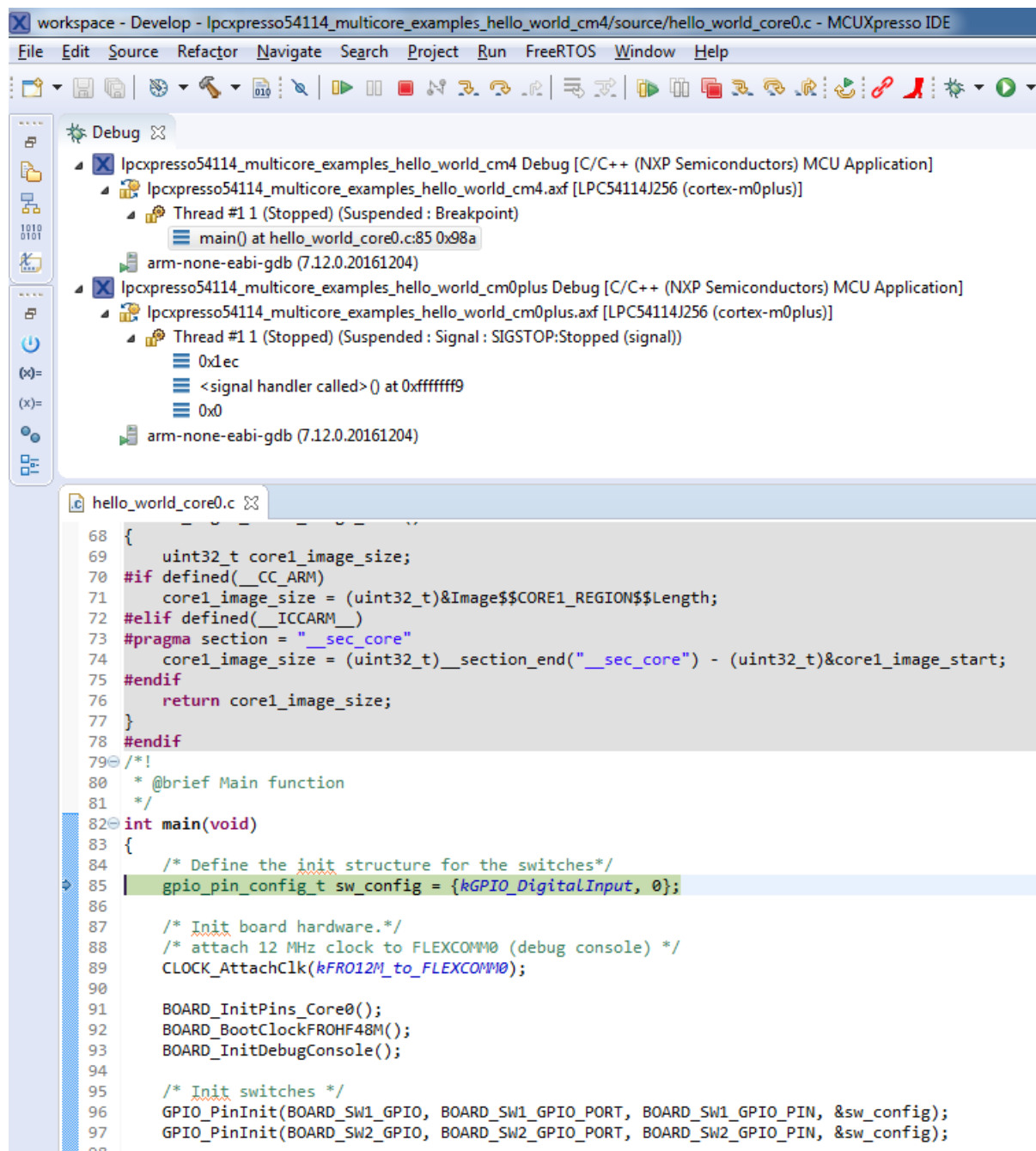
After clicking the “Resume All Debug sessions” button, the hello\_world multicore application runs and a banner is displayed on the terminal. If this is not the case, check your terminal settings and connections.



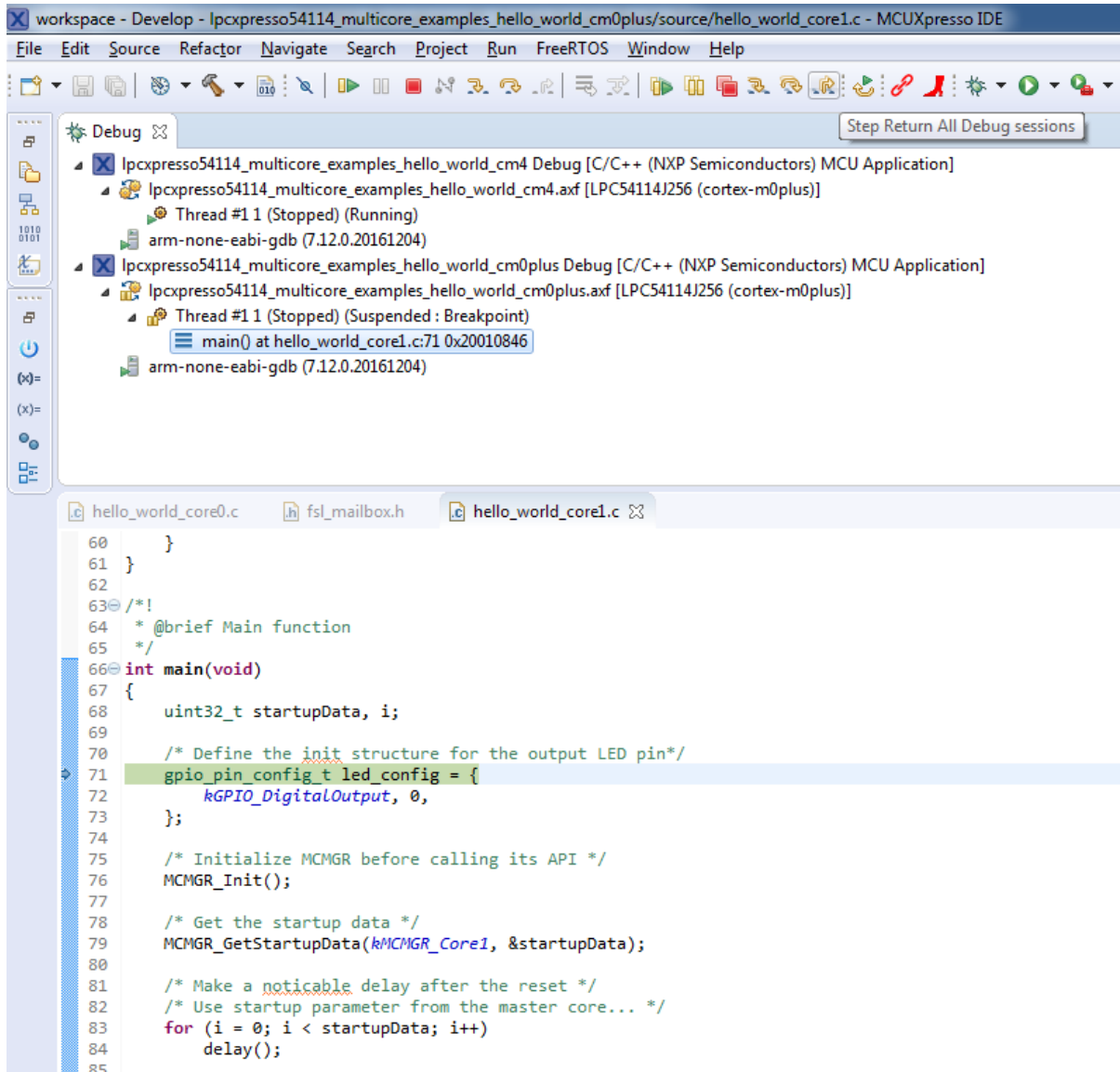
An LED controlled by the auxiliary core starts flashing, indicating that the auxiliary core has been released from the reset and running correctly. It is also possible to debug both sides of the multicore application in parallel. After creating the debug session for the primary core, perform same steps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

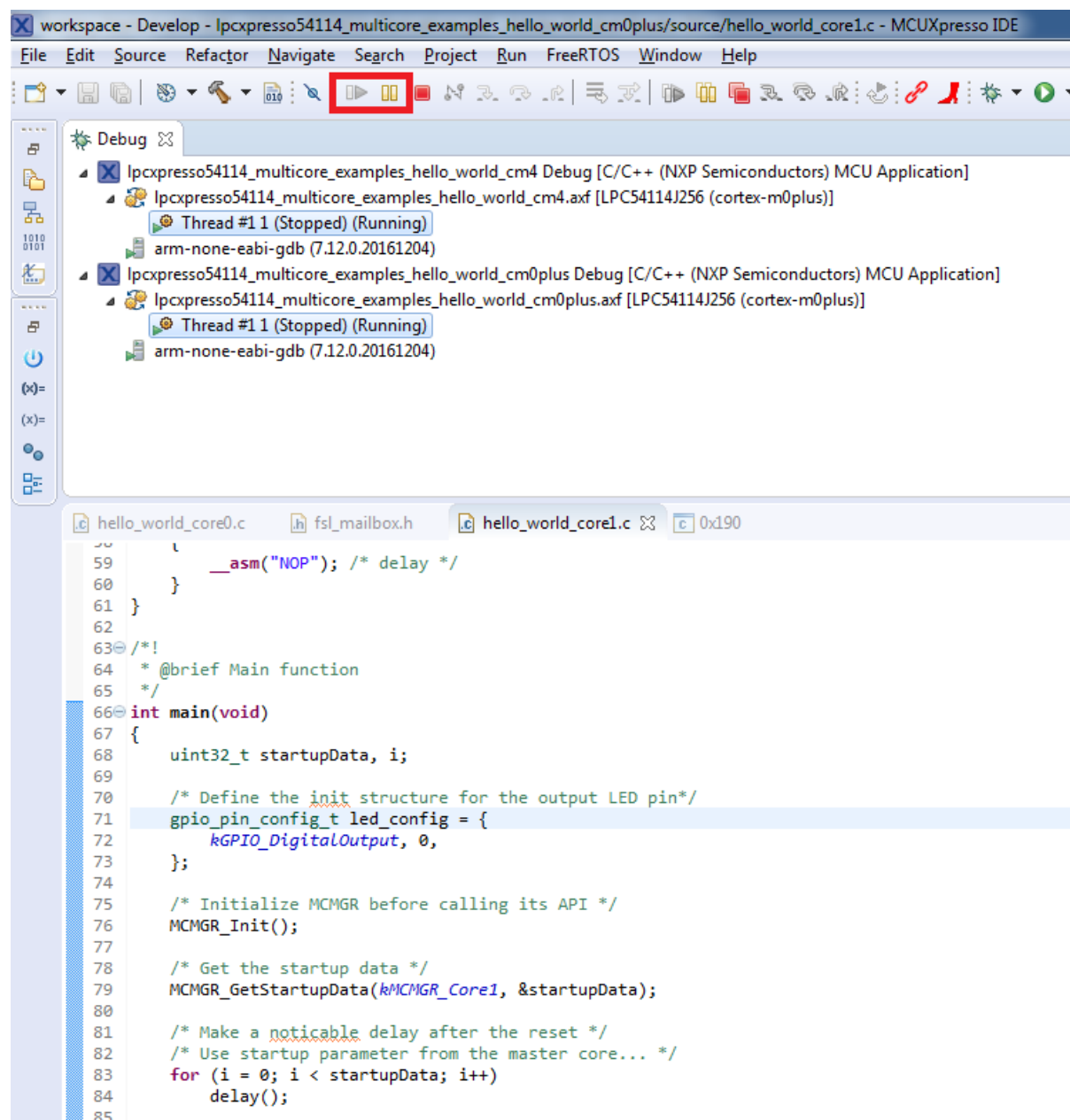


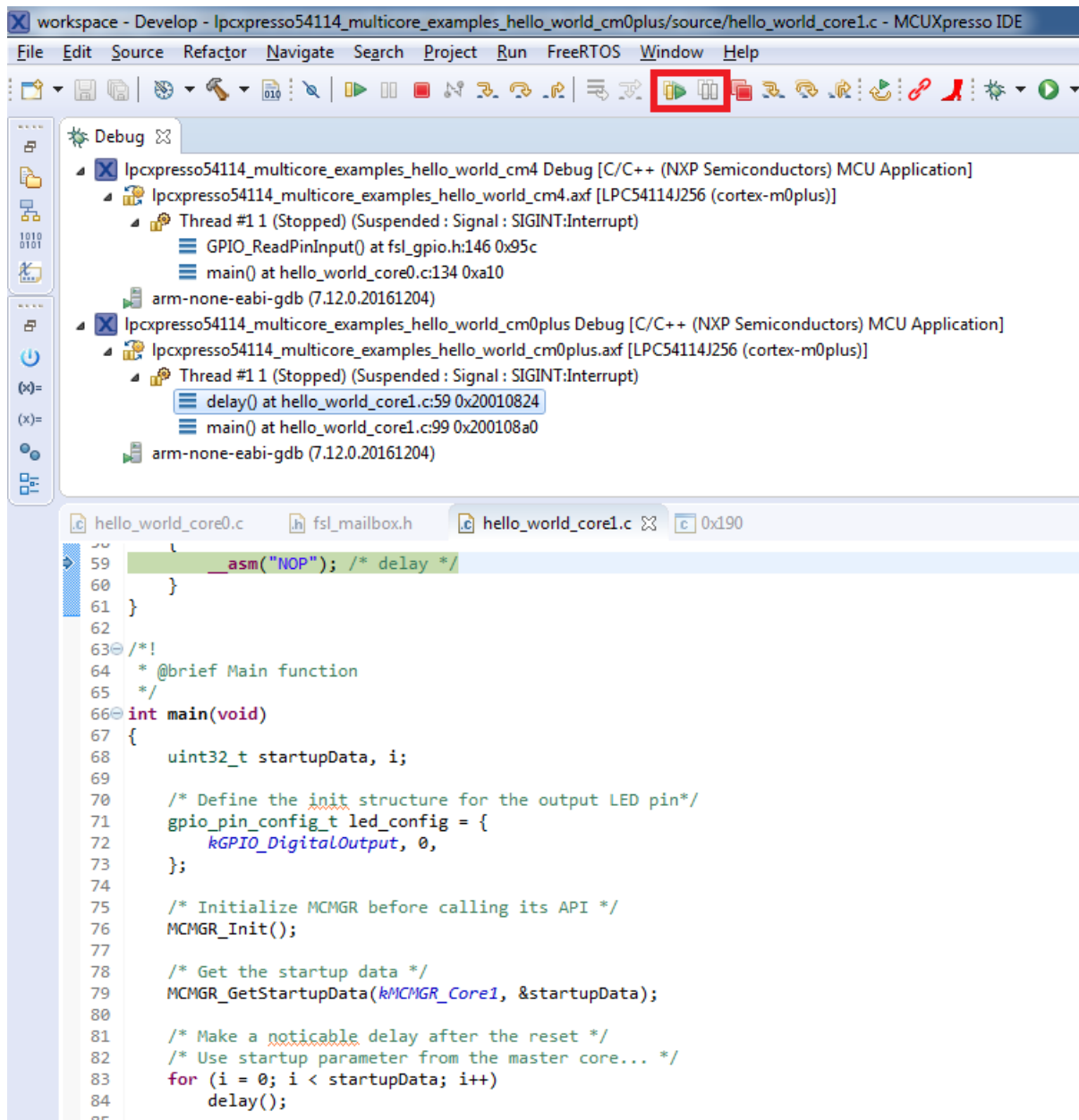


Now, the two debug sessions should be opened, and the debug controls can be used for both debug sessions depending on the debug session selection. Keep the primary core debug session selected by clicking the “Resume” button. The hello\_world multicore application then starts running. The primary core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

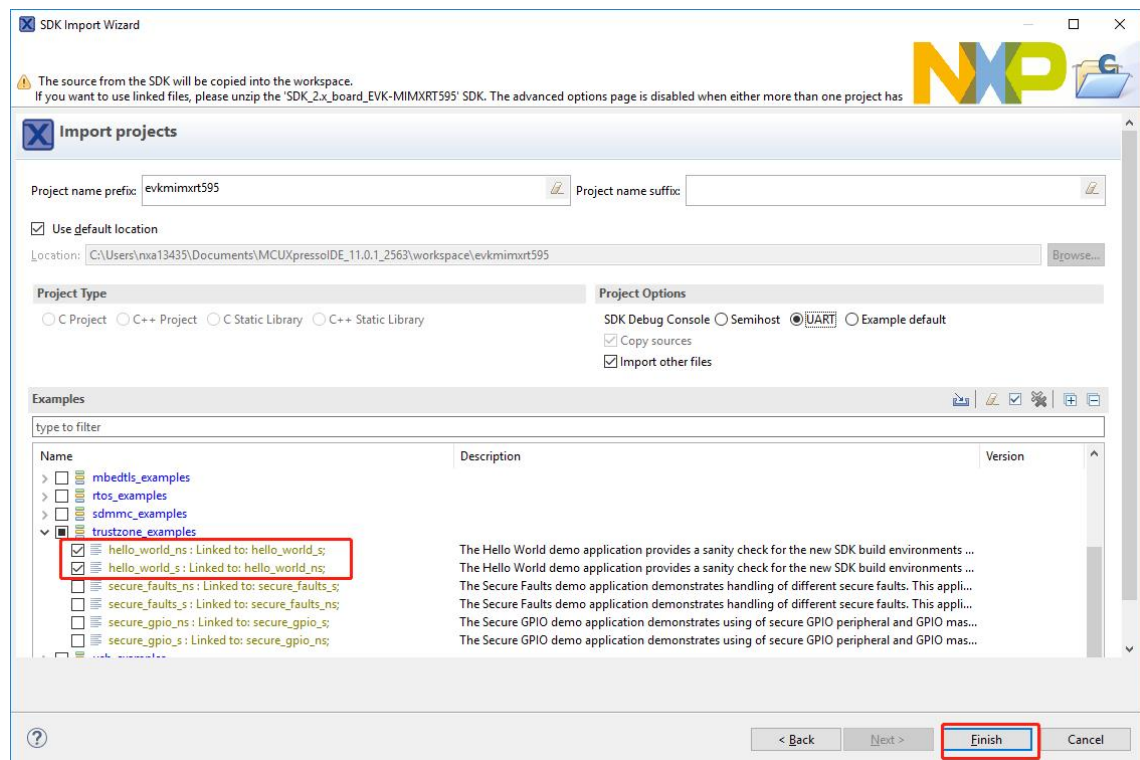




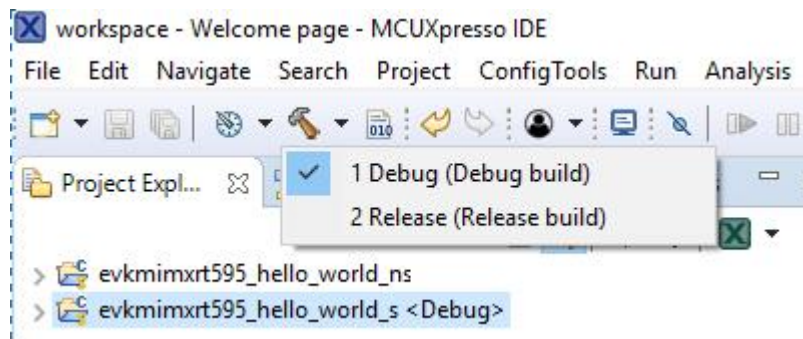
**Build a TrustZone example application** This section describes the steps required to configure MCUXpresso IDE to build, run, and debug TrustZone example applications. The TrustZone version of the hello\_world example application targeted for the MIMXRT595-EVK hardware platform is used as an example,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





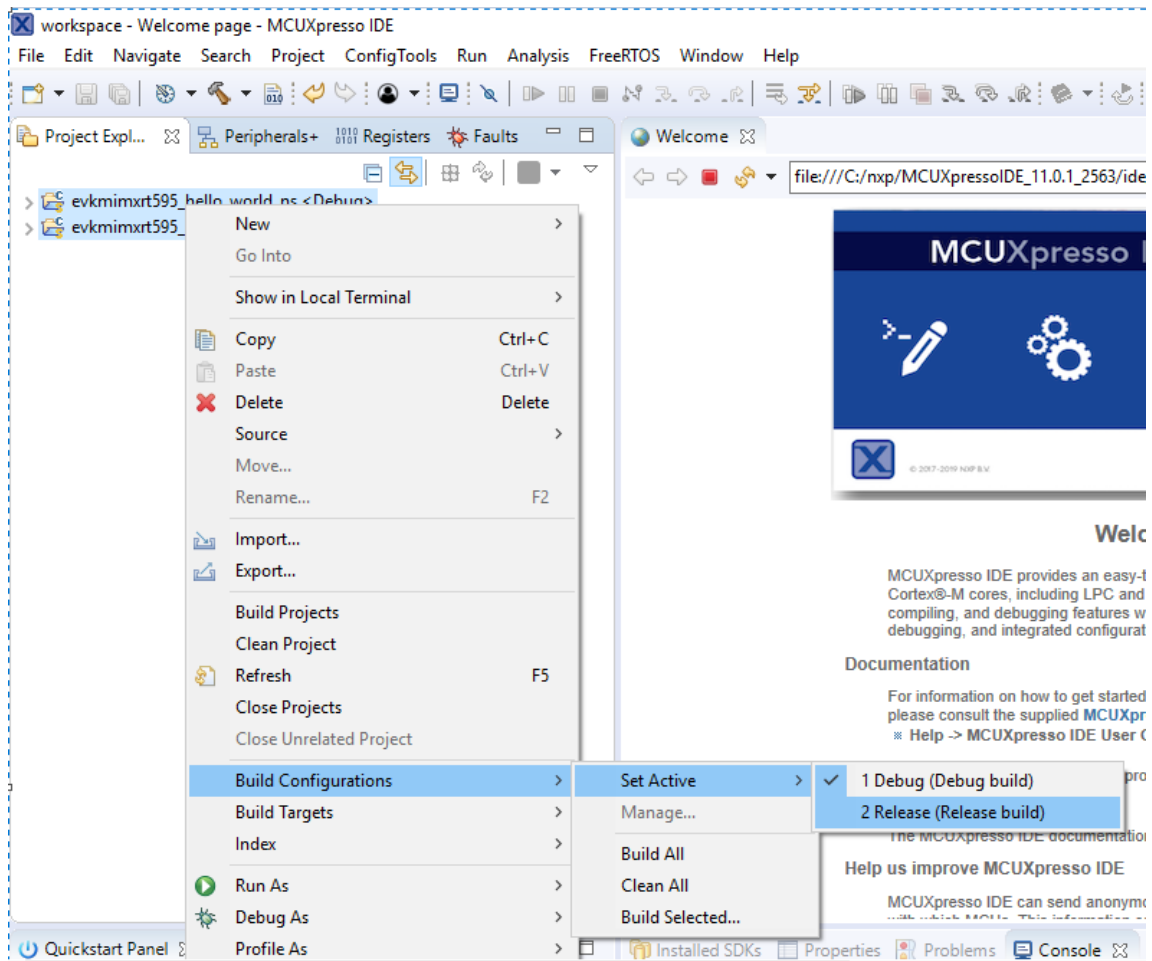
- Now, two projects should be imported into the workspace. To start building the TrustZone application, highlight the `evkmimxrt595_hello_world_s` project (TrustZone master project) in the Project Explorer. Then, choose the appropriate build target, **Debug** or **Release**, by clicking the downward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

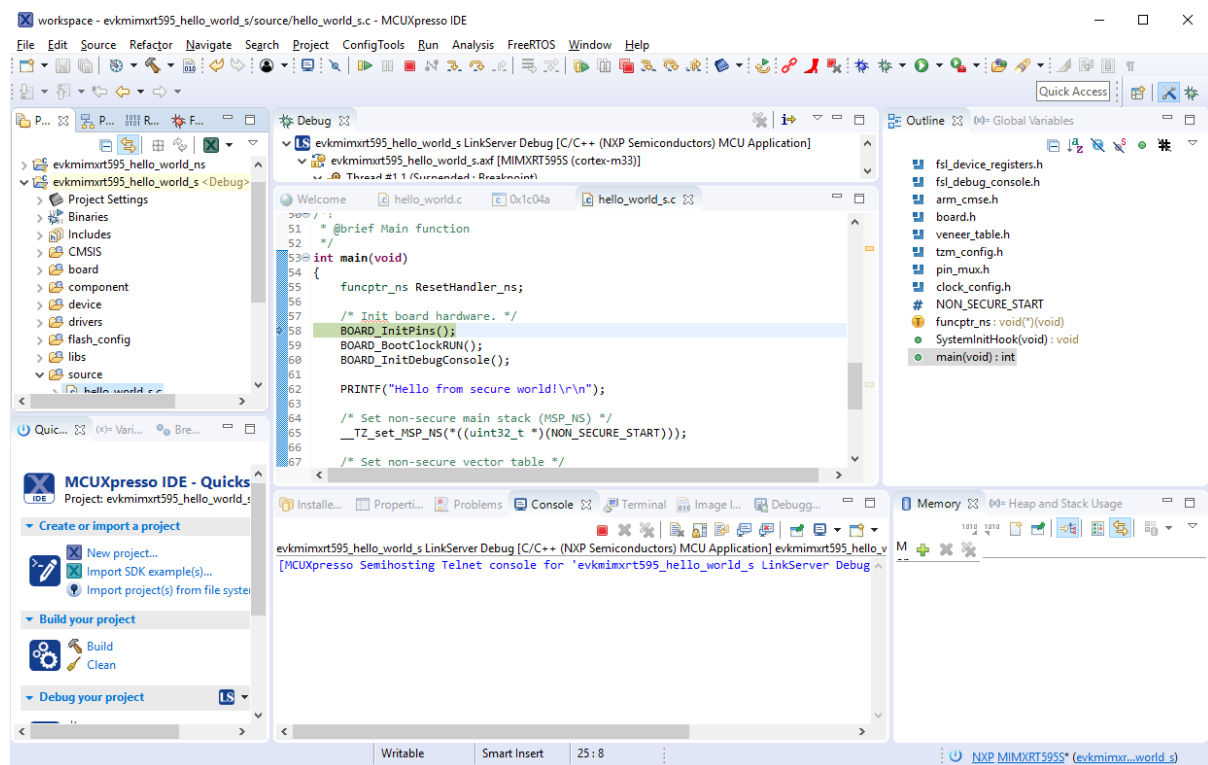
**Note:** When the **Release** build is requested, it is necessary to change the build configuration of both the secure and non-secure application projects first. To do this, select both projects in the Project Explorer view by clicking to select the first project, then using shift-click or control-click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

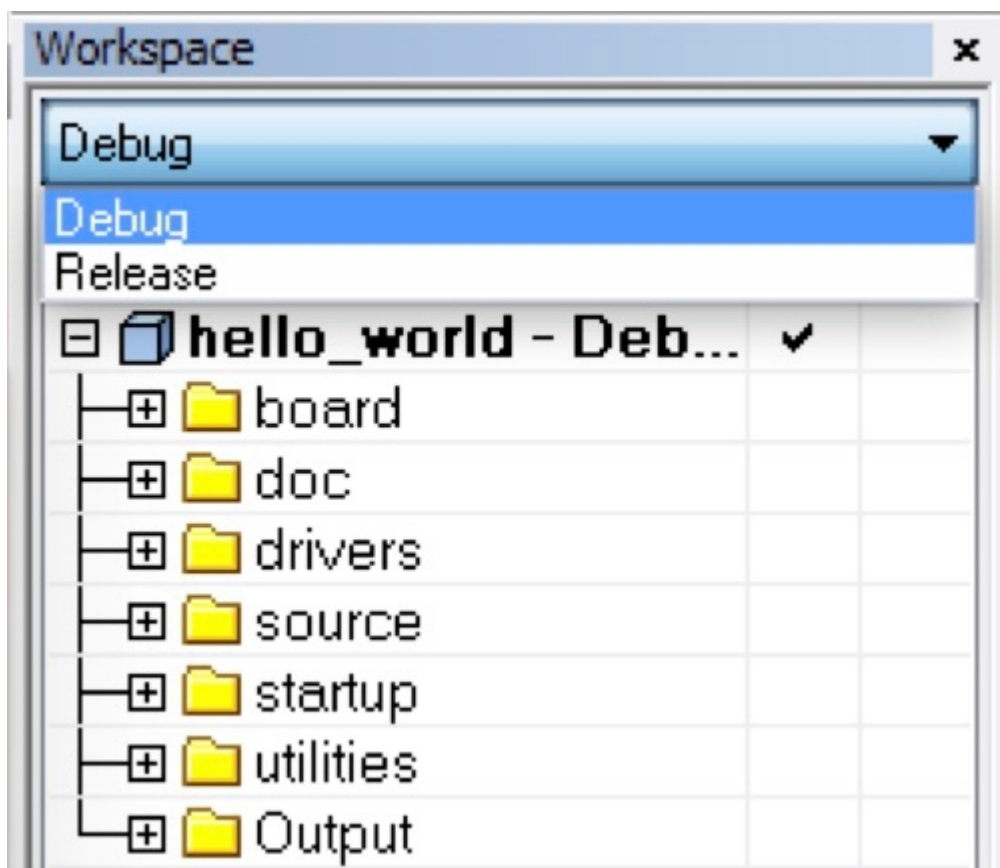
1. Open the desired demo application workspace. Most example application workspace files can be located using the following path:

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

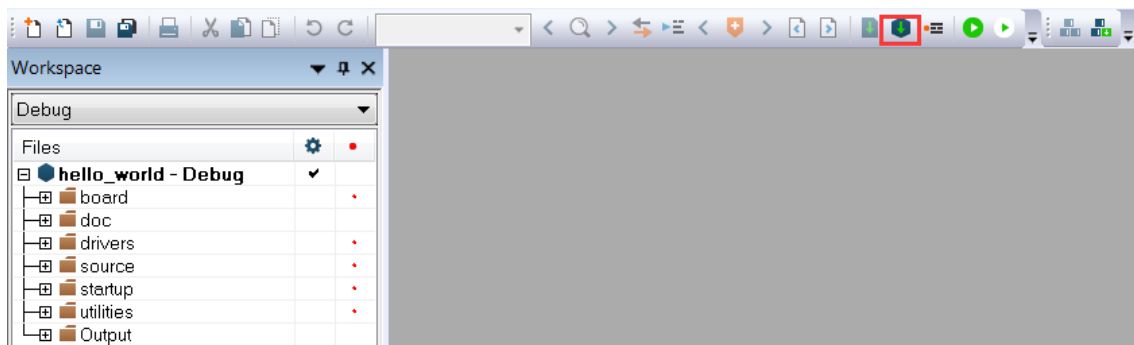
Other example applications may have additional folders in their path.

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

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



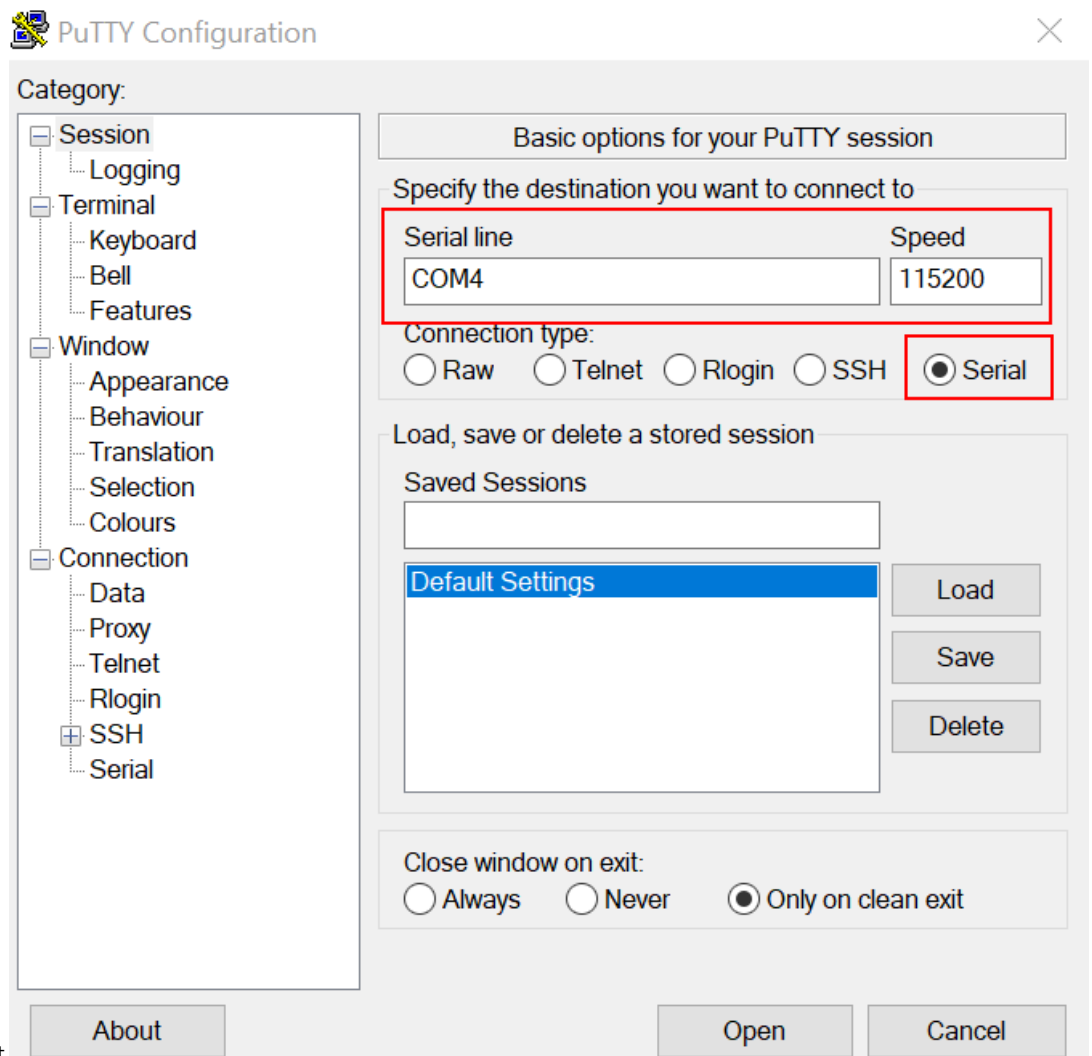
3. To build the demo application, click **Make**, highlighted in red in following figure.



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

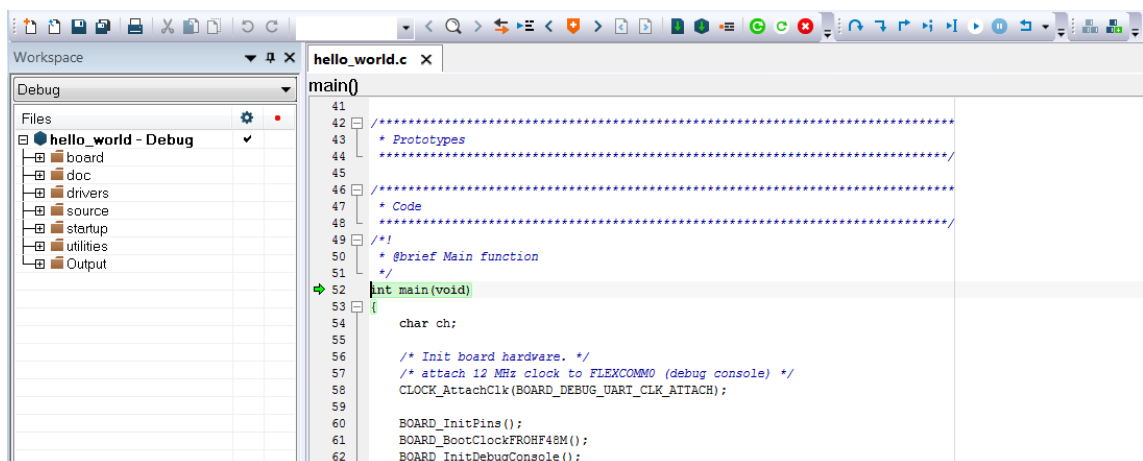


4. 1 stop bit

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
```

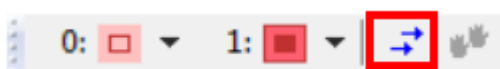
```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm4/iar/hello_world_cm4.eww
```

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.

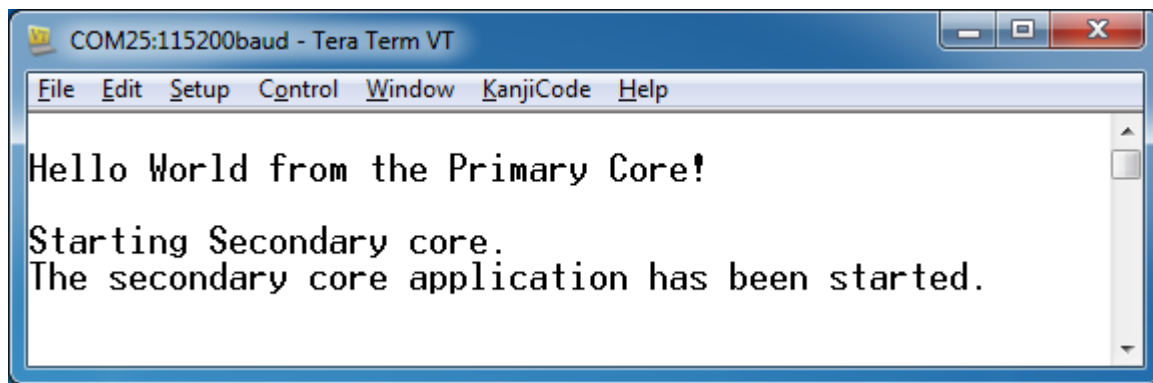
**Run a multicore example application** The primary core debugger handles flashing both primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 4 as described in **Run an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

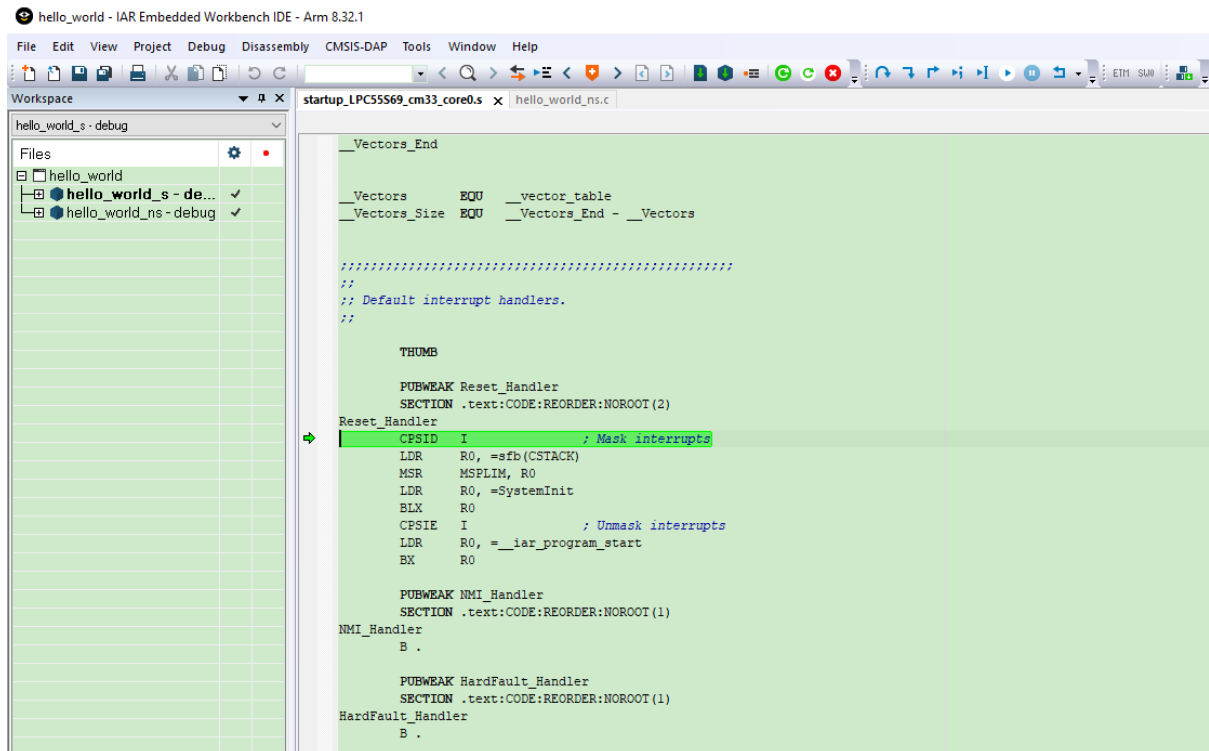
```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_ns/iar/hello_world_  
↪ns.eww
```

```
<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
```

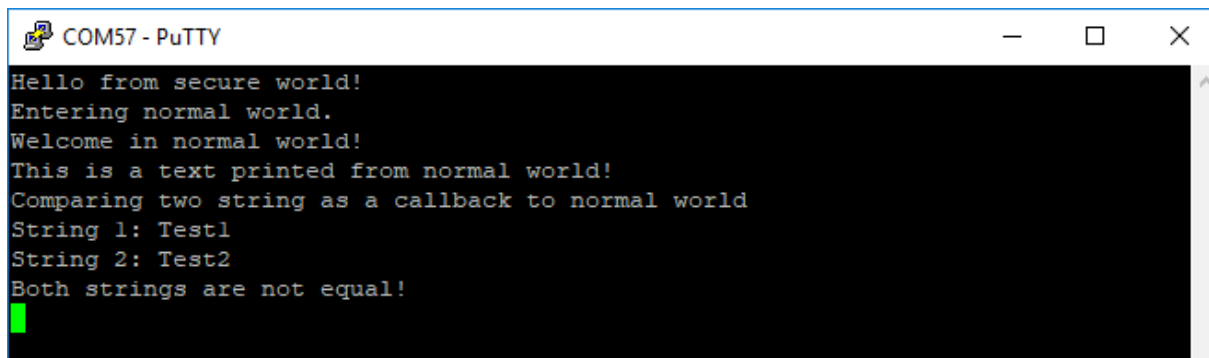
This project `hello_world.eww` contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another. Build both applications separately by clicking **Make**. It is requested to build the application for the secure project first, because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



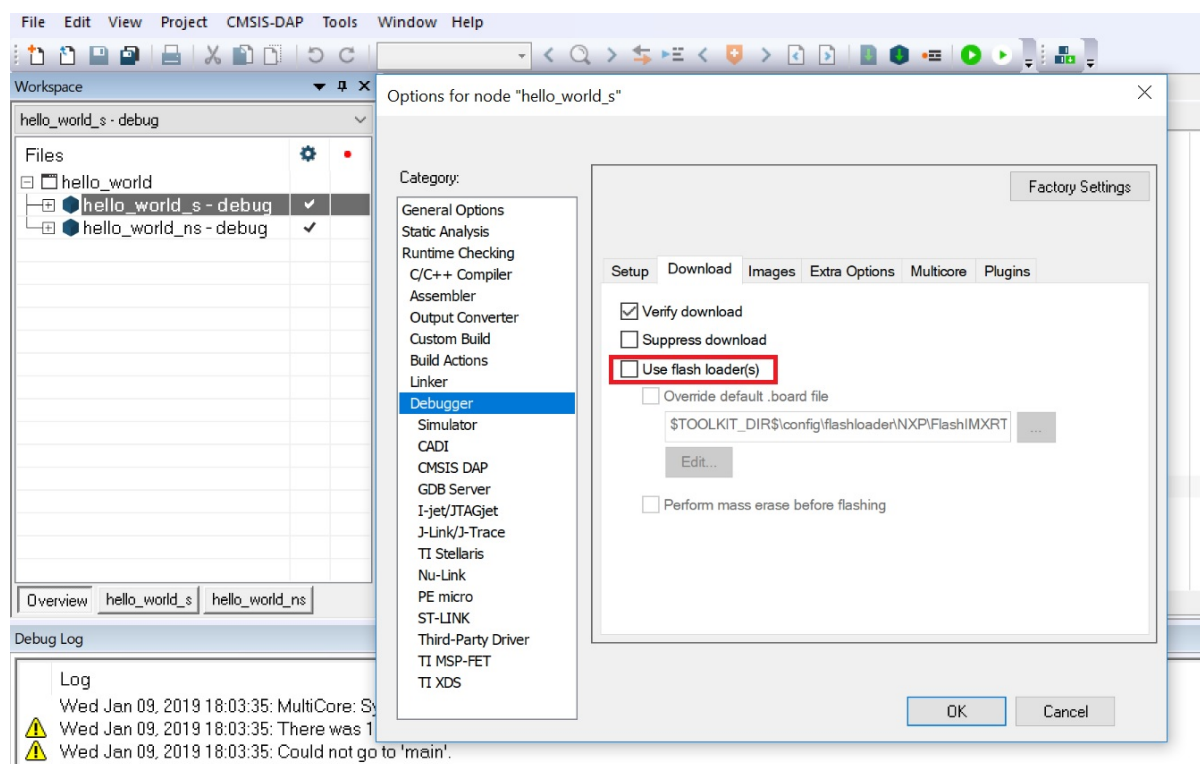
Run the code by clicking **Go** to start the application.

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.



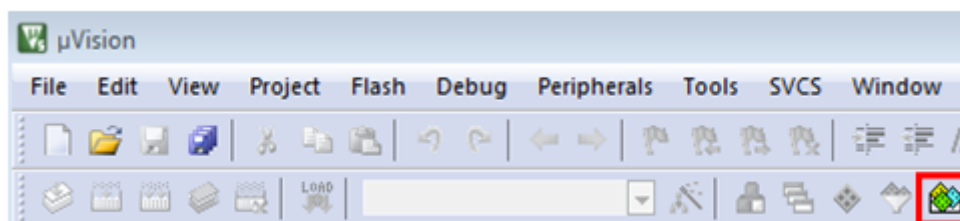


## Run a demo using Keil MDK/μVision

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:



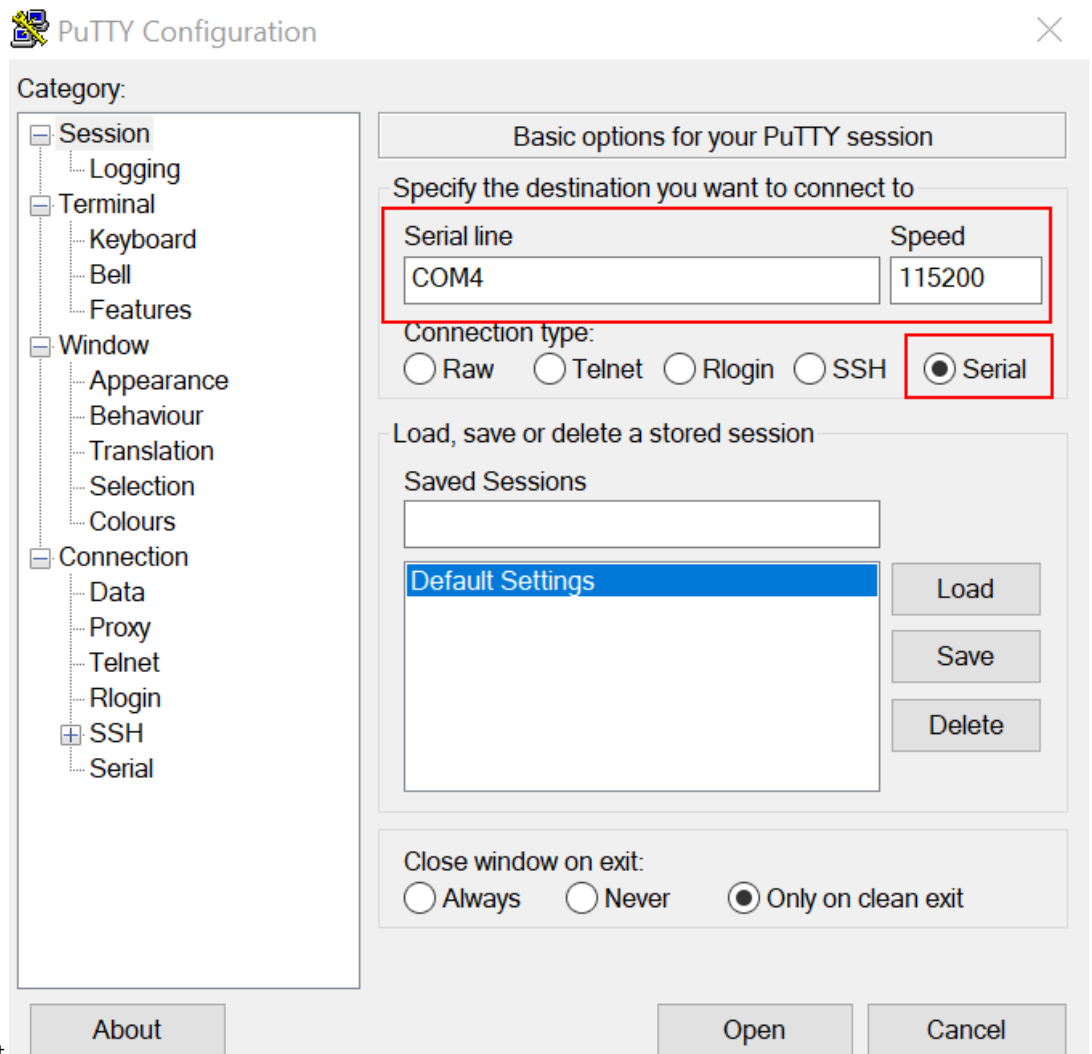
- To build the demo project, select **Rebuild**, highlighted in red.



- The build completes without errors.

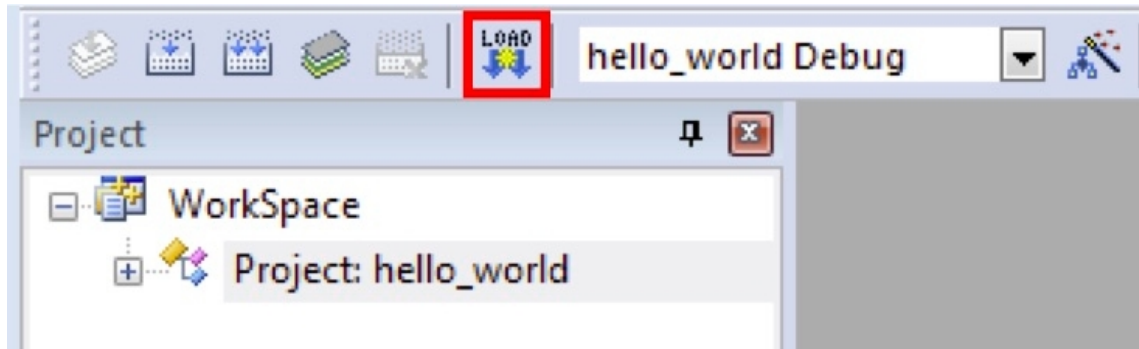
**Run an example application** To download and run the application, perform these steps:

- 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

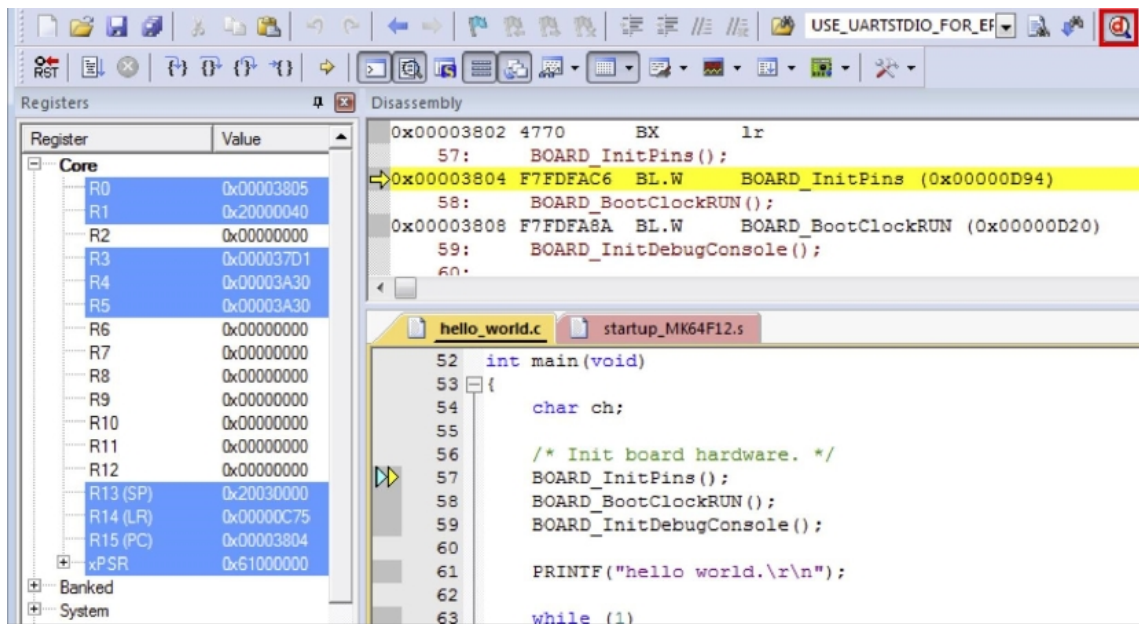


- 1 stop bit

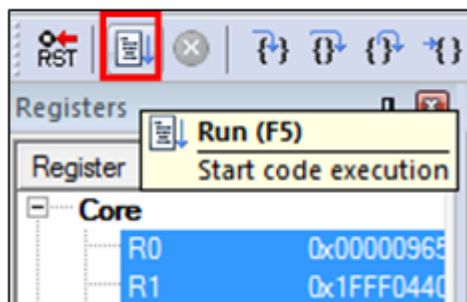
- In  $\mu$ Vision, after the application is built, click the **Download** button to download the application to the target.



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:

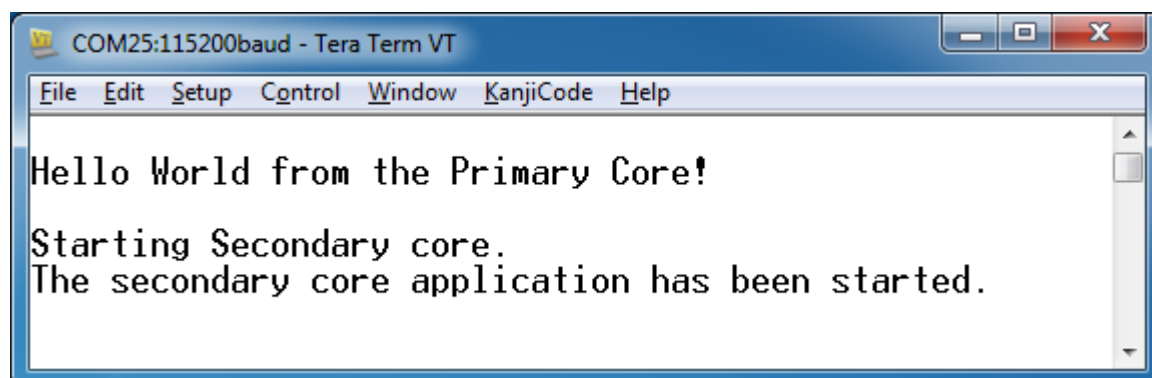
```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm0plus/mdk/hello_world_
→cm0plus.uvmpw
```

```
<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.

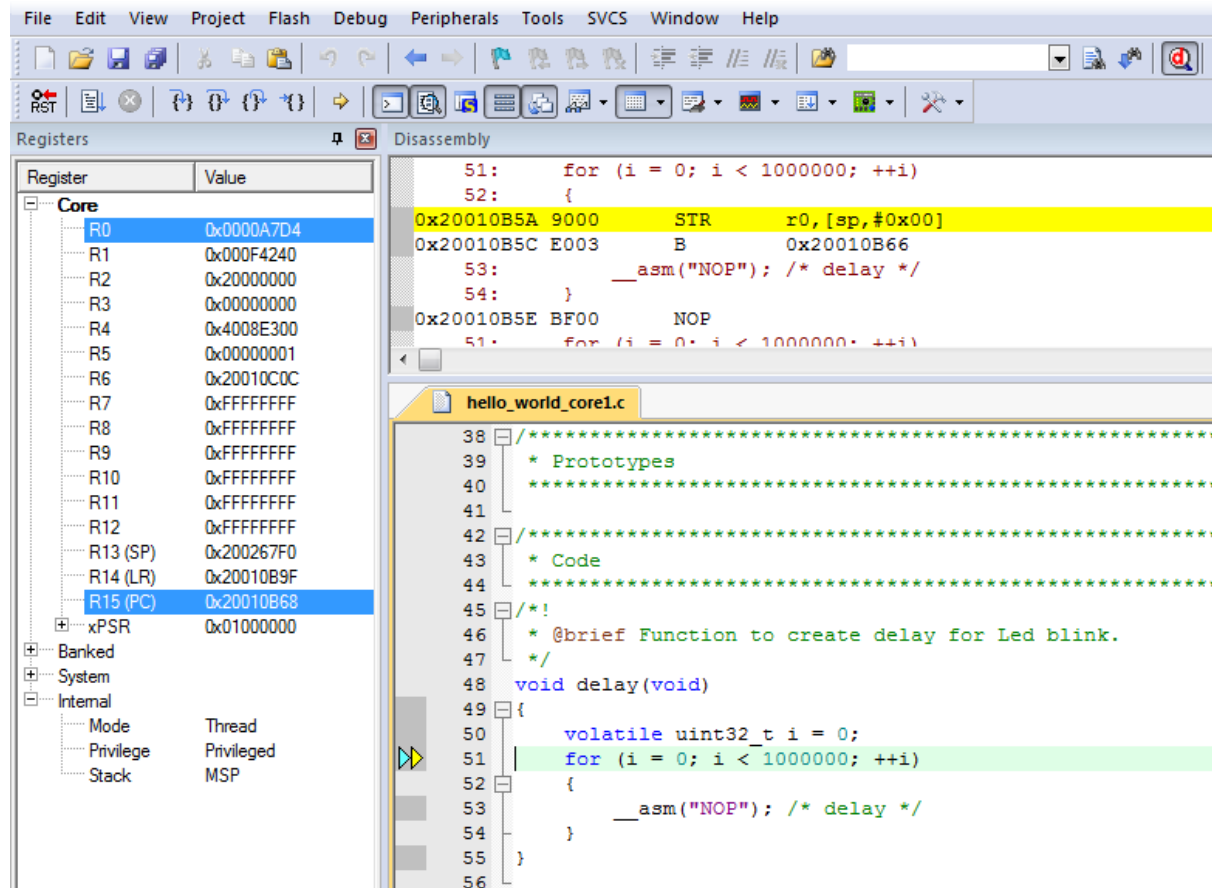
**Run a multicore example application** The primary core debugger flashes both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 5 as described in **Run an example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

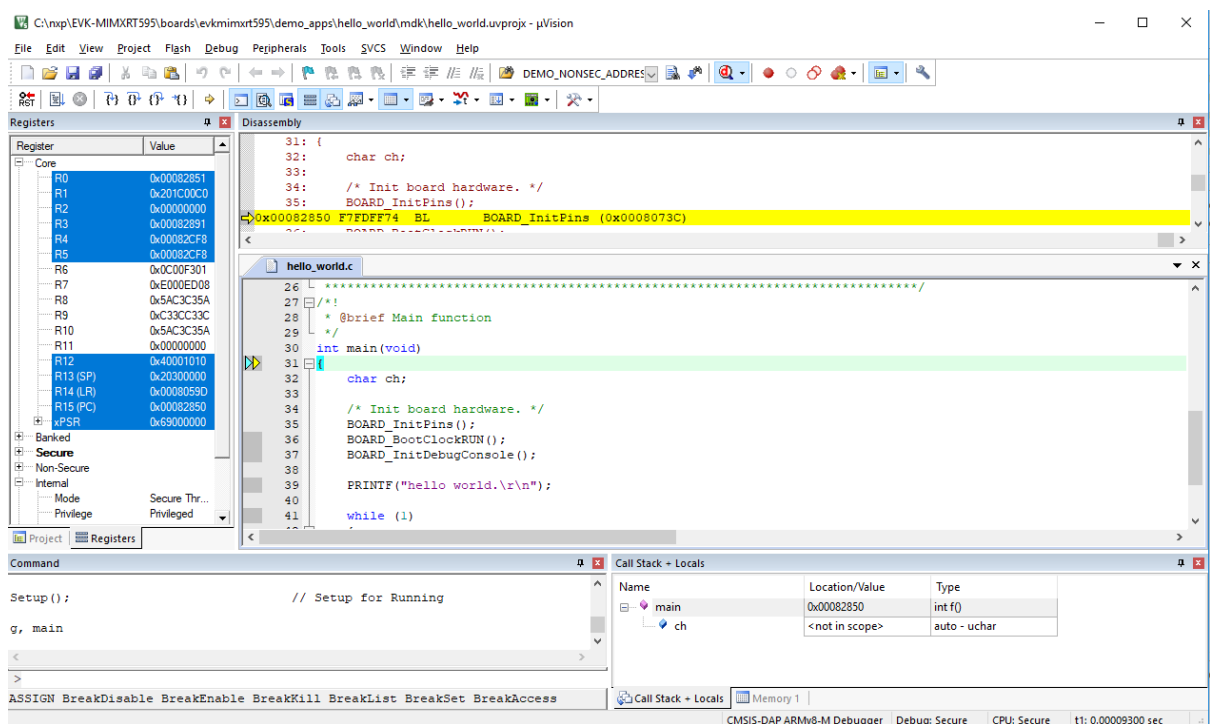
```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_s/mdk/hello_world.  
→ uvmpw
```

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.

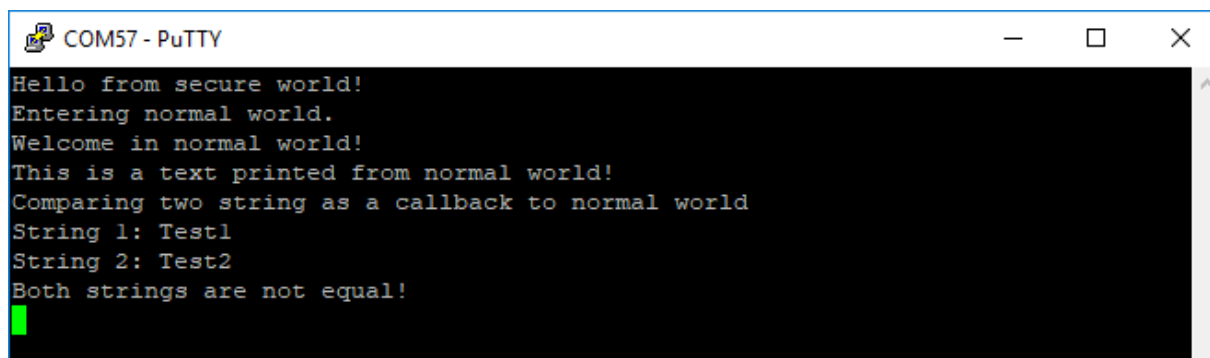
**Run a TrustZone example application** The secure project is configured to download both secure and non-secure output files so debugging can be fully managed from the secure project.

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.

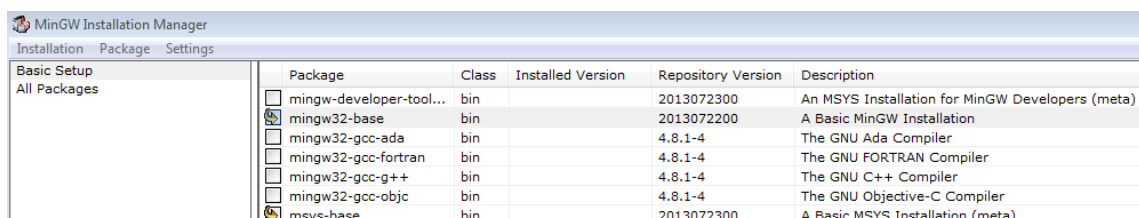
**Install GCC Arm Embedded tool chain** Download and run the installer from GNU Arm Embedded Toolchain. This is the actual toolset (in other words, compiler, linker, and so on). The GCC toolchain should correspond to the latest supported version, as described in **MCUXpresso SDK Release Notes**.

**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.

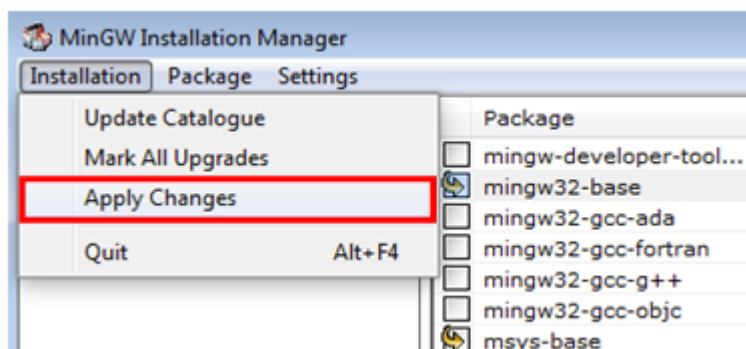
1. Download the latest MinGW mingw-get-setup installer from [MinGW](#).
2. Run the installer. The recommended installation path is `C:\MinGW`, however, you may install to any location.

**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.



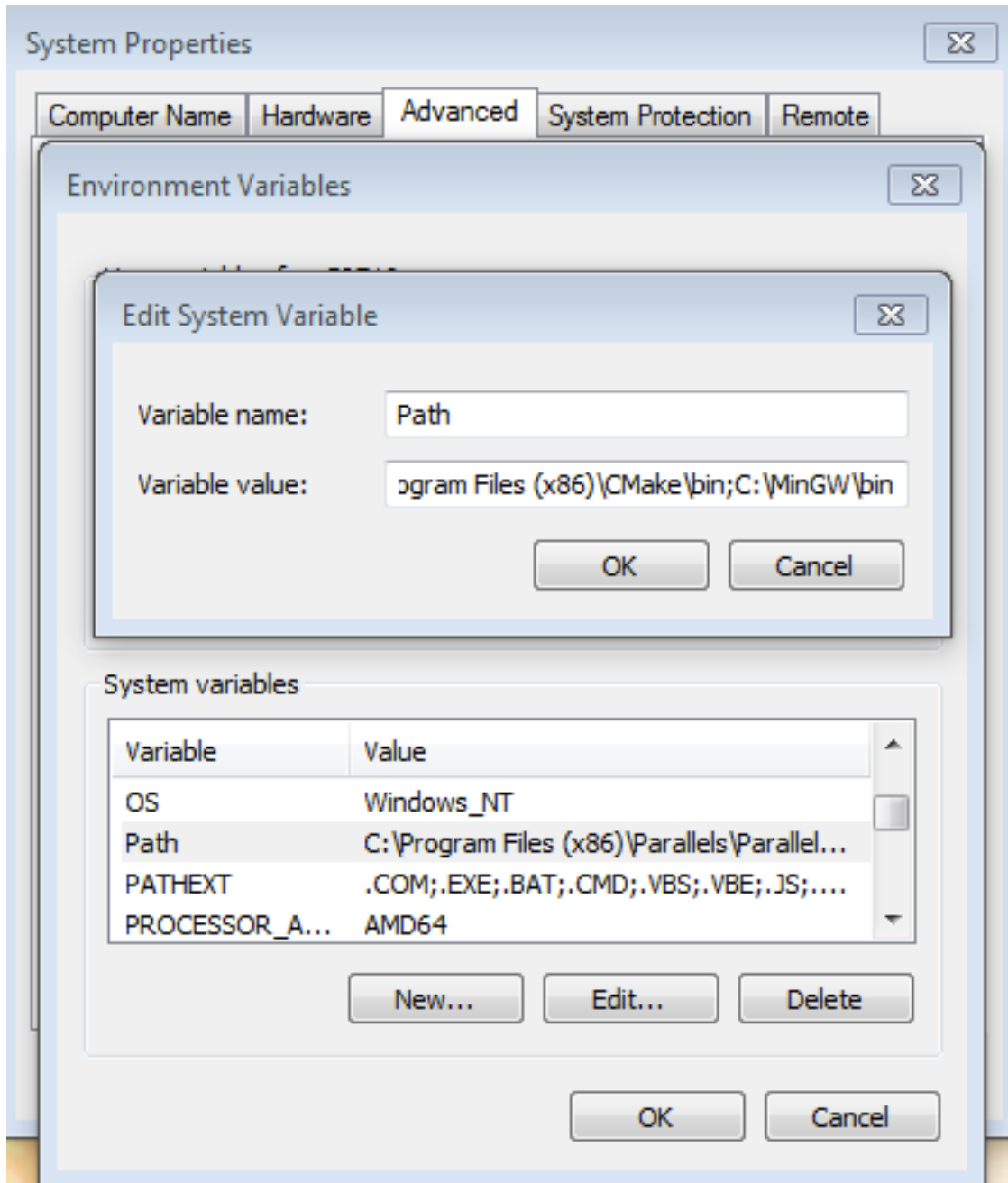
5. Add the appropriate item to the Windows operating system path environment variable. It can be found under **Control Panel->System and Security->System->Advanced System Settings** in the **Environment Variables...** section. The path is:



```
<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.



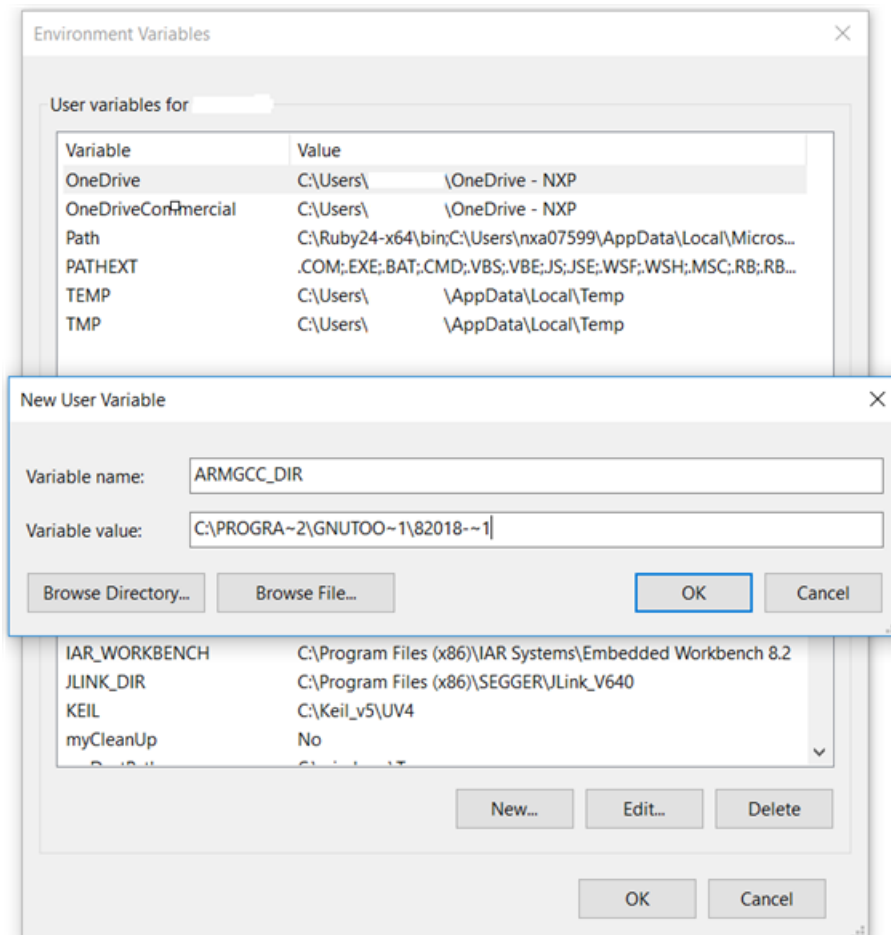
**Add a new system environment variable for ARMGCC\_DIR** Create a new *system* environment variable and name it as ARMGCC\_DIR. The value of this variable should point to the Arm GCC Embedded tool chain installation path. For this example, the path is:

```
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
```

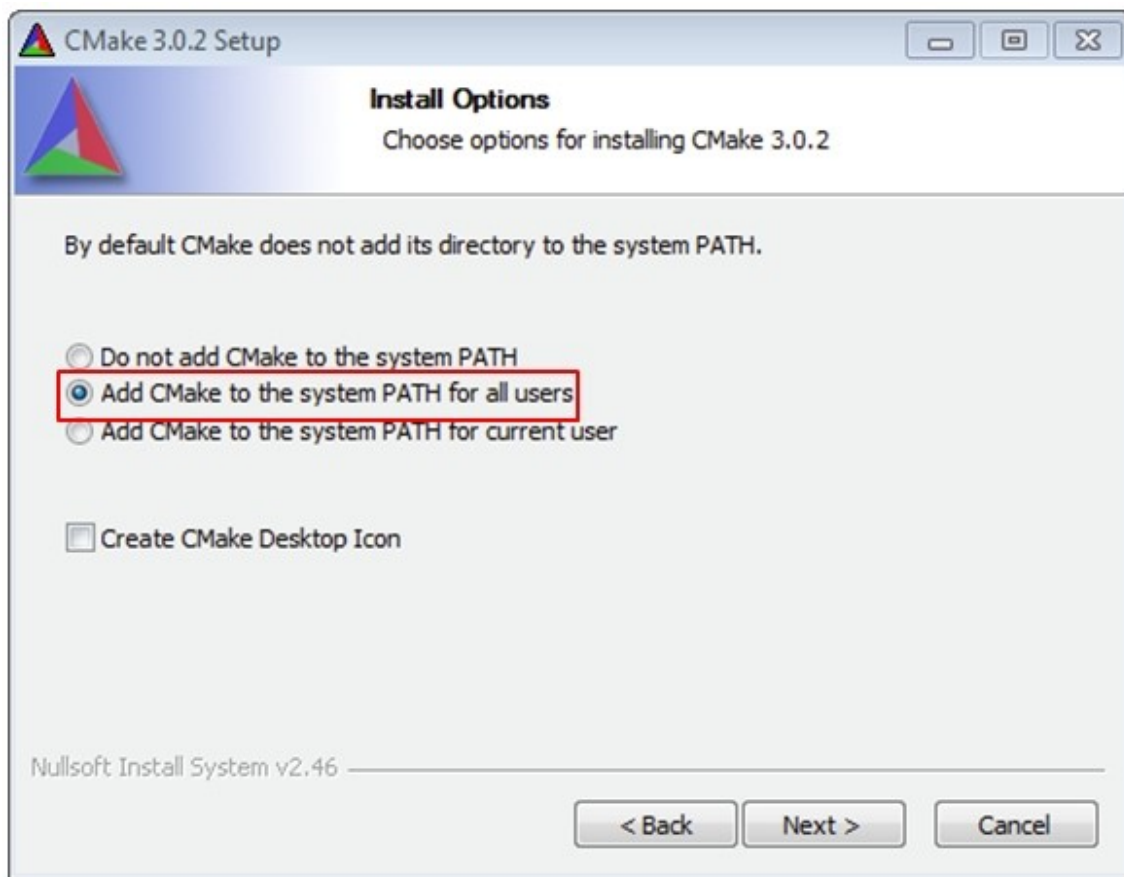


## Install CMake

### 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.

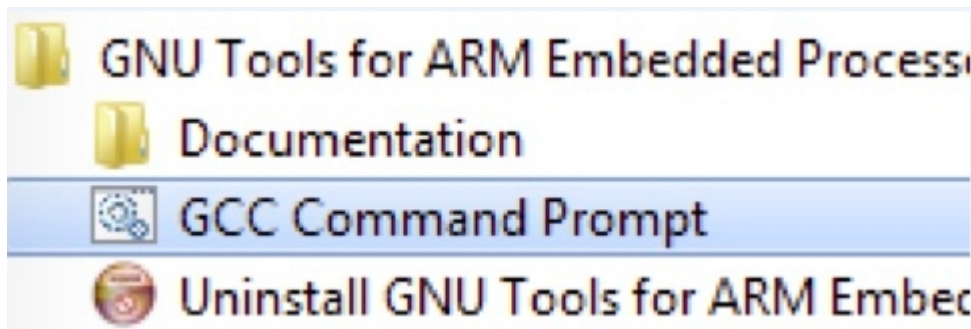
**Linux OS** It depends on the distributions of Linux Operation System. Here we use Ubuntu as an example.

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.

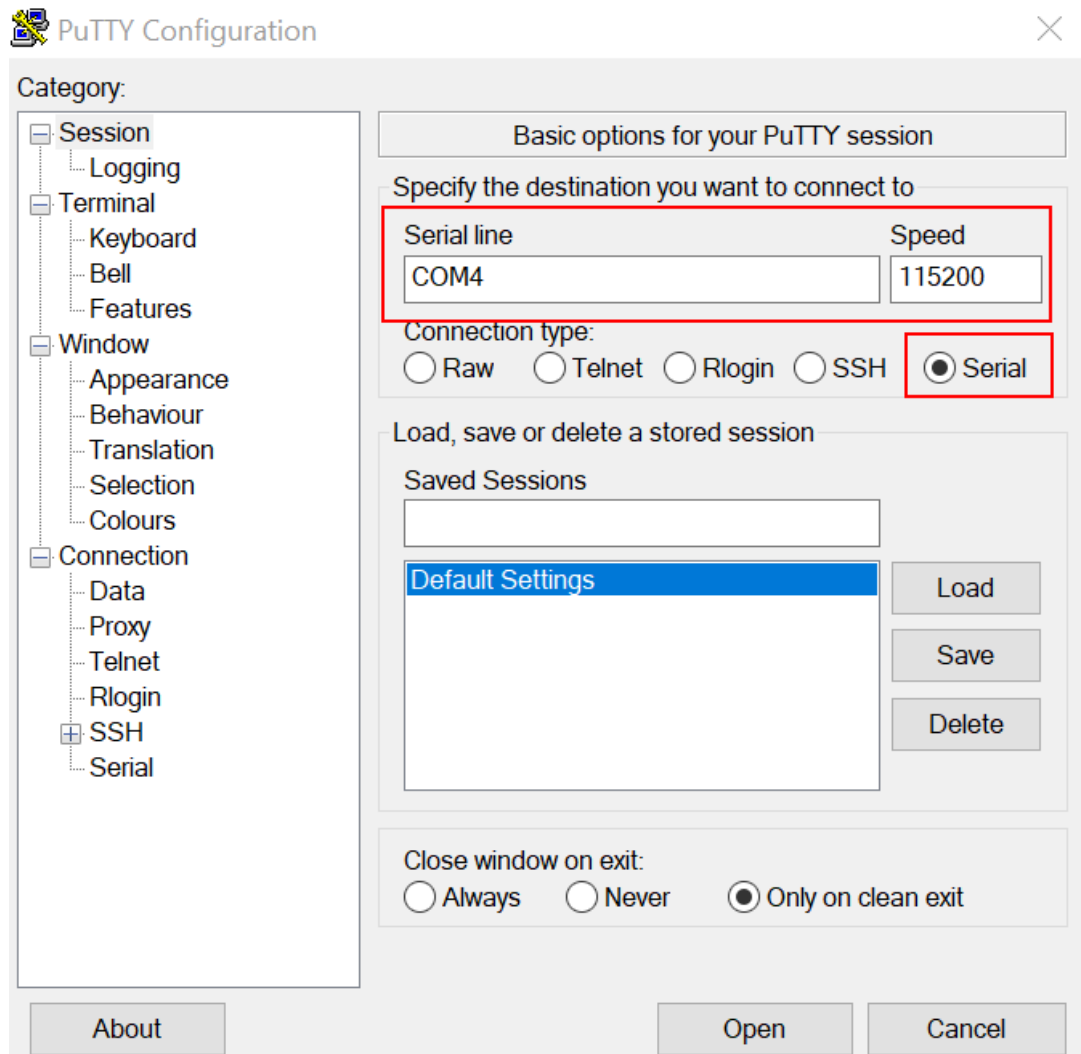
```
[ 84%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.0_FRDM-K64F
/devices/MK64F12/drivers/fsl_smc.c.obj
[ 92%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.0_FRDM-K64F
/devices/MK64F12/drivers/fsl_clock.c.obj
[100%] Linking C executable debug\hello_world.elf
[100%]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

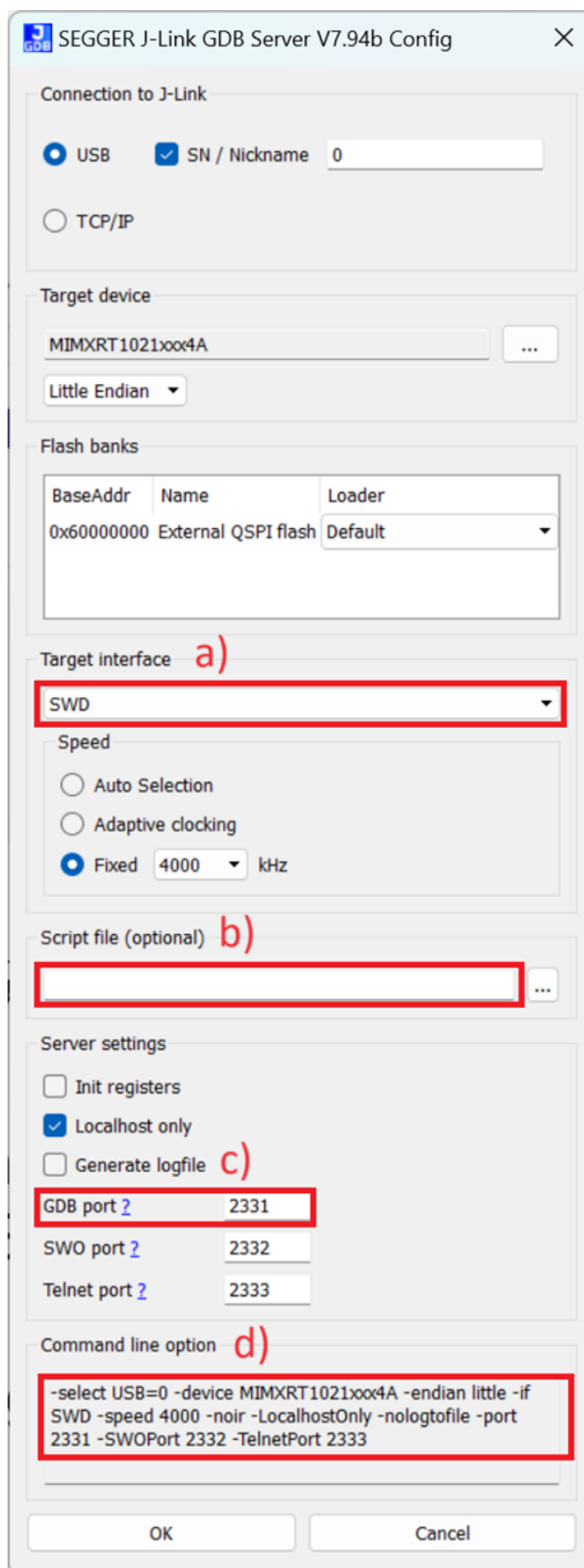


3. To launch the application, open the Windows **Start** menu and select **Programs > SEGGER > J-Link <version> J-Link GDB Server**.

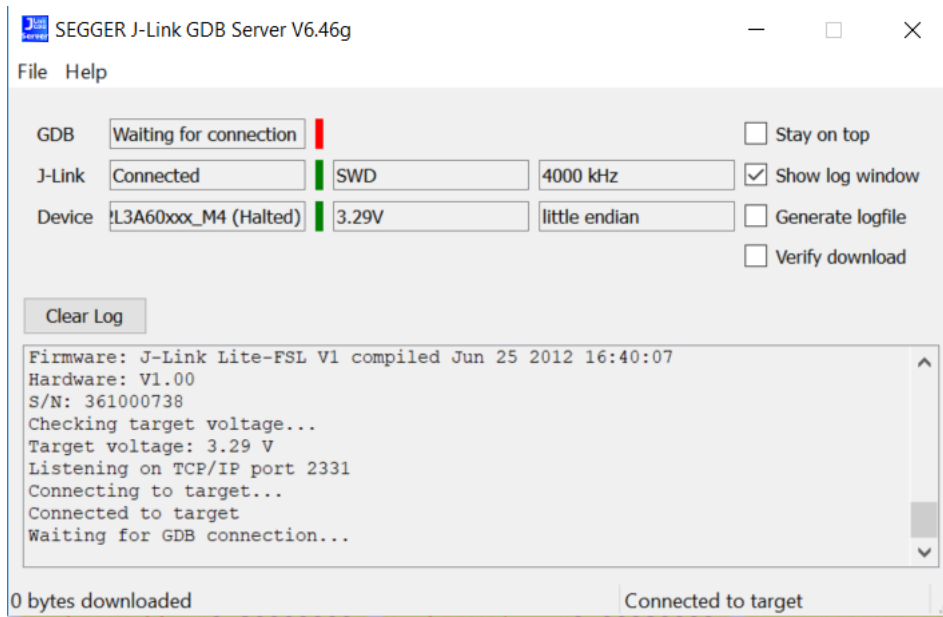
**Note:** It is assumed that the J-Link software is already installed.

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

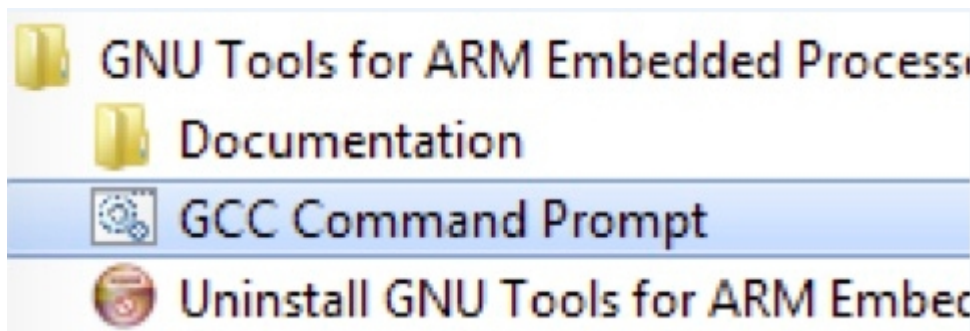
4. Make sure to check the following options.
  1. **Target interface:** The debug connection on board uses internal SWD signaling. In case of a wrong setting J-Link is unable to communicate with device under test.
  2. **Script file:** If required, a J-Link init script file can be used for board initialization. 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**.

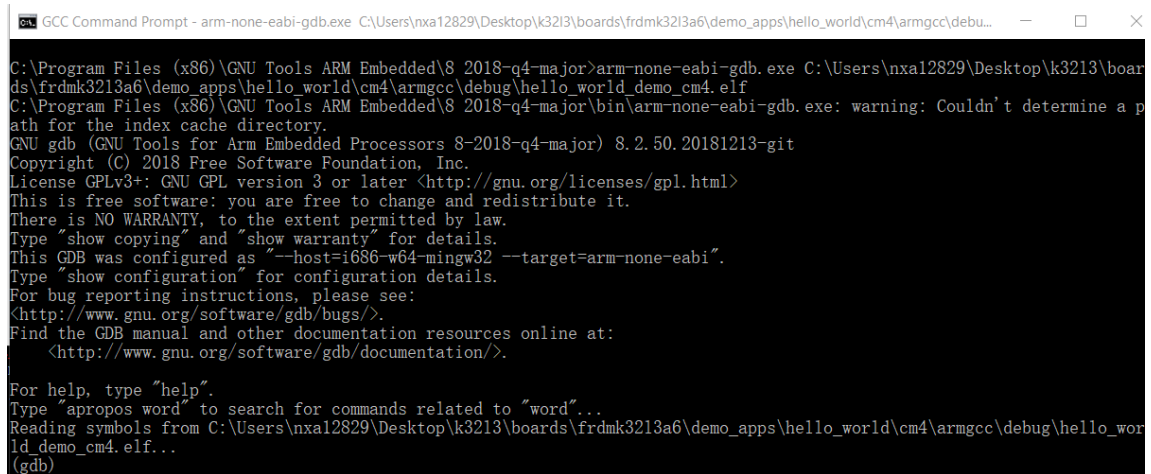


7. Change to the directory that contains the example application output. The output can be found in using one of these paths, depending on the build target selected:

```
<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`.



```

GCC Command Prompt - 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>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 pre-requisites 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:/nxp/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:/nxp/SDK_2.6.0_EVK-MIMXRT595/components/uart/usart_adapter.c
.obj
[ 62%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
flexspi.c.obj
[ 66%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
cache.c.obj
[ 70%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/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:/nxp/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:/nxp/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:/nxp/SDK_2.6.0_EVK-MIMXRT595/components/lists/generic_list.c
.obj
[ 85%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
usart.c.obj
[ 88%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
flexcomm.c.obj
[ 92%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
gpio.c.obj
[ 96%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/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:\nxp\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/fsl_assert.c.obj
[ 60%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_flexspi.c.obj
[ 64%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_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/fsl_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/fsl_flexcomm.c.obj
[ 92%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_gpio.c.obj
[ 96%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_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 . . .

```

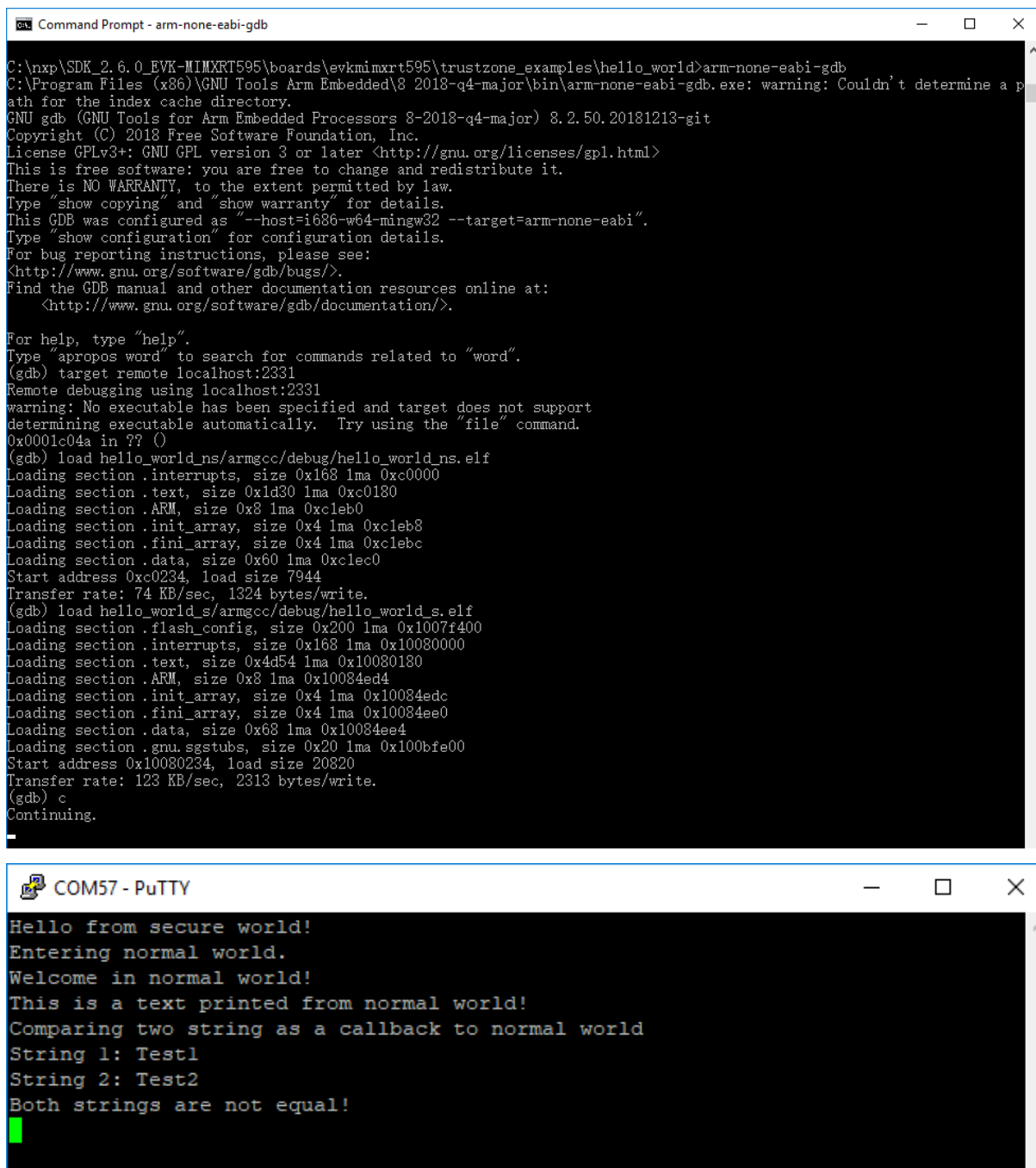
**Run a TrustZone example application** When running a TrustZone application, the same prerequisites for J-Link/J-Link OpenSDA firmware, and the serial console as for the single core application, apply, as described in **Run an example application**.

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.



```

C:\nxp\SDK_2.6.0_EVK-MIMXRT595\boards\evkmimxrt595\trustzone_examples\hello_world>arm-none-eabi-gdb
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major\bin\arm-none-eabi-gdb.exe: warning: Couldn't determine a p
ath 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".
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
warning: No executable has been specified and target does not support
determining executable automatically. Try using the "file" command.
0x0001c04a in ?? ()
(gdb) load hello_world_ns/armgcc/debug/hello_world_ns.elf
Loading section .interrupts, size 0x168 lma 0xc0000
Loading section .text, size 0x1d30 lma 0xc0180
Loading section .ARM, size 0x8 lma 0xc1eb0
Loading section .init_array, size 0x4 lma 0xc1eb8
Loading section .fini_array, size 0x4 lma 0xc1ebc
Loading section .data, size 0x60 lma 0xc1ec0
Start address 0xc0234, load size 7944
Transfer rate: 74 KB/sec, 1324 bytes/write.
(gdb) load hello_world_s/armgcc/debug/hello_world_s.elf
Loading section .flash_config, size 0x200 lma 0x1007f400
Loading section .interrupts, size 0x168 lma 0x10080000
Loading section .text, size 0x4d54 lma 0x10080180
Loading section .ARM, size 0x8 lma 0x10084ed4
Loading section .init_array, size 0x4 lma 0x10084edc
Loading section .fini_array, size 0x4 lma 0x10084ee0
Loading section .data, size 0x68 lma 0x10084ee4
Loading section .gnu.sgstubs, size 0x20 lma 0x100bfe00
Start address 0x10080234, load size 20820
Transfer rate: 123 KB/sec, 2313 bytes/write.
(gdb) c
Continuing.

```

```






COM57 - PuTTY
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.

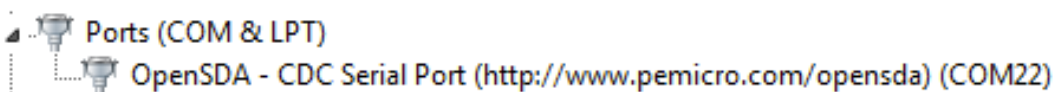
2. **Windows:** To determine the COM port open Device Manager in the Windows operating system. Click the **Start** menu and type **Device Manager** in the search bar.

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.

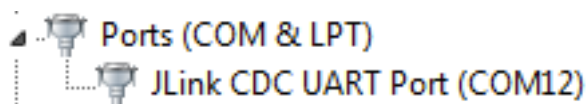
1. **CMSIS-DAP/mbd/DAPLink** interface:



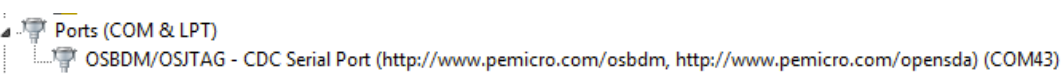
## 2. P&E Micro:



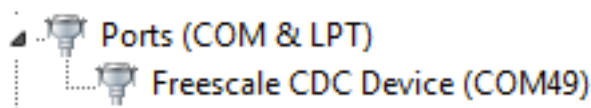
## 3. J-Link:



## 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/mbd 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.

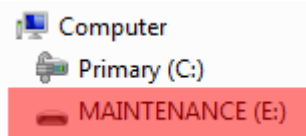
**Updating OpenSDA firmware** Any NXP hardware platform that comes with an OpenSDA-compatible debug interface has the ability to update the OpenSDA firmware. This typically means to switch from the default application (either CMSIS-DAP or P&E Micro) to a SEGGER J-Link. This section contains the steps 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-MCXA346      | 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-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                |
| KW47-LOC          | CMSIS-DAP                 | MCU-Link                |
| LPC845BREAKOUT    | CMSIS-DAP                 | LPC-Link2               |
| LPCXpresso51U68   | CMSIS-DAP                 | LPC-Link2               |
| LPCXpresso54628   | CMSIS-DAP                 | LPC-Link2               |

continues on next page

Table 1 – continued from previous page

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

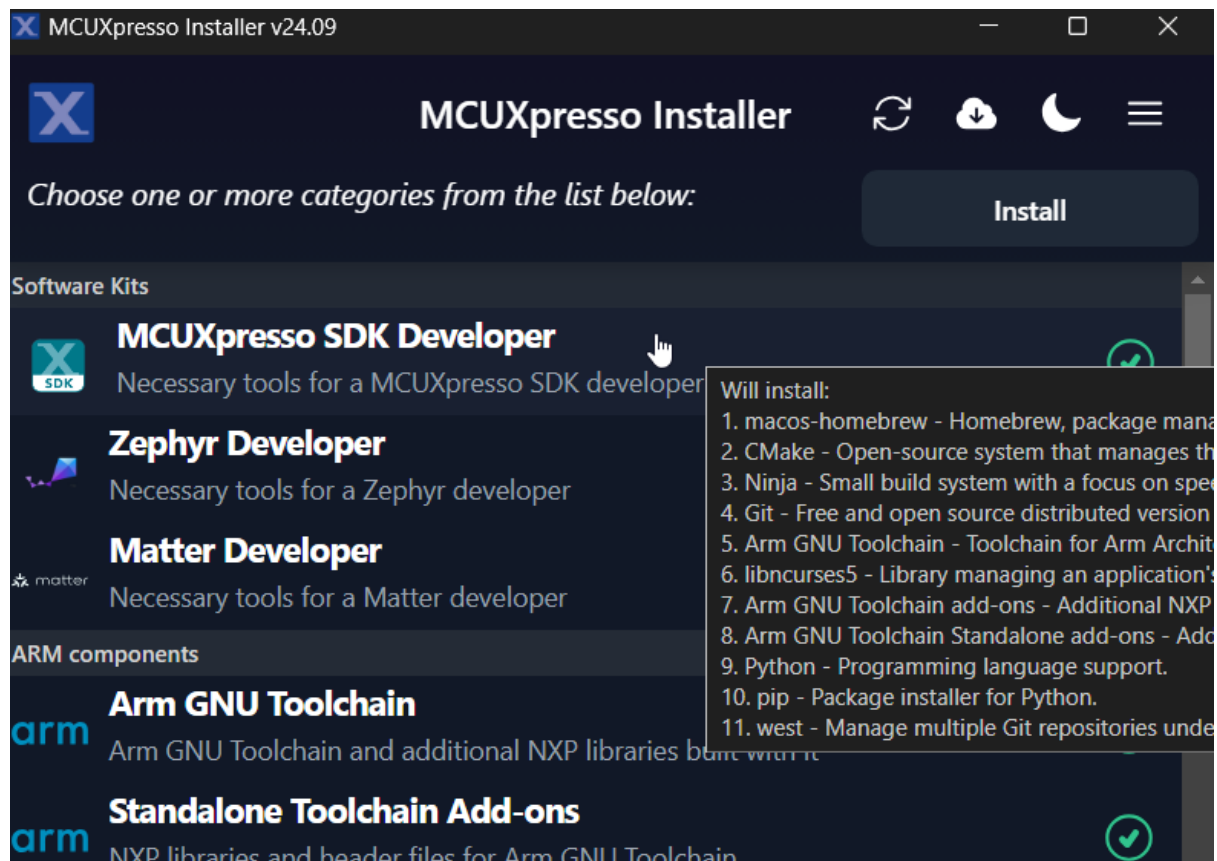
### 1.3.1 Getting Started with MCUXpresso SDK Repository

#### 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:

```

└─ arm
└─ common
└─ install-info

```

- 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

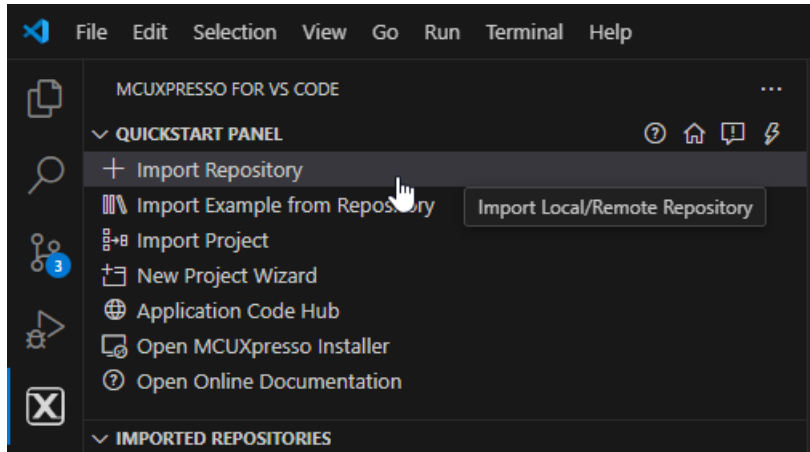
This section explains how to configure MCUXpresso for VS Code to build, run, and debug example applications. This guide uses the `hello_world` demo application as an example. However, these

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](#).

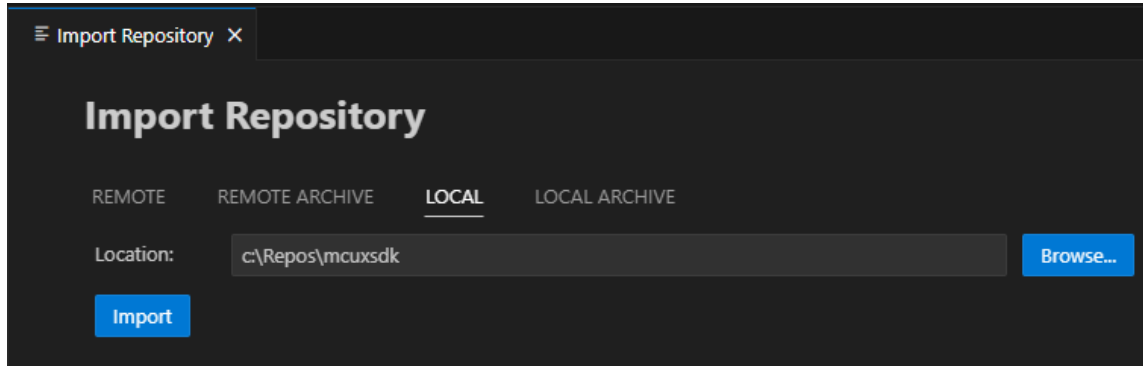
To build an example application:

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

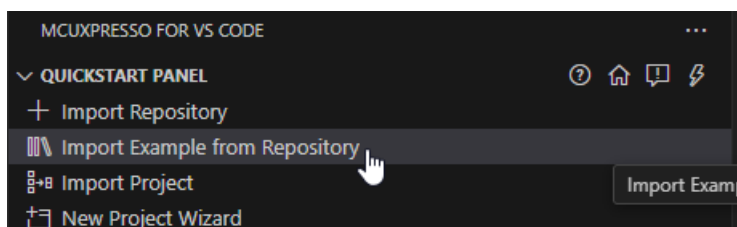


**Note:** You can import the SDK in several ways. Refer to [MCUXpresso for VS Code Wiki](#) for details.

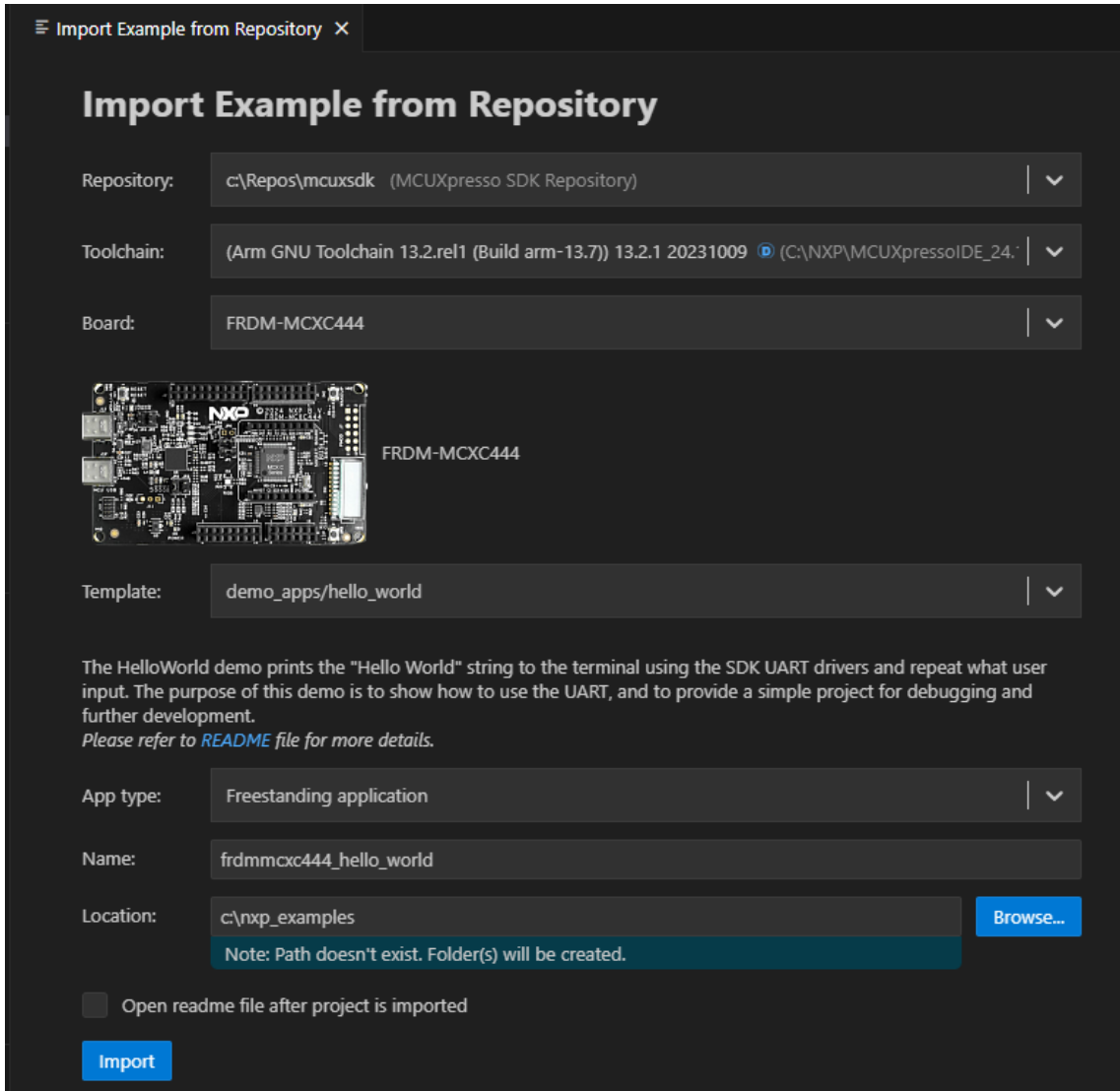
Select **Local** if you've already obtained the SDK as seen in [Get MCUXpresso SDK Repo](#). Select your location and click **Import**.



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) | v

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

Board: FRDM-MCXC444 | v

 FRDM-MCXC444

Template: demo\_apps/hello\_world | v

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 | v

Name: frdmmcxc444\_hello\_world

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

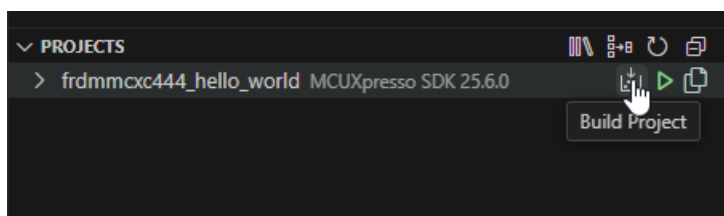
Note: Path doesn't exist. Folder(s) will be created.

☐ 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.

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

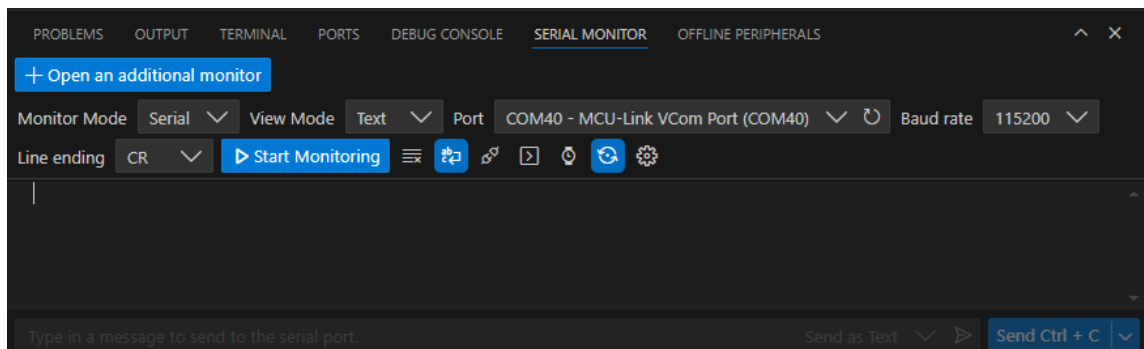
```

[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.

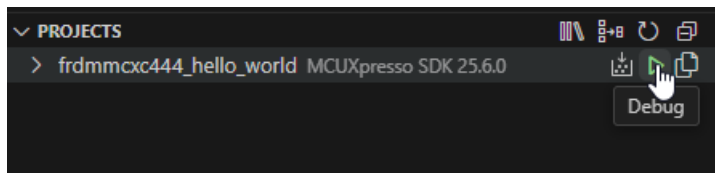
```

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

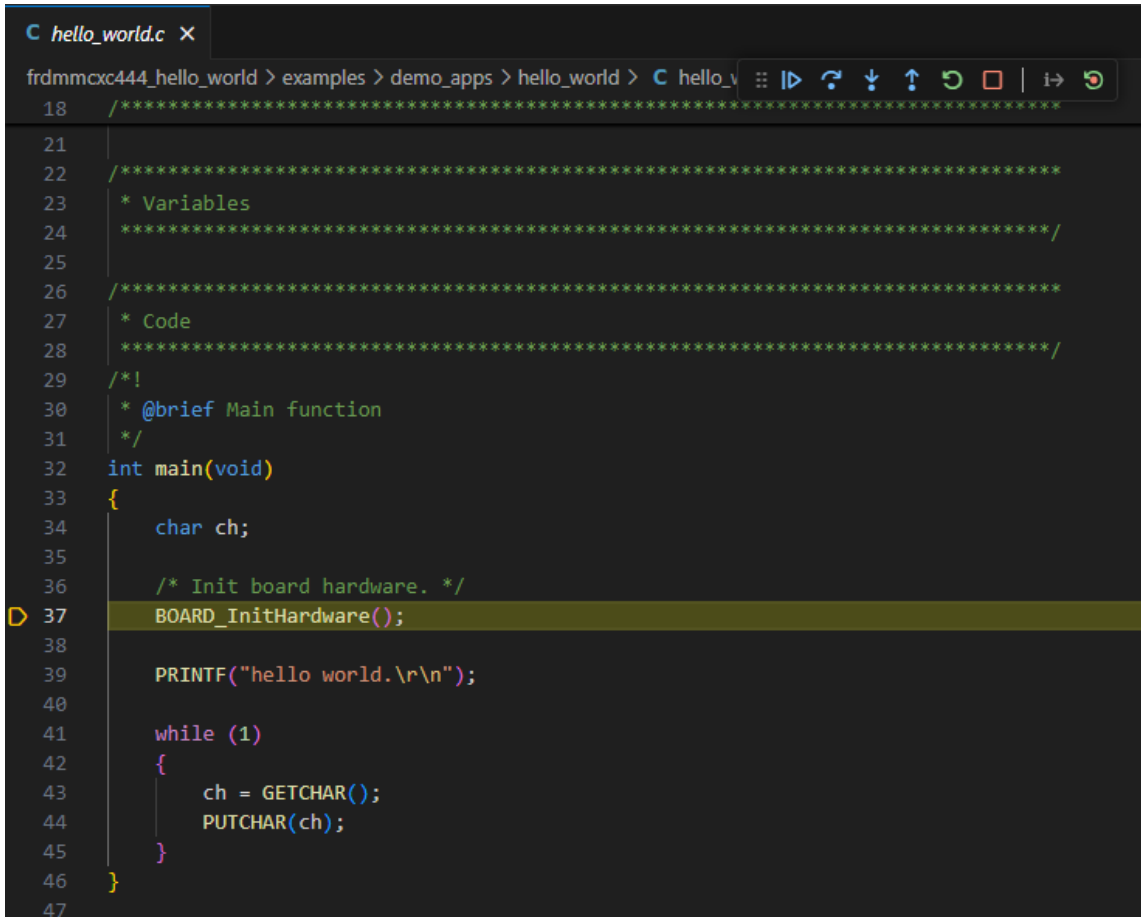
1. Open the **Serial Monitor** from the VS Code's integrated terminal. Select the VCom Port for your device and set the baud rate to 115200.



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.

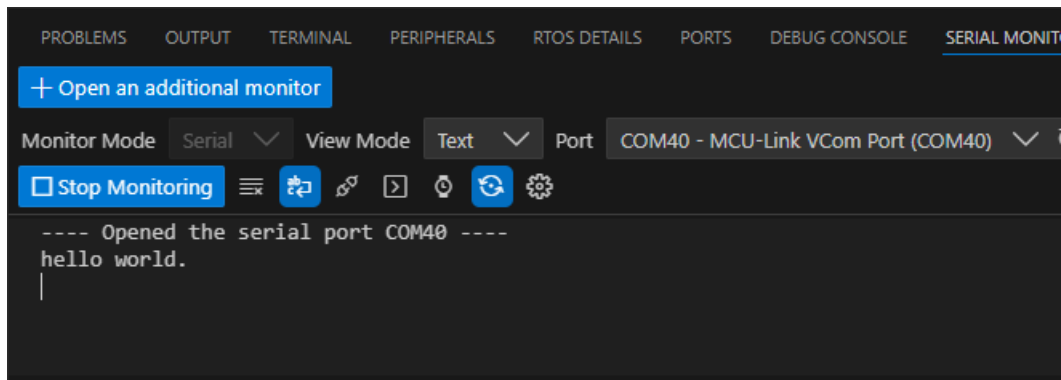


```

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

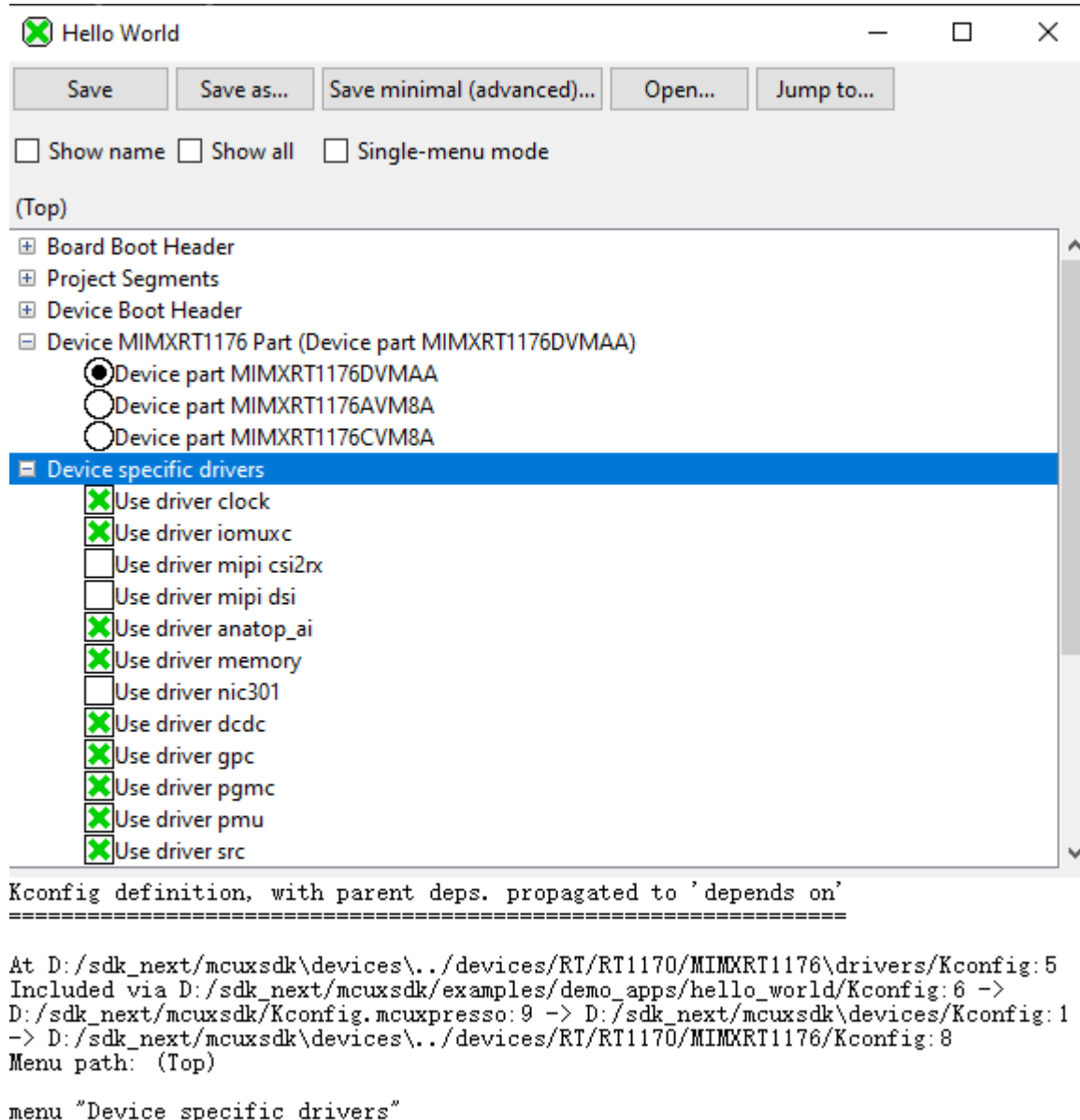
```
west build -b evkbnimxrt1170 examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

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:

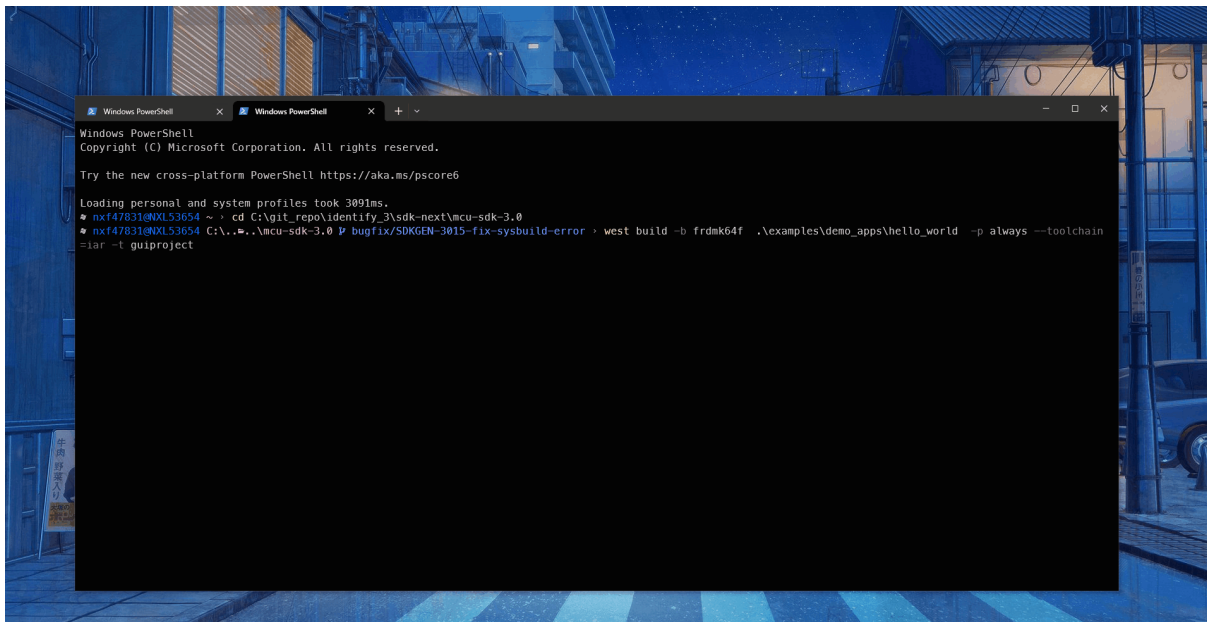
```
west debug -r linkserver
```

**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 5.41
- MCUXpresso for VS Code v25.06
- GCC Arm Embedded Toolchain 14.2.x

## Supported development systems

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

| Development boards   | MCU devices                            |                |                |
|----------------------|--|----------------|----------------|
| <b>FRDM-K32L2A4S</b> | K32L2A31VLH1A,<br><b>K32L2A41VLL1A</b> | K32L2A31VLL1A, | K32L2A41VLH1A, |

## MCUXpresso SDK release package

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

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

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

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

## RTOS

**FreeRTOS** Real-time operating system for microcontrollers from Amazon

## Middleware

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

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

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

**NXP Touch Library** NXP Touch Library

**TinyCBOR** Concise Binary Object Representation (CBOR) Library

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

**MMCAU** The NXP Memory-Mapped Cryptographic Acceleration Unit

**mbedtls** mbedtls SSL/TLS library v2.x

**llhttp** HTTP parser llhttp

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

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

## 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.

### FreeRTOS issue

When generating a new FreeRTOS project with New Project Wizard tool, the user should de-assert the macro option “configUSE\_PORT\_OPTIMISED\_TASK\_SELECTION” in the “FreeRTOSConfig.h” file while it is not being used in the template project for some Cortex-M0+ devices or it may not pass the compiling.

## 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
- 

#### ADC16

[2.3.0]

- Improvements
  - Added new API ADC16\_EnableAsynchronousClockOutput() to enable/disable ADACK output.
  - In ADC16\_GetDefaultConfig(), set enableAsynchronousClock to false.

**[2.2.0]**

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

**[2.1.0]**

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

**[2.0.3]**

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

**[2.0.2]**

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

**[2.0.1]**

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

**[2.0.0]**

- Initial version
- 
- 

**CMP****[2.0.3]**

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

**[2.0.2]**

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

### [2.0.1]

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

### [2.0.0]

- Initial version.
- 

## COMMON

### [2.6.0]

- Bug Fixes
  - Fix CERT-C violations.

### [2.5.0]

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

### [2.4.3]

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

### [2.4.2]

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

### [2.4.1]

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

### [2.4.0]

- New Features
  - Added EnableIRQWithPriority, IRQ\_SetPriority, and IRQ\_ClearPendingIRQ for ARM.
  - Added MSDK\_EnableCpuCycleCounter, MSDK\_GetCpuCycleCount for ARM.

### [2.3.3]

- New Features
  - Added NETC into status group.

**[2.3.2]**

- Improvements
  - Make driver aarch64 compatible

**[2.3.1]**

- Bug Fixes
  - Fixed MAKE\_VERSION overflow on 16-bit platforms.

**[2.3.0]**

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

**[2.2.10]**

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

**[2.2.9]**

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

**[2.2.8]**

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

**[2.2.7]**

- Other Change
  - Added MECC status group definition.

**[2.2.6]**

- Other Change
  - Added more status group definition.
- Bug Fixes
  - Undef \_\_VECTOR\_TABLE to avoid duplicate definition in cmsis\_clang.h



#### [2.2.5]

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

#### [2.2.4]

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

#### [2.2.3]

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

#### [2.2.2]

- New Features
  - Added include RTE\_Components.h for CMSIS pack RTE.

#### [2.2.1]

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

#### [2.2.0]

- New Features
  - Moved SDK\_DelayAtLeastUs function from clock driver to common driver.

#### [2.1.4]

- New Features
  - Added OTFAD into status group.

#### [2.1.3]

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

#### [2.1.2]

- Improvements
  - Add SUPPRESS\_FALL\_THROUGH\_WARNING() macro for the usage of suppressing fallthrough warning.

### [2.1.1]

- Bug Fixes
  - Deleted and optimized repeated macro.

### [2.1.0]

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

### [2.0.2]

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

### [2.0.1]

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

### [2.0.0]

- Initial version.
- 

## CRC

### [2.0.4]

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

### [2.0.3]

- Bug fix:
  - Fix MISRA issues.

### [2.0.2]

- Bug fix:
  - Fix MISRA issues.

### [2.0.1]

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

### [2.0.0]

- Initial version.
- 

## DAC

### [2.0.2]

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

### [2.0.1]

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

### [2.0.0]

- Initial version.
- 

## DMAMUX

### [2.1.2]

- Bug Fixes
  - Add macro FSL\_DMAMUX\_CHANNEL\_NUM to calculate correct DMAMUX channel number when input EDAM channel number.

### [2.1.1]

- Improvements
  - Add macro FSL\_FEATURE\_DMAMUX\_CHANNEL\_NEEDS\_ENDIAN\_CONVERT and DMAMUX\_CHANNEL\_ENDIAN\_CONVERTn to do channel endian convert.

### [2.1.0]

- Improvements
  - Modify the type of parameter source from uint32\_t to int32\_t in the DMA-MUX\_SetSource.

**[2.0.5]**

- Improvements
  - Added feature FSL\_FEATURE\_DMAMUX\_CHCFG\_REGISTER\_WIDTH for the difference of CHCFG register width.

**[2.0.4]**

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

**[2.0.3]**

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

**[2.0.2]**

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

**[2.0.1]**

- Bug Fixes
  - Fixed the build warning issue by changing the type of parameter source from uint8\_t to uint32\_t when setting DMA request source in DMAMUX\_SetSourceChange.

**[2.0.0]**

- Initial version.
- 

**EDMA****[2.4.5]**

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

**[2.4.4]**

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

#### [2.4.3]

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

#### [2.4.2]

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

#### [2.4.1]

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

**FLASH****[3.2.0]**

- New Feature
  - Basic support for FTFC

**[3.1.3]**

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

**[3.1.2]**

- Bug Fixes — Remove redundant comments.

### [3.1.1]

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

### [3.1.0]

- New Feature
  - Support erase flash asynchronously.

### [3.0.2]

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

### [3.0.1]

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

### [3.0.0]

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

### [2.3.1]

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

### [2.3.0]

- New Features
  - Added support for device with LP flash (K3S/G).
  - Added flash prefetch speculation APIs.
- Improvements
  - Refined flash\_cache\_clear function.
  - Reorganized the member of flash\_config\_t struct.

### [2.2.0]

- New Features
  - Supported FTFL device in FLASH\_Swap API.
  - Supported various pflash start addresses.
  - Added support for KV58 in cache clear function.

- Added support for device with secondary flash (KW40).
- Bug Fixes
  - Compiled execute-in-ram functions as PIC binary code for driver use.
  - Added missed flexram properties.
  - Fixed unaligned variable issue for execute-in-ram function code array.

#### [2.1.0]

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

#### [2.0.0]

- Initial version
- 

### FLEXIO

#### [2.3.0]

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

#### [2.2.3]

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

#### [2.2.2]

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

#### [2.2.1]

- Improvements
  - Added doxygen index parameter comment in FLEXIO\_SetClockMode.

#### [2.2.0]

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

#### [2.1.0]

- Improvements
  - Added API FLEXIO\_SetClockMode to set flexio channel counter and source clock.

#### [2.0.4]

- Bug Fixes
  - Fixed MISRA 8.4 issues.

#### [2.0.3]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

#### [2.0.2]

- Improvements
  - Split FLEXIO component which combines all flexio/flexio\_uart/flexio\_i2c/flexio\_i2s drivers into several components: FlexIO component, flexio\_uart component, flexio\_i2c\_master component, and flexio\_i2s component.
- Bug Fixes
  - Fixed MISRA issues
    - \* Fixed rules 10.1, 10.3, 10.4, 10.7, 11.6, 11.9, 14.4, 17.7.

#### [2.0.1]

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

### FLEXIO\_I2C

#### [2.6.1]

- Bug Fixes
  - Fixed coverity issues

#### [2.6.0]

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

#### [2.5.1]

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

**[2.5.0]**

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

**[2.4.0]**

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

**[2.3.0]**

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

**[2.2.0]**

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

**[2.1.7]**

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

### [2.1.6]

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

### [2.1.5]

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

### [2.1.4]

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

### [2.1.3]

- Improvements
  - Changed the prototype of FLEXIO\_I2C\_MasterInit to return kStatus\_Success if initialized successfully or to return kStatus\_InvalidArgument if “(srcClock\_Hz / masterConfig->baudRate\_Bps) / 2 - 1” exceeds 0xFFU.

### [2.1.2]

- Bug Fixes
  - Fixed the FLEXIO I2C issue where the master could not receive data from I2C slave in high baudrate.
  - Fixed the FLEXIO I2C issue where the master could not receive NAK when master sent non-existent addr.
  - Fixed the FLEXIO I2C issue where the master could not get transfer count successfully.
  - Fixed the FLEXIO I2C issue where the master could not receive data successfully when sending data first.
  - Fixed the Dozen mode configuration error in FLEXIO\_I2C\_MasterInit API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.
  - Fixed the issue that FLEXIO\_I2C\_MasterTransferBlocking API called FLEXIO\_I2C\_MasterTransferCreateHandle, which lead to the s\_flexioHandle/s\_flexioIsr/s\_flexioType variable being written. Then, if calling FLEXIO\_I2C\_MasterTransferBlocking API multiple times, the s\_flexioHandle/s\_flexioIsr/s\_flexioType variable would not be written any more due to it being out of range. This lead to the following situation: NonBlocking transfer APIs could not work due to the fail of register IRQ.

**[2.1.1]**

- Bug Fixes
  - Implemented the FLEXIO\_I2C\_MasterTransferBlocking API which is defined in header file but has no implementation in the C file.

**[2.1.0]**

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

**FLEXIO\_I2S****[2.2.2]**

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

**[2.2.1]**

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

**[2.2.0]**

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

**[2.1.6]**

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

**[2.1.5]**

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

**[2.1.4]**

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



### [2.1.3]

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

### [2.1.2]

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

### [2.1.1]

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

### [2.1.0]

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

## FLEXIO\_I2S\_EDMA

### [2.1.9]

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

### [2.1.8]

- Improvements
    - Applied EDMA ERRATA 51327.
- 

## FLEXIO\_SPI

### [2.4.2]

- Bug Fixes
  - Fixed FLEXIO\_SPI\_MasterTransferBlocking and FLEXIO\_SPI\_MasterTransferNonBlocking issue in CS continuous mode, the CS might not be continuous.

**[2.4.1]**

- Bug Fixes
  - Fixed coverity issues

**[2.4.0]**

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

**[2.3.5]**

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

**[2.3.4]**

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

**[2.3.3]**

- Bugfixes
  - Fixed cs-continuous mode.

**[2.3.2]**

- Improvements
  - Changed FLEXIO\_SPI\_DUMMYDATA to 0x00.

**[2.3.1]**

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

**[2.3.0]**

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

### [2.2.1]

- Bug Fixes
  - Fixed bug in FLEXIO\_SPI\_MasterTransferAbortEDMA that when aborting EDMA transfer EDMA\_AbortTransfer should be used rather than EDMA\_StopTransfer.

### [2.2.0]

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

### [2.1.3]

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

### [2.1.2]

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

### [2.1.1]

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

**[2.1.0]**

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

**FLEXIO\_UART****[2.6.2]**

- Bug Fixes
  - Fixed coverity issues

**[2.6.1]**

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

**[2.6.0]**

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

**[2.5.1]**

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

**[2.5.0]**

- Improvements
  - Added API FLEXIO\_UART\_FlushShifters to flush UART fifo.

**[2.4.0]**

- Improvements
  - Use separate data for TX and RX in flexio\_uart\_transfer\_t.
- Bug Fixes
  - Fixed bug that when ring buffer is used, if some data is received in ring buffer first before calling FLEXIO\_UART\_TransferReceiveNonBlocking, the received data count returned by FLEXIO\_UART\_TransferGetReceiveCount is wrong.

### [2.3.0]

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

### [2.2.0]

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

### [2.1.6]

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

### [2.1.5]

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

### [2.1.4]

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

### [2.1.3]

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

### [2.1.2]

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

### [2.1.1]

- Bug Fixes
  - Changed the API name FLEXIO\_UART\_StopRingBuffer to FLEXIO\_UART\_TransferStopRingBuffer to align with the definition in C file.

### [2.1.0]

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

## FLEXIO\_UART\_EDMA

### [2.3.1]

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

### [2.3.0]

- Refer FLEXIO\_UART driver change log to 2.3.0
- 

## 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`”.
- 

## INTMUX

### [2.0.4]

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

### [2.0.3]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 17.7 and 18.1.

### [2.0.2]

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

**[2.0.1]**

- Improvements
  - Added weak function implementations of INTMUX1\_x\_DriverIRQHandler. x ranges from 0 to 7 to support 8 channels.

**[2.0.0]**

- Initial version.
- 

**LLWU****[2.0.5]**

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

**[2.0.4]**

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

**[2.0.3]**

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

**[2.0.2]**

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

**[2.0.1]**

- Other Changes
  - Updates for KL8x.

## [2.0.0]

- Initial version.
- 

## LPI2C

### [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

### [2.4.4]

- 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
- 

### LPIT

#### [2.1.1]

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

#### [2.1.0]

- Improvements
  - Add new function LPIT\_SetTimerValue to set timeout period.

#### [2.0.2]

- Improvements
  - Improved LPIT\_SetTimerPeriod implementation, configure timeout value with LPIT ticks minus 1 generate more correct interval.
  - Added timeout value configuration check for LPIT\_SetTimerPeriod, at least input 3 ticks for calling LPIT\_SetTimerPeriod.
- Bug Fixes
  - Fixed MISRA C-2012 rule 17.7 violations.

#### [2.0.1]

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

#### [2.0.0]

- Initial version.
- 

### LPSPi

#### [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\_CR[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****[2.4.6]**

- 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 LP-SPI\_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

#### [2.2.0]

- 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****[2.9.1]**

- 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 have a negative effect on other IPs that are using the interrupt.

#### [2.2.2]

- Improvements
  - Added software reset feature support.
  - Added software reset API in LPUART\_Init.

#### [2.2.1]

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

#### [2.2.0]

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

#### [2.1.1]

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

#### [2.1.0]

- Improvements
    - Update transactional APIs.
- 

### LPUART\_EDMA

#### [2.4.0]

- Refer LPUART driver change log 2.1.0 to 2.4.0
- 

### MCM

#### [2.2.0]

- Improvements
  - Support platforms with less features.

**[2.1.0]**

- Others
  - Remove byteID from mcm\_lmem\_fault\_attribute\_t for document update.

**[2.0.0]**

- Initial version.
- 

**MMDVSQ****[2.0.4]**

- Improvements
  - Fixed CERT-C issues.

**[2.0.3]**

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

**[2.0.2]**

- Bug fix:
  - Fixed MMDVSQ\_GetExecutionStatus function get execution status wrong.

**[2.0.1]**

- Other changes:
  - Changed name of MMDVSQ\_GetDivideRemainder and MMDVSQ\_GetDivideQuotient functions.

**[2.0.0]**

- Initial version.
- 

**MSCM****[2.0.0]**

- Initial version.
- 

**PMC****[2.0.3]**

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

### [2.0.2]

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

### [2.0.1]

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

### [2.0.0]

- Initial version.
- 

## PORT

### [2.5.1]

- Bug Fixes
  - Fix CERT INT31-C issues.

### [2.5.0]

- Bug Fixes
  - Correct the kPORT\_MuxAsGpio for some platforms.

### [2.4.1]

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

### [2.4.0]

- New Features
  - Updated port\_pin\_config\_t to support input buffer and input invert.

### [2.3.0]

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

### [2.2.0]

- New Features
  - Added new api PORT\_EnablePinDoubleDriveStrength.



**[2.1.1]**

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

**[2.1.0]**

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

**[2.0.2]**

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

**[2.0.1]**

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

**RCM****[2.0.4]**

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

**[2.0.3]**

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

**[2.0.2]**

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

**[2.0.1]**

- Bug Fixes
  - Fixed kRCM\_SourceSw bit shift issue.

## [2.0.0]

- Initial version.
- 

## RTC

### [2.3.3]

- Bug Fixes
  - Fix RTC\_GetDatetime function validating datetime issue.

### [2.3.2]

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

### [2.3.1]

- Bug Fixes
  - Fixed CERT INT31-C violations.

### [2.3.0]

- Improvements
  - Added API RTC\_EnableLPOClock to set 1kHz LPO clock.
  - Added API RTC\_EnableCrystalClock to replace API RTC\_SetClockSource.

### [2.2.2]

- Improvements
  - Refine \_rtc\_interrupt\_enable order.

### [2.2.1]

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

### [2.2.0]

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

**[2.1.0]**

- Improvements
  - Added feature macro check for many features.

**[2.0.0]**

- Initial version.
- 

**SIM****[2.2.0]**

- Improvements
  - Added API to trigger TRGMUX.

**[2.1.3]**

- Improvements
  - Updated function SIM\_GetUniqueId to support different register names.

**[2.1.2]**

- Bug Fixes
  - Fixed SIM\_GetUniqueId bug that could not get UIDH.

**[2.1.1]**

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

**[2.1.0]**

- Improvements
  - Added new APIs: SIM\_GetRfAddr() and SIM\_EnableSystickClock().

**[2.0.0]**

- Initial version.
- 

**SMC****[2.0.7]**

- Bug Fixes
  - Fixed MISRA-2012 issue 10.3.

#### [2.0.6]

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

#### [2.0.5]

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

#### [2.0.4]

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

#### [2.0.3]

- Improvements
  - Added APIs SMC\_PreEnterStopModes, SMC\_PreEnterWaitModes, SMC\_PostExitWaitModes, and SMC\_PostExitStopModes.

#### [2.0.2]

- Bug Fixes
  - Added DSB before WFI while ISB after WFI.
- Other Changes
  - Updated SMC\_SetPowerModeVlppw implementation.

#### [2.0.1]

- Other Changes
  - Updated for KL8x.

#### [2.0.0]

- Initial version.
- 

## TPM

#### [2.3.5]

- New Feature
  - Added IRQ handler entry for TPM2.

**[2.3.4]**

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

**[2.3.3]**

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

**[2.3.2]**

- Bug Fixes
  - Fixed ERR008085 TPM writing the TPMx\_MOD or TPMx\_CnV registers more than once may fail when the timer is disabled.

**[2.3.1]**

- Bug Fixes
  - Fixed compilation error when macro FSL\_SDK\_DISABLE\_DRIVER\_CLOCK\_CONTROL is 1.

**[2.3.0]**

- Improvements
  - Create callback feature for TPM match and timer overflow interrupts.

**[2.2.4]**

- Improvements
  - Add feature macros(FSL\_FEATURE\_TPM\_HAS\_GLOBAL\_TIME\_BASE\_EN, FSL\_FEATURE\_TPM\_HAS\_GLOBAL\_TIME\_BASE\_SYNC).

**[2.2.3]**

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

**[2.2.2]**

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

**[2.2.1]**

- Bug Fixes
  - Fixed CCM issue by splitting function from TPM\_SetupPwm() function to reduce function complexity.
  - Fixed violations of MISRA C-2012 rule 17.7.

### [2.2.0]

- Improvements
  - Added TPM\_SetChannelPolarity to support select channel input/output polarity.
  - Added TPM\_EnableChannelExtTrigger to support enable external trigger input to be used by channel.
  - Added TPM\_CalculateCounterClkDiv to help calculates the counter clock prescaler.
  - Added TPM\_GetChannelValue to support get TPM channel value.
  - Added new TPM configuration.
    - \* syncGlobalTimeBase
    - \* extTriggerPolarity
    - \* chnlPolarity
  - Added new PWM signal configuration.
    - \* secPauseLevel
- Bug Fixes
  - Fixed TPM\_SetupPwm can't configure 0% combined PWM issues.

### [2.1.1]

- Improvements
  - Add feature macro for PWM pause level select feature.

### [2.1.0]

- Improvements
  - Added TPM\_EnableChannel and TPM\_DisableChannel APIs.
  - Added new PWM signal configuration.
    - \* pauseLevel - Support select output level when counter first enabled or paused.
    - \* enableComplementary - Support enable/disable generate complementary PWM signal.
    - \* deadTimeValue - Support deadtime insertion for each pair of channels in combined PWM mode.
- Bug Fixes
  - Fixed issues about channel MSnB:MSnA and ELSnB:ELSnA bit fields and CnV register change request acknowledgement. Writes to these bits are ignored when the interval between successive writes is less than the TPM clock period.

### [2.0.8]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.4 ,10.7 and 14.4.

### [2.0.7]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.4 and 17.7.

**[2.0.6]**

- Bug Fixes
  - Fixed Out-of-bounds issue.

**[2.0.5]**

- Bug Fixes
  - Fixed MISRA-2012 rules.
    - \* Rule 10.6, 10.7

**[2.0.4]**

- Bug Fixes
  - Fixed ERR050050 in functions TPM\_SetupPwm/TPM\_UpdatePwmDutycycle. When TPM was configured in EPWM mode as PS = 0, the compare event was missed on the first reload/overflow after writing 1 to the CnV register.

**[2.0.3]**

- Bug Fixes
  - MISRA-2012 issue fixed.
    - \* Fixed rules: rule-12.1, rule-17.7, rule-16.3, rule-14.4, rule-1.3, rule-10.4, rule-10.3, rule-10.7, rule-10.1, rule-10.6, and rule-18.1.

**[2.0.2]**

- Bug Fixes
  - Fixed issues in functions TPM\_SetupPwm/TPM\_UpdateChnEdgeLevelSelect/TPM\_SetupInputCapture/TPM\_SetupOutputCompare/TPM\_SetupDualEdgeCapture, wait acknowledgement when the channel is disabled.

**[2.0.1]**

- Bug Fixes
  - Fixed TPM\_UpdateChnEdgeLevelSelect ACK wait issue.
  - Fixed the issue that TPM\_SetupDualEdgeCapture could not set FILTER register.
  - Fixed TPM\_UpdateChnEdgeLevelSelect ACK wait issue.

**[2.0.0]**

- Initial version.
- 

**TRGMUX****[2.0.1]**

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

## [2.0.0]

- Initial version.
- 

## TRNG

### [2.0.18]

- Bug fix:
  - TRNG health checks now done in software on RT5xx and RT6xx.

### [2.0.17]

- New features:
  - Add support for RT700.

### [2.0.16]

- Improvements:
  - Added support for Dual oscillator mode.

### [2.0.15]

- Other changes:
  - Changed TRNG\_USER\_CONFIG\_DEFAULT\_XXX values according to latest recommended by design team.

### [2.0.14]

- New features:
  - Add support for RW610 and RW612.

### [2.0.13]

- Bug fix:
  - After deepsleep it might return error, added clearing bits in TRNG\_GetRandomData() and generating new entropy.
  - Modified reloading entropy in TRNG\_GetRandomData(), for some data length it doesn't reloading entropy correctly.

### [2.0.12]

- Bug fix:
  - For KW34A4\_SERIES, KW35A4\_SERIES, KW36A4\_SERIES set TRNG\_USER\_CONFIG\_DEFAULT\_OSC\_DIV to kTRNG\_RingOscDiv8.



**[2.0.11]**

- Bug fix:
  - Add clearing pending errors in TRNG\_Init().

**[2.0.10]**

- Bug Fix:
  - Fixed doxygen issues.

**[2.0.9]**

- Bug Fix:
  - Fix HIS\_CCM metrics issues.

**[2.0.8]**

- Bug fix:
  - For K32L2A41A\_SERIES set TRNG\_USER\_CONFIG\_DEFAULT\_OSC\_DIV to kTRNG\_RingOscDiv4.

**[2.0.7]**

- Bug fix:
  - Fix MISRA 2004 issue rule 12.5.

**[2.0.6]**

- Bug fix:
  - For KW35Z4\_SERIES set TRNG\_USER\_CONFIG\_DEFAULT\_OSC\_DIV to kTRNG\_RingOscDiv8.

**[2.0.5]**

- Improvements:
  - For FRQMIN, FRQMAX and OSCDIV, add possibility to use device specific preprocessor macro to define default value in TRNG user configuration structure.

**[2.0.4]**

- Bug Fix:
  - Fix MISRA-2012 issues.
    - \* Rule 10.1, rule 10.3, rule 13.5, rule 16.1.

**[2.0.3]**

- Improvements:
  - update TRNG\_Init to restart new entropy generation.

### [2.0.2]

- Improvements:
  - fix MISRA issues
    - \* Rule 14.4.

### [2.0.1]

- New features:
  - Set default OSCDIV for Kinetis devices KL8x and KL28Z.
- Other changes:
  - Changed default OSCDIV for K81 to divide by 2.

### [2.0.0]

- Initial version.
- 

## TSI\_V4

### [2.1.3]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules:
    - \* Rule 10.1, 10.3, 10.4, 10.8, 12.2, 14.4, 17.7.

### [2.1.2]

- Bug Fixes
  - Fixed w1c issues in status handling API.
  - Fixed register naming error in API “static inline void TSI\_EnableEndOfScanDmaTransferOnly(TSI\_Type \*base, bool enable)”.
  - Removed redundant status flags clear APIs when enable interrupts.

### [2.1.1]

- New Features
  - Changed void TSI\_DeInit(TSI\_Type \*base) to void TSI\_Deinit(TSI\_Type \*base).

### [2.0.1]

- Other Changes
    - Changed default configuration structure member order.
-

## TSTMR

### [2.0.2]

- Improvements
  - Support 24MHz clock source.
- Bugfix
  - Fix MISRA C-2012 Rule 10.4 issue.
  - Read of TSTMR HIGH must follow TSTMR LOW atomically: require masking interrupt around 2 LSB / MSB accesses.

### [2.0.1]

- Bugfix
  - Restrict to read with 32-bit accesses only.
  - Restrict that TSTMR LOW read occurs first, followed by the TSTMR HIGH read.

### [2.0.0]

- Initial version.
- 

## VREF

### [2.1.3]

- Improvements
  - Add timeout for APIs with dfmea issues.

### [2.1.2]

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

### [2.1.1]

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

### [2.1.0]

- Improvements
  - Added new functions to support L5K board: added VREF\_SetTrim2V1Val() and VREF\_GetTrim2V1Val() functions to supply 2V1 output mode.

#### [2.0.0]

- Initial version.
- 

## WDOG32

#### [2.2.0]

- Improvements
  - Added while loop timeout config value for WDOG32 reconfiguration and unlock sequence.
  - Change the return type of WDOG32\_Init, WDOG32\_Deinit and WDOG32\_Unlock from void to status\_t.

#### [2.1.0]

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

#### [2.0.4]

- Improvements
  - To ensure that the reconfiguration is inside 128 bus clocks unlock window, put all reconfiguration APIs in quick access code section.

#### [2.0.3]

- Bug Fixes
  - Fixed the noncompliance issue of the reference document.
    - \* Waited until for new configuration to take effect by checking the RCS bit field.
    - \* Waited until for registers to be unlocked by checking the ULK bit field.
- Improvements
  - Added 128 bus clocks delay ensures a smooth transition before restarting the counter with the new configuration when there is no RCS status bit.

#### [2.0.2]

- Bug Fixes
  - MISRA C-2012 issue fixed.
    - \* Fixed rules, containing: rule-10.3, rule-14.4, rule-15.5.
  - Fixed the issue of the inseparable process interrupted by other interrupt source.
    - \* WDOG32\_Refresh

**[2.0.1]**

- Bug Fixes
  - WDOG must be configured within its configuration time period.
    - \* Added WDOG32\_Init API to quick access section.
    - \* Defined register variable in WDOG32\_Init API.

**[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.

[K32L2A41A](#)

## 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 FreeMASTER

[freemaster](#)

### 1.7.2 FreeRTOS

[FreeRTOS](#)

### 1.7.3 File systemFatfs

fatfs



# Chapter 2

## K32L2A41A

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

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

Initializes the ADC16 module.

#### Parameters

- `base` – ADC16 peripheral base address.
- `config` – Pointer to configuration structure. See “*adc16\_config\_t*”.

`void ADC16_Deinit(ADC_Type *base)`

De-initializes the ADC16 module.

#### Parameters

- `base` – ADC16 peripheral base address.

`void ADC16_GetDefaultConfig(adc16_config_t *config)`

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

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

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

#### Parameters

- `config` – Pointer to the configuration structure.

`status_t ADC16_DoAutoCalibration(ADC_Type *base)`

Automates the hardware calibration.

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

#### Parameters

- `base` – ADC16 peripheral base address.

**Return values**

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

**Returns**

Execution status.

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

Sets the offset value for the conversion result.

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

**Parameters**

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

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

Enables generating the DMA trigger when the conversion is complete.

**Parameters**

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

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

Enables the hardware trigger mode.

**Parameters**

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

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

Sets the channel mux mode.

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

**Parameters**

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

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

Configures the hardware compare mode.

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

**Parameters**

- `base` – ADC16 peripheral base address.



- `config` – Pointer to the “`adc16_hardware_compare_config_t`” structure. Passing “NULL” disables the feature.

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

Sets the hardware average mode.

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

#### Parameters

- `base` – ADC16 peripheral base address.
- `mode` – Setting the hardware average mode. See “`adc16_hardware_average_mode_t`”.

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

Configures the PGA for the converter’s front end.

#### Parameters

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

`uint32_t ADC16_GetStatusFlags(ADC_Type *base)`

Gets the status flags of the converter.

#### Parameters

- `base` – ADC16 peripheral base address.

#### Returns

Flags’ mask if indicated flags are asserted. See “`_adc16_status_flags`”.

`void ADC16_ClearStatusFlags(ADC_Type *base, uint32_t mask)`

Clears the status flags of the converter.

#### Parameters

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

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

Enable/disable ADC Asynchronous clock output to other modules.

#### Parameters

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

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

Configures the conversion channel.

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

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

#### Parameters

- base – ADC16 peripheral base address.
- channelGroup – Channel group index.
- config – Pointer to the “adc16\_channel\_config\_t” structure for the conversion channel.

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

Gets the conversion value.

#### Parameters

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

#### Returns

Conversion value.

```
uint32_t ADC16_GetChannelStatusFlags(ADC_Type *base, uint32_t channelGroup)
```

Gets the status flags of channel.

#### Parameters

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

#### Returns

Flags’ mask if indicated flags are asserted. See “\_adc16\_channel\_status\_flags”.

```
FSL_ADC16_DRIVER_VERSION
```

ADC16 driver version 2.3.0.

```
enum _adc16_channel_status_flags
```

Channel status flags.

*Values:*

```
enumerator kADC16_ChannelConversionDoneFlag
```

Conversion done.

```
enum _adc16_status_flags
```

Converter status flags.

*Values:*

enumerator kADC16\_ActiveFlag

Converter is active.

enumerator kADC16\_CalibrationFailedFlag

Calibration is failed.

enum \_adc\_channel\_mux\_mode

Channel multiplexer mode for each channel.

For some ADC16 channels, there are two pin selections in channel multiplexer. For example, ADC0\_SE4a and ADC0\_SE4b are the different channels that share the same channel number.

*Values:*

enumerator kADC16\_ChannelMuxA

For channel with channel mux a.

enumerator kADC16\_ChannelMuxB

For channel with channel mux b.

enum \_adc16\_clock\_divider

Clock divider for the converter.

*Values:*

enumerator kADC16\_ClockDivider1

For divider 1 from the input clock to the module.

enumerator kADC16\_ClockDivider2

For divider 2 from the input clock to the module.

enumerator kADC16\_ClockDivider4

For divider 4 from the input clock to the module.

enumerator kADC16\_ClockDivider8

For divider 8 from the input clock to the module.

enum \_adc16\_resolution

Converter's resolution.

*Values:*

enumerator kADC16\_Resolution8or9Bit

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

enumerator kADC16\_Resolution12or13Bit

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

enumerator kADC16\_Resolution10or11Bit

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

enumerator kADC16\_ResolutionSE8Bit

Single End 8-bit.

enumerator kADC16\_ResolutionSE12Bit

Single End 12-bit.

enumerator kADC16\_ResolutionSE10Bit

Single End 10-bit.

enumerator kADC16\_ResolutionDF9Bit

Differential Sample 9-bit.

enumerator kADC16\_\_ResolutionDF13Bit  
Differential Sample 13-bit.

enumerator kADC16\_\_ResolutionDF11Bit  
Differential Sample 11-bit.

enum \_\_adc16\_\_clock\_\_source  
Clock source.

*Values:*

enumerator kADC16\_\_ClockSourceAlt0  
Selection 0 of the clock source.

enumerator kADC16\_\_ClockSourceAlt1  
Selection 1 of the clock source.

enumerator kADC16\_\_ClockSourceAlt2  
Selection 2 of the clock source.

enumerator kADC16\_\_ClockSourceAlt3  
Selection 3 of the clock source.

enumerator kADC16\_\_ClockSourceAsynchronousClock  
Using internal asynchronous clock.

enum \_\_adc16\_\_long\_\_sample\_\_mode  
Long sample mode.

*Values:*

enumerator kADC16\_\_LongSampleCycle24  
20 extra ADCK cycles, 24 ADCK cycles total.

enumerator kADC16\_\_LongSampleCycle16  
12 extra ADCK cycles, 16 ADCK cycles total.

enumerator kADC16\_\_LongSampleCycle10  
6 extra ADCK cycles, 10 ADCK cycles total.

enumerator kADC16\_\_LongSampleCycle6  
2 extra ADCK cycles, 6 ADCK cycles total.

enumerator kADC16\_\_LongSampleDisabled  
Disable the long sample feature.

enum \_\_adc16\_\_reference\_\_voltage\_\_source  
Reference voltage source.

*Values:*

enumerator kADC16\_\_ReferenceVoltageSourceVref  
For external pins pair of VrefH and VrefL.

enumerator kADC16\_\_ReferenceVoltageSourceValt  
For alternate reference pair of ValtH and ValtL.

enum \_\_adc16\_\_hardware\_\_average\_\_mode  
Hardware average mode.

*Values:*

enumerator kADC16\_\_HardwareAverageCount4  
For hardware average with 4 samples.

enumerator kADC16\_\_HardwareAverageCount8

For hardware average with 8 samples.

enumerator kADC16\_\_HardwareAverageCount16

For hardware average with 16 samples.

enumerator kADC16\_\_HardwareAverageCount32

For hardware average with 32 samples.

enumerator kADC16\_\_HardwareAverageDisabled

Disable the hardware average feature.

enum \_\_adc16\_\_hardware\_\_compare\_\_mode

Hardware compare mode.

*Values:*

enumerator kADC16\_\_HardwareCompareMode0

$x < \text{value1}$ .

enumerator kADC16\_\_HardwareCompareMode1

$x > \text{value1}$ .

enumerator kADC16\_\_HardwareCompareMode2

if  $\text{value1} \leq \text{value2}$ , then  $x < \text{value1} \mid \mid x > \text{value2}$ ; else,  $\text{value1} > x > \text{value2}$ .

enumerator kADC16\_\_HardwareCompareMode3

if  $\text{value1} \leq \text{value2}$ , then  $\text{value1} \leq x \leq \text{value2}$ ; else  $x \geq \text{value1} \mid \mid x \leq \text{value2}$ .

enum \_\_adc16\_\_pga\_\_gain

PGA's Gain mode.

*Values:*

enumerator kADC16\_\_PGAGainValueOf1

For amplifier gain of 1.

enumerator kADC16\_\_PGAGainValueOf2

For amplifier gain of 2.

enumerator kADC16\_\_PGAGainValueOf4

For amplifier gain of 4.

enumerator kADC16\_\_PGAGainValueOf8

For amplifier gain of 8.

enumerator kADC16\_\_PGAGainValueOf16

For amplifier gain of 16.

enumerator kADC16\_\_PGAGainValueOf32

For amplifier gain of 32.

enumerator kADC16\_\_PGAGainValueOf64

For amplifier gain of 64.

typedef enum \_\_adc\_channel\_mux\_mode adc16\_channel\_mux\_mode\_t

Channel multiplexer mode for each channel.

For some ADC16 channels, there are two pin selections in channel multiplexer. For example, ADC0\_SE4a and ADC0\_SE4b are the different channels that share the same channel number.

typedef enum \_\_adc16\_clock\_divider adc16\_clock\_divider\_t

Clock divider for the converter.

```
typedef enum _adc16_resolution adc16_resolution_t
    Converter's resolution.

typedef enum _adc16_clock_source adc16_clock_source_t
    Clock source.

typedef enum _adc16_long_sample_mode adc16_long_sample_mode_t
    Long sample mode.

typedef enum _adc16_reference_voltage_source adc16_reference_voltage_source_t
    Reference voltage source.

typedef enum _adc16_hardware_average_mode adc16_hardware_average_mode_t
    Hardware average mode.

typedef enum _adc16_hardware_compare_mode adc16_hardware_compare_mode_t
    Hardware compare mode.

typedef enum _adc16_pga_gain adc16_pga_gain_t
    PGA's Gain mode.

typedef struct _adc16_config adc16_config_t
    ADC16 converter configuration.

typedef struct _adc16_hardware_compare_config adc16_hardware_compare_config_t
    ADC16 Hardware comparison configuration.

typedef struct _adc16_channel_config adc16_channel_config_t
    ADC16 channel conversion configuration.

typedef struct _adc16_pga_config adc16_pga_config_t
    ADC16 programmable gain amplifier configuration.

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

### Public Members

```
adc16_reference_voltage_source_t referenceVoltageSource
    Select the reference voltage source.

adc16_clock_source_t clockSource
    Select the input clock source to converter.

bool enableAsynchronousClock
    Enable the asynchronous clock output.

adc16_clock_divider_t clockDivider
    Select the divider of input clock source.

adc16_resolution_t resolution
    Select the sample resolution mode.

adc16_long_sample_mode_t longSampleMode
    Select the long sample mode.

bool enableHighSpeed
    Enable the high-speed mode.

bool enableLowPower
    Enable low power.
```

bool enableContinuousConversion

Enable continuous conversion mode.

*adc16\_hardware\_average\_mode\_t* hardwareAverageMode

Set hardware average mode.

struct *\_adc16\_hardware\_compare\_config*

*#include <fsl\_adc16.h>* ADC16 Hardware comparison configuration.

### Public Members

*adc16\_hardware\_compare\_mode\_t* hardwareCompareMode

Select the hardware compare mode. See “*adc16\_hardware\_compare\_mode\_t*”.

int16\_t value1

Setting value1 for hardware compare mode.

int16\_t value2

Setting value2 for hardware compare mode.

struct *\_adc16\_channel\_config*

*#include <fsl\_adc16.h>* ADC16 channel conversion configuration.

### Public Members

uint32\_t channelNumber

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

bool enableInterruptOnConversionCompleted

Generate an interrupt request once the conversion is completed.

bool enableDifferentialConversion

Using Differential sample mode.

struct *\_adc16\_pga\_config*

*#include <fsl\_adc16.h>* ADC16 programmable gain amplifier configuration.

### Public Members

*adc16\_pga\_gain\_t* pgaGain

Setting PGA gain.

bool enableRunInNormalMode

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

bool disablePgaChopping

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

bool enableRunInOffsetMeasurement

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

## 2.2 Clock Driver

enum \_clock\_name

Clock name used to get clock frequency.

*Values:*

enumerator kCLOCK\_CoreSysClk

Core/system clock

enumerator kCLOCK\_PlatClk

Platform clock

enumerator kCLOCK\_BusClk

Bus clock

enumerator kCLOCK\_FlexBusClk

FlexBus clock

enumerator kCLOCK\_FlashClk

Flash clock

enumerator kCLOCK\_ScgSysOscClk

SCG system OSC clock. (SYSOSC)

enumerator kCLOCK\_ScgSircClk

SCG SIRC clock.

enumerator kCLOCK\_ScgFircClk

SCG FIRC clock.

enumerator kCLOCK\_ScgSysPllClk

SCG system PLL clock. (SYSPLL)

enumerator kCLOCK\_ScgSysOscAsyncDiv1Clk

SOSCDIV1\_CLK.

enumerator kCLOCK\_ScgSysOscAsyncDiv2Clk

SOSCDIV2\_CLK.

enumerator kCLOCK\_ScgSysOscAsyncDiv3Clk

SOSCDIV3\_CLK.

enumerator kCLOCK\_ScgSircAsyncDiv1Clk

SIRCDIV1\_CLK.

enumerator kCLOCK\_ScgSircAsyncDiv2Clk

SIRCDIV2\_CLK.

enumerator kCLOCK\_ScgSircAsyncDiv3Clk

SIRCDIV3\_CLK.

enumerator kCLOCK\_ScgFircAsyncDiv1Clk

FIRCDIV1\_CLK.

enumerator kCLOCK\_ScgFircAsyncDiv2Clk

FIRCDIV2\_CLK.

enumerator kCLOCK\_ScgFircAsyncDiv3Clk

FIRCDIV3\_CLK.



enumerator kCLOCK\_ScgSysPllAsyncDiv1Clk  
SPLLDIV1\_CLK.

enumerator kCLOCK\_ScgSysPllAsyncDiv2Clk  
SPLLDIV2\_CLK.

enumerator kCLOCK\_ScgSysPllAsyncDiv3Clk  
SPLLDIV3\_CLK.

enumerator kCLOCK\_LpoClk  
LPO clock

enumerator kCLOCK\_Osc32kClk  
External OSC 32K clock (OSC32KCLK)

enumerator kCLOCK\_ErClk  
ERCLK. The external reference clock from SCG.

enum \_clock\_ip\_src

Clock source for peripherals that support various clock selections.

*Values:*

enumerator kCLOCK\_IpSrcNoneOrExt  
Clock is off or external clock is used.

enumerator kCLOCK\_IpSrcSysOscAsync  
System Oscillator async clock.

enumerator kCLOCK\_IpSrcSircAsync  
Slow IRC async clock.

enumerator kCLOCK\_IpSrcFircAsync  
Fast IRC async clock.

enumerator kCLOCK\_IpSrcSysPllAsync  
System PLL async clock.

enum \_clock\_ip\_name

Peripheral clock name definition used for clock gate, clock source and clock divider setting. It is defined as the corresponding register address.

*Values:*

enumerator kCLOCK\_IpInvalid

enumerator kCLOCK\_Dma0

enumerator kCLOCK\_Flash0

enumerator kCLOCK\_Dmamux0

enumerator kCLOCK\_Intmux0

enumerator kCLOCK\_Tpm2

enumerator kCLOCK\_Lpit0

enumerator kCLOCK\_Lptmr0

enumerator kCLOCK\_Rtc0

enumerator kCLOCK\_Lpspi2

enumerator kCLOCK\_\_Lpi2c2  
enumerator kCLOCK\_\_Lpuart2  
enumerator kCLOCK\_\_Sai0  
enumerator kCLOCK\_\_Emvsim0  
enumerator kCLOCK\_\_Usbfs0  
enumerator kCLOCK\_\_PortA  
enumerator kCLOCK\_\_PortB  
enumerator kCLOCK\_\_PortC  
enumerator kCLOCK\_\_PortD  
enumerator kCLOCK\_\_PortE  
enumerator kCLOCK\_\_Tsi0  
enumerator kCLOCK\_\_Adc0  
enumerator kCLOCK\_\_Dac0  
enumerator kCLOCK\_\_Cmp0  
enumerator kCLOCK\_\_Vref0  
enumerator kCLOCK\_\_Crc0  
enumerator kCLOCK\_\_Trng0  
enumerator kCLOCK\_\_Tpm0  
enumerator kCLOCK\_\_Tpm1  
enumerator kCLOCK\_\_Lptmr1  
enumerator kCLOCK\_\_Lpspi0  
enumerator kCLOCK\_\_Lpspi1  
enumerator kCLOCK\_\_Lpi2c0  
enumerator kCLOCK\_\_Lpi2c1  
enumerator kCLOCK\_\_Lpuart0  
enumerator kCLOCK\_\_Lpuart1  
enumerator kCLOCK\_\_Flexio0  
enumerator kCLOCK\_\_Cmp1

SCG status return codes.

*Values:*

enumerator kStatus\_SCG\_Busy  
Clock is busy.  
enumerator kStatus\_SCG\_InvalidSrc  
Invalid source.

enum \_scg\_sys\_clk

SCG system clock type.

*Values:*

enumerator kSCG\_SysClkSlow

System slow clock.

enumerator kSCG\_SysClkCore

Core clock.

enum \_scg\_sys\_clk\_src

SCG system clock source.

*Values:*

enumerator kSCG\_SysClkSrcSysOsc

System OSC.

enumerator kSCG\_SysClkSrcSirc

Slow IRC.

enumerator kSCG\_SysClkSrcFirc

Fast IRC.

enumerator kSCG\_SysClkSrcSysPll

System PLL.

enum \_scg\_sys\_clk\_div

SCG system clock divider value.

*Values:*

enumerator kSCG\_SysClkDivBy1

Divided by 1.

enumerator kSCG\_SysClkDivBy2

Divided by 2.

enumerator kSCG\_SysClkDivBy3

Divided by 3.

enumerator kSCG\_SysClkDivBy4

Divided by 4.

enumerator kSCG\_SysClkDivBy5

Divided by 5.

enumerator kSCG\_SysClkDivBy6

Divided by 6.

enumerator kSCG\_SysClkDivBy7

Divided by 7.

enumerator kSCG\_SysClkDivBy8

Divided by 8.

enumerator kSCG\_SysClkDivBy9

Divided by 9.

enumerator kSCG\_SysClkDivBy10

Divided by 10.

enumerator kSCG\_SysClkDivBy11

Divided by 11.

enumerator kSCG\_SysClkDivBy12

Divided by 12.

enumerator kSCG\_SysClkDivBy13

Divided by 13.

enumerator kSCG\_SysClkDivBy14

Divided by 14.

enumerator kSCG\_SysClkDivBy15

Divided by 15.

enumerator kSCG\_SysClkDivBy16

Divided by 16.

enum \_clock\_clkout\_src

SCG clock out configuration (CLKOUTSEL).

*Values:*

enumerator kClockClkoutSelScgSlow

SCG slow clock.

enumerator kClockClkoutSelSysOsc

System OSC.

enumerator kClockClkoutSelSirc

Slow IRC.

enumerator kClockClkoutSelFirc

Fast IRC.

enumerator kClockClkoutSelSysPll

System PLL.

enum \_scg\_async\_clk

SCG asynchronous clock type.

*Values:*

enumerator kSCG\_AsyncDiv1Clk

The async clock by DIV1, e.g. SOSCDIV1\_CLK, SIRCDIV1\_CLK.

enumerator kSCG\_AsyncDiv2Clk

The async clock by DIV2, e.g. SOSCDIV2\_CLK, SIRCDIV2\_CLK.

enumerator kSCG\_AsyncDiv3Clk

The async clock by DIV3, e.g. SOSCDIV3\_CLK, SIRCDIV3\_CLK.

enum scg\_async\_clk\_div

SCG asynchronous clock divider value.

*Values:*

enumerator kSCG\_AsyncClkDisable

Clock output is disabled.

enumerator kSCG\_AsyncClkDivBy1

Divided by 1.

enumerator kSCG\_\_AsyncClkDivBy2  
Divided by 2.

enumerator kSCG\_\_AsyncClkDivBy4  
Divided by 4.

enumerator kSCG\_\_AsyncClkDivBy8  
Divided by 8.

enumerator kSCG\_\_AsyncClkDivBy16  
Divided by 16.

enumerator kSCG\_\_AsyncClkDivBy32  
Divided by 32.

enumerator kSCG\_\_AsyncClkDivBy64  
Divided by 64.

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

*Values:*

enumerator kSCG\_\_SysOscMonitorDisable  
Monitor disabled.

enumerator kSCG\_\_SysOscMonitorInt  
Interrupt when the system OSC error is detected.

enumerator kSCG\_\_SysOscMonitorReset  
Reset when the system OSC error is detected.

Oscillator capacitor load setting.

*Values:*

enumerator kSCG\_\_SysOscCap2P  
2 pF capacitor load

enumerator kSCG\_\_SysOscCap4P  
4 pF capacitor load

enumerator kSCG\_\_SysOscCap8P  
8 pF capacitor load

enumerator kSCG\_\_SysOscCap16P  
16 pF capacitor load

enum \_\_scg\_\_sosc\_\_mode  
OSC work mode.

*Values:*

enumerator kSCG\_\_SysOscModeExt  
Use external clock.

enumerator kSCG\_\_SysOscModeOscLowPower  
Oscillator low power.

enumerator kSCG\_\_SysOscModeOscHighGain  
Oscillator high gain.

OSC enable mode.

*Values:*

enumerator kSCG\_SysOscEnable  
Enable OSC clock.

enumerator kSCG\_SysOscEnableInStop  
Enable OSC in stop mode.

enumerator kSCG\_SysOscEnableInLowPower  
Enable OSC in low power mode.

enumerator kSCG\_SysOscEnableErClk  
Enable OSCERCLK.

enum \_scg\_sirc\_range

SCG slow IRC clock frequency range.

*Values:*

enumerator kSCG\_SircRangeLow  
Slow IRC low range clock (2 MHz, 4 MHz for i.MX 7 ULP).

enumerator kSCG\_SircRangeHigh  
Slow IRC high range clock (8 MHz, 16 MHz for i.MX 7 ULP).

SIRC enable mode.

*Values:*

enumerator kSCG\_SircEnable  
Enable SIRC clock.

enumerator kSCG\_SircEnableInStop  
Enable SIRC in stop mode.

enumerator kSCG\_SircEnableInLowPower  
Enable SIRC in low power mode.

enum \_scg\_firc\_trim\_mode

SCG fast IRC trim mode.

*Values:*

enumerator kSCG\_FircTrimNonUpdate  
FIRC 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 scg\_firc\_trim\_config\_t.

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

enum \_scg\_firc\_trim\_div

SCG fast IRC trim predivided value for system OSC.

*Values:*

enumerator kSCG\_FircTrimDivBy1  
Divided by 1.

enumerator kSCG\_FircTrimDivBy128  
Divided by 128.

enumerator kSCG\_FircTrimDivBy256  
Divided by 256.

enumerator kSCG\_FircTrimDivBy512  
Divided by 512.

enumerator kSCG\_FircTrimDivBy1024  
Divided by 1024.

enumerator kSCG\_FircTrimDivBy2048  
Divided by 2048.

enum \_\_scg\_firc\_trim\_src  
SCG fast IRC trim source.

*Values:*

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

enumerator kSCG\_FircTrimSrcSysOsc  
System OSC.

enum \_\_scg\_firc\_range  
SCG fast IRC clock frequency range.

*Values:*

enumerator kSCG\_FircRange48M  
Fast IRC is trimmed to 48 MHz.

enumerator kSCG\_FircRange52M  
Fast IRC is trimmed to 52 MHz.

enumerator kSCG\_FircRange56M  
Fast IRC is trimmed to 56 MHz.

enumerator kSCG\_FircRange60M  
Fast IRC is trimmed to 60 MHz.

FIRC enable mode.

*Values:*

enumerator kSCG\_FircEnable  
Enable FIRC clock.

enumerator kSCG\_FircEnableInStop  
Enable FIRC in stop mode.

enumerator kSCG\_FircEnableInLowPower  
Enable FIRC in low power mode.

enumerator kSCG\_FircDisableRegulator  
Disable regulator.

enum \_\_scg\_pll\_src  
SCG system PLL clock source.

*Values:*

enumerator kSCG\_SysPllSrcSysOsc  
System PLL clock source is system OSC.

enumerator kSCG\_SysPllSrcFirc  
System PLL clock source is fast IRC.

enum \_scg\_spll\_monitor\_mode  
SCG system PLL monitor mode.

*Values:*

enumerator kSCG\_SysPllMonitorDisable  
Monitor disabled.

enumerator kSCG\_SysPllMonitorInt  
Interrupt when the system PLL error is detected.

enumerator kSCG\_SysPllMonitorReset  
Reset when the system PLL error is detected.

SPLL enable mode.

*Values:*

enumerator kSCG\_SysPllEnable  
Enable SPLL clock.

enumerator kSCG\_SysPllEnableInStop  
Enable SPLL in stop mode.

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

typedef enum \_clock\_ip\_src clock\_ip\_src\_t  
Clock source for peripherals that support various clock selections.

typedef enum \_clock\_ip\_name clock\_ip\_name\_t  
Peripheral clock name definition used for clock gate, clock source and clock divider setting.  
It is defined as the corresponding register address.

typedef enum \_scg\_sys\_clk scg\_sys\_clk\_t  
SCG system clock type.

typedef enum \_scg\_sys\_clk\_src scg\_sys\_clk\_src\_t  
SCG system clock source.

typedef enum \_scg\_sys\_clk\_div scg\_sys\_clk\_div\_t  
SCG system clock divider value.

typedef struct \_scg\_sys\_clk\_config scg\_sys\_clk\_config\_t  
SCG system clock configuration.

typedef enum \_clock\_clkout\_src clock\_clkout\_src\_t  
SCG clock out configuration (CLKOUTSEL).

typedef enum \_scg\_async\_clk scg\_async\_clk\_t  
SCG asynchronous clock type.

typedef enum \_scg\_async\_clk\_div scg\_async\_clk\_div\_t  
SCG asynchronous clock divider value.

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



```
typedef enum _scg_sosc_mode scg_sosc_mode_t
    OSC work mode.

typedef struct _scg_sosc_config scg_sosc_config_t
    SCG system OSC configuration.

typedef enum _scg_sirc_range scg_sirc_range_t
    SCG slow IRC clock frequency range.

typedef struct _scg_sirc_config scg_sirc_config_t
    SCG slow IRC clock configuration.

typedef enum _scg_firc_trim_mode scg_firc_trim_mode_t
    SCG fast IRC trim mode.

typedef enum _scg_firc_trim_div scg_firc_trim_div_t
    SCG fast IRC trim predivided value for system OSC.

typedef enum _scg_firc_trim_src scg_firc_trim_src_t
    SCG fast IRC trim source.

typedef struct _scg_firc_trim_config scg_firc_trim_config_t
    SCG fast IRC clock trim configuration.

typedef enum _scg_firc_range scg_firc_range_t
    SCG fast IRC clock frequency range.

typedef struct _scg_firc_config_t scg_firc_config_t
    SCG fast IRC clock configuration.

typedef enum _scg_spill_src scg_spill_src_t
    SCG system PLL clock source.

typedef enum _scg_spill_monitor_mode scg_spill_monitor_mode_t
    SCG system PLL monitor mode.

typedef struct _scg_spill_config scg_spill_config_t
    SCG system PLL configuration.

volatile uint32_t g_xtal0Freq
```

External XTAL0 (OSC0/SYSOSC) clock frequency.

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

```
CLOCK_InitSysOsc(...);
CLOCK_SetXtal0Freq(8000000);
```

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

```
static inline void CLOCK_EnableClock(clock_ip_name_t name)
    Enable the clock for specific IP.
```

#### Parameters

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

```
static inline void CLOCK_DisableClock(clock_ip_name_t name)
    Disable the clock for specific IP.
```

#### Parameters

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

static inline bool `CLOCK_IsEnabledByOtherCore(clock_ip_name_t name)`

Check whether the clock is already enabled and configured by any other core.

#### Parameters

- `name` – Which peripheral to check, see `clock_ip_name_t`.

#### Returns

True if clock is already enabled, otherwise false.

static inline void `CLOCK_SetIpSrc(clock_ip_name_t name, clock_ip_src_t src)`

Set the clock source for specific IP module.

Set the clock source for specific IP, not all modules need to set the clock source, should only use this function for the modules need source setting.

#### Parameters

- `name` – Which peripheral to check, see `clock_ip_name_t`.
- `src` – Clock source to set.

static inline void `CLOCK_SetIpSrcDiv(clock_ip_name_t name, clock_ip_src_t src, uint8_t divValue, uint8_t fracValue)`

Set the clock source and divider for specific IP module.

Set the clock source and divider for specific IP, not all modules need to set the clock source and divider, should only use this function for the modules need source and divider setting.

Divider output clock = Divider input clock x [(fracValue+1)/(divValue+1)].

#### Parameters

- `name` – Which peripheral to check, see `clock_ip_name_t`.
- `src` – Clock source to set.
- `divValue` – The divider value.
- `fracValue` – The fraction multiply value.

uint32\_t `CLOCK_GetFreq(clock_name_t clockName)`

Gets the clock frequency for a specific clock name.

This function checks the current clock configurations and then calculates the clock frequency for a specific clock name defined in `clock_name_t`.

#### Parameters

- `clockName` – Clock names defined in `clock_name_t`

#### Returns

Clock frequency value in hertz

uint32\_t `CLOCK_GetCoreSysClkFreq(void)`

Get the core clock or system clock frequency.

#### Returns

Clock frequency in Hz.

uint32\_t `CLOCK_GetPlatClkFreq(void)`

Get the platform clock frequency.

#### Returns

Clock frequency in Hz.

uint32\_t CLOCK\_GetBusClkFreq(void)

Get the bus clock frequency.

**Returns**

Clock frequency in Hz.

uint32\_t CLOCK\_GetFlashClkFreq(void)

Get the flash clock frequency.

**Returns**

Clock frequency in Hz.

uint32\_t CLOCK\_GetOsc32kClkFreq(void)

Get the OSC 32K clock frequency (OSC32KCLK).

**Returns**

Clock frequency in Hz.

uint32\_t CLOCK\_GetErClkFreq(void)

Get the external reference clock frequency (ERCLK).

**Returns**

Clock frequency in Hz.

uint32\_t CLOCK\_GetIpFreq(*clock\_ip\_name\_t* name)

Gets the clock frequency for a specific IP module.

This function gets the IP module clock frequency based on PCC registers. It is only used for the IP modules which could select clock source by PCC[PCS].

**Parameters**

- name – Which peripheral to get, see *clock\_ip\_name\_t*.

**Returns**

Clock frequency value in hertz

bool CLOCK\_EnableUsbfs0Clock(*clock\_ip\_src\_t* src, uint32\_t freq)

Enable USB FS clock.

**Parameters**

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

**Return values**

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

static inline void CLOCK\_DisableUsbfs0Clock(void)

Disable USB FS clock.

Disable USB FS clock.

FSL\_CLOCK\_DRIVER\_VERSION

CLOCK driver version 2.3.2.

SDK\_DEVICE\_MAXIMUM\_CPU\_CLOCK\_FREQUENCY

DMAMUX\_CLOCKS

Clock ip name array for DMAMUX.

RTC\_CLOCKS

Clock ip name array for RTC.

SAI\_CLOCKS

Clock ip name array for SAI.

PORT\_CLOCKS

Clock ip name array for PORT.

LPI2C\_CLOCKS

Clock ip name array for LPI2C.

FLEXIO\_CLOCKS

Clock ip name array for FLEXIO.

TSI\_CLOCKS

Clock ip name array for TSI.

EMVSIM\_CLOCKS

Clock ip name array for EMVSIM.

EDMA\_CLOCKS

Clock ip name array for EDMA.

LPUART\_CLOCKS

Clock ip name array for LPUART.

DAC\_CLOCKS

Clock ip name array for DAC.

LPTMR\_CLOCKS

Clock ip name array for LPTMR.

ADC16\_CLOCKS

Clock ip name array for ADC16.

INTMUX\_CLOCKS

Clock ip name array for INTMUX.

TRNG\_CLOCKS

Clock ip name array for TRNG.

LPSPI\_CLOCKS

Clock ip name array for LPSPI.

VREF\_CLOCKS

Clock ip name array for VREF.

TPM\_CLOCKS

Clock ip name array for TPM.

LPIT\_CLOCKS

Clock ip name array for LPIT.

CRC\_CLOCKS

Clock ip name array for CRC.

FLASH\_CLOCKS

Clock ip name array for FLASH.

CMP\_CLOCKS

Clock ip name array for CMP.

LPO\_CLK\_FREQ

LPO clock frequency.

kCLOCK\_Osc0ErClk

For compatible with other MCG platforms.

kCLOCK\_Er32kClk

For compatible with other MCG platforms.

CLOCK\_GetOsc0ErClkFreq

For compatible with other MCG platforms.

CLOCK\_GetEr32kClkFreq

For compatible with other MCG platforms.

uint32\_t CLOCK\_GetSysClkFreq(*scg\_sys\_clk\_t* type)

Gets the SCG system clock frequency.

This function gets the SCG system clock frequency. These clocks are used for core, platform, external, and bus clock domains.

#### Parameters

- type – Which type of clock to get, core clock or slow clock.

#### Returns

Clock frequency.

static inline void CLOCK\_SetVlprModeSysClkConfig(const *scg\_sys\_clk\_config\_t* \*config)

Sets the system clock configuration for VLPR mode.

This function sets the system clock configuration for VLPR mode.

#### Parameters

- config – Pointer to the configuration.

static inline void CLOCK\_SetRunModeSysClkConfig(const *scg\_sys\_clk\_config\_t* \*config)

Sets the system clock configuration for RUN mode.

This function sets the system clock configuration for RUN mode.

#### Parameters

- config – Pointer to the configuration.

static inline void CLOCK\_SetHsruntimeModeSysClkConfig(const *scg\_sys\_clk\_config\_t* \*config)

Sets the system clock configuration for HSRUN mode.

This function sets the system clock configuration for HSRUN mode.

#### Parameters

- config – Pointer to the configuration.

static inline void CLOCK\_GetCurSysClkConfig(*scg\_sys\_clk\_config\_t* \*config)

Gets the system clock configuration in the current power mode.

This function gets the system configuration in the current power mode.

#### Parameters

- config – Pointer to the configuration.

static inline void CLOCK\_SetClkOutSel(*clock\_clkout\_src\_t* setting)

Sets the clock out selection.

This function sets the clock out selection (CLKOUTSEL).

#### Parameters

- setting – The selection to set.

**Returns**

The current clock out selection.

*status\_t* CLOCK\_InitSysOsc(const *scg\_sosc\_config\_t* \*config)

Initializes the SCG system OSC.

This function enables the SCG system OSC clock according to the configuration.

---

**Note:** This function can't detect whether the system OSC has been enabled and used by an IP.

---

**Parameters**

- config – Pointer to the configuration structure.

**Return values**

- kStatus\_Success – System OSC is initialized.
- kStatus\_SCG\_Busy – System OSC has been enabled and is used by the system clock.
- kStatus\_ReadOnly – System OSC control register is locked.

*status\_t* CLOCK\_DeinitSysOsc(void)

De-initializes the SCG system OSC.

This function disables the SCG system OSC clock.

---

**Note:** This function can't detect whether the system OSC is used by an IP.

---

**Return values**

- kStatus\_Success – System OSC is deinitialized.
- kStatus\_SCG\_Busy – System OSC is used by the system clock.
- kStatus\_ReadOnly – System OSC control register is locked.

static inline void CLOCK\_SetSysOscAsyncClkDiv(*scg\_async\_clk\_t* asyncClk, *scg\_async\_clk\_div\_t* divider)

Set the asynchronous clock divider.

---

**Note:** There might be glitch when changing the asynchronous divider, so make sure the asynchronous clock is not used while changing divider.

---

**Parameters**

- asyncClk – Which asynchronous clock to configure.
- divider – The divider value to set.

*uint32\_t* CLOCK\_GetSysOscFreq(void)

Gets the SCG system OSC clock frequency (SYSOSC).

**Returns**

Clock frequency; If the clock is invalid, returns 0.

uint32\_t CLOCK\_GetSysOscAsyncFreq(*scg\_async\_clk\_t* type)

Gets the SCG asynchronous clock frequency from the system OSC.

**Parameters**

- type – The asynchronous clock type.

**Returns**

Clock frequency; If the clock is invalid, returns 0.

static inline bool CLOCK\_IsSysOscErr(void)

Checks whether the system OSC clock error occurs.

**Returns**

True if the error occurs, false if not.

static inline void CLOCK\_ClearSysOscErr(void)

Clears the system OSC clock error.

static inline 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.

static inline bool CLOCK\_IsSysOscValid(void)

Checks whether the system OSC clock is valid.

**Returns**

True if clock is valid, false if not.

*status\_t* CLOCK\_InitSirc(const *scg\_sirc\_config\_t* \*config)

Initializes the SCG slow IRC clock.

This function enables the SCG slow IRC clock according to the configuration.

---

**Note:** This function can't detect whether the system OSC has been enabled and used by an IP.

---

**Parameters**

- config – Pointer to the configuration structure.

**Return values**

- kStatus\_Success – SIRC is initialized.
- kStatus\_SCG\_Busy – SIRC has been enabled and is used by system clock.
- kStatus\_ReadOnly – SIRC control register is locked.

*status\_t* CLOCK\_DeinitSirc(void)

De-initializes the SCG slow IRC.

This function disables the SCG slow IRC.

---

**Note:** This function can't detect whether the SIRC is used by an IP.

---

**Return values**

- `kStatus_Success` – SIRC is deinitialized.
- `kStatus_SCG_Busy` – SIRC is used by system clock.
- `kStatus_ReadOnly` – SIRC control register is locked.

`static inline void CLOCK_SetSircAsyncClkDiv(scg_async_clk_t asyncClk, scg_async_clk_div_t divider)`

Set the asynchronous clock divider.

---

**Note:** There might be glitch when changing the asynchronous divider, so make sure the asynchronous clock is not used while changing divider.

---

#### Parameters

- `asyncClk` – Which asynchronous clock to configure.
- `divider` – The divider value to set.

`uint32_t CLOCK_GetSircFreq(void)`

Gets the SCG SIRC clock frequency.

#### Returns

Clock frequency; If the clock is invalid, returns 0.

`uint32_t CLOCK_GetSircAsyncFreq(scg_async_clk_t type)`

Gets the SCG asynchronous clock frequency from the SIRC.

#### Parameters

- `type` – The asynchronous clock type.

#### Returns

Clock frequency; If the clock is invalid, returns 0.

`static inline bool CLOCK_IsSircValid(void)`

Checks whether the SIRC clock is valid.

#### Returns

True if clock is valid, false if not.

`status_t CLOCK_InitFirc(const scg_firc_config_t *config)`

Initializes the SCG fast IRC clock.

This function enables the SCG fast IRC clock according to the configuration.

---

**Note:** This function can't detect whether the FIRC has been enabled and used by an IP.

---

#### Parameters

- `config` – Pointer to the configuration structure.

#### Return values

- `kStatus_Success` – FIRC is initialized.
- `kStatus_SCG_Busy` – FIRC has been enabled and is used by the system clock.
- `kStatus_ReadOnly` – FIRC control register is locked.



*status\_t* CLOCK\_DeinitFirc(void)

De-initializes the SCG fast IRC.

This function disables the SCG fast IRC.

---

**Note:** This function can't detect whether the FIRC is used by an IP.

---

#### Return values

- kStatus\_Success – FIRC is deinitialized.
- kStatus\_SCG\_Busy – FIRC is used by the system clock.
- kStatus\_ReadOnly – FIRC control register is locked.

static inline void CLOCK\_SetFircAsyncClkDiv(*scg\_async\_clk\_t* asyncClk, *scg\_async\_clk\_div\_t* divider)

Set the asynchronous clock divider.

---

**Note:** There might be glitch when changing the asynchronous divider, so make sure the asynchronous clock is not used while changing divider.

---

#### Parameters

- asyncClk – Which asynchronous clock to configure.
- divider – The divider value to set.

uint32\_t CLOCK\_GetFircFreq(void)

Gets the SCG FIRC clock frequency.

#### Returns

Clock frequency; If the clock is invalid, returns 0.

uint32\_t CLOCK\_GetFircAsyncFreq(*scg\_async\_clk\_t* type)

Gets the SCG asynchronous clock frequency from the FIRC.

#### Parameters

- type – The asynchronous clock type.

#### Returns

Clock frequency; If the clock is invalid, returns 0.

static inline bool CLOCK\_IsFircErr(void)

Checks whether the FIRC clock error occurs.

#### Returns

True if the error occurs, false if not.

static inline void CLOCK\_ClearFircErr(void)

Clears the FIRC clock error.

static inline bool CLOCK\_IsFircValid(void)

Checks whether the FIRC clock is valid.

#### Returns

True if clock is valid, false if not.

```
uint32_t CLOCK_GetSysPllMultDiv(uint32_t refFreq, uint32_t desireFreq, uint8_t *mult, uint8_t *prediv)
```

Calculates the MULT and PREDIV for the PLL.

This function calculates the proper MULT and PREDIV to generate the desired PLL output frequency with the input reference clock frequency. It returns the closest frequency match that the PLL can generate. The corresponding MULT/PREDIV are returned with parameters. If the desired frequency is not valid, this function returns 0.

#### Parameters

- refFreq – The input reference clock frequency.
- desireFreq – The desired output clock frequency.
- mult – The value of MULT.
- prediv – The value of PREDIV.

#### Returns

The PLL output frequency with the MULT and PREDIV; If the desired frequency can't be generated, this function returns 0U.

```
status_t CLOCK_InitSysPll(const scg_spll_config_t *config)
```

Initializes the SCG system PLL.

This function enables the SCG system PLL clock according to the configuration. The system PLL can use the system OSC or FIRC as the clock source. Ensure that the source clock is valid before calling this function.

Example code for initializing SPL clock output:

```
const scg_spll_config_t g_scgSysPllConfig = { .enableMode = kSCG_SysPllEnable,  
                                              .monitorMode = kSCG_SysPllMonitorDisable,  
                                              .div1 = kSCG_AsyncClkDivBy1,  
                                              .div2 = kSCG_AsyncClkDisable,  
                                              .div3 = kSCG_AsyncClkDivBy2,  
                                              .src = kSCG_SysPllSrcFirc,  
                                              .isBypassSelected = false,  
                                              .isPfdSelected = false,  
                                              .prediv = 5U,  
                                              .pfdClkout = kSCG_AuxPllPfd0Clk,
```

---

**Note:** This function can't detect whether the system PLL has been enabled and used by an IP.

---

#### Parameters

- config – Pointer to the configuration structure.

#### Return values

- kStatus\_Success – System PLL is initialized.
- kStatus\_SCG\_Busy – System PLL has been enabled and is used by the system clock.
- kStatus\_ReadOnly – System PLL control register is locked.

```
status_t CLOCK_DeinitSysPll(void)
```

De-initializes the SCG system PLL.

This function disables the SCG system PLL.

---

**Note:** This function can't detect whether the system PLL is used by an IP.

---

### Return values

- `kStatus_Success` – system PLL is deinitialized.
- `kStatus_SCG_Busy` – system PLL is used by the system clock.
- `kStatus_ReadOnly` – System PLL control register is locked.

```
static inline void CLOCK_SetSysPllAsyncClkDiv(scg_async_clk_t asyncClk, scg_async_clk_div_t divider)
```

Set the asynchronous clock divider.

---

**Note:** There might be glitch when changing the asynchronous divider, so make sure the asynchronous clock is not used while changing divider.

---

### Parameters

- `asyncClk` – Which asynchronous clock to configure.
- `divider` – The divider value to set.

```
uint32_t CLOCK_GetSysPllFreq(void)
```

Gets the SCG system PLL clock frequency.

### Returns

Clock frequency; If the clock is invalid, returns 0.

```
uint32_t CLOCK_GetSysPllAsyncFreq(scg_async_clk_t type)
```

Gets the SCG asynchronous clock frequency from the system PLL.

### Parameters

- `type` – The asynchronous clock type.

### Returns

Clock frequency; If the clock is invalid, returns 0.

```
static inline bool CLOCK_IsSysPllErr(void)
```

Checks whether the system PLL clock error occurs.

### Returns

True if an error occurs, false if not.

```
static inline void CLOCK_ClearSysPllErr(void)
```

Clears the system PLL clock error.

```
static inline void CLOCK_SetSysPllMonitorMode(scg_spll_monitor_mode_t mode)
```

Sets the system PLL monitor mode.

This function sets the system PLL 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.

```
static inline bool CLOCK_IsSysPllValid(void)
```

Checks whether the system PLL clock is valid.

### Returns

True if the clock is valid, false if not.

static inline void CLOCK\_SetXtal0Freq(uint32\_t freq)

Sets the XTAL0 frequency based on board settings.

**Parameters**

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

uint32\_t divSlow

Slow clock divider, see scg\_sys\_clk\_div\_t.

uint32\_t \_\_pad0\_\_

Reserved.

uint32\_t \_\_pad1\_\_

Reserved.

uint32\_t \_\_pad2\_\_

Reserved.

uint32\_t divCore

Core clock divider, see scg\_sys\_clk\_div\_t.

uint32\_t \_\_pad3\_\_

Reserved.

uint32\_t src

System clock source, see scg\_sys\_clk\_src\_t.

uint32\_t \_\_pad4\_\_

reserved.

uint32\_t freq

System OSC frequency.

scg\_sosc\_monitor\_mode\_t monitorMode

Clock monitor mode selected.

uint8\_t enableMode

Enable mode, OR'ed value of \_scg\_sosc\_enable\_mode.

scg\_async\_clk\_div\_t div1

SOSCDIV1 value.

scg\_async\_clk\_div\_t div2

SOSCDIV2 value.

scg\_async\_clk\_div\_t div3

SOSCDIV3 value.

uint32\_t capLoad

Capacitor load, OR'ed value of \_scg\_sosc\_cap\_load.

scg\_sosc\_mode\_t workMode

OSC work mode.

uint32\_t enableMode

Enable mode, OR'ed value of \_scg\_sirc\_enable\_mode.

scg\_async\_clk\_div\_t div1

SIRCDIV1 value.

scg\_async\_clk\_div\_t div2

SIRCDIV2 value.

*scg\_async\_clk\_div\_t* div3  
SIRCDIV3 value.

*scg\_sirc\_range\_t* range  
Slow IRC frequency range.

*scg\_firc\_trim\_mode\_t* trimMode  
FIRC trim mode.

*scg\_firc\_trim\_src\_t* trimSrc  
Trim source.

*scg\_firc\_trim\_div\_t* trimDiv  
Trim predivided value for the system OSC.

uint8\_t trimCoar  
Trim coarse value; Irrelevant if trimMode is kSCG\_FircTrimUpdate.

uint8\_t trimFine  
Trim fine value; Irrelevant if trimMode is kSCG\_FircTrimUpdate.

uint32\_t enableMode  
Enable mode, OR'ed value of *\_scg\_firc\_enable\_mode*.

*scg\_async\_clk\_div\_t* div1  
FIRCDIV1 value.

*scg\_async\_clk\_div\_t* div2  
FIRCDIV2 value.

*scg\_async\_clk\_div\_t* div3  
FIRCDIV3 value.

*scg\_firc\_range\_t* range  
Fast IRC frequency range.

const *scg\_firc\_trim\_config\_t* \*trimConfig  
Pointer to the FIRC trim configuration; set NULL to disable trim.

uint8\_t enableMode  
Enable mode, OR'ed value of *\_scg\_spill\_enable\_mode*

*scg\_spill\_monitor\_mode\_t* monitorMode  
Clock monitor mode selected.

*scg\_async\_clk\_div\_t* div1  
SPLLDIV1 value.

*scg\_async\_clk\_div\_t* div2  
SPLLDIV2 value.

*scg\_async\_clk\_div\_t* div3  
SPLLDIV3 value.

*scg\_spill\_src\_t* src  
Clock source.

uint8\_t prediv  
PLL reference clock divider.

uint8\_t mult  
System PLL multiplier.

**FSL\_SDK\_DISABLE\_DRIVER\_CLOCK\_CONTROL**

Configure whether driver controls clock.

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

---

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

---

```
struct _scg_sys_clk_config
    #include <fsl_clock.h> SCG system clock configuration.

struct _scg_sosc_config
    #include <fsl_clock.h> SCG system OSC configuration.

struct _scg_sirc_config
    #include <fsl_clock.h> SCG slow IRC clock configuration.

struct _scg_firc_trim_config
    #include <fsl_clock.h> SCG fast IRC clock trim configuration.

struct _scg_firc_config_t
    #include <fsl_clock.h> SCG fast IRC clock configuration.

struct _scg_spll_config
    #include <fsl_clock.h> SCG system PLL configuration.
```

## 2.3 CMP: Analog Comparator Driver

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

Initializes the CMP.

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

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

### Parameters

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

```
void CMP_Deinit(CMP_Type *base)
```

De-initializes the CMP module.

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

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

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

**Parameters**

- base – CMP peripheral base address.

static inline void CMP\_Enable(CMP\_Type \*base, bool enable)

Enables/disables the CMP module.

**Parameters**

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

void CMP\_GetDefaultConfig(*cmp\_config\_t* \*config)

Initializes the CMP user configuration structure.

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

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

**Parameters**

- config – Pointer to the configuration structure.

void CMP\_SetInputChannels(CMP\_Type \*base, uint8\_t positiveChannel, uint8\_t negativeChannel)

Sets the input channels for the comparator.

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

**Parameters**

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

void CMP\_EnableDMA(CMP\_Type \*base, bool enable)

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

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

**Parameters**

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

static inline void CMP\_EnableWindowMode(CMP\_Type \*base, bool enable)

Enables/disables the window mode.

**Parameters**

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

static inline void CMP\_EnablePassThroughMode(CMP\_Type \*base, bool enable)  
Enables/disables the pass through mode.

**Parameters**

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

void CMP\_SetFilterConfig(CMP\_Type \*base, const *cmp\_filter\_config\_t* \*config)  
Configures the filter.

**Parameters**

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

void CMP\_SetDACConfig(CMP\_Type \*base, const *cmp\_dac\_config\_t* \*config)  
Configures the internal DAC.

**Parameters**

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

void CMP\_EnableInterrupts(CMP\_Type \*base, uint32\_t mask)  
Enables the interrupts.

**Parameters**

- base – CMP peripheral base address.
- mask – Mask value for interrupts. See “\_cmp\_interrupt\_enable”.

void CMP\_DisableInterrupts(CMP\_Type \*base, uint32\_t mask)  
Disables the interrupts.

**Parameters**

- base – CMP peripheral base address.
- mask – Mask value for interrupts. See “\_cmp\_interrupt\_enable”.

uint32\_t CMP\_GetStatusFlags(CMP\_Type \*base)  
Gets the status flags.

**Parameters**

- base – CMP peripheral base address.

**Returns**

Mask value for the asserted flags. See “\_cmp\_status\_flags”.

void CMP\_ClearStatusFlags(CMP\_Type \*base, uint32\_t mask)  
Clears the status flags.

**Parameters**

- base – CMP peripheral base address.
- mask – Mask value for the flags. See “\_cmp\_status\_flags”.

FSL\_CMP\_DRIVER\_VERSION  
CMP driver version 2.0.3.

enum \_cmp\_interrupt\_enable  
Interrupt enable/disable mask.

*Values:*



enumerator kCMP\_OutputRisingInterruptEnable

Comparator interrupt enable rising.

enumerator kCMP\_OutputFallingInterruptEnable

Comparator interrupt enable falling.

enum \_cmp\_status\_flags

Status flags' mask.

*Values:*

enumerator kCMP\_OutputRisingEventFlag

Rising-edge on the comparison output has occurred.

enumerator kCMP\_OutputFallingEventFlag

Falling-edge on the comparison output has occurred.

enumerator kCMP\_OutputAssertEventFlag

Return the current value of the analog comparator output.

enum \_cmp\_hysteresis\_mode

CMP Hysteresis mode.

*Values:*

enumerator kCMP\_HysteresisLevel0

Hysteresis level 0.

enumerator kCMP\_HysteresisLevel1

Hysteresis level 1.

enumerator kCMP\_HysteresisLevel2

Hysteresis level 2.

enumerator kCMP\_HysteresisLevel3

Hysteresis level 3.

enum \_cmp\_reference\_voltage\_source

CMP Voltage Reference source.

*Values:*

enumerator kCMP\_VrefSourceVin1

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

enumerator kCMP\_VrefSourceVin2

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

typedef enum \_cmp\_hysteresis\_mode cmp\_hysteresis\_mode\_t

CMP Hysteresis mode.

typedef enum \_cmp\_reference\_voltage\_source cmp\_reference\_voltage\_source\_t

CMP Voltage Reference source.

typedef struct \_cmp\_config cmp\_config\_t

Configures the comparator.

typedef struct \_cmp\_filter\_config cmp\_filter\_config\_t

Configures the filter.

typedef struct \_cmp\_dac\_config cmp\_dac\_config\_t

Configures the internal DAC.

struct \_cmp\_config

#include <fsl\_cmp.h> Configures the comparator.

### Public Members

bool enableCmp

Enable the CMP module.

cmp\_hysteresis\_mode\_t hysteresisMode

CMP Hysteresis mode.

bool enableHighSpeed

Enable High-speed (HS) comparison mode.

bool enableInvertOutput

Enable the inverted comparator output.

bool useUnfilteredOutput

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

bool enablePinOut

The comparator output is available on the associated pin.

bool enableTriggerMode

Enable the trigger mode.

struct \_cmp\_filter\_config

#include <fsl\_cmp.h> Configures the filter.

### Public Members

bool enableSample

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

uint8\_t filterCount

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

uint8\_t filterPeriod

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

struct \_cmp\_dac\_config

#include <fsl\_cmp.h> Configures the internal DAC.

### Public Members

cmp\_reference\_voltage\_source\_t referenceVoltageSource

Supply voltage reference source.

uint8\_t DACValue

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

## 2.4 CRC: Cyclic Redundancy Check Driver

FSL\_CRC\_DRIVER\_VERSION

CRC driver version. Version 2.0.4.

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 DAC: Digital-to-Analog Converter Driver

void `DAC_Init(DAC_Type *base, const dac_config_t *config)`

Initializes the DAC module.

This function initializes the DAC module including the following operations.

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

### Parameters

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

void `DAC_Deinit(DAC_Type *base)`

De-initializes the DAC module.

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

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

### Parameters

- `base` – DAC peripheral base address.

void DAC\_GetDefaultConfig(*dac\_config\_t* \*config)

Initializes the DAC user configuration structure.

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

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

#### Parameters

- config – Pointer to the configuration structure. See “*dac\_config\_t*”.

static inline void DAC\_Enable(DAC\_Type \*base, bool enable)

Enables the DAC module.

#### Parameters

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

static inline void DAC\_EnableBuffer(DAC\_Type \*base, bool enable)

Enables the DAC buffer.

#### Parameters

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

void DAC\_SetBufferConfig(DAC\_Type \*base, const *dac\_buffer\_config\_t* \*config)

Configures the CMP buffer.

#### Parameters

- base – DAC peripheral base address.
- config – Pointer to the configuration structure. See “*dac\_buffer\_config\_t*”.

void DAC\_GetDefaultBufferConfig(*dac\_buffer\_config\_t* \*config)

Initializes the DAC buffer configuration structure.

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

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

#### Parameters

- config – Pointer to the configuration structure. See “*dac\_buffer\_config\_t*”.

static inline void DAC\_EnableBufferDMA(DAC\_Type \*base, bool enable)

Enables the DMA for DAC buffer.

#### Parameters

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

void DAC\_SetBufferValue(DAC\_Type \*base, uint8\_t index, uint16\_t value)

Sets the value for items in the buffer.

#### Parameters

- `base` – DAC peripheral base address.
- `index` – Setting the index for items in the buffer. The available index should not exceed the size of the DAC buffer.
- `value` – Setting the value for items in the buffer. 12-bits are available.

`static inline void DAC_DoSoftwareTriggerBuffer(DAC_Type *base)`

Triggers the buffer using software and updates the read pointer of the DAC buffer.

This function triggers the function using software. The read pointer of the DAC buffer is updated with one step after this function is called. Changing the read pointer depends on the buffer's work mode.

#### Parameters

- `base` – DAC peripheral base address.

`static inline uint8_t DAC_GetBufferReadPointer(DAC_Type *base)`

Gets the current read pointer of the DAC buffer.

This function gets the current read pointer of the DAC buffer. The current output value depends on the item indexed by the read pointer. It is updated either by a software trigger or a hardware trigger.

#### Parameters

- `base` – DAC peripheral base address.

#### Returns

The current read pointer of the DAC buffer.

`void DAC_SetBufferReadPointer(DAC_Type *base, uint8_t index)`

Sets the current read pointer of the DAC buffer.

This function sets the current read pointer of the DAC buffer. The current output value depends on the item indexed by the read pointer. It is updated either by a software trigger or a hardware trigger. After the read pointer changes, the DAC output value also changes.

#### Parameters

- `base` – DAC peripheral base address.
- `index` – Setting an index value for the pointer.

`void DAC_EnableBufferInterrupts(DAC_Type *base, uint32_t mask)`

Enables interrupts for the DAC buffer.

#### Parameters

- `base` – DAC peripheral base address.
- `mask` – Mask value for interrupts. See “`_dac_buffer_interrupt_enable`”.

`void DAC_DisableBufferInterrupts(DAC_Type *base, uint32_t mask)`

Disables interrupts for the DAC buffer.

#### Parameters

- `base` – DAC peripheral base address.
- `mask` – Mask value for interrupts. See “`_dac_buffer_interrupt_enable`”.

`uint8_t DAC_GetBufferStatusFlags(DAC_Type *base)`

Gets the flags of events for the DAC buffer.

#### Parameters

- `base` – DAC peripheral base address.

**Returns**

Mask value for the asserted flags. See “\_dac\_buffer\_status\_flags”.

void DAC\_ClearBufferStatusFlags(DAC\_Type \*base, uint32\_t mask)

Clears the flags of events for the DAC buffer.

**Parameters**

- base – DAC peripheral base address.
- mask – Mask value for flags. See “\_dac\_buffer\_status\_flags\_t”.

FSL\_DAC\_DRIVER\_VERSION

DAC driver version 2.0.2.

enum \_dac\_buffer\_status\_flags

DAC buffer flags.

*Values:*

enumerator kDAC\_BufferWatermarkFlag

DAC Buffer Watermark Flag.

enumerator kDAC\_BufferReadPointerTopPositionFlag

DAC Buffer Read Pointer Top Position Flag.

enumerator kDAC\_BufferReadPointerBottomPositionFlag

DAC Buffer Read Pointer Bottom Position Flag.

enum \_dac\_buffer\_interrupt\_enable

DAC buffer interrupts.

*Values:*

enumerator kDAC\_BufferWatermarkInterruptEnable

DAC Buffer Watermark Interrupt Enable.

enumerator kDAC\_BufferReadPointerTopInterruptEnable

DAC Buffer Read Pointer Top Flag Interrupt Enable.

enumerator kDAC\_BufferReadPointerBottomInterruptEnable

DAC Buffer Read Pointer Bottom Flag Interrupt Enable

enum \_dac\_reference\_voltage\_source

DAC reference voltage source.

*Values:*

enumerator kDAC\_ReferenceVoltageSourceVref1

The DAC selects DACREF\_1 as the reference voltage.

enumerator kDAC\_ReferenceVoltageSourceVref2

The DAC selects DACREF\_2 as the reference voltage.

enum \_dac\_buffer\_trigger\_mode

DAC buffer trigger mode.

*Values:*

enumerator kDAC\_BufferTriggerByHardwareMode

The DAC hardware trigger is selected.

enumerator kDAC\_BufferTriggerBySoftwareMode

The DAC software trigger is selected.



enum `_dac_buffer_watermark`

DAC buffer watermark.

*Values:*

enumerator `kDAC_BufferWatermark1Word`

1 word away from the upper limit.

enumerator `kDAC_BufferWatermark2Word`

2 words away from the upper limit.

enumerator `kDAC_BufferWatermark3Word`

3 words away from the upper limit.

enumerator `kDAC_BufferWatermark4Word`

4 words away from the upper limit.

enum `_dac_buffer_work_mode`

DAC buffer work mode.

*Values:*

enumerator `kDAC_BufferWorkAsNormalMode`

Normal mode.

enumerator `kDAC_BufferWorkAsSwingMode`

Swing mode.

enumerator `kDAC_BufferWorkAsOneTimeScanMode`

One-Time Scan mode.

enumerator `kDAC_BufferWorkAsFIFOMode`

FIFO mode.

typedef enum `_dac_reference_voltage_source` `dac_reference_voltage_source_t`

DAC reference voltage source.

typedef enum `_dac_buffer_trigger_mode` `dac_buffer_trigger_mode_t`

DAC buffer trigger mode.

typedef enum `_dac_buffer_watermark` `dac_buffer_watermark_t`

DAC buffer watermark.

typedef enum `_dac_buffer_work_mode` `dac_buffer_work_mode_t`

DAC buffer work mode.

typedef struct `_dac_config` `dac_config_t`

DAC module configuration.

typedef struct `_dac_buffer_config` `dac_buffer_config_t`

DAC buffer configuration.

struct `_dac_config`

`#include <fsl_dac.h>` DAC module configuration.

### Public Members

`dac_reference_voltage_source_t` `referenceVoltageSource`

Select the DAC reference voltage source.

bool `enableLowPowerMode`

Enable the low-power mode.

```
struct _dac_buffer_config
#include <fsl_dac.h> DAC buffer configuration.
```

### Public Members

*dac\_buffer\_trigger\_mode\_t* triggerMode

Select the buffer's trigger mode.

*dac\_buffer\_watermark\_t* watermark

Select the buffer's watermark.

*dac\_buffer\_work\_mode\_t* workMode

Select the buffer's work mode.

uint8\_t upperLimit

Set the upper limit for the buffer index. Normally, 0-15 is available for a buffer with 16 items.

## 2.6 DMAMUX: Direct Memory Access Multiplexer Driver

void DMAMUX\_Init(DMAMUX\_Type \*base)

Initializes the DMAMUX peripheral.

This function ungates the DMAMUX clock.

### Parameters

- base – DMAMUX peripheral base address.

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

Deinitializes the DMAMUX peripheral.

This function gates the DMAMUX clock.

### Parameters

- base – DMAMUX peripheral base address.

static inline void DMAMUX\_EnableChannel(DMAMUX\_Type \*base, uint32\_t channel)

Enables the DMAMUX channel.

This function enables the DMAMUX channel.

### Parameters

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

static inline void DMAMUX\_DisableChannel(DMAMUX\_Type \*base, uint32\_t channel)

Disables the DMAMUX channel.

This function disables the DMAMUX channel.

---

**Note:** The user must disable the DMAMUX channel before configuring it.

---

### Parameters

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

static inline void DMAMUX\_SetSource(DMAMUX\_Type \*base, uint32\_t channel, int32\_t source)  
Configures the DMAMUX channel source.

**Parameters**

- base – DMAMUX peripheral base address.
- channel – DMAMUX channel number.
- source – Channel source, which is used to trigger the DMA transfer. User need to use the `dma_request_source_t` type as the input parameter.

static inline void DMAMUX\_EnablePeriodTrigger(DMAMUX\_Type \*base, uint32\_t channel)  
Enables the DMAMUX period trigger.

This function enables the DMAMUX period trigger feature.

**Parameters**

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

static inline void DMAMUX\_DisablePeriodTrigger(DMAMUX\_Type \*base, uint32\_t channel)  
Disables the DMAMUX period trigger.

This function disables the DMAMUX period trigger.

**Parameters**

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

FSL\_DMAMUX\_DRIVER\_VERSION  
DMAMUX driver version 2.1.1.

## 2.7 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

void EDMA\_Init(DMA\_Type \*base, const *edma\_config\_t* \*config)

Initializes the eDMA peripheral.

This function ungates the eDMA clock and configures the eDMA peripheral according to the configuration structure. All emda enabled request will be cleared in this function.

---

**Note:** This function enables the minor loop map feature.

---

**Parameters**

- base – eDMA peripheral base address.
- config – A pointer to the configuration structure, see “`edma_config_t`”.

void EDMA\_Deinit(DMA\_Type \*base)

Deinitializes the eDMA peripheral.

This function gates the eDMA clock.

**Parameters**

- base – eDMA peripheral base address.

```
void EDMA__InstallTCD(DMA_Type *base, uint32_t channel, edma_tcd_t *tcd)
```

Push content of TCD structure into hardware TCD register.

#### Parameters

- base – EDMA peripheral base address.
- channel – EDMA channel number.
- tcd – Point to TCD structure.

```
void EDMA__GetDefaultConfig(edma_config_t *config)
```

Gets the eDMA default configuration structure.

This function sets the configuration structure to default values. The default configuration is set to the following values.

```
config.enableContinuousLinkMode = false;  
config.enableHaltOnError = true;  
config.enableRoundRobinArbitration = false;  
config.enableDebugMode = false;
```

#### Parameters

- config – A pointer to the eDMA configuration structure.

```
static inline void EDMA__EnableContinuousChannelLinkMode(DMA_Type *base, bool enable)
```

Enable/Disable continuous channel link mode.

---

**Note:** Do not use continuous link mode with a channel linking to itself if there is only one minor loop iteration per service request, for example, if the channel's NBYTES value is the same as either the source or destination size. The same data transfer profile can be achieved by simply increasing the NBYTES value, which provides more efficient, faster processing.

---

#### Parameters

- base – EDMA peripheral base address.
- enable – true is enable, false is disable.

```
static inline void EDMA__EnableMinorLoopMapping(DMA_Type *base, bool enable)
```

Enable/Disable minor loop mapping.

The TCDn.word2 is redefined to include individual enable fields, an offset field, and the NBYTES field.

#### Parameters

- base – EDMA peripheral base address.
- enable – true is enable, false is disable.

```
void EDMA__ResetChannel(DMA_Type *base, uint32_t channel)
```

Sets all TCD registers to default values.

This function sets TCD registers for this channel to default values.

---

**Note:** This function must not be called while the channel transfer is ongoing or it causes unpredictable results.

---

---

**Note:** This function enables the auto stop request feature.

---

### Parameters

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

```
void EDMA_SetTransferConfig(DMA_Type *base, uint32_t channel, const edma_transfer_config_t
                           *config, edma_tcd_t *nextTcd)
```

Configures the eDMA transfer attribute.

This function configures the transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the TCD address. Example:

```
edma_transfer_t config;
edma_tcd_t tcd;
config.srcAddr = ..;
config.destAddr = ..;
...
EDMA_SetTransferConfig(DMA0, channel, &config, &stcd);
```

---

**Note:** If nextTcd is not NULL, it means scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the eDMA\_ResetChannel.

---

### Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- config – Pointer to eDMA transfer configuration structure.
- nextTcd – Point to TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

```
void EDMA_SetMinorOffsetConfig(DMA_Type *base, uint32_t channel, const
                               edma_minor_offset_config_t *config)
```

Configures the eDMA minor offset feature.

The minor offset means that the signed-extended value is added to the source address or destination address after each minor loop.

### Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- config – A pointer to the minor offset configuration structure.

```
void EDMA_SetChannelPreemptionConfig(DMA_Type *base, uint32_t channel, const
                                     edma_channel_preemption_config_t *config)
```

Configures the eDMA channel preemption feature.

This function configures the channel preemption attribute and the priority of the channel.

### Parameters

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

- `config` – A pointer to the channel preemption configuration structure.

```
void EDMA_SetChannelLink(DMA_Type *base, uint32_t channel, edma_channel_link_type_t linkType, uint32_t linkedChannel)
```

Sets the channel link for the eDMA transfer.

This function configures either the minor link or the major link mode. The minor link means that the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

---

**Note:** Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

---

### Parameters

- `base` – eDMA peripheral base address.
- `channel` – eDMA channel number.
- `linkType` – A channel link type, which can be one of the following:
  - `kEDMA_LinkNone`
  - `kEDMA_MinorLink`
  - `kEDMA_MajorLink`
- `linkedChannel` – The linked channel number.

```
void EDMA_SetBandWidth(DMA_Type *base, uint32_t channel, edma_bandwidth_t bandWidth)
```

Sets the bandwidth for the eDMA transfer.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

### Parameters

- `base` – eDMA peripheral base address.
- `channel` – eDMA channel number.
- `bandWidth` – A bandwidth setting, which can be one of the following:
  - `kEDMABandwidthStallNone`
  - `kEDMABandwidthStall4Cycle`
  - `kEDMABandwidthStall8Cycle`

```
void EDMA_SetModulo(DMA_Type *base, uint32_t channel, edma_modulo_t srcModulo, edma_modulo_t destModulo)
```

Sets the source modulo and the destination modulo for the eDMA transfer.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

### Parameters

- `base` – eDMA peripheral base address.
- `channel` – eDMA channel number.
- `srcModulo` – A source modulo value.
- `destModulo` – A destination modulo value.

`static inline void EDMA__EnableAsyncRequest(DMA_Type *base, uint32_t channel, bool enable)`  
Enables an async request for the eDMA transfer.

**Parameters**

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- enable – The command to enable (true) or disable (false).

`static inline void EDMA__EnableAutoStopRequest(DMA_Type *base, uint32_t channel, bool enable)`

Enables an auto stop request for the eDMA transfer.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

**Parameters**

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- enable – The command to enable (true) or disable (false).

`void EDMA__EnableChannelInterrupts(DMA_Type *base, uint32_t channel, uint32_t mask)`  
Enables the interrupt source for the eDMA transfer.

**Parameters**

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

`void EDMA__DisableChannelInterrupts(DMA_Type *base, uint32_t channel, uint32_t mask)`  
Disables the interrupt source for the eDMA transfer.

**Parameters**

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of the interrupt source to be set. Use the defined `edma_interrupt_enable_t` type.

`void EDMA__SetMajorOffsetConfig(DMA_Type *base, uint32_t channel, int32_t sourceOffset, int32_t destOffset)`

Configures the eDMA channel TCD major offset feature.

Adjustment value added to the source address at the completion of the major iteration count

**Parameters**

- base – eDMA peripheral base address.
- channel – edma channel number.
- sourceOffset – source address offset will be applied to source address after major loop done.
- destOffset – destination address offset will be applied to source address after major loop done.

void EDMA\_TcdReset(*edma\_tcd\_t* \*tcd)

Sets all fields to default values for the TCD structure.

This function sets all fields for this TCD structure to default value.

---

**Note:** This function enables the auto stop request feature.

---

#### Parameters

- tcd – Pointer to the TCD structure.

void EDMA\_TcdSetTransferConfig(*edma\_tcd\_t* \*tcd, const *edma\_transfer\_config\_t* \*config, *edma\_tcd\_t* \*nextTcd)

Configures the eDMA TCD transfer attribute.

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The TCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

```
edma_transfer_t config = {  
    ...  
}  
edma_tcd_t tcd __aligned(32);  
edma_tcd_t nextTcd __aligned(32);  
EDMA_TcdSetTransferConfig(&tcd, &config, &nextTcd);
```

---

**Note:** TCD address should be 32 bytes aligned or it causes an eDMA error.

---

---

**Note:** If the nextTcd is not NULL, the scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the EDMA\_TcdReset.

---

#### Parameters

- tcd – Pointer to the TCD structure.
- config – Pointer to eDMA transfer configuration structure.
- nextTcd – Pointer to the next TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

void EDMA\_TcdSetMinorOffsetConfig(*edma\_tcd\_t* \*tcd, const *edma\_minor\_offset\_config\_t* \*config)

Configures the eDMA TCD minor offset feature.

A minor offset is a signed-extended value added to the source address or a destination address after each minor loop.

#### Parameters

- tcd – A point to the TCD structure.
- config – A pointer to the minor offset configuration structure.

void EDMA\_TcdSetChannelLink(*edma\_tcd\_t* \*tcd, *edma\_channel\_link\_type\_t* linkType, uint32\_t linkedChannel)

Sets the channel link for the eDMA TCD.



This function configures either a minor link or a major link. The minor link means the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

---

**Note:** Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

---

#### Parameters

- `tcd` – Point to the TCD structure.
- `linkType` – Channel link type, it can be one of:
  - `kEDMA_LinkNone`
  - `kEDMA_MinorLink`
  - `kEDMA_MajorLink`
- `linkedChannel` – The linked channel number.

`static inline void EDMA__TcdSetBandWidth(edma_tcd_t *tcd, edma_bandwidth_t bandWidth)`

Sets the bandwidth for the eDMA TCD.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

#### Parameters

- `tcd` – A pointer to the TCD structure.
- `bandWidth` – A bandwidth setting, which can be one of the following:
  - `kEDMABandwidthStallNone`
  - `kEDMABandwidthStall4Cycle`
  - `kEDMABandwidthStall8Cycle`

`void EDMA__TcdSetModulo(edma_tcd_t *tcd, edma_modulo_t srcModulo, edma_modulo_t destModulo)`

Sets the source modulo and the destination modulo for the eDMA TCD.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

#### Parameters

- `tcd` – A pointer to the TCD structure.
- `srcModulo` – A source modulo value.
- `destModulo` – A destination modulo value.

`static inline void EDMA__TcdEnableAutoStopRequest(edma_tcd_t *tcd, bool enable)`

Sets the auto stop request for the eDMA TCD.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

#### Parameters

- `tcd` – A pointer to the TCD structure.
- `enable` – The command to enable (true) or disable (false).

void EDMA\_\_TcdEnableInterrupts(*edma\_tcd\_t* \*tcd, uint32\_t mask)

Enables the interrupt source for the eDMA TCD.

**Parameters**

- tcd – Point to the TCD structure.
- mask – The mask of interrupt source to be set. Users need to use the defined *edma\_interrupt\_enable\_t* type.

void EDMA\_\_TcdDisableInterrupts(*edma\_tcd\_t* \*tcd, uint32\_t mask)

Disables the interrupt source for the eDMA TCD.

**Parameters**

- tcd – Point to the TCD structure.
- mask – The mask of interrupt source to be set. Users need to use the defined *edma\_interrupt\_enable\_t* type.

void EDMA\_\_TcdSetMajorOffsetConfig(*edma\_tcd\_t* \*tcd, int32\_t sourceOffset, int32\_t destOffset)

Configures the eDMA TCD major offset feature.

Adjustment value added to the source address at the completion of the major iteration count

**Parameters**

- tcd – A point to the TCD structure.
- sourceOffset – source address offset 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(DMA\_Type \*base, uint32\_t channel)

Enables the eDMA hardware channel request.

This function enables the hardware channel request.

**Parameters**

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

static inline void EDMA\_\_DisableChannelRequest(DMA\_Type \*base, uint32\_t channel)

Disables the eDMA hardware channel request.

This function disables the hardware channel request.

**Parameters**

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

static inline void EDMA\_\_TriggerChannelStart(DMA\_Type \*base, uint32\_t channel)

Starts the eDMA transfer by using the software trigger.

This function starts a minor loop transfer.

**Parameters**

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

uint32\_t EDMA\_GetRemainingMajorLoopCount(DMA\_Type \*base, uint32\_t channel)

Gets the remaining major loop count from the eDMA current channel TCD.

This function checks the TCD (Task Control Descriptor) status for a specified eDMA channel and returns the number of major loop count that has not finished.

---

**Note:** 1. This function can only be used to get unfinished major loop count of transfer without the next TCD, or it might be inaccuracy.

- a. The unfinished/remaining transfer bytes cannot be obtained directly from registers while the channel is running. Because to calculate the remaining bytes, the initial NBYTES configured in DMA\_TCDn\_NBYTES\_MLNO register is needed while the eDMA IP does not support getting it while a channel is active. In another word, the NBYTES value reading is always the actual (decrementing) NBYTES value the dma\_engine is working with while a channel is running. Consequently, to get the remaining transfer bytes, a software-saved initial value of NBYTES (for example copied before enabling the channel) is needed. The formula to calculate it is shown below: RemainingBytes = RemainingMajorLoopCount \* NBYTES(initially configured)
- 

#### Parameters

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

#### Returns

Major loop count which has not been transferred yet for the current TCD.

static inline uint32\_t EDMA\_GetErrorStatusFlags(DMA\_Type \*base)

Gets the eDMA channel error status flags.

#### Parameters

- base – eDMA peripheral base address.

#### Returns

The mask of error status flags. Users need to use the \_edma\_error\_status\_flags type to decode the return variables.

uint32\_t EDMA\_GetChannelStatusFlags(DMA\_Type \*base, uint32\_t channel)

Gets the eDMA channel status flags.

#### Parameters

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

#### Returns

The mask of channel status flags. Users need to use the \_edma\_channel\_status\_flags type to decode the return variables.

void EDMA\_ClearChannelStatusFlags(DMA\_Type \*base, uint32\_t channel, uint32\_t mask)

Clears the eDMA channel status flags.

#### Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of channel status to be cleared. Users need to use the defined \_edma\_channel\_status\_flags type.

void EDMA\_CreateHandle(*edma\_handle\_t* \*handle, DMA\_Type \*base, uint32\_t channel)

Creates the eDMA handle.

This function is called if using the transactional API for eDMA. This function initializes the internal state of the eDMA handle.

#### Parameters

- handle – eDMA handle pointer. The eDMA handle stores callback function and parameters.
- base – eDMA peripheral base address.
- channel – eDMA channel number.

void EDMA\_InstallTCDMemory(*edma\_handle\_t* \*handle, *edma\_tcd\_t* \*tcdPool, uint32\_t tcdSize)

Installs the TCDs memory pool into the eDMA handle.

This function is called after the EDMA\_CreateHandle to use scatter/gather feature. This function shall only be used while users need to use scatter gather mode. Scatter gather mode enables EDMA to load a new transfer control block (tcd) in hardware, and automatically reconfigure that DMA channel for a new transfer. Users need to prepare tcd memory and also configure tcds using interface EDMA\_SubmitTransfer.

#### Parameters

- handle – eDMA handle pointer.
- tcdPool – A memory pool to store TCDs. It must be 32 bytes aligned.
- tcdSize – The number of TCD slots.

void EDMA\_SetCallback(*edma\_handle\_t* \*handle, *edma\_callback* callback, void \*userData)

Installs a callback function for the eDMA transfer.

This callback is called in the eDMA IRQ handler. Use the callback to do something after the current major loop transfer completes. This function will be called every time one tcd finished transfer.

#### Parameters

- handle – eDMA handle pointer.
- callback – eDMA callback function pointer.
- userData – A parameter for the callback function.

void EDMA\_PrepareTransferConfig(*edma\_transfer\_config\_t* \*config, void \*srcAddr, uint32\_t srcWidth, int16\_t srcOffset, void \*destAddr, uint32\_t destWidth, int16\_t destOffset, uint32\_t bytesEachRequest, uint32\_t transferBytes)

Prepares the eDMA transfer structure configurations.

This function prepares the transfer configuration structure according to the user input.

---

**Note:** The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

---

#### Parameters

- config – The user configuration structure of type *edma\_transfer\_t*.
- srcAddr – eDMA transfer source address.
- srcWidth – eDMA transfer source address width(bytes).

- srcOffset – source address offset.
- destAddr – eDMA transfer destination address.
- destWidth – eDMA transfer destination address width(bytes).
- destOffset – destination address offset.
- bytesEachRequest – eDMA transfer bytes per channel request.
- transferBytes – eDMA transfer bytes to be transferred.

```
void EDMA__PrepareTransfer(edma_transfer_config_t *config, void *srcAddr, uint32_t srcWidth,  
                          void *destAddr, uint32_t destWidth, uint32_t bytesEachRequest,  
                          uint32_t transferBytes, edma_transfer_type_t transferType)
```

Prepares the eDMA transfer structure.

This function prepares the transfer configuration structure according to the user input.

---

**Note:** The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

---

#### Parameters

- config – The user configuration structure of type *edma\_transfer\_t*.
- srcAddr – eDMA transfer source address.
- srcWidth – eDMA transfer source address width(bytes).
- destAddr – eDMA transfer destination address.
- destWidth – eDMA transfer destination address width(bytes).
- bytesEachRequest – eDMA transfer bytes per channel request.
- transferBytes – eDMA transfer bytes to be transferred.
- transferType – eDMA transfer type.

```
status_t EDMA__SubmitTransfer(edma_handle_t *handle, const edma_transfer_config_t *config)
```

Submits the eDMA transfer request.

This function submits the eDMA transfer request according to the transfer configuration structure. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function EDMA\_InstallTCDMemory before.

#### Parameters

- handle – eDMA handle pointer.
- config – Pointer to eDMA transfer configuration structure.

#### Return values

- kStatus\_EDMA\_Success – It means submit transfer request succeed.
- kStatus\_EDMA\_QueueFull – It means TCD queue is full. Submit transfer request is not allowed.
- kStatus\_EDMA\_Busy – It means the given channel is busy, need to submit request later.

```
void EDMA__StartTransfer(edma_handle_t *handle)
```

eDMA starts transfer.

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

**Parameters**

- handle – eDMA handle pointer.

void EDMA\_StopTransfer(*edma\_handle\_t* \*handle)

eDMA stops transfer.

This function disables the channel request to pause the transfer. Users can call EDMA\_StartTransfer() again to resume the transfer.

**Parameters**

- handle – eDMA handle pointer.

void EDMA\_AbortTransfer(*edma\_handle\_t* \*handle)

eDMA aborts transfer.

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

**Parameters**

- handle – DMA handle pointer.

static inline uint32\_t EDMA\_GetUnusedTCDNumber(*edma\_handle\_t* \*handle)

Get unused TCD slot number.

This function gets current tcd index which is run. If the TCD pool pointer is NULL, it will return 0.

**Parameters**

- handle – DMA handle pointer.

**Returns**

The unused tcd slot number.

static inline uint32\_t EDMA\_GetNextTCDAddress(*edma\_handle\_t* \*handle)

Get the next tcd address.

This function gets the next tcd address. If this is last TCD, return 0.

**Parameters**

- handle – DMA handle pointer.

**Returns**

The next TCD address.

void EDMA\_HandleIRQ(*edma\_handle\_t* \*handle)

eDMA IRQ handler for the current major loop transfer completion.

This function clears the channel major interrupt flag and calls the callback function if it is not NULL.

Note: For the case using TCD queue, when the major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Assertion of an optional interrupt request also occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor (if scatter/gather is enabled).

For instance, when the time interrupt of TCD[0] happens, the TCD[1] has already been loaded into the eDMA engine. As sga and sga\_index are calculated based on the DLAST\_SGA bitfield lies in the TCD\_CSR register, the sga\_index in this case should be 2 (DLAST\_SGA of TCD[1] stores the address of TCD[2]). Thus, the “tcdUsed” updated should be (tcdUsed - 2U) which indicates the number of TCDs can be loaded in the memory pool (because TCD[0] and TCD[1] have been loaded into the eDMA engine at this point already.).

For the last two continuous ISRs in a scatter/gather process, they both load the last TCD (The last ISR does not load a new TCD) from the memory pool to the eDMA engine when major loop completes. Therefore, ensure that the header and tcdUsed updated are identical for them. tcdUsed are both 0 in this case as no TCD to be loaded.

See the “eDMA basic data flow” in the eDMA Functional description section of the Reference Manual for further details.

### Parameters

- handle – eDMA handle pointer.

FSL\_EDMA\_DRIVER\_VERSION

eDMA driver version

Version 2.4.5.

enum \_edma\_transfer\_size

eDMA transfer configuration

*Values:*

enumerator kEDMA\_TransferSize1Bytes

Source/Destination data transfer size is 1 byte every time

enumerator kEDMA\_TransferSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kEDMA\_TransferSize4Bytes

Source/Destination data transfer size is 4 bytes every time

enumerator kEDMA\_TransferSize8Bytes

Source/Destination data transfer size is 8 bytes every time

enumerator kEDMA\_TransferSize16Bytes

Source/Destination data transfer size is 16 bytes every time

enumerator kEDMA\_TransferSize32Bytes

Source/Destination data transfer size is 32 bytes every time

enum \_edma\_modulo

eDMA modulo configuration

*Values:*

enumerator kEDMA\_ModuloDisable

Disable modulo

enumerator kEDMA\_Modulo2bytes

Circular buffer size is 2 bytes.

enumerator kEDMA\_Modulo4bytes

Circular buffer size is 4 bytes.

enumerator kEDMA\_Modulo8bytes

Circular buffer size is 8 bytes.

enumerator kEDMA\_Modulo16bytes

Circular buffer size is 16 bytes.

enumerator kEDMA\_Modulo32bytes

Circular buffer size is 32 bytes.

enumerator kEDMA\_Modulo64bytes

Circular buffer size is 64 bytes.

enumerator kEDMA\_Modulo128bytes  
Circular buffer size is 128 bytes.

enumerator kEDMA\_Modulo256bytes  
Circular buffer size is 256 bytes.

enumerator kEDMA\_Modulo512bytes  
Circular buffer size is 512 bytes.

enumerator kEDMA\_Modulo1Kbytes  
Circular buffer size is 1 K bytes.

enumerator kEDMA\_Modulo2Kbytes  
Circular buffer size is 2 K bytes.

enumerator kEDMA\_Modulo4Kbytes  
Circular buffer size is 4 K bytes.

enumerator kEDMA\_Modulo8Kbytes  
Circular buffer size is 8 K bytes.

enumerator kEDMA\_Modulo16Kbytes  
Circular buffer size is 16 K bytes.

enumerator kEDMA\_Modulo32Kbytes  
Circular buffer size is 32 K bytes.

enumerator kEDMA\_Modulo64Kbytes  
Circular buffer size is 64 K bytes.

enumerator kEDMA\_Modulo128Kbytes  
Circular buffer size is 128 K bytes.

enumerator kEDMA\_Modulo256Kbytes  
Circular buffer size is 256 K bytes.

enumerator kEDMA\_Modulo512Kbytes  
Circular buffer size is 512 K bytes.

enumerator kEDMA\_Modulo1Mbytes  
Circular buffer size is 1 M bytes.

enumerator kEDMA\_Modulo2Mbytes  
Circular buffer size is 2 M bytes.

enumerator kEDMA\_Modulo4Mbytes  
Circular buffer size is 4 M bytes.

enumerator kEDMA\_Modulo8Mbytes  
Circular buffer size is 8 M bytes.

enumerator kEDMA\_Modulo16Mbytes  
Circular buffer size is 16 M bytes.

enumerator kEDMA\_Modulo32Mbytes  
Circular buffer size is 32 M bytes.

enumerator kEDMA\_Modulo64Mbytes  
Circular buffer size is 64 M bytes.

enumerator kEDMA\_Modulo128Mbytes  
Circular buffer size is 128 M bytes.



enumerator kEDMA\_Modulo256Mbytes  
Circular buffer size is 256 M bytes.

enumerator kEDMA\_Modulo512Mbytes  
Circular buffer size is 512 M bytes.

enumerator kEDMA\_Modulo1Gbytes  
Circular buffer size is 1 G bytes.

enumerator kEDMA\_Modulo2Gbytes  
Circular buffer size is 2 G bytes.

enum \_edma\_bandwidth  
Bandwidth control.

*Values:*

enumerator kEDMA\_BandwidthStallNone  
No eDMA engine stalls.

enumerator kEDMA\_BandwidthStall4Cycle  
eDMA engine stalls for 4 cycles after each read/write.

enumerator kEDMA\_BandwidthStall8Cycle  
eDMA engine stalls for 8 cycles after each read/write.

enum \_edma\_channel\_link\_type  
Channel link type.

*Values:*

enumerator kEDMA\_LinkNone  
No channel link

enumerator kEDMA\_MinorLink  
Channel link after each minor loop

enumerator kEDMA\_MajorLink  
Channel link while major loop count exhausted

\_edma\_channel\_status\_flags eDMA channel status flags.

*Values:*

enumerator kEDMA\_DoneFlag  
DONE flag, set while transfer finished, CITER value exhausted

enumerator kEDMA\_ErrorFlag  
eDMA error flag, an error occurred in a transfer

enumerator kEDMA\_InterruptFlag  
eDMA interrupt flag, set while an interrupt occurred of this channel

\_edma\_error\_status\_flags eDMA channel error status flags.

*Values:*

enumerator kEDMA\_DestinationBusErrorFlag  
Bus error on destination address

enumerator kEDMA\_SourceBusErrorFlag  
Bus error on the source address

enumerator kEDMA\_ScatterGatherErrorFlag  
Error on the Scatter/Gather address, not 32byte aligned.

enumerator kEDMA\_NbytesErrorFlag  
NBYTES/CITER configuration error

enumerator kEDMA\_DestinationOffsetErrorFlag  
Destination offset not aligned with destination size

enumerator kEDMA\_DestinationAddressErrorFlag  
Destination address not aligned with destination size

enumerator kEDMA\_SourceOffsetErrorFlag  
Source offset not aligned with source size

enumerator kEDMA\_SourceAddressErrorFlag  
Source address not aligned with source size

enumerator kEDMA\_ErrorChannelFlag  
Error channel number of the cancelled channel number

enumerator kEDMA\_ChannelPriorityErrorFlag  
Channel priority is not unique.

enumerator kEDMA\_TransferCanceledFlag  
Transfer cancelled

enumerator kEDMA\_ValidFlag  
No error occurred, this bit is 0. Otherwise, it is 1.

enum \_edma\_interrupt\_enable  
eDMA interrupt source

*Values:*

enumerator kEDMA\_ErrorInterruptEnable  
Enable interrupt while channel error occurs.

enumerator kEDMA\_MajorInterruptEnable  
Enable interrupt while major count exhausted.

enumerator kEDMA\_HalfInterruptEnable  
Enable interrupt while major count to half value.

enum \_edma\_transfer\_type  
eDMA transfer type

*Values:*

enumerator kEDMA\_MemoryToMemory  
Transfer from memory to memory

enumerator kEDMA\_PeripheralToMemory  
Transfer from peripheral to memory

enumerator kEDMA\_MemoryToPeripheral  
Transfer from memory to peripheral

enumerator kEDMA\_PeripheralToPeripheral  
Transfer from Peripheral to peripheral

\_edma\_transfer\_status eDMA transfer status

*Values:*

enumerator `kStatus_EDMA_QueueFull`

TCD queue is full.

enumerator `kStatus_EDMA_Busy`

Channel is busy and can't handle the transfer request.

typedef enum `_edma_transfer_size` `edma_transfer_size_t`

eDMA transfer configuration

typedef enum `_edma_modulo` `edma_modulo_t`

eDMA modulo configuration

typedef enum `_edma_bandwidth` `edma_bandwidth_t`

Bandwidth control.

typedef enum `_edma_channel_link_type` `edma_channel_link_type_t`

Channel link type.

typedef enum `_edma_interrupt_enable` `edma_interrupt_enable_t`

eDMA interrupt source

typedef enum `_edma_transfer_type` `edma_transfer_type_t`

eDMA transfer type

typedef struct `_edma_config` `edma_config_t`

eDMA global configuration structure.

typedef struct `_edma_transfer_config` `edma_transfer_config_t`

eDMA transfer configuration

This structure configures the source/destination transfer attribute.

typedef struct `_edma_channel_Preemption_config` `edma_channel_Preemption_config_t`

eDMA channel priority configuration

typedef struct `_edma_minor_offset_config` `edma_minor_offset_config_t`

eDMA minor offset configuration

typedef struct `_edma_tcd` `edma_tcd_t`

eDMA TCD.

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

typedef void (\*`edma_callback`)(struct `_edma_handle` \*`handle`, void \*`userData`, bool `transferDone`, uint32\_t `tcds`)

Define callback function for eDMA.

This callback function is called in the EDMA interrupt handle. In normal mode, run into callback function means the transfer users need is done. In scatter gather mode, run into callback function means a transfer control block (tcd) is finished. Not all transfer finished, users can get the finished tcd numbers using interface `EDMA_GetUnusedTCDNumber`.

**Param handle**

EDMA handle pointer, users shall not touch the values inside.

**Param userData**

The callback user parameter pointer. Users can use this parameter to involve things users need to change in EDMA callback function.

**Param transferDone**

If the current loaded transfer done. In normal mode it means if all transfer done. In scatter gather mode, this parameter shows is the current transfer

block in EDMA register is done. As the load of core is different, it will be different if the new tcd loaded into EDMA registers while this callback called. If true, it always means new tcd still not loaded into registers, while false means new tcd already loaded into registers.

**Param tcDs**

How many tcDs are done from the last callback. This parameter only used in scatter gather mode. It tells user how many tcDs are finished between the last callback and this.

```
typedef struct _edma_handle edma_handle_t  
    eDMA transfer handle structure
```

```
DMA_DCHPRI_INDEX(channel)  
    Compute the offset unit from DCHPRI3.
```

```
struct _edma_config  
    #include <fsl_edma.h> eDMA global configuration structure.
```

**Public Members**

bool enableContinuousLinkMode

Enable (true) continuous link mode. Upon minor loop completion, the channel activates again if that channel has a minor loop channel link enabled and the link channel is itself.

bool enableHaltOnError

Enable (true) transfer halt on error. Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

bool enableRoundRobinArbitration

Enable (true) round robin channel arbitration method or fixed priority arbitration is used for channel selection

bool enableDebugMode

Enable(true) eDMA debug mode. When in debug mode, the eDMA stalls the start of a new channel. Executing channels are allowed to complete.

```
struct _edma_transfer_config  
    #include <fsl_edma.h> eDMA transfer configuration  
    This structure configures the source/destination transfer attribute.
```

**Public Members**

uint32\_t srcAddr

Source data address.

uint32\_t destAddr

Destination data address.

*edma\_transfer\_size\_t* srcTransferSize

Source data transfer size.

*edma\_transfer\_size\_t* destTransferSize

Destination data transfer size.

int16\_t srcOffset

Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.

int16\_t destOffset

Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.

uint32\_t minorLoopBytes

Bytes to transfer in a minor loop

uint32\_t majorLoopCounts

Major loop iteration count.

struct \_\_edma\_channel\_Preemption\_config

*#include <fsl\_edma.h>* eDMA channel priority configuration

### Public Members

bool enableChannelPreemption

If true: a channel can be suspended by other channel with higher priority

bool enablePreemptAbility

If true: a channel can suspend other channel with low priority

uint8\_t channelPriority

Channel priority

struct \_\_edma\_minor\_offset\_config

*#include <fsl\_edma.h>* eDMA minor offset configuration

### Public Members

bool enableSrcMinorOffset

Enable(true) or Disable(false) source minor loop offset.

bool enableDestMinorOffset

Enable(true) or Disable(false) destination minor loop offset.

uint32\_t minorOffset

Offset for a minor loop mapping.

struct \_\_edma\_tcd

*#include <fsl\_edma.h>* eDMA TCD.

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

### Public Members

\_\_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\_handle  
#include <fsl\_edma.h> eDMA transfer handle structure

### Public Members

*edma\_callback* callback  
Callback function for major count exhausted.

void \*userData  
Callback function parameter.

DMA\_Type \*base  
eDMA peripheral base address.

*edma\_tcd\_t* \*tcdPool  
Pointer to memory stored TCDs.

uint8\_t channel  
eDMA channel number.

volatile int8\_t header  
The first TCD index. Should point to the next TCD to be loaded into the eDMA engine.

volatile int8\_t tail  
The last TCD index. Should point to the next TCD to be stored into the memory pool.

volatile int8\_t tcdUsed  
The number of used TCD slots. Should reflect the number of TCDs can be used/loaded in the memory.

volatile int8\_t tcdSize  
The total number of TCD slots in the queue.

uint8\_t flags  
The status of the current channel.

## 2.8 FGPIO Driver

`void FGPIO_PinInit(FGPIO_Type *base, uint32_t pin, const gpio_pin_config_t *config)`

Initializes a FGPIO pin used by the board.

To initialize the FGPIO driver, define a pin configuration, as either input or output, in the user file. Then, call the `FGPIO_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` – FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
- `pin` – FGPIO port pin number
- `config` – FGPIO pin configuration pointer

`static inline void FGPIO_PinWrite(FGPIO_Type *base, uint32_t pin, uint8_t output)`

Sets the output level of the multiple FGPIO pins to the logic 1 or 0.

### Parameters

- `base` – FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
- `pin` – FGPIO pin number
- `output` – FGPIO pin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

`static inline void FGPIO_PortSet(FGPIO_Type *base, uint32_t mask)`

Sets the output level of the multiple FGPIO pins to the logic 1.

### Parameters

- `base` – FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
- `mask` – FGPIO pin number macro

`static inline void FGPIO_PortClear(FGPIO_Type *base, uint32_t mask)`

Sets the output level of the multiple FGPIO pins to the logic 0.

### Parameters

- `base` – FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
- `mask` – FGPIO pin number macro

```
static inline void FGPIO_PortToggle(FGPIO_Type *base, uint32_t mask)
```

Reverses the current output logic of the multiple FGPIO pins.

**Parameters**

- base – FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
- mask – FGPIO pin number macro

```
static inline uint32_t FGPIO_PinRead(FGPIO_Type *base, uint32_t pin)
```

Reads the current input value of the FGPIO port.

**Parameters**

- base – FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
- pin – FGPIO pin number

**Return values**

FGPIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

```
uint32_t FGPIO_PortGetInterruptFlags(FGPIO_Type *base)
```

Reads the FGPIO port interrupt status flag.

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level-sensitive interrupt that remains asserted, the flag is set again immediately.

**Parameters**

- base – FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)

**Return values**

The – current FGPIO port interrupt status flags, for example, 0x00010001 means the pin 0 and 17 have the interrupt.

```
void FGPIO_PortClearInterruptFlags(FGPIO_Type *base, uint32_t mask)
```

Clears the multiple FGPIO pin interrupt status flag.

**Parameters**

- base – FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
- mask – FGPIO pin number macro

## 2.9 C90TFS Flash Driver

## 2.10 FlexIO: FlexIO Driver

## 2.11 FlexIO Driver



void FLEXIO\_GetDefaultConfig(*flexio\_config\_t* \*userConfig)

Gets the default configuration to configure the FlexIO module. The configuration can be used directly to call the FLEXIO\_Configure().

Example:

```
flexio_config_t config;
FLEXIO_GetDefaultConfig(&config);
```

#### Parameters

- userConfig – pointer to flexio\_config\_t structure

void FLEXIO\_Init(FLEXIO\_Type \*base, const *flexio\_config\_t* \*userConfig)

Configures the FlexIO with a FlexIO configuration. The configuration structure can be filled by the user or be set with default values by FLEXIO\_GetDefaultConfig().

Example

```
flexio_config_t config = {
    .enableFlexio = true,
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false
};
FLEXIO_Configure(base, &config);
```

#### Parameters

- base – FlexIO peripheral base address
- userConfig – pointer to flexio\_config\_t structure

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

Gates the FlexIO clock. Call this API to stop the FlexIO clock.

---

**Note:** After calling this API, call the FLEXIO\_Init to use the FlexIO module.

---

#### Parameters

- base – FlexIO peripheral base address

uint32\_t FLEXIO\_GetInstance(FLEXIO\_Type \*base)

Get instance number for FLEXIO module.

#### Parameters

- base – FLEXIO peripheral base address.

void FLEXIO\_Reset(FLEXIO\_Type \*base)

Resets the FlexIO module.

#### Parameters

- base – FlexIO peripheral base address

static inline void FLEXIO\_Enable(FLEXIO\_Type \*base, bool enable)

Enables the FlexIO module operation.

#### Parameters

- base – FlexIO peripheral base address
- enable – true to enable, false to disable.

```
static inline uint32_t FLEXIO_ReadPinInput(FLEXIO_Type *base)
```

Reads the input data on each of the FlexIO pins.

#### Parameters

- base – FlexIO peripheral base address

#### Returns

FlexIO pin input data

```
static inline uint8_t FLEXIO_GetShifterState(FLEXIO_Type *base)
```

Gets the current state pointer for state mode use.

#### Parameters

- base – FlexIO peripheral base address

#### Returns

current State pointer

```
void FLEXIO_SetShifterConfig(FLEXIO_Type *base, uint8_t index, const flexio_shifter_config_t *shifterConfig)
```

Configures the shifter with the shifter configuration. The configuration structure covers both the SHIFTCTL and SHIFTCFG registers. To configure the shifter to the proper mode, select which timer controls the shifter to shift, whether to generate start bit/stop bit, and the polarity of start bit and stop bit.

#### Example

```
flexio_shifter_config_t config = {  
.timerSelect = 0,  
.timerPolarity = kFLEXIO_ShifterTimerPolarityOnPositive,  
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,  
.pinPolarity = kFLEXIO_PinActiveLow,  
.shifterMode = kFLEXIO_ShifterModeTransmit,  
.inputSource = kFLEXIO_ShifterInputFromPin,  
.shifterStop = kFLEXIO_ShifterStopBitHigh,  
.shifterStart = kFLEXIO_ShifterStartBitLow  
};  
FLEXIO_SetShifterConfig(base, &config);
```

#### Parameters

- base – FlexIO peripheral base address
- index – Shifter index
- shifterConfig – Pointer to flexio\_shifter\_config\_t structure

```
void FLEXIO_SetTimerConfig(FLEXIO_Type *base, uint8_t index, const flexio_timer_config_t *timerConfig)
```

Configures the timer with the timer configuration. The configuration structure covers both the TIMCTL and TIMCFG registers. To configure the timer to the proper mode, select trigger source for timer and the timer pin output and the timing for timer.

#### Example

```
flexio_timer_config_t config = {  
.triggerSelect = FLEXIO_TIMER_TRIGGER_SEL_SHIFTNSTAT(0),  
.triggerPolarity = kFLEXIO_TimerTriggerPolarityActiveLow,  
.triggerSource = kFLEXIO_TimerTriggerSourceInternal,  
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,  
.pinSelect = 0,  
.pinPolarity = kFLEXIO_PinActiveHigh,  
.timerMode = kFLEXIO_TimerModeDual8BitBaudBit,
```

(continues on next page)

(continued from previous page)

```
.timerOutput = kFLEXIO_TimerOutputZeroNotAffectedByReset,
.timerDecrement = kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput,
.timerReset = kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput,
.timerDisable = kFLEXIO_TimerDisableOnTimerCompare,
.timerEnable = kFLEXIO_TimerEnableOnTriggerHigh,
.timerStop = kFLEXIO_TimerStopBitEnableOnTimerDisable,
.timerStart = kFLEXIO_TimerStartBitEnabled
};
FLEXIO_SetTimerConfig(base, &config);
```

**Parameters**

- base – FlexIO peripheral base address
- index – Timer index
- timerConfig – Pointer to the flexio\_timer\_config\_t structure

```
static inline void FLEXIO_SetClockMode(FLEXIO_Type *base, uint8_t index,
                                     flexio_timer_decrement_source_t clocksource)
```

This function set the value of the prescaler on flexio channels.

**Parameters**

- base – Pointer to the FlexIO simulated peripheral type.
- index – Timer index
- clocksource – Set clock value

```
static inline void FLEXIO_EnableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
```

Enables the shifter status interrupt. The interrupt generates when the corresponding SSF is set.

---

**Note:** For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

---

**Parameters**

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « shifter index)

```
static inline void FLEXIO_DisableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
```

Disables the shifter status interrupt. The interrupt won't generate when the corresponding SSF is set.

---

**Note:** For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

---

**Parameters**

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « shifter index)

static inline void FLEXIO\_EnableShifterErrorInterrupts(FLEXIO\_Type \*base, uint32\_t mask)  
Enables the shifter error interrupt. The interrupt generates when the corresponding SEF is set.

---

**Note:** For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

---

#### Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by  $(1 \ll \text{shifter index})$

static inline void FLEXIO\_DisableShifterErrorInterrupts(FLEXIO\_Type \*base, uint32\_t mask)  
Disables the shifter error interrupt. The interrupt won't generate when the corresponding SEF is set.

---

**Note:** For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

---

#### Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by  $(1 \ll \text{shifter index})$

static inline void FLEXIO\_EnableTimerStatusInterrupts(FLEXIO\_Type \*base, uint32\_t mask)  
Enables the timer status interrupt. The interrupt generates when the corresponding SSF is set.

---

**Note:** For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using  $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

---

#### Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by  $(1 \ll \text{timer index})$

static inline void FLEXIO\_DisableTimerStatusInterrupts(FLEXIO\_Type \*base, uint32\_t mask)  
Disables the timer status interrupt. The interrupt won't generate when the corresponding SSF is set.

---

**Note:** For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using  $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

---

#### Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by  $(1 \ll \text{timer index})$

static inline uint32\_t FLEXIO\_GetShifterStatusFlags(FLEXIO\_Type \*base)  
Gets the shifter status flags.

#### Parameters

- base – FlexIO peripheral base address

**Returns**

Shifter status flags

```
static inline void FLEXIO_ClearShifterStatusFlags(FLEXIO_Type *base, uint32_t mask)
```

Clears the shifter status flags.

---

**Note:** For clearing multiple shifter status flags, for example, two shifter status flags, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

---

**Parameters**

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by  $(1 \ll \text{shifter index})$

```
static inline uint32_t FLEXIO_GetShifterErrorFlags(FLEXIO_Type *base)
```

Gets the shifter error flags.

**Parameters**

- base – FlexIO peripheral base address

**Returns**

Shifter error flags

```
static inline void FLEXIO_ClearShifterErrorFlags(FLEXIO_Type *base, uint32_t mask)
```

Clears the shifter error flags.

---

**Note:** For clearing multiple shifter error flags, for example, two shifter error flags, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

---

**Parameters**

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by  $(1 \ll \text{shifter index})$

```
static inline uint32_t FLEXIO_GetTimerStatusFlags(FLEXIO_Type *base)
```

Gets the timer status flags.

**Parameters**

- base – FlexIO peripheral base address

**Returns**

Timer status flags

```
static inline void FLEXIO_ClearTimerStatusFlags(FLEXIO_Type *base, uint32_t mask)
```

Clears the timer status flags.

---

**Note:** For clearing multiple timer status flags, for example, two timer status flags, can calculate the mask by using  $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

---

**Parameters**

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by  $(1 \ll \text{timer index})$

static inline void FLEXIO\_EnableShifterStatusDMA(FLEXIO\_Type \*base, uint32\_t mask, bool enable)

Enables/disables the shifter status DMA. The DMA request generates when the corresponding SSF is set.

---

**Note:** For multiple shifter status DMA enables, for example, calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

---

#### Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by  $(1 \ll \text{shifter index})$
- enable – True to enable, false to disable.

uint32\_t FLEXIO\_GetShifterBufferAddress(FLEXIO\_Type \*base, flexio\_shifter\_buffer\_type\_t type, uint8\_t index)

Gets the shifter buffer address for the DMA transfer usage.

#### Parameters

- base – FlexIO peripheral base address
- type – Shifter type of flexio\_shifter\_buffer\_type\_t
- index – Shifter index

#### Returns

Corresponding shifter buffer index

status\_t FLEXIO\_RegisterHandleIRQ(void \*base, void \*handle, flexio\_isr\_t isr)

Registers the handle and the interrupt handler for the FlexIO-simulated peripheral.

#### Parameters

- base – Pointer to the FlexIO simulated peripheral type.
- handle – Pointer to the handler for FlexIO simulated peripheral.
- isr – FlexIO simulated peripheral interrupt handler.

#### Return values

- kStatus\_Success – Successfully create the handle.
- kStatus\_OutOfRange – The FlexIO type/handle/ISR table out of range.

status\_t FLEXIO\_UnregisterHandleIRQ(void \*base)

Unregisters the handle and the interrupt handler for the FlexIO-simulated peripheral.

#### Parameters

- base – Pointer to the FlexIO simulated peripheral type.

#### Return values

- kStatus\_Success – Successfully create the handle.
- kStatus\_OutOfRange – The FlexIO type/handle/ISR table out of range.

FSL\_FLEXIO\_DRIVER\_VERSION

FlexIO driver version.

enum \_flexio\_timer\_trigger\_polarity

Define time of timer trigger polarity.

*Values:*

enumerator kFLEXIO\_\_TimerTriggerPolarityActiveHigh  
Active high.

enumerator kFLEXIO\_\_TimerTriggerPolarityActiveLow  
Active low.

enum \_flexio\_timer\_trigger\_source

Define type of timer trigger source.

*Values:*

enumerator kFLEXIO\_\_TimerTriggerSourceExternal  
External trigger selected.

enumerator kFLEXIO\_\_TimerTriggerSourceInternal  
Internal trigger selected.

enum \_flexio\_pin\_config

Define type of timer/shifter pin configuration.

*Values:*

enumerator kFLEXIO\_\_PinConfigOutputDisabled  
Pin output disabled.

enumerator kFLEXIO\_\_PinConfigOpenDrainOrBidirection  
Pin open drain or bidirectional output enable.

enumerator kFLEXIO\_\_PinConfigBidirectionOutputData  
Pin bidirectional output data.

enumerator kFLEXIO\_\_PinConfigOutput  
Pin output.

enum \_flexio\_pin\_polarity

Definition of pin polarity.

*Values:*

enumerator kFLEXIO\_\_PinActiveHigh  
Active high.

enumerator kFLEXIO\_\_PinActiveLow  
Active low.

enum \_flexio\_timer\_mode

Define type of timer work mode.

*Values:*

enumerator kFLEXIO\_\_TimerModeDisabled  
Timer Disabled.

enumerator kFLEXIO\_\_TimerModeDual8BitBaudBit  
Dual 8-bit counters baud/bit mode.

enumerator kFLEXIO\_\_TimerModeDual8BitPWM  
Dual 8-bit counters PWM mode.

enumerator kFLEXIO\_\_TimerModeSingle16Bit

Single 16-bit counter mode.

enumerator kFLEXIO\_\_TimerModeDual8BitPWMLow

Dual 8-bit counters PWM Low mode.

enum \_flexio\_timer\_output

Define type of timer initial output or timer reset condition.

*Values:*

enumerator kFLEXIO\_\_TimerOutputOneNotAffectedByReset

Logic one when enabled and is not affected by timer reset.

enumerator kFLEXIO\_\_TimerOutputZeroNotAffectedByReset

Logic zero when enabled and is not affected by timer reset.

enumerator kFLEXIO\_\_TimerOutputOneAffectedByReset

Logic one when enabled and on timer reset.

enumerator kFLEXIO\_\_TimerOutputZeroAffectedByReset

Logic zero when enabled and on timer reset.

enum \_flexio\_timer\_decrement\_source

Define type of timer decrement.

*Values:*

enumerator kFLEXIO\_\_TimerDecSrcOnFlexIOClockShiftTimerOutput

Decrement counter on FlexIO clock, Shift clock equals Timer output.

enumerator kFLEXIO\_\_TimerDecSrcOnTriggerInputShiftTimerOutput

Decrement counter on Trigger input (both edges), Shift clock equals Timer output.

enumerator kFLEXIO\_\_TimerDecSrcOnPinInputShiftPinInput

Decrement counter on Pin input (both edges), Shift clock equals Pin input.

enumerator kFLEXIO\_\_TimerDecSrcOnTriggerInputShiftTriggerInput

Decrement counter on Trigger input (both edges), Shift clock equals Trigger input.

enum \_flexio\_timer\_reset\_condition

Define type of timer reset condition.

*Values:*

enumerator kFLEXIO\_\_TimerResetNever

Timer never reset.

enumerator kFLEXIO\_\_TimerResetOnTimerPinEqualToTimerOutput

Timer reset on Timer Pin equal to Timer Output.

enumerator kFLEXIO\_\_TimerResetOnTimerTriggerEqualToTimerOutput

Timer reset on Timer Trigger equal to Timer Output.

enumerator kFLEXIO\_\_TimerResetOnTimerPinRisingEdge

Timer reset on Timer Pin rising edge.

enumerator kFLEXIO\_\_TimerResetOnTimerTriggerRisingEdge

Timer reset on Trigger rising edge.

enumerator kFLEXIO\_\_TimerResetOnTimerTriggerBothEdge

Timer reset on Trigger rising or falling edge.



enum \_flexio\_timer\_disable\_condition

Define type of timer disable condition.

*Values:*

enumerator kFLEXIO\_\_TimerDisableNever

Timer never disabled.

enumerator kFLEXIO\_\_TimerDisableOnPreTimerDisable

Timer disabled on Timer N-1 disable.

enumerator kFLEXIO\_\_TimerDisableOnTimerCompare

Timer disabled on Timer compare.

enumerator kFLEXIO\_\_TimerDisableOnTimerCompareTriggerLow

Timer disabled on Timer compare and Trigger Low.

enumerator kFLEXIO\_\_TimerDisableOnPinBothEdge

Timer disabled on Pin rising or falling edge.

enumerator kFLEXIO\_\_TimerDisableOnPinBothEdgeTriggerHigh

Timer disabled on Pin rising or falling edge provided Trigger is high.

enumerator kFLEXIO\_\_TimerDisableOnTriggerFallingEdge

Timer disabled on Trigger falling edge.

enum \_flexio\_timer\_enable\_condition

Define type of timer enable condition.

*Values:*

enumerator kFLEXIO\_\_TimerEnabledAlways

Timer always enabled.

enumerator kFLEXIO\_\_TimerEnableOnPrevTimerEnable

Timer enabled on Timer N-1 enable.

enumerator kFLEXIO\_\_TimerEnableOnTriggerHigh

Timer enabled on Trigger high.

enumerator kFLEXIO\_\_TimerEnableOnTriggerHighPinHigh

Timer enabled on Trigger high and Pin high.

enumerator kFLEXIO\_\_TimerEnableOnPinRisingEdge

Timer enabled on Pin rising edge.

enumerator kFLEXIO\_\_TimerEnableOnPinRisingEdgeTriggerHigh

Timer enabled on Pin rising edge and Trigger high.

enumerator kFLEXIO\_\_TimerEnableOnTriggerRisingEdge

Timer enabled on Trigger rising edge.

enumerator kFLEXIO\_\_TimerEnableOnTriggerBothEdge

Timer enabled on Trigger rising or falling edge.

enum \_flexio\_timer\_stop\_bit\_condition

Define type of timer stop bit generate condition.

*Values:*

enumerator kFLEXIO\_\_TimerStopBitDisabled

Stop bit disabled.

enumerator kFLEXIO\_TimerStopBitEnableOnTimerCompare

Stop bit is enabled on timer compare.

enumerator kFLEXIO\_TimerStopBitEnableOnTimerDisable

Stop bit is enabled on timer disable.

enumerator kFLEXIO\_TimerStopBitEnableOnTimerCompareDisable

Stop bit is enabled on timer compare and timer disable.

enum \_flexio\_timer\_start\_bit\_condition

Define type of timer start bit generate condition.

*Values:*

enumerator kFLEXIO\_TimerStartBitDisabled

Start bit disabled.

enumerator kFLEXIO\_TimerStartBitEnabled

Start bit enabled.

enum \_flexio\_timer\_output\_state

FlexIO as PWM channel output state.

*Values:*

enumerator kFLEXIO\_PwmLow

The output state of PWM channel is low

enumerator kFLEXIO\_PwmHigh

The output state of PWM channel is high

enum \_flexio\_shifter\_timer\_polarity

Define type of timer polarity for shifter control.

*Values:*

enumerator kFLEXIO\_ShifterTimerPolarityOnPositive

Shift on positive edge of shift clock.

enumerator kFLEXIO\_ShifterTimerPolarityOnNegative

Shift on negative edge of shift clock.

enum \_flexio\_shifter\_mode

Define type of shifter working mode.

*Values:*

enumerator kFLEXIO\_ShifterDisabled

Shifter is disabled.

enumerator kFLEXIO\_ShifterModeReceive

Receive mode.

enumerator kFLEXIO\_ShifterModeTransmit

Transmit mode.

enumerator kFLEXIO\_ShifterModeMatchStore

Match store mode.

enumerator kFLEXIO\_ShifterModeMatchContinuous

Match continuous mode.

enumerator kFLEXIO\_ShifterModeState

SHIFTBUF contents are used for storing programmable state attributes.

enumerator kFLEXIO\_\_ShifterModeLogic

SHIFTBUF contents are used for implementing programmable logic look up table.

enum \_\_flexio\_\_shifter\_\_input\_\_source

Define type of shifter input source.

*Values:*

enumerator kFLEXIO\_\_ShifterInputFromPin

Shifter input from pin.

enumerator kFLEXIO\_\_ShifterInputFromNextShifterOutput

Shifter input from Shifter N+1.

enum \_\_flexio\_\_shifter\_\_stop\_\_bit

Define of STOP bit configuration.

*Values:*

enumerator kFLEXIO\_\_ShifterStopBitDisable

Disable shifter stop bit.

enumerator kFLEXIO\_\_ShifterStopBitLow

Set shifter stop bit to logic low level.

enumerator kFLEXIO\_\_ShifterStopBitHigh

Set shifter stop bit to logic high level.

enum \_\_flexio\_\_shifter\_\_start\_\_bit

Define type of START bit configuration.

*Values:*

enumerator kFLEXIO\_\_ShifterStartBitDisabledLoadDataOnEnable

Disable shifter start bit, transmitter loads data on enable.

enumerator kFLEXIO\_\_ShifterStartBitDisabledLoadDataOnShift

Disable shifter start bit, transmitter loads data on first shift.

enumerator kFLEXIO\_\_ShifterStartBitLow

Set shifter start bit to logic low level.

enumerator kFLEXIO\_\_ShifterStartBitHigh

Set shifter start bit to logic high level.

enum \_\_flexio\_\_shifter\_\_buffer\_\_type

Define FlexIO shifter buffer type.

*Values:*

enumerator kFLEXIO\_\_ShifterBuffer

Shifter Buffer N Register.

enumerator kFLEXIO\_\_ShifterBufferBitSwapped

Shifter Buffer N Bit Byte Swapped Register.

enumerator kFLEXIO\_\_ShifterBufferByteSwapped

Shifter Buffer N Byte Swapped Register.

enumerator kFLEXIO\_\_ShifterBufferBitByteSwapped

Shifter Buffer N Bit Swapped Register.

enumerator kFLEXIO\_\_ShifterBufferNibbleByteSwapped

Shifter Buffer N Nibble Byte Swapped Register.

enumerator `kFLEXIO_ShifterBufferHalfWordSwapped`  
Shifter Buffer N Half Word Swapped Register.

enumerator `kFLEXIO_ShifterBufferNibbleSwapped`  
Shifter Buffer N Nibble Swapped Register.

typedef enum `_flexio_timer_trigger_polarity` `flexio_timer_trigger_polarity_t`  
Define time of timer trigger polarity.

typedef enum `_flexio_timer_trigger_source` `flexio_timer_trigger_source_t`  
Define type of timer trigger source.

typedef enum `_flexio_pin_config` `flexio_pin_config_t`  
Define type of timer/shifter pin configuration.

typedef enum `_flexio_pin_polarity` `flexio_pin_polarity_t`  
Definition of pin polarity.

typedef enum `_flexio_timer_mode` `flexio_timer_mode_t`  
Define type of timer work mode.

typedef enum `_flexio_timer_output` `flexio_timer_output_t`  
Define type of timer initial output or timer reset condition.

typedef enum `_flexio_timer_decrement_source` `flexio_timer_decrement_source_t`  
Define type of timer decrement.

typedef enum `_flexio_timer_reset_condition` `flexio_timer_reset_condition_t`  
Define type of timer reset condition.

typedef enum `_flexio_timer_disable_condition` `flexio_timer_disable_condition_t`  
Define type of timer disable condition.

typedef enum `_flexio_timer_enable_condition` `flexio_timer_enable_condition_t`  
Define type of timer enable condition.

typedef enum `_flexio_timer_stop_bit_condition` `flexio_timer_stop_bit_condition_t`  
Define type of timer stop bit generate condition.

typedef enum `_flexio_timer_start_bit_condition` `flexio_timer_start_bit_condition_t`  
Define type of timer start bit generate condition.

typedef enum `_flexio_timer_output_state` `flexio_timer_output_state_t`  
FlexIO as PWM channel output state.

typedef enum `_flexio_shifter_timer_polarity` `flexio_shifter_timer_polarity_t`  
Define type of timer polarity for shifter control.

typedef enum `_flexio_shifter_mode` `flexio_shifter_mode_t`  
Define type of shifter working mode.

typedef enum `_flexio_shifter_input_source` `flexio_shifter_input_source_t`  
Define type of shifter input source.

typedef enum `_flexio_shifter_stop_bit` `flexio_shifter_stop_bit_t`  
Define of STOP bit configuration.

typedef enum `_flexio_shifter_start_bit` `flexio_shifter_start_bit_t`  
Define type of START bit configuration.

typedef enum `_flexio_shifter_buffer_type` `flexio_shifter_buffer_type_t`  
Define FlexIO shifter buffer type.

```
typedef struct _flexio_config flexio_config_t
    Define FlexIO user configuration structure.
typedef struct _flexio_timer_config flexio_timer_config_t
    Define FlexIO timer configuration structure.
typedef struct _flexio_shifter_config flexio_shifter_config_t
    Define FlexIO shifter configuration structure.
typedef void (*flexio_isr_t)(void *base, void *handle)
    typedef for FlexIO simulated driver interrupt handler.
FLEXIO_Type *const s_flexioBases[]
    Pointers to flexio bases for each instance.
const clock_ip_name_t s_flexioClocks[]
    Pointers to flexio clocks for each instance.
FLEXIO_TIMER_TRIGGER_SEL_PININPUT(x)
    Calculate FlexIO timer trigger.
FLEXIO_TIMER_TRIGGER_SEL_SHIFThSTAT(x)
FLEXIO_TIMER_TRIGGER_SEL_TIMn(x)
struct _flexio_config
    #include <fsl_flexio.h> Define FlexIO user configuration structure.
```

### Public Members

```
bool enableFlexio
    Enable/disable FlexIO module
bool enableInDoze
    Enable/disable FlexIO operation in doze mode
bool enableInDebug
    Enable/disable FlexIO operation in debug mode
bool enableFastAccess
    Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to
    be at least twice the frequency of the bus clock.
struct _flexio_timer_config
    #include <fsl_flexio.h> Define FlexIO timer configuration structure.
```

### Public Members

```
uint32_t triggerSelect
    The internal trigger selection number using MACROs.
flexio_timer_trigger_polarity_t triggerPolarity
    Trigger Polarity.
flexio_timer_trigger_source_t triggerSource
    Trigger Source, internal (see 'trgsel') or external.
flexio_pin_config_t pinConfig
    Timer Pin Configuration.
```

uint32\_t pinSelect

Timer Pin number Select.

*flexio\_pin\_polarity\_t* pinPolarity

Timer Pin Polarity.

*flexio\_timer\_mode\_t* timerMode

Timer work Mode.

*flexio\_timer\_output\_t* timerOutput

Configures the initial state of the Timer Output and whether it is affected by the Timer reset.

*flexio\_timer\_decrement\_source\_t* timerDecrement

Configures the source of the Timer decrement and the source of the Shift clock.

*flexio\_timer\_reset\_condition\_t* timerReset

Configures the condition that causes the timer counter (and optionally the timer output) to be reset.

*flexio\_timer\_disable\_condition\_t* timerDisable

Configures the condition that causes the Timer to be disabled and stop decrementing.

*flexio\_timer\_enable\_condition\_t* timerEnable

Configures the condition that causes the Timer to be enabled and start decrementing.

*flexio\_timer\_stop\_bit\_condition\_t* timerStop

Timer STOP Bit generation.

*flexio\_timer\_start\_bit\_condition\_t* timerStart

Timer STRAT Bit generation.

uint32\_t timerCompare

Value for Timer Compare N Register.

struct *\_flexio\_shifter\_config*

*#include <fsl\_flexio.h>* Define FlexIO shifter configuration structure.

## Public Members

uint32\_t timerSelect

Selects which Timer is used for controlling the logic/shift register and generating the Shift clock.

*flexio\_shifter\_timer\_polarity\_t* timerPolarity

Timer Polarity.

*flexio\_pin\_config\_t* pinConfig

Shifter Pin Configuration.

uint32\_t pinSelect

Shifter Pin number Select.

*flexio\_pin\_polarity\_t* pinPolarity

Shifter Pin Polarity.

*flexio\_shifter\_mode\_t* shifterMode

Configures the mode of the Shifter.

uint32\_t parallelWidth

Configures the parallel width when using parallel mode.

*flexio\_shifter\_input\_source\_t* inputSource  
Selects the input source for the shifter.

*flexio\_shifter\_stop\_bit\_t* shifterStop  
Shifter STOP bit.

*flexio\_shifter\_start\_bit\_t* shifterStart  
Shifter START bit.

## 2.12 FlexIO eDMA I2S Driver

```
void FLEXIO_I2S_TransferTxCreateHandleEDMA(FLEXIO_I2S_Type *base,
                                           flexio_i2s_edma_handle_t *handle,
                                           flexio_i2s_edma_callback_t callback, void
                                           *userData, edma_handle_t *dmaHandle)
```

Initializes the FlexIO I2S eDMA handle.

This function initializes the FlexIO I2S master DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

### Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer.
- callback – FlexIO I2S eDMA callback function called while finished a block.
- userData – User parameter for callback.
- dmaHandle – eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

```
void FLEXIO_I2S_TransferRxCreateHandleEDMA(FLEXIO_I2S_Type *base,
                                           flexio_i2s_edma_handle_t *handle,
                                           flexio_i2s_edma_callback_t callback, void
                                           *userData, edma_handle_t *dmaHandle)
```

Initializes the FlexIO I2S Rx eDMA handle.

This function initializes the FlexIO I2S slave DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

### Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer.
- callback – FlexIO I2S eDMA callback function called while finished a block.
- userData – User parameter for callback.
- dmaHandle – eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

```
void FLEXIO_I2S_TransferSetFormatEDMA(FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t
                                       *handle, flexio_i2s_format_t *format, uint32_t
                                       srcClock_Hz)
```

Configures the FlexIO I2S Tx audio format.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to format.

#### Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer
- format – Pointer to FlexIO I2S audio data format structure.
- srcClock\_Hz – FlexIO I2S clock source frequency in Hz, it should be 0 while in slave mode.

*status\_t* FLEXIO\_I2S\_TransferSendEDMA(*FLEXIO\_I2S\_Type* \*base, *flexio\_i2s\_edma\_handle\_t* \*handle, *flexio\_i2s\_transfer\_t* \*xfer)

Performs a non-blocking FlexIO I2S transfer using DMA.

---

**Note:** This interface returned immediately after transfer initiates. Users should call FLEXIO\_I2S\_GetTransferStatus to poll the transfer status and check whether the FlexIO I2S transfer is finished.

---

#### Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- xfer – Pointer to DMA transfer structure.

#### Return values

- kStatus\_Success – Start a FlexIO I2S eDMA send successfully.
- kStatus\_InvalidArgument – The input arguments is invalid.
- kStatus\_TxBusy – FlexIO I2S is busy sending data.

*status\_t* FLEXIO\_I2S\_TransferReceiveEDMA(*FLEXIO\_I2S\_Type* \*base, *flexio\_i2s\_edma\_handle\_t* \*handle, *flexio\_i2s\_transfer\_t* \*xfer)

Performs a non-blocking FlexIO I2S receive using eDMA.

---

**Note:** This interface returned immediately after transfer initiates. Users should call FLEXIO\_I2S\_GetReceiveRemainingBytes to poll the transfer status and check whether the FlexIO I2S transfer is finished.

---

#### Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- xfer – Pointer to DMA transfer structure.

#### Return values

- kStatus\_Success – Start a FlexIO I2S eDMA receive successfully.
- kStatus\_InvalidArgument – The input arguments is invalid.
- kStatus\_RxBusy – FlexIO I2S is busy receiving data.



```
void FLEXIO_I2S_TransferAbortSendEDMA(FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t
                                     *handle)
```

Aborts a FlexIO I2S transfer using eDMA.

#### Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.

```
void FLEXIO_I2S_TransferAbortReceiveEDMA(FLEXIO_I2S_Type *base,
                                         flexio_i2s_edma_handle_t *handle)
```

Aborts a FlexIO I2S receive using eDMA.

#### Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.

```
status_t FLEXIO_I2S_TransferGetSendCountEDMA(FLEXIO_I2S_Type *base,
                                              flexio_i2s_edma_handle_t *handle, size_t
                                              *count)
```

Gets the remaining bytes to be sent.

#### Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- count – Bytes sent.

#### Return values

- kStatus\_Success – Succeed get the transfer count.
- kStatus\_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

```
status_t FLEXIO_I2S_TransferGetReceiveCountEDMA(FLEXIO_I2S_Type *base,
                                                flexio_i2s_edma_handle_t *handle, size_t
                                                *count)
```

Get the remaining bytes to be received.

#### Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- count – Bytes received.

#### Return values

- kStatus\_Success – Succeed get the transfer count.
- kStatus\_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

```
FSL_FLEXIO_I2S_EDMA_DRIVER_VERSION
```

FlexIO I2S EDMA driver version 2.1.9.

```
typedef struct flexio_i2s_edma_handle flexio_i2s_edma_handle_t
```

```
typedef void (*flexio_i2s_edma_callback_t)(FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t
*handle, status_t status, void *userData)
```

FlexIO I2S eDMA transfer callback function for finish and error.

```
struct _flexio_i2s_edma_handle
```

*#include <fsl\_flexio\_i2s\_edma.h>* FlexIO I2S DMA transfer handle, users should not touch the content of the handle.

### Public Members

*edma\_handle\_t* \*dmaHandle

DMA handler for FlexIO I2S send

uint8\_t bytesPerFrame

Bytes in a frame

uint8\_t nbytes

eDMA minor byte transfer count initially configured.

uint32\_t state

Internal state for FlexIO I2S eDMA transfer

*flexio\_i2s\_edma\_callback\_t* callback

Callback for users while transfer finish or error occurred

void \*userData

User callback parameter

*edma\_tcd\_t* tcd[(4U) + 1U]

TCD pool for eDMA transfer.

*flexio\_i2s\_transfer\_t* queue[(4U)]

Transfer queue storing queued transfer.

size\_t transferSize[(4U)]

Data bytes need to transfer

volatile uint8\_t queueUser

Index for user to queue transfer.

volatile uint8\_t queueDriver

Index for driver to get the transfer data and size

## 2.13 FlexIO eDMA SPI Driver

```
status_t FLEXIO_SPI_MasterTransferCreateHandleEDMA(FLEXIO_SPI_Type *base,  
                                                    flexio_spi_master_edma_handle_t  
                                                    *handle,  
                                                    flexio_spi_master_edma_transfer_callback_t  
                                                    callback, void *userData,  
                                                    edma_handle_t *txHandle,  
                                                    edma_handle_t *rxHandle)
```

Initializes the FlexIO SPI master eDMA handle.

This function initializes the FlexIO SPI master eDMA handle which can be used for other FlexIO SPI master transactional APIs. For a specified FlexIO SPI instance, call this API once to get the initialized handle.

### Parameters

- base – Pointer to FLEXIO\_SPI\_Type structure.
- handle – Pointer to flexio\_spi\_master\_edma\_handle\_t structure to store the transfer state.

- callback – SPI callback, NULL means no callback.
- userData – callback function parameter.
- txHandle – User requested eDMA handle for FlexIO SPI RX eDMA transfer.
- rxHandle – User requested eDMA handle for FlexIO SPI TX eDMA transfer.

**Return values**

- kStatus\_\_Success – Successfully create the handle.
- kStatus\_\_OutOfRange – The FlexIO SPI eDMA type/handle table out of range.

*status\_t* FLEXIO\_SPI\_MasterTransferEDMA(*FLEXIO\_SPI\_Type* \*base,  
  *flexio\_spi\_master\_edma\_handle\_t* \*handle,  
  *flexio\_spi\_transfer\_t* \*xfer)

Performs a non-blocking FlexIO SPI transfer using eDMA.

---

**Note:** This interface returns immediately after transfer initiates. Call FLEXIO\_SPI\_MasterGetTransferCountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

---

**Parameters**

- base – Pointer to FLEXIO\_SPI\_Type structure.
- handle – Pointer to flexio\_spi\_master\_edma\_handle\_t structure to store the transfer state.
- xfer – Pointer to FlexIO SPI transfer structure.

**Return values**

- kStatus\_\_Success – Successfully start a transfer.
- kStatus\_\_InvalidArgument – Input argument is invalid.
- kStatus\_FLEXIO\_SPI\_Busy – FlexIO SPI is not idle, is running another transfer.

*void* FLEXIO\_SPI\_MasterTransferAbortEDMA(*FLEXIO\_SPI\_Type* \*base,  
  *flexio\_spi\_master\_edma\_handle\_t* \*handle)

Aborts a FlexIO SPI transfer using eDMA.

**Parameters**

- base – Pointer to FLEXIO\_SPI\_Type structure.
- handle – FlexIO SPI eDMA handle pointer.

*status\_t* FLEXIO\_SPI\_MasterTransferGetCountEDMA(*FLEXIO\_SPI\_Type* \*base,  
  *flexio\_spi\_master\_edma\_handle\_t* \*handle,  
  *size\_t* \*count)

Gets the number of bytes transferred so far using FlexIO SPI master eDMA.

**Parameters**

- base – Pointer to FLEXIO\_SPI\_Type structure.
- handle – FlexIO SPI eDMA handle pointer.
- count – Number of bytes transferred so far by the non-blocking transaction.

```
static inline void FLEXIO_SPI_SlaveTransferCreateHandleEDMA(FLEXIO_SPI_Type *base,  
                                                         flexio_spi_slave_edma_handle_t  
                                                         *handle,  
                                                         flexio_spi_slave_edma_transfer_callback_t  
                                                         callback, void *userData,  
                                                         edma_handle_t *txHandle,  
                                                         edma_handle_t *rxHandle)
```

Initializes the FlexIO SPI slave eDMA handle.

This function initializes the FlexIO SPI slave eDMA handle.

#### Parameters

- base – Pointer to *FLEXIO\_SPI\_Type* structure.
- handle – Pointer to *flexio\_spi\_slave\_edma\_handle\_t* structure to store the transfer state.
- callback – SPI callback, NULL means no callback.
- userData – callback function parameter.
- txHandle – User requested eDMA handle for FlexIO SPI TX eDMA transfer.
- rxHandle – User requested eDMA handle for FlexIO SPI RX eDMA transfer.

```
status_t FLEXIO_SPI_SlaveTransferEDMA(FLEXIO_SPI_Type *base,  
                                       flexio_spi_slave_edma_handle_t *handle,  
                                       flexio_spi_transfer_t *xfer)
```

Performs a non-blocking FlexIO SPI transfer using eDMA.

---

**Note:** This interface returns immediately after transfer initiates. Call *FLEXIO\_SPI\_SlaveGetTransferCountEDMA* to poll the transfer status and check whether the FlexIO SPI transfer is finished.

---

#### Parameters

- base – Pointer to *FLEXIO\_SPI\_Type* structure.
- handle – Pointer to *flexio\_spi\_slave\_edma\_handle\_t* structure to store the transfer state.
- xfer – Pointer to FlexIO SPI transfer structure.

#### Return values

- *kStatus\_Success* – Successfully start a transfer.
- *kStatus\_InvalidArgument* – Input argument is invalid.
- *kStatus\_FLEXIO\_SPI\_Busy* – FlexIO SPI is not idle, is running another transfer.

```
static inline void FLEXIO_SPI_SlaveTransferAbortEDMA(FLEXIO_SPI_Type *base,  
                                                    flexio_spi_slave_edma_handle_t  
                                                    *handle)
```

Aborts a FlexIO SPI transfer using eDMA.

#### Parameters

- base – Pointer to *FLEXIO\_SPI\_Type* structure.
- handle – Pointer to *flexio\_spi\_slave\_edma\_handle\_t* structure to store the transfer state.

```
static inline status_t FLEXIO_SPI_SlaveTransferGetCountEDMA(FLEXIO_SPI_Type *base,
                                                         flexio_spi_slave_edma_handle_t
                                                         *handle, size_t *count)
```

Gets the number of bytes transferred so far using FlexIO SPI slave eDMA.

#### Parameters

- base – Pointer to *FLEXIO\_SPI\_Type* structure.
- handle – FlexIO SPI eDMA handle pointer.
- count – Number of bytes transferred so far by the non-blocking transaction.

```
FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION
```

FlexIO SPI EDMA driver version.

```
typedef struct flexio_spi_master_edma_handle flexio_spi_master_edma_handle_t
```

typedef for *flexio\_spi\_master\_edma\_handle\_t* in advance.

```
typedef flexio_spi_master_edma_handle_t flexio_spi_slave_edma_handle_t
```

Slave handle is the same with master handle.

```
typedef void (*flexio_spi_master_edma_transfer_callback_t)(FLEXIO_SPI_Type *base,
                                                         flexio_spi_master_edma_handle_t *handle,
                                                         status_t status, void *userData)
```

FlexIO SPI master callback for finished transmit.

```
typedef void (*flexio_spi_slave_edma_transfer_callback_t)(FLEXIO_SPI_Type *base,
                                                         flexio_spi_slave_edma_handle_t *handle,
                                                         status_t status, void *userData)
```

FlexIO SPI slave callback for finished transmit.

```
struct flexio_spi_master_edma_handle
```

#include <fsl\_flexio\_spi\_edma.h> FlexIO SPI eDMA transfer handle, users should not touch the content of the handle.

#### Public Members

```
size_t transferSize
```

Total bytes to be transferred.

```
uint8_t nbytes
```

eDMA minor byte transfer count initially configured.

```
bool txInProgress
```

Send transfer in progress

```
bool rxInProgress
```

Receive transfer in progress

```
edma_handle_t *txHandle
```

DMA handler for SPI send

```
edma_handle_t *rxHandle
```

DMA handler for SPI receive

```
flexio_spi_master_edma_transfer_callback_t callback
```

Callback for SPI DMA transfer

```
void *userData
```

User Data for SPI DMA callback

## 2.14 FlexIO eDMA UART Driver

```
status_t FLEXIO_UART_TransferCreateHandleEDMA(FLEXIO_UART_Type *base,  
                                              flexio_uart_edma_handle_t *handle,  
                                              flexio_uart_edma_transfer_callback_t  
                                              callback, void *userData, edma_handle_t  
                                              *txEdmaHandle, edma_handle_t  
                                              *rxEdmaHandle)
```

Initializes the UART handle which is used in transactional functions.

### Parameters

- base – Pointer to *FLEXIO\_UART\_Type*.
- handle – Pointer to *flexio\_uart\_edma\_handle\_t* structure.
- callback – The callback function.
- userData – The parameter of the callback function.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.

### Return values

- *kStatus\_Success* – Successfully create the handle.
- *kStatus\_OutOfRange* – The FlexIO SPI eDMA type/handle table out of range.

```
status_t FLEXIO_UART_TransferSendEDMA(FLEXIO_UART_Type *base,  
                                       flexio_uart_edma_handle_t *handle,  
                                       flexio_uart_transfer_t *xfer)
```

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent out, the send callback function is called.

### Parameters

- base – Pointer to *FLEXIO\_UART\_Type*
- handle – UART handle pointer.
- xfer – UART eDMA transfer structure, see *flexio\_uart\_transfer\_t*.

### Return values

- *kStatus\_Success* – if succeed, others failed.
- *kStatus\_FLEXIO\_UART\_TxBusy* – Previous transfer on going.

```
status_t FLEXIO_UART_TransferReceiveEDMA(FLEXIO_UART_Type *base,  
                                          flexio_uart_edma_handle_t *handle,  
                                          flexio_uart_transfer_t *xfer)
```

Receives data using eDMA.

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

### Parameters

- base – Pointer to *FLEXIO\_UART\_Type*
- handle – Pointer to *flexio\_uart\_edma\_handle\_t* structure
- xfer – UART eDMA transfer structure, see *flexio\_uart\_transfer\_t*.

### Return values

- kStatus\_Success – if succeed, others failed.
- kStatus\_UART\_RxBusy – Previous transfer on going.

```
void FLEXIO_UART_TransferAbortSendEDMA(FLEXIO_UART_Type *base,  
                                         flexio_uart_edma_handle_t *handle)
```

Aborts the sent data which using eDMA.

This function aborts sent data which using eDMA.

#### Parameters

- base – Pointer to FLEXIO\_UART\_Type
- handle – Pointer to flexio\_uart\_edma\_handle\_t structure

```
void FLEXIO_UART_TransferAbortReceiveEDMA(FLEXIO_UART_Type *base,  
                                           flexio_uart_edma_handle_t *handle)
```

Aborts the receive data which using eDMA.

This function aborts the receive data which using eDMA.

#### Parameters

- base – Pointer to FLEXIO\_UART\_Type
- handle – Pointer to flexio\_uart\_edma\_handle\_t structure

```
status_t FLEXIO_UART_TransferGetSendCountEDMA(FLEXIO_UART_Type *base,  
                                                flexio_uart_edma_handle_t *handle,  
                                                size_t *count)
```

Gets the number of bytes sent out.

This function gets the number of bytes sent out.

#### Parameters

- base – Pointer to FLEXIO\_UART\_Type
- handle – Pointer to flexio\_uart\_edma\_handle\_t structure
- count – Number of bytes sent so far by the non-blocking transaction.

#### Return values

- kStatus\_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus\_Success – Successfully return the count.

```
status_t FLEXIO_UART_TransferGetReceiveCountEDMA(FLEXIO_UART_Type *base,  
                                                  flexio_uart_edma_handle_t *handle,  
                                                  size_t *count)
```

Gets the number of bytes received.

This function gets the number of bytes received.

#### Parameters

- base – Pointer to FLEXIO\_UART\_Type
- handle – Pointer to flexio\_uart\_edma\_handle\_t structure
- count – Number of bytes received so far by the non-blocking transaction.

#### Return values

- kStatus\_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus\_Success – Successfully return the count.

FSL\_FLEXIO\_UART\_EDMA\_DRIVER\_VERSION

FlexIO UART EDMA driver version.

typedef struct *flexio\_uart\_edma\_handle* flexio\_uart\_edma\_handle\_t

typedef void (\*flexio\_uart\_edma\_transfer\_callback\_t)(FLEXIO\_UART\_Type \*base,  
*flexio\_uart\_edma\_handle\_t* \*handle, *status\_t* status, void \*userData)

UART transfer callback function.

struct *flexio\_uart\_edma\_handle*

*#include <fsl\_flexio\_uart\_edma.h>* UART eDMA handle.

### Public Members

*flexio\_uart\_edma\_transfer\_callback\_t* callback

Callback function.

void \*userData

UART callback function parameter.

size\_t txDataSizeAll

Total bytes to be sent.

size\_t rxDataSizeAll

Total bytes to be received.

*edma\_handle\_t* \*txEdmaHandle

The eDMA TX channel used.

*edma\_handle\_t* \*rxEdmaHandle

The eDMA RX channel used.

uint8\_t nbytes

eDMA minor byte transfer count initially configured.

volatile uint8\_t txState

TX transfer state.

volatile uint8\_t rxState

RX transfer state

## 2.15 FlexIO I2C Master Driver

*status\_t* FLEXIO\_I2C\_CheckForBusyBus(FLEXIO\_I2C\_Type \*base)

Make sure the bus isn't already pulled down.

Check the FLEXIO pin status to see whether either of SDA and SCL pin is pulled down.

### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure..

### Return values

- kStatus\_Success –
- kStatus\_FLEXIO\_I2C\_Busy –



*status\_t* FLEXIO\_I2C\_MasterInit(*FLEXIO\_I2C\_Type* \*base, *flexio\_i2c\_master\_config\_t* \*masterConfig, uint32\_t srcClock\_Hz)

Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO I2C hardware configuration.

#### Example

```
FLEXIO_I2C_Type base = {
    .flexioBase = FLEXIO,
    .SDAPinIndex = 0,
    .SCLPinIndex = 1,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_i2c_master_config_t config = {
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .baudRate_Bps = 100000
};
FLEXIO_I2C_MasterInit(base, &config, srcClock_Hz);
```

#### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure.
- masterConfig – Pointer to flexio\_i2c\_master\_config\_t structure.
- srcClock\_Hz – FlexIO source clock in Hz.

#### Return values

- kStatus\_Success – Initialization successful
- kStatus\_InvalidArgument – The source clock exceed upper range limitation

void FLEXIO\_I2C\_MasterDeinit(*FLEXIO\_I2C\_Type* \*base)

De-initializes the FlexIO I2C master peripheral. Calling this API Resets the FlexIO I2C master shifer and timer config, module can't work unless the FLEXIO\_I2C\_MasterInit is called.

#### Parameters

- base – pointer to FLEXIO\_I2C\_Type structure.

void FLEXIO\_I2C\_MasterGetDefaultConfig(*flexio\_i2c\_master\_config\_t* \*masterConfig)

Gets the default configuration to configure the FlexIO module. The configuration can be used directly for calling the FLEXIO\_I2C\_MasterInit().

#### Example:

```
flexio_i2c_master_config_t config;
FLEXIO_I2C_MasterGetDefaultConfig(&config);
```

#### Parameters

- masterConfig – Pointer to flexio\_i2c\_master\_config\_t structure.

static inline void FLEXIO\_I2C\_MasterEnable(*FLEXIO\_I2C\_Type* \*base, bool enable)

Enables/disables the FlexIO module operation.

#### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure.
- enable – Pass true to enable module, false does not have any effect.

uint32\_t FLEXIO\_I2C\_MasterGetStatusFlags(*FLEXIO\_I2C\_Type* \*base)

Gets the FlexIO I2C master status flags.

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure

**Returns**

Status flag, use status flag to AND `_flexio_i2c_master_status_flags` can get the related status.

void FLEXIO\_I2C\_MasterClearStatusFlags(*FLEXIO\_I2C\_Type* \*base, uint32\_t mask)

Clears the FlexIO I2C master status flags.

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure.
- mask – Status flag. The parameter can be any combination of the following values:
  - kFLEXIO\_I2C\_RxFullFlag
  - kFLEXIO\_I2C\_ReceiveNakFlag

void FLEXIO\_I2C\_MasterEnableInterrupts(*FLEXIO\_I2C\_Type* \*base, uint32\_t mask)

Enables the FlexIO i2c master interrupt requests.

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure.
- mask – Interrupt source. Currently only one interrupt request source:
  - kFLEXIO\_I2C\_TransferCompleteInterruptEnable

void FLEXIO\_I2C\_MasterDisableInterrupts(*FLEXIO\_I2C\_Type* \*base, uint32\_t mask)

Disables the FlexIO I2C master interrupt requests.

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure.
- mask – Interrupt source.

void FLEXIO\_I2C\_MasterSetBaudRate(*FLEXIO\_I2C\_Type* \*base, uint32\_t baudRate\_Bps,  
uint32\_t srcClock\_Hz)

Sets the FlexIO I2C master transfer baudrate.

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure
- baudRate\_Bps – the baud rate value in HZ
- srcClock\_Hz – source clock in HZ

void FLEXIO\_I2C\_MasterStart(*FLEXIO\_I2C\_Type* \*base, uint8\_t address, *flexio\_i2c\_direction\_t* direction)

Sends START + 7-bit address to the bus.

---

**Note:** This API should be called when the transfer configuration is ready to send a START signal and 7-bit address to the bus. This is a non-blocking API, which returns directly after the address is put into the data register but the address transfer is not finished on the bus. Ensure that the kFLEXIO\_I2C\_RxFullFlag status is asserted before calling this API.

---

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure.
- address – 7-bit address.
- direction – transfer direction. This parameter is one of the values in flexio\_i2c\_direction\_t:
  - kFLEXIO\_I2C\_Write: Transmit
  - kFLEXIO\_I2C\_Read: Receive

void FLEXIO\_I2C\_MasterStop(*FLEXIO\_I2C\_Type* \*base)

Sends the stop signal on the bus.

#### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure.

void FLEXIO\_I2C\_MasterRepeatedStart(*FLEXIO\_I2C\_Type* \*base)

Sends the repeated start signal on the bus.

#### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure.

void FLEXIO\_I2C\_MasterAbortStop(*FLEXIO\_I2C\_Type* \*base)

Sends the stop signal when transfer is still on-going.

#### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure.

void FLEXIO\_I2C\_MasterEnableAck(*FLEXIO\_I2C\_Type* \*base, bool enable)

Configures the sent ACK/NAK for the following byte.

#### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure.
- enable – True to configure send ACK, false configure to send NAK.

*status\_t* FLEXIO\_I2C\_MasterSetTransferCount(*FLEXIO\_I2C\_Type* \*base, uint16\_t count)

Sets the number of bytes to be transferred from a start signal to a stop signal.

---

**Note:** Call this API before a transfer begins because the timer generates a number of clocks according to the number of bytes that need to be transferred.

---

#### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure.
- count – Number of bytes need to be transferred from a start signal to a re-start/stop signal

#### Return values

- kStatus\_Success – Successfully configured the count.
- kStatus\_InvalidArgument – Input argument is invalid.

static inline void FLEXIO\_I2C\_MasterWriteByte(*FLEXIO\_I2C\_Type* \*base, uint32\_t data)

Writes one byte of data to the I2C bus.

---

**Note:** This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

---

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure.
- data – a byte of data.

static inline uint8\_t FLEXIO\_I2C\_MasterReadByte(*FLEXIO\_I2C\_Type* \*base)

Reads one byte of data from the I2C bus.

---

**Note:** This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the data is ready in the register.

---

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure.

**Returns**

data byte read.

*status\_t* FLEXIO\_I2C\_MasterWriteBlocking(*FLEXIO\_I2C\_Type* \*base, const uint8\_t \*txBuff, uint8\_t txSize)

Sends a buffer of data in bytes.

---

**Note:** This function blocks via polling until all bytes have been sent.

---

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure.
- txBuff – The data bytes to send.
- txSize – The number of data bytes to send.

**Return values**

- kStatus\_Success – Successfully write data.
- kStatus\_FLEXIO\_I2C\_Nak – Receive NAK during writing data.
- kStatus\_FLEXIO\_I2C\_Timeout – Timeout polling status flags.

*status\_t* FLEXIO\_I2C\_MasterReadBlocking(*FLEXIO\_I2C\_Type* \*base, uint8\_t \*rxBuff, uint8\_t rxSize)

Receives a buffer of bytes.

---

**Note:** This function blocks via polling until all bytes have been received.

---

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure.
- rxBuff – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

**Return values**

- kStatus\_Success – Successfully read data.
- kStatus\_FLEXIO\_I2C\_Timeout – Timeout polling status flags.

*status\_t* FLEXIO\_I2C\_MasterTransferBlocking(*FLEXIO\_I2C\_Type* \*base,  
flexio\_i2c\_master\_transfer\_t \*xfer)

Performs a master polling transfer on the I2C bus.

---

**Note:** The API does not return until the transfer succeeds or fails due to receiving NAK.

---

#### Parameters

- base – pointer to FLEXIO\_I2C\_Type structure.
- xfer – pointer to flexio\_i2c\_master\_transfer\_t structure.

#### Returns

status of status\_t.

*status\_t* FLEXIO\_I2C\_MasterTransferCreateHandle(*FLEXIO\_I2C\_Type* \*base,  
flexio\_i2c\_master\_handle\_t \*handle,  
flexio\_i2c\_master\_transfer\_callback\_t  
callback, void \*userData)

Initializes the I2C handle which is used in transactional functions.

#### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure.
- handle – Pointer to flexio\_i2c\_master\_handle\_t structure to store the transfer state.
- callback – Pointer to user callback function.
- userData – User param passed to the callback function.

#### Return values

- kStatus\_Success – Successfully create the handle.
- kStatus\_OutOfRange – The FlexIO type/handle/isr table out of range.

*status\_t* FLEXIO\_I2C\_MasterTransferNonBlocking(*FLEXIO\_I2C\_Type* \*base,  
flexio\_i2c\_master\_handle\_t \*handle,  
flexio\_i2c\_master\_transfer\_t \*xfer)

Performs a master interrupt non-blocking transfer on the I2C bus.

---

**Note:** The API returns immediately after the transfer initiates. Call FLEXIO\_I2C\_MasterTransferGetCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus\_FLEXIO\_I2C\_Busy, the transfer is finished.

---

#### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure
- handle – Pointer to flexio\_i2c\_master\_handle\_t structure which stores the transfer state
- xfer – pointer to flexio\_i2c\_master\_transfer\_t structure

#### Return values

- kStatus\_Success – Successfully start a transfer.
- kStatus\_FLEXIO\_I2C\_Busy – FlexIO I2C is not idle, is running another transfer.

```
status_t FLEXIO_I2C_MasterTransferGetCount(FLEXIO_I2C_Type *base,  
                                           flexio_i2c_master_handle_t *handle, size_t  
                                           *count)
```

Gets the master transfer status during a interrupt non-blocking transfer.

#### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure.
- handle – Pointer to flexio\_i2c\_master\_handle\_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

#### Return values

- kStatus\_InvalidArgument – count is Invalid.
- kStatus\_NoTransferInProgress – There is not a non-blocking transaction currently in progress.
- kStatus\_Success – Successfully return the count.

```
void FLEXIO_I2C_MasterTransferAbort(FLEXIO_I2C_Type *base, flexio_i2c_master_handle_t  
                                   *handle)
```

Aborts an interrupt non-blocking transfer early.

---

**Note:** This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

---

#### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure
- handle – Pointer to flexio\_i2c\_master\_handle\_t structure which stores the transfer state

```
void FLEXIO_I2C_MasterTransferHandleIRQ(void *i2cType, void *i2cHandle)
```

Master interrupt handler.

#### Parameters

- i2cType – Pointer to FLEXIO\_I2C\_Type structure
- i2cHandle – Pointer to flexio\_i2c\_master\_transfer\_t structure

```
FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION
```

FlexIO I2C transfer status.

*Values:*

enumerator kStatus\_FLEXIO\_I2C\_Busy  
I2C is busy doing transfer.

enumerator kStatus\_FLEXIO\_I2C\_Idle  
I2C is busy doing transfer.

enumerator kStatus\_FLEXIO\_I2C\_Nak  
NAK received during transfer.

enumerator kStatus\_FLEXIO\_I2C\_Timeout  
Timeout polling status flags.

`enum _flexio_i2c_master_interrupt`  
Define FlexIO I2C master interrupt mask.  
*Values:*  
`enumerator kFLEXIO_I2C_TxEmptyInterruptEnable`  
Tx buffer empty interrupt enable.  
`enumerator kFLEXIO_I2C_RxFullInterruptEnable`  
Rx buffer full interrupt enable.

`enum _flexio_i2c_master_status_flags`  
Define FlexIO I2C master status mask.  
*Values:*  
`enumerator kFLEXIO_I2C_TxEmptyFlag`  
Tx shifter empty flag.  
`enumerator kFLEXIO_I2C_RxFullFlag`  
Rx shifter full/Transfer complete flag.  
`enumerator kFLEXIO_I2C_ReceiveNakFlag`  
Receive NAK flag.

`enum _flexio_i2c_direction`  
Direction of master transfer.  
*Values:*  
`enumerator kFLEXIO_I2C_Write`  
Master send to slave.  
`enumerator kFLEXIO_I2C_Read`  
Master receive from slave.

`typedef enum _flexio_i2c_direction flexio_i2c_direction_t`  
Direction of master transfer.

`typedef struct _flexio_i2c_type FLEXIO_I2C_Type`  
Define FlexIO I2C master access structure typedef.

`typedef struct _flexio_i2c_master_config flexio_i2c_master_config_t`  
Define FlexIO I2C master user configuration structure.

`typedef struct _flexio_i2c_master_transfer flexio_i2c_master_transfer_t`  
Define FlexIO I2C master transfer structure.

`typedef struct _flexio_i2c_master_handle flexio_i2c_master_handle_t`  
FlexIO I2C master handle typedef.

`typedef void (*flexio_i2c_master_transfer_callback_t)(FLEXIO_I2C_Type *base,  
flexio_i2c_master_handle_t *handle, status_t status, void *userData)`  
FlexIO I2C master transfer callback typedef.

`I2C_RETRY_TIMES`  
Retry times for waiting flag.

`struct _flexio_i2c_type`  
`#include <fsl_flexio_i2c_master.h>` Define FlexIO I2C master access structure typedef.

### Public Members

FLEXIO\_Type \*flexioBase

FlexIO base pointer.

uint8\_t SDAPinIndex

Pin select for I2C SDA.

uint8\_t SCLPinIndex

Pin select for I2C SCL.

uint8\_t shifterIndex[2]

Shifter index used in FlexIO I2C.

uint8\_t timerIndex[3]

Timer index used in FlexIO I2C.

uint32\_t baudrate

Master transfer baudrate, used to calculate delay time.

struct \_flexio\_i2c\_master\_config

*#include <fsl\_flexio\_i2c\_master.h>* Define FlexIO I2C master user configuration structure.

### Public Members

bool enableMaster

Enables the FlexIO I2C peripheral at initialization time.

bool enableInDoze

Enable/disable FlexIO operation in doze mode.

bool enableInDebug

Enable/disable FlexIO operation in debug mode.

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32\_t baudRate\_Bps

Baud rate in Bps.

struct \_flexio\_i2c\_master\_transfer

*#include <fsl\_flexio\_i2c\_master.h>* Define FlexIO I2C master transfer structure.

### Public Members

uint32\_t flags

Transfer flag which controls the transfer, reserved for FlexIO I2C.

uint8\_t slaveAddress

7-bit slave address.

*flexio\_i2c\_direction\_t* direction

Transfer direction, read or write.

uint32\_t subaddress

Sub address. Transferred MSB first.

uint8\_t subaddressSize

Size of sub address.



```
uint8_t volatile *data
    Transfer buffer.
volatile size_t dataSize
    Transfer size.
struct _flexio_i2c_master_handle
    #include <fsl_flexio_i2c_master.h> Define FlexIO I2C master handle structure.
```

### Public Members

```
flexio_i2c_master_transfer_t transfer
    FlexIO I2C master transfer copy.
size_t transferSize
    Total bytes to be transferred.
uint8_t state
    Transfer state maintained during transfer.
flexio_i2c_master_transfer_callback_t completionCallback
    Callback function called at transfer event. Callback function called at transfer event.
void *userData
    Callback parameter passed to callback function.
bool needRestart
    Whether master needs to send re-start signal.
```

## 2.16 FlexIO I2S Driver

```
void FLEXIO_I2S_Init(FLEXIO_I2S_Type *base, const flexio_i2s_config_t *config)
    Initializes the FlexIO I2S.
```

This API configures FlexIO pins and shifter to I2S and configures the FlexIO I2S with a configuration structure. The configuration structure can be filled by the user, or be set with default values by FLEXIO\_I2S\_GetDefaultConfig().

---

**Note:** This API should be called at the beginning of the application to use the FlexIO I2S driver. Otherwise, any access to the FlexIO I2S module can cause hard fault because the clock is not enabled.

---

### Parameters

- base – FlexIO I2S base pointer
- config – FlexIO I2S configure structure.

```
void FLEXIO_I2S_GetDefaultConfig(flexio_i2s_config_t *config)
    Sets the FlexIO I2S configuration structure to default values.
```

The purpose of this API is to get the configuration structure initialized for use in FLEXIO\_I2S\_Init(). Users may use the initialized structure unchanged in FLEXIO\_I2S\_Init() or modify some fields of the structure before calling FLEXIO\_I2S\_Init().

### Parameters

- config – pointer to master configuration structure

void FLEXIO\_I2S\_Deinit(*FLEXIO\_I2S\_Type* \*base)

De-initializes the FlexIO I2S.

Calling this API resets the FlexIO I2S shifter and timer config. After calling this API, call the FLEXIO\_I2S\_Init to use the FlexIO I2S module.

**Parameters**

- base – FlexIO I2S base pointer

static inline void FLEXIO\_I2S\_Enable(*FLEXIO\_I2S\_Type* \*base, bool enable)

Enables/disables the FlexIO I2S module operation.

**Parameters**

- base – Pointer to FLEXIO\_I2S\_Type
- enable – True to enable, false dose not have any effect.

uint32\_t FLEXIO\_I2S\_GetStatusFlags(*FLEXIO\_I2S\_Type* \*base)

Gets the FlexIO I2S status flags.

**Parameters**

- base – Pointer to FLEXIO\_I2S\_Type structure

**Returns**

Status flag, which are ORed by the enumerators in the \_flexio\_i2s\_status\_flags.

void FLEXIO\_I2S\_EnableInterrupts(*FLEXIO\_I2S\_Type* \*base, uint32\_t mask)

Enables the FlexIO I2S interrupt.

This function enables the FlexIO UART interrupt.

**Parameters**

- base – Pointer to FLEXIO\_I2S\_Type structure
- mask – interrupt source

void FLEXIO\_I2S\_DisableInterrupts(*FLEXIO\_I2S\_Type* \*base, uint32\_t mask)

Disables the FlexIO I2S interrupt.

This function enables the FlexIO UART interrupt.

**Parameters**

- base – pointer to FLEXIO\_I2S\_Type structure
- mask – interrupt source

static inline void FLEXIO\_I2S\_TxEnableDMA(*FLEXIO\_I2S\_Type* \*base, bool enable)

Enables/disables the FlexIO I2S Tx DMA requests.

**Parameters**

- base – FlexIO I2S base pointer
- enable – True means enable DMA, false means disable DMA.

static inline void FLEXIO\_I2S\_RxEnableDMA(*FLEXIO\_I2S\_Type* \*base, bool enable)

Enables/disables the FlexIO I2S Rx DMA requests.

**Parameters**

- base – FlexIO I2S base pointer
- enable – True means enable DMA, false means disable DMA.

```
static inline uint32_t FLEXIO_I2S_TxGetDataRegisterAddress(FLEXIO_I2S_Type *base)
```

Gets the FlexIO I2S send data register address.

This function returns the I2S data register address, mainly used by DMA/eDMA.

#### Parameters

- base – Pointer to *FLEXIO\_I2S\_Type* structure

#### Returns

FlexIO i2s send data register address.

```
static inline uint32_t FLEXIO_I2S_RxGetDataRegisterAddress(FLEXIO_I2S_Type *base)
```

Gets the FlexIO I2S receive data register address.

This function returns the I2S data register address, mainly used by DMA/eDMA.

#### Parameters

- base – Pointer to *FLEXIO\_I2S\_Type* structure

#### Returns

FlexIO i2s receive data register address.

```
void FLEXIO_I2S_MasterSetFormat(FLEXIO_I2S_Type *base, flexio_i2s_format_t *format,  
                                uint32_t srcClock_Hz)
```

Configures the FlexIO I2S audio format in master mode.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

#### Parameters

- base – Pointer to *FLEXIO\_I2S\_Type* structure
- format – Pointer to FlexIO I2S audio data format structure.
- srcClock\_Hz – I2S master clock source frequency in Hz.

```
void FLEXIO_I2S_SlaveSetFormat(FLEXIO_I2S_Type *base, flexio_i2s_format_t *format)
```

Configures the FlexIO I2S audio format in slave mode.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

#### Parameters

- base – Pointer to *FLEXIO\_I2S\_Type* structure
- format – Pointer to FlexIO I2S audio data format structure.

```
status_t FLEXIO_I2S_WriteBlocking(FLEXIO_I2S_Type *base, uint8_t bitWidth, uint8_t *txData,  
                                   size_t size)
```

Sends data using a blocking method.

---

**Note:** This function blocks via polling until data is ready to be sent.

---

#### Parameters

- base – FlexIO I2S base pointer.
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- txData – Pointer to the data to be written.
- size – Bytes to be written.

#### Return values

- `kStatus_Success` – Successfully write data.
- `kStatus_FLEXIO_I2C_Timeout` – Timeout polling status flags.

```
static inline void FLEXIO_I2S_WriteData(FLEXIO_I2S_Type *base, uint8_t bitWidth, uint32_t data)
```

Writes data into a data register.

#### Parameters

- `base` – FlexIO I2S base pointer.
- `bitWidth` – How many bits in a audio word, usually 8/16/24/32 bits.
- `data` – Data to be written.

```
status_t FLEXIO_I2S_ReadBlocking(FLEXIO_I2S_Type *base, uint8_t bitWidth, uint8_t *rxData, size_t size)
```

Receives a piece of data using a blocking method.

---

**Note:** This function blocks via polling until data is ready to be sent.

---

#### Parameters

- `base` – FlexIO I2S base pointer
- `bitWidth` – How many bits in a audio word, usually 8/16/24/32 bits.
- `rxData` – Pointer to the data to be read.
- `size` – Bytes to be read.

#### Return values

- `kStatus_Success` – Successfully read data.
- `kStatus_FLEXIO_I2C_Timeout` – Timeout polling status flags.

```
static inline uint32_t FLEXIO_I2S_ReadData(FLEXIO_I2S_Type *base)
```

Reads a data from the data register.

#### Parameters

- `base` – FlexIO I2S base pointer

#### Returns

Data read from data register.

```
void FLEXIO_I2S_TransferTxCreateHandle(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, flexio_i2s_callback_t callback, void *userData)
```

Initializes the FlexIO I2S handle.

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

#### Parameters

- `base` – Pointer to `FLEXIO_I2S_Type` structure
- `handle` – Pointer to `flexio_i2s_handle_t` structure to store the transfer state.
- `callback` – FlexIO I2S callback function, which is called while finished a block.
- `userData` – User parameter for the FlexIO I2S callback.

```
void FLEXIO_I2S_TransferSetFormat(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle,  
                                flexio_i2s_format_t *format, uint32_t srcClock_Hz)
```

Configures the FlexIO I2S audio format.

Audio format can be changed at run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

#### Parameters

- base – Pointer to FLEXIO\_I2S\_Type structure.
- handle – FlexIO I2S handle pointer.
- format – Pointer to audio data format structure.
- srcClock\_Hz – FlexIO I2S bit clock source frequency in Hz. This parameter should be 0 while in slave mode.

```
void FLEXIO_I2S_TransferRxCreateHandle(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle,  
                                       flexio_i2s_callback_t callback, void *userData)
```

Initializes the FlexIO I2S receive handle.

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

#### Parameters

- base – Pointer to FLEXIO\_I2S\_Type structure.
- handle – Pointer to flexio\_i2s\_handle\_t structure to store the transfer state.
- callback – FlexIO I2S callback function, which is called while finished a block.
- userData – User parameter for the FlexIO I2S callback.

```
status_t FLEXIO_I2S_TransferSendNonBlocking(FLEXIO_I2S_Type *base, flexio_i2s_handle_t  
                                             *handle, flexio_i2s_transfer_t *xfer)
```

Performs an interrupt non-blocking send transfer on FlexIO I2S.

---

**Note:** The API returns immediately after transfer initiates. Call FLEXIO\_I2S\_GetRemainingBytes to poll the transfer status and check whether the transfer is finished. If the return status is 0, the transfer is finished.

---

#### Parameters

- base – Pointer to FLEXIO\_I2S\_Type structure.
- handle – Pointer to flexio\_i2s\_handle\_t structure which stores the transfer state
- xfer – Pointer to flexio\_i2s\_transfer\_t structure

#### Return values

- kStatus\_Success – Successfully start the data transmission.
- kStatus\_FLEXIO\_I2S\_TxBusy – Previous transmission still not finished, data not all written to TX register yet.
- kStatus\_InvalidArgument – The input parameter is invalid.

```
status_t FLEXIO_I2S_TransferReceiveNonBlocking(FLEXIO_I2S_Type *base, flexio_i2s_handle_t  
                                                *handle, flexio_i2s_transfer_t *xfer)
```

Performs an interrupt non-blocking receive transfer on FlexIO I2S.

---

**Note:** The API returns immediately after transfer initiates. Call FLEXIO\_I2S\_GetRemainingBytes to poll the transfer status to check whether the transfer is finished. If the return status is 0, the transfer is finished.

---

#### Parameters

- base – Pointer to FLEXIO\_I2S\_Type structure.
- handle – Pointer to flexio\_i2s\_handle\_t structure which stores the transfer state
- xfer – Pointer to flexio\_i2s\_transfer\_t structure

#### Return values

- kStatus\_Success – Successfully start the data receive.
- kStatus\_FLEXIO\_I2S\_RxBusy – Previous receive still not finished.
- kStatus\_InvalidArgument – The input parameter is invalid.

void FLEXIO\_I2S\_TransferAbortSend(*FLEXIO\_I2S\_Type* \*base, *flexio\_i2s\_handle\_t* \*handle)  
Aborts the current send.

---

**Note:** This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

---

#### Parameters

- base – Pointer to FLEXIO\_I2S\_Type structure.
- handle – Pointer to flexio\_i2s\_handle\_t structure which stores the transfer state

void FLEXIO\_I2S\_TransferAbortReceive(*FLEXIO\_I2S\_Type* \*base, *flexio\_i2s\_handle\_t* \*handle)  
Aborts the current receive.

---

**Note:** This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

---

#### Parameters

- base – Pointer to FLEXIO\_I2S\_Type structure.
- handle – Pointer to flexio\_i2s\_handle\_t structure which stores the transfer state

*status\_t* FLEXIO\_I2S\_TransferGetSendCount(*FLEXIO\_I2S\_Type* \*base, *flexio\_i2s\_handle\_t* \*handle, *size\_t* \*count)

Gets the remaining bytes to be sent.

#### Parameters

- base – Pointer to FLEXIO\_I2S\_Type structure.
- handle – Pointer to flexio\_i2s\_handle\_t structure which stores the transfer state
- count – Bytes sent.

#### Return values

- `kStatus_Success` – Succeed get the transfer count.
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

`status_t FLEXIO_I2S_TransferGetReceiveCount(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, size_t *count)`

Gets the remaining bytes to be received.

#### Parameters

- `base` – Pointer to `FLEXIO_I2S_Type` structure.
- `handle` – Pointer to `flexio_i2s_handle_t` structure which stores the transfer state
- `count` – Bytes recieved.

#### Return values

- `kStatus_Success` – Succeed get the transfer count.
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

#### Returns

count Bytes received.

`void FLEXIO_I2S_TransferTxHandleIRQ(void *i2sBase, void *i2sHandle)`

Tx interrupt handler.

#### Parameters

- `i2sBase` – Pointer to `FLEXIO_I2S_Type` structure.
- `i2sHandle` – Pointer to `flexio_i2s_handle_t` structure

`void FLEXIO_I2S_TransferRxHandleIRQ(void *i2sBase, void *i2sHandle)`

Rx interrupt handler.

#### Parameters

- `i2sBase` – Pointer to `FLEXIO_I2S_Type` structure.
- `i2sHandle` – Pointer to `flexio_i2s_handle_t` structure.

`FSL_FLEXIO_I2S_DRIVER_VERSION`

FlexIO I2S driver version 2.2.2.

FlexIO I2S transfer status.

Values:

enumerator `kStatus_FLEXIO_I2S_Idle`

FlexIO I2S is in idle state

enumerator `kStatus_FLEXIO_I2S_TxBusy`

FlexIO I2S Tx is busy

enumerator `kStatus_FLEXIO_I2S_RxBusy`

FlexIO I2S Rx is busy

enumerator `kStatus_FLEXIO_I2S_Error`

FlexIO I2S error occurred

enumerator `kStatus_FLEXIO_I2S_QueueFull`

FlexIO I2S transfer queue is full.

enumerator kStatus\_FLEXIO\_I2S\_Timeout  
FlexIO I2S timeout polling status flags.

enum \_flexio\_i2s\_master\_slave  
Master or slave mode.

*Values:*

enumerator kFLEXIO\_I2S\_Master  
Master mode

enumerator kFLEXIO\_I2S\_Slave  
Slave mode

\_flexio\_i2s\_interrupt\_enable Define FlexIO FlexIO I2S interrupt mask.

*Values:*

enumerator kFLEXIO\_I2S\_TxDataRegEmptyInterruptEnable  
Transmit buffer empty interrupt enable.

enumerator kFLEXIO\_I2S\_RxDataRegFullInterruptEnable  
Receive buffer full interrupt enable.

\_flexio\_i2s\_status\_flags Define FlexIO FlexIO I2S status mask.

*Values:*

enumerator kFLEXIO\_I2S\_TxDataRegEmptyFlag  
Transmit buffer empty flag.

enumerator kFLEXIO\_I2S\_RxDataRegFullFlag  
Receive buffer full flag.

enum \_flexio\_i2s\_sample\_rate  
Audio sample rate.

*Values:*

enumerator kFLEXIO\_I2S\_SampleRate8KHz  
Sample rate 8000Hz

enumerator kFLEXIO\_I2S\_SampleRate11025Hz  
Sample rate 11025Hz

enumerator kFLEXIO\_I2S\_SampleRate12KHz  
Sample rate 12000Hz

enumerator kFLEXIO\_I2S\_SampleRate16KHz  
Sample rate 16000Hz

enumerator kFLEXIO\_I2S\_SampleRate22050Hz  
Sample rate 22050Hz

enumerator kFLEXIO\_I2S\_SampleRate24KHz  
Sample rate 24000Hz

enumerator kFLEXIO\_I2S\_SampleRate32KHz  
Sample rate 32000Hz

enumerator kFLEXIO\_I2S\_SampleRate44100Hz  
Sample rate 44100Hz



```

    enumerator kFLEXIO_I2S_SampleRate48KHz
        Sample rate 48000Hz
    enumerator kFLEXIO_I2S_SampleRate96KHz
        Sample rate 96000Hz
enum _flexio_i2s_word_width
    Audio word width.
    Values:
    enumerator kFLEXIO_I2S_WordWidth8bits
        Audio data width 8 bits
    enumerator kFLEXIO_I2S_WordWidth16bits
        Audio data width 16 bits
    enumerator kFLEXIO_I2S_WordWidth24bits
        Audio data width 24 bits
    enumerator kFLEXIO_I2S_WordWidth32bits
        Audio data width 32 bits
typedef struct _flexio_i2s_type FLEXIO_I2S_Type
    Define FlexIO I2S access structure typedef.
typedef enum _flexio_i2s_master_slave flexio_i2s_master_slave_t
    Master or slave mode.
typedef struct _flexio_i2s_config flexio_i2s_config_t
    FlexIO I2S configure structure.
typedef struct _flexio_i2s_format flexio_i2s_format_t
    FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx.
typedef enum _flexio_i2s_sample_rate flexio_i2s_sample_rate_t
    Audio sample rate.
typedef enum _flexio_i2s_word_width flexio_i2s_word_width_t
    Audio word width.
typedef struct _flexio_i2s_transfer flexio_i2s_transfer_t
    Define FlexIO I2S transfer structure.
typedef struct _flexio_i2s_handle flexio_i2s_handle_t
typedef void (*flexio_i2s_callback_t)(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle,
    status_t status, void *userData)
    FlexIO I2S xfer callback prototype.
I2S_RETRY_TIMES
    Retry times for waiting flag.
FLEXIO_I2S_XFER_QUEUE_SIZE
    FlexIO I2S transfer queue size, user can refine it according to use case.
struct _flexio_i2s_type
    #include <fsl_flexio_i2s.h> Define FlexIO I2S access structure typedef.

```

### Public Members

FLEXIO\_Type \*flexioBase

FlexIO base pointer

uint8\_t txPinIndex

Tx data pin index in FlexIO pins

uint8\_t rxPinIndex

Rx data pin index

uint8\_t bclkPinIndex

Bit clock pin index

uint8\_t fsPinIndex

Frame sync pin index

uint8\_t txShifterIndex

Tx data shifter index

uint8\_t rxShifterIndex

Rx data shifter index

uint8\_t bclkTimerIndex

Bit clock timer index

uint8\_t fsTimerIndex

Frame sync timer index

struct \_flexio\_i2s\_config

*#include <fsl\_flexio\_i2s.h>* FlexIO I2S configure structure.

### Public Members

bool enableI2S

Enable FlexIO I2S

*flexio\_i2s\_master\_slave\_t* masterSlave

Master or slave

*flexio\_pin\_polarity\_t* txPinPolarity

Tx data pin polarity, active high or low

*flexio\_pin\_polarity\_t* rxPinPolarity

Rx data pin polarity

*flexio\_pin\_polarity\_t* bclkPinPolarity

Bit clock pin polarity

*flexio\_pin\_polarity\_t* fsPinPolarity

Frame sync pin polarity

*flexio\_shifter\_timer\_polarity\_t* txTimerPolarity

Tx data valid on bclk rising or falling edge

*flexio\_shifter\_timer\_polarity\_t* rxTimerPolarity

Rx data valid on bclk rising or falling edge

struct \_flexio\_i2s\_format

*#include <fsl\_flexio\_i2s.h>* FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx.

### Public Members

uint8\_t bitWidth  
Bit width of audio data, always 8/16/24/32 bits

uint32\_t sampleRate\_Hz  
Sample rate of the audio data

struct \_flexio\_i2s\_transfer  
*#include <fsl\_flexio\_i2s.h>* Define FlexIO I2S transfer structure.

### Public Members

uint8\_t \*data  
Data buffer start pointer

size\_t dataSize  
Bytes to be transferred.

struct \_flexio\_i2s\_handle  
*#include <fsl\_flexio\_i2s.h>* Define FlexIO I2S handle structure.

### Public Members

uint32\_t state  
Internal state

*flexio\_i2s\_callback\_t* callback  
Callback function called at transfer event

void \*userData  
Callback parameter passed to callback function

uint8\_t bitWidth  
Bit width for transfer, 8/16/24/32bits

*flexio\_i2s\_transfer\_t* queue[(4U)]  
Transfer queue storing queued transfer

size\_t transferSize[(4U)]  
Data bytes need to transfer

volatile uint8\_t queueUser  
Index for user to queue transfer

volatile uint8\_t queueDriver  
Index for driver to get the transfer data and size

## 2.17 FlexIO SPI Driver

void FLEXIO\_SPI\_MasterInit(*FLEXIO\_SPI\_Type* \*base, *flexio\_spi\_master\_config\_t* \*masterConfig, uint32\_t srcClock\_Hz)

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI master hardware, and configures the FlexIO SPI with FlexIO SPI master configuration. The configuration structure can be filled by the user, or be set with default values by the FLEXIO\_SPI\_MasterGetDefaultConfig().

### Example

```
FLEXIO_SPI_Type spiDev = {  
    .flexioBase = FLEXIO,  
    .SDOPinIndex = 0,  
    .SDIPinIndex = 1,  
    .SCKPinIndex = 2,  
    .CSnPinIndex = 3,  
    .shifterIndex = {0,1},  
    .timerIndex = {0,1}  
};  
flexio_spi_master_config_t config = {  
    .enableMaster = true,  
    .enableInDoze = false,  
    .enableInDebug = true,  
    .enableFastAccess = false,  
    .baudRate_Bps = 500000,  
    .phase = kFLEXIO_SPI_ClockPhaseFirstEdge,  
    .direction = kFLEXIO_SPI_MsbFirst,  
    .dataMode = kFLEXIO_SPI_8BitMode  
};  
FLEXIO_SPI_MasterInit(&spiDev, &config, srcClock_Hz);
```

---

**Note:** 1.FlexIO SPI master only support CPOL = 0, which means clock inactive low. 2.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI master communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by  $2*2=4$ . If FlexIO SPI master communicates with FlexIO SPI slave, the maximum baud rate is FlexIO clock frequency divided by  $(1.5+2.5)*2=8$ .

---

### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- masterConfig – Pointer to the flexio\_spi\_master\_config\_t structure.
- srcClock\_Hz – FlexIO source clock in Hz.

void FLEXIO\_SPI\_MasterDeinit(*FLEXIO\_SPI\_Type* \*base)

Resets the FlexIO SPI timer and shifter config.

### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type.

void FLEXIO\_SPI\_MasterGetDefaultConfig(*flexio\_spi\_master\_config\_t* \*masterConfig)

Gets the default configuration to configure the FlexIO SPI master. The configuration can be used directly by calling the FLEXIO\_SPI\_MasterConfigure(). Example:

```
flexio_spi_master_config_t masterConfig;  
FLEXIO_SPI_MasterGetDefaultConfig(&masterConfig);
```

### Parameters

- masterConfig – Pointer to the flexio\_spi\_master\_config\_t structure.

void FLEXIO\_SPI\_SlaveInit(*FLEXIO\_SPI\_Type* \*base, *flexio\_spi\_slave\_config\_t* \*slaveConfig)

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI slave hardware configuration, and configures the FlexIO SPI with FlexIO SPI slave configuration. The configuration structure can be filled by the user, or be set with default values by the FLEXIO\_SPI\_SlaveGetDefaultConfig().

**Note:** 1. Only one timer is needed in the FlexIO SPI slave. As a result, the second timer index is ignored. 2. FlexIO SPI slave only support CPOL = 0, which means clock inactive low. 3. For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI slave communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by  $3 \times 2 = 6$ . If FlexIO SPI slave communicates with FlexIO SPI master, the maximum baud rate is FlexIO clock frequency divided by  $(1.5 + 2.5) \times 2 = 8$ .

Example

```
FLEXIO_SPI_Type spiDev = {
    .flexioBase = FLEXIO,
    .SDOPinIndex = 0,
    .SDIPinIndex = 1,
    .SCKPinIndex = 2,
    .CSnPinIndex = 3,
    .shifterIndex = {0,1},
    .timerIndex = {0}
};
flexio_spi_slave_config_t config = {
    .enableSlave = true,
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
    .direction = kFLEXIO_SPI_MsbFirst,
    .dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_SlaveInit(&spiDev, &config);
```

### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- slaveConfig – Pointer to the flexio\_spi\_slave\_config\_t structure.

void FLEXIO\_SPI\_SlaveDeinit(*FLEXIO\_SPI\_Type* \*base)

Gates the FlexIO clock.

### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type.

void FLEXIO\_SPI\_SlaveGetDefaultConfig(*flexio\_spi\_slave\_config\_t* \*slaveConfig)

Gets the default configuration to configure the FlexIO SPI slave. The configuration can be used directly for calling the FLEXIO\_SPI\_SlaveConfigure(). Example:

```
flexio_spi_slave_config_t slaveConfig;
FLEXIO_SPI_SlaveGetDefaultConfig(&slaveConfig);
```

### Parameters

- slaveConfig – Pointer to the flexio\_spi\_slave\_config\_t structure.

uint32\_t FLEXIO\_SPI\_GetStatusFlags(*FLEXIO\_SPI\_Type* \*base)

Gets FlexIO SPI status flags.

### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.

**Returns**

status flag; Use the status flag to AND the following flag mask and get the status.

- kFLEXIO\_SPI\_TxEmptyFlag
- kFLEXIO\_SPI\_RxEmptyFlag

void FLEXIO\_SPI\_ClearStatusFlags(*FLEXIO\_SPI\_Type* \*base, uint32\_t mask)

Clears FlexIO SPI status flags.

**Parameters**

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- mask – status flag The parameter can be any combination of the following values:
  - kFLEXIO\_SPI\_TxEmptyFlag
  - kFLEXIO\_SPI\_RxEmptyFlag

void FLEXIO\_SPI\_EnableInterrupts(*FLEXIO\_SPI\_Type* \*base, uint32\_t mask)

Enables the FlexIO SPI interrupt.

This function enables the FlexIO SPI interrupt.

**Parameters**

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- mask – interrupt source. The parameter can be any combination of the following values:
  - kFLEXIO\_SPI\_RxFullInterruptEnable
  - kFLEXIO\_SPI\_TxEmptyInterruptEnable

void FLEXIO\_SPI\_DisableInterrupts(*FLEXIO\_SPI\_Type* \*base, uint32\_t mask)

Disables the FlexIO SPI interrupt.

This function disables the FlexIO SPI interrupt.

**Parameters**

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- mask – interrupt source The parameter can be any combination of the following values:
  - kFLEXIO\_SPI\_RxFullInterruptEnable
  - kFLEXIO\_SPI\_TxEmptyInterruptEnable

void FLEXIO\_SPI\_EnableDMA(*FLEXIO\_SPI\_Type* \*base, uint32\_t mask, bool enable)

Enables/disables the FlexIO SPI transmit DMA. This function enables/disables the FlexIO SPI Tx DMA, which means that asserting the kFLEXIO\_SPI\_TxEmptyFlag does/doesn't trigger the DMA request.

**Parameters**

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- mask – SPI DMA source.
- enable – True means enable DMA, false means disable DMA.

static inline uint32\_t FLEXIO\_SPI\_GetTxDataRegisterAddress(*FLEXIO\_SPI\_Type* \*base,  
flexio\_spi\_shift\_direction\_t  
direction)

Gets the FlexIO SPI transmit data register address for MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

#### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- direction – Shift direction of MSB first or LSB first.

#### Returns

FlexIO SPI transmit data register address.

```
static inline uint32_t FLEXIO_SPI_GetRxDataRegisterAddress(FLEXIO_SPI_Type *base,  
                                                         flexio_spi_shift_direction_t  
                                                         direction)
```

Gets the FlexIO SPI receive data register address for the MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

#### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- direction – Shift direction of MSB first or LSB first.

#### Returns

FlexIO SPI receive data register address.

```
static inline void FLEXIO_SPI_Enable(FLEXIO_SPI_Type *base, bool enable)
```

Enables/disables the FlexIO SPI module operation.

#### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type.
- enable – True to enable, false does not have any effect.

```
void FLEXIO_SPI_MasterSetBaudRate(FLEXIO_SPI_Type *base, uint32_t baudRate_Bps,  
                                  uint32_t srcClockHz)
```

Sets baud rate for the FlexIO SPI transfer, which is only used for the master.

#### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- baudRate\_Bps – Baud Rate needed in Hz.
- srcClockHz – SPI source clock frequency in Hz.

```
static inline void FLEXIO_SPI_WriteData(FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t  
                                       direction, uint32_t data)
```

Writes one byte of data, which is sent using the MSB method.

---

**Note:** This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

---

#### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- direction – Shift direction of MSB first or LSB first.
- data – 8/16/32 bit data.

```
static inline uint32_t FLEXIO_SPI_ReadData(FLEXIO_SPI_Type *base,  
                                           flexio_spi_shift_direction_t direction)
```

Reads 8 bit/16 bit data.

---

**Note:** This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

---

#### Parameters

- base – Pointer to the *FLEXIO\_SPI\_Type* structure.
- direction – Shift direction of MSB first or LSB first.

#### Returns

8 bit/16 bit data received.

```
status_t FLEXIO_SPI_WriteBlocking(FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t  
                                  direction, const uint8_t *buffer, size_t size)
```

Sends a buffer of data bytes.

---

**Note:** This function blocks using the polling method until all bytes have been sent.

---

#### Parameters

- base – Pointer to the *FLEXIO\_SPI\_Type* structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The data bytes to send.
- size – The number of data bytes to send.

#### Return values

- kStatus\_Success – Successfully create the handle.
- kStatus\_FLEXIO\_SPI\_Timeout – The transfer timed out and was aborted.

```
status_t FLEXIO_SPI_ReadBlocking(FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t  
                                  direction, uint8_t *buffer, size_t size)
```

Receives a buffer of bytes.

---

**Note:** This function blocks using the polling method until all bytes have been received.

---

#### Parameters

- base – Pointer to the *FLEXIO\_SPI\_Type* structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The buffer to store the received bytes.
- size – The number of data bytes to be received.

#### Return values

- kStatus\_Success – Successfully create the handle.
- kStatus\_FLEXIO\_SPI\_Timeout – The transfer timed out and was aborted.



*status\_t* FLEXIO\_SPI\_MasterTransferBlocking(*FLEXIO\_SPI\_Type* \*base, *flexio\_spi\_transfer\_t* \*xfer)

Receives a buffer of bytes.

---

**Note:** This function blocks via polling until all bytes have been received.

---

#### Parameters

- base – pointer to FLEXIO\_SPI\_Type structure
- xfer – FlexIO SPI transfer structure, see flexio\_spi\_transfer\_t.

#### Return values

- kStatus\_Success – Successfully create the handle.
- kStatus\_FLEXIO\_SPI\_Timeout – The transfer timed out and was aborted.

void FLEXIO\_SPI\_FlushShifters(*FLEXIO\_SPI\_Type* \*base)

Flush tx/rx shifters.

#### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.

*status\_t* FLEXIO\_SPI\_MasterTransferCreateHandle(*FLEXIO\_SPI\_Type* \*base,  
flexio\_spi\_master\_handle\_t \*handle,  
flexio\_spi\_master\_transfer\_callback\_t  
callback, void \*userData)

Initializes the FlexIO SPI Master handle, which is used in transactional functions.

#### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- handle – Pointer to the flexio\_spi\_master\_handle\_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

#### Return values

- kStatus\_Success – Successfully create the handle.
- kStatus\_OutOfRange – The FlexIO type/handle/ISR table out of range.

*status\_t* FLEXIO\_SPI\_MasterTransferNonBlocking(*FLEXIO\_SPI\_Type* \*base,  
flexio\_spi\_master\_handle\_t \*handle,  
flexio\_spi\_transfer\_t \*xfer)

Master transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

#### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- handle – Pointer to the flexio\_spi\_master\_handle\_t structure to store the transfer state.
- xfer – FlexIO SPI transfer structure. See flexio\_spi\_transfer\_t.

#### Return values

- kStatus\_Success – Successfully start a transfer.

- `kStatus_InvalidArgument` – Input argument is invalid.
- `kStatus_FLEXIO_SPI_Busy` – SPI is not idle, is running another transfer.

`void FLEXIO_SPI_MasterTransferAbort(FLEXIO_SPI_Type *base, flexio_spi_master_handle_t *handle)`

Aborts the master data transfer, which used IRQ.

#### Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to the `flexio_spi_master_handle_t` structure to store the transfer state.

`status_t FLEXIO_SPI_MasterTransferGetCount(FLEXIO_SPI_Type *base, flexio_spi_master_handle_t *handle, size_t *count)`

Gets the data transfer status which used IRQ.

#### Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to the `flexio_spi_master_handle_t` structure to store the transfer state.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

#### Return values

- `kStatus_InvalidArgument` – count is Invalid.
- `kStatus_Success` – Successfully return the count.

`void FLEXIO_SPI_MasterTransferHandleIRQ(void *spiType, void *spiHandle)`

FlexIO SPI master IRQ handler function.

#### Parameters

- `spiType` – Pointer to the `FLEXIO_SPI_Type` structure.
- `spiHandle` – Pointer to the `flexio_spi_master_handle_t` structure to store the transfer state.

`status_t FLEXIO_SPI_SlaveTransferCreateHandle(FLEXIO_SPI_Type *base, flexio_spi_slave_handle_t *handle, flexio_spi_slave_transfer_callback_t callback, void *userData)`

Initializes the FlexIO SPI Slave handle, which is used in transactional functions.

#### Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.
- `callback` – The callback function.
- `userData` – The parameter of the callback function.

#### Return values

- `kStatus_Success` – Successfully create the handle.
- `kStatus_OutOfRange` – The FlexIO type/handle/ISR table out of range.

```
status_t FLEXIO_SPI_SlaveTransferNonBlocking(FLEXIO_SPI_Type *base,  
                                             flexio_spi_slave_handle_t *handle,  
                                             flexio_spi_transfer_t *xfer)
```

Slave transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

#### Parameters

- handle – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.
- base – Pointer to the `FLEXIO_SPI_Type` structure.
- xfer – FlexIO SPI transfer structure. See `flexio_spi_transfer_t`.

#### Return values

- `kStatus_Success` – Successfully start a transfer.
- `kStatus_InvalidArgument` – Input argument is invalid.
- `kStatus_FLEXIO_SPI_Busy` – SPI is not idle; it is running another transfer.

```
static inline void FLEXIO_SPI_SlaveTransferAbort(FLEXIO_SPI_Type *base,  
                                                flexio_spi_slave_handle_t *handle)
```

Aborts the slave data transfer which used IRQ, share same API with master.

#### Parameters

- base – Pointer to the `FLEXIO_SPI_Type` structure.
- handle – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.

```
static inline status_t FLEXIO_SPI_SlaveTransferGetCount(FLEXIO_SPI_Type *base,  
                                                       flexio_spi_slave_handle_t *handle,  
                                                       size_t *count)
```

Gets the data transfer status which used IRQ, share same API with master.

#### Parameters

- base – Pointer to the `FLEXIO_SPI_Type` structure.
- handle – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

#### Return values

- `kStatus_InvalidArgument` – count is Invalid.
- `kStatus_Success` – Successfully return the count.

```
void FLEXIO_SPI_SlaveTransferHandleIRQ(void *spiType, void *spiHandle)
```

FlexIO SPI slave IRQ handler function.

#### Parameters

- spiType – Pointer to the `FLEXIO_SPI_Type` structure.
- spiHandle – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.

```
FSL_FLEXIO_SPI_DRIVER_VERSION
```

FlexIO SPI driver version.

Error codes for the FlexIO SPI driver.

*Values:*

enumerator kStatus\_FLEXIO\_SPI\_Busy  
FlexIO SPI is busy.

enumerator kStatus\_FLEXIO\_SPI\_Idle  
SPI is idle

enumerator kStatus\_FLEXIO\_SPI\_Error  
FlexIO SPI error.

enumerator kStatus\_FLEXIO\_SPI\_Timeout  
FlexIO SPI timeout polling status flags.

enum \_flexio\_spi\_clock\_phase  
FlexIO SPI clock phase configuration.

*Values:*

enumerator kFLEXIO\_SPI\_ClockPhaseFirstEdge  
First edge on SPCK occurs at the middle of the first cycle of a data transfer.

enumerator kFLEXIO\_SPI\_ClockPhaseSecondEdge  
First edge on SPCK occurs at the start of the first cycle of a data transfer.

enum \_flexio\_spi\_shift\_direction  
FlexIO SPI data shifter direction options.

*Values:*

enumerator kFLEXIO\_SPI\_MsbFirst  
Data transfers start with most significant bit.

enumerator kFLEXIO\_SPI\_LsbFirst  
Data transfers start with least significant bit.

enum \_flexio\_spi\_data\_bitcount\_mode  
FlexIO SPI data length mode options.

*Values:*

enumerator kFLEXIO\_SPI\_8BitMode  
8-bit data transmission mode.

enumerator kFLEXIO\_SPI\_16BitMode  
16-bit data transmission mode.

enumerator kFLEXIO\_SPI\_32BitMode  
32-bit data transmission mode.

enum \_flexio\_spi\_interrupt\_enable  
Define FlexIO SPI interrupt mask.

*Values:*

enumerator kFLEXIO\_SPI\_TxEmptyInterruptEnable  
Transmit buffer empty interrupt enable.

enumerator kFLEXIO\_SPI\_RxFullInterruptEnable  
Receive buffer full interrupt enable.

enum \_flexio\_spi\_status\_flags

Define FlexIO SPI status mask.

*Values:*

enumerator kFLEXIO\_SPI\_TxBufferEmptyFlag

Transmit buffer empty flag.

enumerator kFLEXIO\_SPI\_RxBufferFullFlag

Receive buffer full flag.

enum \_flexio\_spi\_dma\_enable

Define FlexIO SPI DMA mask.

*Values:*

enumerator kFLEXIO\_SPI\_TxDmaEnable

Tx DMA request source

enumerator kFLEXIO\_SPI\_RxDmaEnable

Rx DMA request source

enumerator kFLEXIO\_SPI\_DmaAllEnable

All DMA request source

enum \_flexio\_spi\_transfer\_flags

Define FlexIO SPI transfer flags.

---

**Note:** Use kFLEXIO\_SPI\_csContinuous and one of the other flags to OR together to form the transfer flag.

---

*Values:*

enumerator kFLEXIO\_SPI\_8bitMsb

FlexIO SPI 8-bit MSB first

enumerator kFLEXIO\_SPI\_8bitLsb

FlexIO SPI 8-bit LSB first

enumerator kFLEXIO\_SPI\_16bitMsb

FlexIO SPI 16-bit MSB first

enumerator kFLEXIO\_SPI\_16bitLsb

FlexIO SPI 16-bit LSB first

enumerator kFLEXIO\_SPI\_32bitMsb

FlexIO SPI 32-bit MSB first

enumerator kFLEXIO\_SPI\_32bitLsb

FlexIO SPI 32-bit LSB first

enumerator kFLEXIO\_SPI\_csContinuous

Enable the CS signal continuous mode

typedef enum \_flexio\_spi\_clock\_phase flexio\_spi\_clock\_phase\_t

FlexIO SPI clock phase configuration.

typedef enum \_flexio\_spi\_shift\_direction flexio\_spi\_shift\_direction\_t

FlexIO SPI data shifter direction options.

typedef enum \_flexio\_spi\_data\_bitcount\_mode flexio\_spi\_data\_bitcount\_mode\_t

FlexIO SPI data length mode options.

```
typedef struct _flexio_spi_type FLEXIO_SPI_Type
    Define FlexIO SPI access structure typedef.

typedef struct _flexio_spi_master_config flexio_spi_master_config_t
    Define FlexIO SPI master configuration structure.

typedef struct _flexio_spi_slave_config flexio_spi_slave_config_t
    Define FlexIO SPI slave configuration structure.

typedef struct _flexio_spi_transfer flexio_spi_transfer_t
    Define FlexIO SPI transfer structure.

typedef struct _flexio_spi_master_handle flexio_spi_master_handle_t
    typedef for flexio_spi_master_handle_t in advance.

typedef flexio_spi_master_handle_t flexio_spi_slave_handle_t
    Slave handle is the same with master handle.

typedef void (*flexio_spi_master_transfer_callback_t)(FLEXIO_SPI_Type *base,
flexio_spi_master_handle_t *handle, status_t status, void *userData)
    FlexIO SPI master callback for finished transmit.

typedef void (*flexio_spi_slave_transfer_callback_t)(FLEXIO_SPI_Type *base,
flexio_spi_slave_handle_t *handle, status_t status, void *userData)
    FlexIO SPI slave callback for finished transmit.

FLEXIO_SPI_DUMMYDATA
    FlexIO SPI dummy transfer data, the data is sent while txData is NULL.

SPI_RETRY_TIMES
    Retry times for waiting flag.

FLEXIO_SPI_XFER_DATA_FORMAT(flag)
    Get the transfer data format of width and bit order.

struct _flexio_spi_type
    #include <fsl_flexio_spi.h> Define FlexIO SPI access structure typedef.
```

### Public Members

FLEXIO\_Type \*flexioBase  
FlexIO base pointer.

uint8\_t SDOPinIndex  
Pin select for data output. To set SDO pin in Hi-Z state, user needs to mux the pin as GPIO input and disable all pull up/down in application.

uint8\_t SDIPinIndex  
Pin select for data input.

uint8\_t SCKPinIndex  
Pin select for clock.

uint8\_t CSnPinIndex  
Pin select for enable.

uint8\_t shifterIndex[2]  
Shifter index used in FlexIO SPI.

uint8\_t timerIndex[2]  
Timer index used in FlexIO SPI.

```
struct _flexio_spi_master_config
```

```
#include <fsl_flexio_spi.h> Define FlexIO SPI master configuration structure.
```

### Public Members

```
bool enableMaster
```

Enable/disable FlexIO SPI master after configuration.

```
bool enableInDoze
```

Enable/disable FlexIO operation in doze mode.

```
bool enableInDebug
```

Enable/disable FlexIO operation in debug mode.

```
bool enableFastAccess
```

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

```
uint32_t baudRate_Bps
```

Baud rate in Bps.

```
flexio_spi_clock_phase_t phase
```

Clock phase.

```
flexio_spi_data_bitcount_mode_t dataMode
```

8bit or 16bit mode.

```
struct _flexio_spi_slave_config
```

```
#include <fsl_flexio_spi.h> Define FlexIO SPI slave configuration structure.
```

### Public Members

```
bool enableSlave
```

Enable/disable FlexIO SPI slave after configuration.

```
bool enableInDoze
```

Enable/disable FlexIO operation in doze mode.

```
bool enableInDebug
```

Enable/disable FlexIO operation in debug mode.

```
bool enableFastAccess
```

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

```
flexio_spi_clock_phase_t phase
```

Clock phase.

```
flexio_spi_data_bitcount_mode_t dataMode
```

8bit or 16bit mode.

```
struct _flexio_spi_transfer
```

```
#include <fsl_flexio_spi.h> Define FlexIO SPI transfer structure.
```

### Public Members

```
const uint8_t *txData
```

Send buffer.

```
uint8_t *rxData
    Receive buffer.

size_t dataSize
    Transfer bytes.

uint8_t flags
    FlexIO SPI control flag, MSB first or LSB first.

struct _flexio_spi_master_handle
    #include <fsl_flexio_spi.h> Define FlexIO SPI handle structure.
```

### Public Members

```
const uint8_t *txData
    Transfer buffer.

uint8_t *rxData
    Receive buffer.

size_t transferSize
    Total bytes to be transferred.

volatile size_t txRemainingBytes
    Send data remaining in bytes.

volatile size_t rxRemainingBytes
    Receive data remaining in bytes.

volatile uint32_t state
    FlexIO SPI internal state.

uint8_t bytePerFrame
    SPI mode, 2bytes or 1byte in a frame

flexio_spi_shift_direction_t direction
    Shift direction.

flexio_spi_master_transfer_callback_t callback
    FlexIO SPI callback.

void *userData
    Callback parameter.

bool isCsContinuous
    Is current transfer using CS continuous mode.

uint32_t timer1Cfg
    TIMER1 TIMCFG regiser value backup.
```

## 2.18 FlexIO UART Driver

```
status_t FLEXIO_UART_Init(FLEXIO_UART_Type *base, const flexio_uart_config_t *userConfig,
                           uint32_t srcClock_Hz)
```

Ungates the FlexIO clock, resets the FlexIO module, configures FlexIO UART hardware, and configures the FlexIO UART with FlexIO UART configuration. The configuration structure can be filled by the user or be set with default values by FLEXIO\_UART\_GetDefaultConfig().

Example



```

FLEXIO_UART_Type base = {
    .flexioBase = FLEXIO,
    .TxPinIndex = 0,
    .RxPinIndex = 1,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_uart_config_t config = {
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .baudRate_Bps = 115200U,
    .bitCountPerChar = 8
};
FLEXIO_UART_Init(base, &config, srcClock_Hz);

```

### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- userConfig – Pointer to the flexio\_uart\_config\_t structure.
- srcClock\_Hz – FlexIO source clock in Hz.

### Return values

- kStatus\_Success – Configuration success.
- kStatus\_FLEXIO\_UART\_BaudrateNotSupport – Baudrate is not supported for current clock source frequency.

void FLEXIO\_UART\_Deinit(*FLEXIO\_UART\_Type* \*base)

Resets the FlexIO UART shifter and timer config.

---

**Note:** After calling this API, call the FLEXIO\_UART\_Init to use the FlexIO UART module.

---

### Parameters

- base – Pointer to FLEXIO\_UART\_Type structure

void FLEXIO\_UART\_GetDefaultConfig(*flexio\_uart\_config\_t* \*userConfig)

Gets the default configuration to configure the FlexIO UART. The configuration can be used directly for calling the FLEXIO\_UART\_Init(). Example:

```

flexio_uart_config_t config;
FLEXIO_UART_GetDefaultConfig(&userConfig);

```

### Parameters

- userConfig – Pointer to the flexio\_uart\_config\_t structure.

uint32\_t FLEXIO\_UART\_GetStatusFlags(*FLEXIO\_UART\_Type* \*base)

Gets the FlexIO UART status flags.

### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.

### Returns

FlexIO UART status flags.

void FLEXIO\_UART\_ClearStatusFlags(*FLEXIO\_UART\_Type* \*base, uint32\_t mask)

Gets the FlexIO UART status flags.

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- mask – Status flag. The parameter can be any combination of the following values:
  - kFLEXIO\_UART\_TxDataRegEmptyFlag
  - kFLEXIO\_UART\_RxEmptyFlag
  - kFLEXIO\_UART\_RxOverRunFlag

void FLEXIO\_UART\_EnableInterrupts(*FLEXIO\_UART\_Type* \*base, uint32\_t mask)

Enables the FlexIO UART interrupt.

This function enables the FlexIO UART interrupt.

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- mask – Interrupt source.

void FLEXIO\_UART\_DisableInterrupts(*FLEXIO\_UART\_Type* \*base, uint32\_t mask)

Disables the FlexIO UART interrupt.

This function disables the FlexIO UART interrupt.

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- mask – Interrupt source.

static inline uint32\_t FLEXIO\_UART\_GetTxDataRegisterAddress(*FLEXIO\_UART\_Type* \*base)

Gets the FlexIO UART transmit data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.

#### Returns

FlexIO UART transmit data register address.

static inline uint32\_t FLEXIO\_UART\_GetRxDataRegisterAddress(*FLEXIO\_UART\_Type* \*base)

Gets the FlexIO UART receive data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.

#### Returns

FlexIO UART receive data register address.

static inline void FLEXIO\_UART\_EnableTxDMA(*FLEXIO\_UART\_Type* \*base, bool enable)

Enables/disables the FlexIO UART transmit DMA. This function enables/disables the FlexIO UART Tx DMA, which means asserting the kFLEXIO\_UART\_TxDataRegEmptyFlag does/doesn't trigger the DMA request.

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- enable – True to enable, false to disable.

static inline void FLEXIO\_UART\_EnableRxDMA(*FLEXIO\_UART\_Type* \*base, bool enable)

Enables/disables the FlexIO UART receive DMA. This function enables/disables the FlexIO UART Rx DMA, which means asserting kFLEXIO\_UART\_RxDataRegFullFlag does/doesn't trigger the DMA request.

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- enable – True to enable, false to disable.

static inline void FLEXIO\_UART\_Enable(*FLEXIO\_UART\_Type* \*base, bool enable)

Enables/disables the FlexIO UART module operation.

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type.
- enable – True to enable, false does not have any effect.

static inline void FLEXIO\_UART\_WriteByte(*FLEXIO\_UART\_Type* \*base, const uint8\_t \*buffer)

Writes one byte of data.

---

**Note:** This is a non-blocking API, which returns directly after the data is put into the data register. Ensure that the TxEmptyFlag is asserted before calling this API.

---

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- buffer – The data bytes to send.

static inline void FLEXIO\_UART\_ReadByte(*FLEXIO\_UART\_Type* \*base, uint8\_t \*buffer)

Reads one byte of data.

---

**Note:** This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

---

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- buffer – The buffer to store the received bytes.

*status\_t* FLEXIO\_UART\_WriteBlocking(*FLEXIO\_UART\_Type* \*base, const uint8\_t \*txData, size\_t txSize)

Sends a buffer of data bytes.

---

**Note:** This function blocks using the polling method until all bytes have been sent.

---

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- txData – The data bytes to send.
- txSize – The number of data bytes to send.

#### Return values

- kStatus\_FLEXIO\_UART\_Timeout – Transmission timed out and was aborted.

- `kStatus_Success` – Successfully wrote all data.

`status_t FLEXIO_UART_ReadBlocking(FLEXIO_UART_Type *base, uint8_t *rxData, size_t rxSize)`

Receives a buffer of bytes.

---

**Note:** This function blocks using the polling method until all bytes have been received.

---

### Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `rxData` – The buffer to store the received bytes.
- `rxSize` – The number of data bytes to be received.

### Return values

- `kStatus_FLEXIO_UART_Timeout` – Transmission timed out and was aborted.
- `kStatus_Success` – Successfully received all data.

`status_t FLEXIO_UART_TransferCreateHandle(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, flexio_uart_transfer_callback_t callback, void *userData)`

Initializes the UART handle.

This function initializes the FlexIO UART handle, which can be used for other FlexIO UART transactional APIs. Call this API once to get the initialized handle.

The UART driver supports the “background” receiving, which means that users can set up a RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn’t call the `FLEXIO_UART_TransferReceiveNonBlocking()` API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly. The ring buffer is disabled if passing `NULL` as `ringBuffer`.

### Parameters

- `base` – to `FLEXIO_UART_Type` structure.
- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.
- `callback` – The callback function.
- `userData` – The parameter of the callback function.

### Return values

- `kStatus_Success` – Successfully create the handle.
- `kStatus_OutOfRange` – The FlexIO type/handle/ISR table out of range.

`void FLEXIO_UART_TransferStartRingBuffer(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, uint8_t *ringBuffer, size_t ringBufferSize)`

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn’t call the `UART_ReceiveNonBlocking()` API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly.

---

**Note:** When using the RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, only 31 bytes are used for saving data.

---

### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- handle – Pointer to the flexio\_uart\_handle\_t structure to store the transfer state.
- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – Size of the ring buffer.

void FLEXIO\_UART\_TransferStopRingBuffer(*FLEXIO\_UART\_Type* \*base, *flexio\_uart\_handle\_t* \*handle)

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- handle – Pointer to the flexio\_uart\_handle\_t structure to store the transfer state.

*status\_t* FLEXIO\_UART\_TransferSendNonBlocking(*FLEXIO\_UART\_Type* \*base,  
*flexio\_uart\_handle\_t* \*handle,  
*flexio\_uart\_transfer\_t* \*xfer)

Transmits a buffer of data using the interrupt method.

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in ISR, the FlexIO UART driver calls the callback function and passes the kStatus\_FLEXIO\_UART\_TxIdle as status parameter.

---

**Note:** The kStatus\_FLEXIO\_UART\_TxIdle is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out.

---

### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- handle – Pointer to the flexio\_uart\_handle\_t structure to store the transfer state.
- xfer – FlexIO UART transfer structure. See flexio\_uart\_transfer\_t.

### Return values

- kStatus\_Success – Successfully starts the data transmission.
- kStatus\_UART\_TxBusy – Previous transmission still not finished, data not written to the TX register.

void FLEXIO\_UART\_TransferAbortSend(*FLEXIO\_UART\_Type* \*base, *flexio\_uart\_handle\_t* \*handle)

Aborts the interrupt-driven data transmit.

This function aborts the interrupt-driven data sending. Get the remainBytes to find out how many bytes are still not sent out.

### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- handle – Pointer to the flexio\_uart\_handle\_t structure to store the transfer state.

*status\_t* FLEXIO\_UART\_TransferGetSendCount(*FLEXIO\_UART\_Type* \*base, *flexio\_uart\_handle\_t* \*handle, *size\_t* \*count)

Gets the number of bytes sent.

This function gets the number of bytes sent driven by interrupt.

### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- handle – Pointer to the flexio\_uart\_handle\_t structure to store the transfer state.
- count – Number of bytes sent so far by the non-blocking transaction.

### Return values

- kStatus\_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus\_Success – Successfully return the count.

*status\_t* FLEXIO\_UART\_TransferReceiveNonBlocking(*FLEXIO\_UART\_Type* \*base, *flexio\_uart\_handle\_t* \*handle, *flexio\_uart\_transfer\_t* \*xfer, *size\_t* \*receivedBytes)

Receives a buffer of data using the interrupt method.

This function receives data using the interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in ring buffer is copied and the parameter *receivedBytes* shows how many bytes are copied from the ring buffer. After copying, if the data in ring buffer is not enough to read, the receive request is saved by the UART driver. When new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter *kStatus\_UART\_RxIdle*. For example, if the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer, the 5 bytes are copied to *xfer->data*. This function returns with the parameter *receivedBytes* set to 5. For the last 5 bytes, newly arrived data is saved from the *xfer->data[5]*. When 5 bytes are received, the UART driver notifies upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to *xfer->data*. When all data is received, the upper layer is notified.

### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- handle – Pointer to the flexio\_uart\_handle\_t structure to store the transfer state.
- xfer – UART transfer structure. See *flexio\_uart\_transfer\_t*.
- receivedBytes – Bytes received from the ring buffer directly.

### Return values

- kStatus\_Success – Successfully queue the transfer into the transmit queue.
- kStatus\_FLEXIO\_UART\_RxBusy – Previous receive request is not finished.

```
void FLEXIO_UART_TransferAbortReceive(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle)
```

Aborts the receive data which was using IRQ.

This function aborts the receive data which was using IRQ.

#### Parameters

- base – Pointer to the *FLEXIO\_UART\_Type* structure.
- handle – Pointer to the *flexio\_uart\_handle\_t* structure to store the transfer state.

```
status_t FLEXIO_UART_TransferGetReceiveCount(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, size_t *count)
```

Gets the number of bytes received.

This function gets the number of bytes received driven by interrupt.

#### Parameters

- base – Pointer to the *FLEXIO\_UART\_Type* structure.
- handle – Pointer to the *flexio\_uart\_handle\_t* structure to store the transfer state.
- count – Number of bytes received so far by the non-blocking transaction.

#### Return values

- *kStatus\_NoTransferInProgress* – transfer has finished or no transfer in progress.
- *kStatus\_Success* – Successfully return the count.

```
void FLEXIO_UART_TransferHandleIRQ(void *uartType, void *uartHandle)
```

FlexIO UART IRQ handler function.

This function processes the FlexIO UART transmit and receives the IRQ request.

#### Parameters

- uartType – Pointer to the *FLEXIO\_UART\_Type* structure.
- uartHandle – Pointer to the *flexio\_uart\_handle\_t* structure to store the transfer state.

```
void FLEXIO_UART_FlushShifters(FLEXIO_UART_Type *base)
```

Flush tx/rx shifters.

#### Parameters

- base – Pointer to the *FLEXIO\_UART\_Type* structure.

```
FSL_FLEXIO_UART_DRIVER_VERSION
```

FlexIO UART driver version.

Error codes for the UART driver.

*Values:*

enumerator *kStatus\_FLEXIO\_UART\_TxBusy*  
Transmitter is busy.

enumerator *kStatus\_FLEXIO\_UART\_RxBusy*  
Receiver is busy.

enumerator kStatus\_FLEXIO\_UART\_TxIdle

UART transmitter is idle.

enumerator kStatus\_FLEXIO\_UART\_RxIdle

UART receiver is idle.

enumerator kStatus\_FLEXIO\_UART\_ERROR

ERROR happens on UART.

enumerator kStatus\_FLEXIO\_UART\_RxRingBufferOverrun

UART RX software ring buffer overrun.

enumerator kStatus\_FLEXIO\_UART\_RxHardwareOverrun

UART RX receiver overrun.

enumerator kStatus\_FLEXIO\_UART\_Timeout

UART times out.

enumerator kStatus\_FLEXIO\_UART\_BaudrateNotSupport

Baudrate is not supported in current clock source

enum \_flexio\_uart\_bit\_count\_per\_char

FlexIO UART bit count per char.

*Values:*

enumerator kFLEXIO\_UART\_7BitsPerChar

7-bit data characters

enumerator kFLEXIO\_UART\_8BitsPerChar

8-bit data characters

enumerator kFLEXIO\_UART\_9BitsPerChar

9-bit data characters

enum \_flexio\_uart\_interrupt\_enable

Define FlexIO UART interrupt mask.

*Values:*

enumerator kFLEXIO\_UART\_TxDataRegEmptyInterruptEnable

Transmit buffer empty interrupt enable.

enumerator kFLEXIO\_UART\_RxDataRegFullInterruptEnable

Receive buffer full interrupt enable.

enum \_flexio\_uart\_status\_flags

Define FlexIO UART status mask.

*Values:*

enumerator kFLEXIO\_UART\_TxDataRegEmptyFlag

Transmit buffer empty flag.

enumerator kFLEXIO\_UART\_RxDataRegFullFlag

Receive buffer full flag.

enumerator kFLEXIO\_UART\_RxOverRunFlag

Receive buffer over run flag.

typedef enum \_flexio\_uart\_bit\_count\_per\_char flexio\_uart\_bit\_count\_per\_char\_t

FlexIO UART bit count per char.



```
typedef struct _flexio_uart_type FLEXIO_UART_Type
    Define FlexIO UART access structure typedef.
typedef struct _flexio_uart_config flexio_uart_config_t
    Define FlexIO UART user configuration structure.
typedef struct _flexio_uart_transfer flexio_uart_transfer_t
    Define FlexIO UART transfer structure.
typedef struct _flexio_uart_handle flexio_uart_handle_t
typedef void (*flexio_uart_transfer_callback_t)(FLEXIO_UART_Type *base, flexio_uart_handle_t
*handle, status_t status, void *userData)
    FlexIO UART transfer callback function.
UART_RETRY_TIMES
    Retry times for waiting flag.
struct _flexio_uart_type
    #include <fsl_flexio_uart.h> Define FlexIO UART access structure typedef.
```

### Public Members

```
FLEXIO_Type *flexioBase
    FlexIO base pointer.
uint8_t TxPinIndex
    Pin select for UART_Tx.
uint8_t RxPinIndex
    Pin select for UART_Rx.
uint8_t shifterIndex[2]
    Shifter index used in FlexIO UART.
uint8_t timerIndex[2]
    Timer index used in FlexIO UART.
struct _flexio_uart_config
    #include <fsl_flexio_uart.h> Define FlexIO UART user configuration structure.
```

### Public Members

```
bool enableUart
    Enable/disable FlexIO UART TX & RX.
bool enableInDoze
    Enable/disable FlexIO operation in doze mode
bool enableInDebug
    Enable/disable FlexIO operation in debug mode
bool enableFastAccess
    Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to
    be at least twice the frequency of the bus clock.
uint32_t baudRate_Bps
    Baud rate in Bps.
```

*flexio\_uart\_bit\_count\_per\_char\_t* bitCountPerChar  
number of bits, 7/8/9 -bit

struct *\_flexio\_uart\_transfer*

*#include <fsl\_flexio\_uart.h>* Define FlexIO UART transfer structure.

### Public Members

size\_t dataSize

Transfer size

struct *\_flexio\_uart\_handle*

*#include <fsl\_flexio\_uart.h>* Define FLEXIO UART handle structure.

### Public Members

const uint8\_t \*volatile txData

Address of remaining data to send.

volatile size\_t txDataSize

Size of the remaining data to send.

uint8\_t \*volatile rxData

Address of remaining data to receive.

volatile size\_t rxDataSize

Size of the remaining data to receive.

size\_t txDataSizeAll

Total bytes to be sent.

size\_t rxDataSizeAll

Total bytes to be received.

uint8\_t \*rxRingBuffer

Start address of the receiver ring buffer.

size\_t rxRingBufferSize

Size of the ring buffer.

volatile uint16\_t rxRingBufferHead

Index for the driver to store received data into ring buffer.

volatile uint16\_t rxRingBufferTail

Index for the user to get data from the ring buffer.

*flexio\_uart\_transfer\_callback\_t* callback

Callback function.

void \*userData

UART callback function parameter.

volatile uint8\_t txState

TX transfer state.

volatile uint8\_t rxState

RX transfer state

union *\_\_unnamed42\_\_*

**Public Members**

uint8\_t \*data

The buffer of data to be transfer.

uint8\_t \*rxData

The buffer to receive data.

const uint8\_t \*txData

The buffer of data to be sent.

## 2.19 ftfx adapter

## 2.20 Ftfx CACHE Driver

enum \_ftfx\_cache\_ram\_func\_constants

Constants for execute-in-RAM flash function.

*Values:*

enumerator kFTFx\_CACHE\_RamFuncMaxSizeInWords

The maximum size of execute-in-RAM function.

typedef struct \_flash\_prefetch\_speculation\_status ftfx\_prefetch\_speculation\_status\_t

FTFx prefetch speculation status.

typedef struct \_ftfx\_cache\_config ftfx\_cache\_config\_t

FTFx cache driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

status\_t FTFx\_CACHE\_Init(ftfx\_cache\_config\_t \*config)

Initializes the global FTFx cache structure members.

This function checks and initializes the Flash module for the other FTFx cache APIs.

**Parameters**

- config – Pointer to the storage for the driver runtime state.

**Return values**

- kStatus\_FTFx\_Success – API was executed successfully.
- kStatus\_FTFx\_InvalidArgument – An invalid argument is provided.
- kStatus\_FTFx\_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.

status\_t FTFx\_CACHE\_ClearCachePrefetchSpeculation(ftfx\_cache\_config\_t \*config, bool isPreProcess)

Process the cache/prefetch/speculation to the flash.

**Parameters**

- config – A pointer to the storage for the driver runtime state.
- isPreProcess – The possible option used to control flash cache/prefetch/speculation

**Return values**

- kStatus\_FTFx\_Success – API was executed successfully.

- `kStatus_FTFx_InvalidArgument` – Invalid argument is provided.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.

`status_t FTFx_CACHE_PflashSetPrefetchSpeculation(ftfx_prefetch_speculation_status_t *speculationStatus)`

Sets the PFlash prefetch speculation to the intended speculation status.

#### Parameters

- `speculationStatus` – The expected protect status to set to the PFlash protection register. Each bit is

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidSpeculationOption` – An invalid speculation option argument is provided.

`status_t FTFx_CACHE_PflashGetPrefetchSpeculation(ftfx_prefetch_speculation_status_t *speculationStatus)`

Gets the PFlash prefetch speculation status.

#### Parameters

- `speculationStatus` – Speculation status returned by the PFlash IP.

#### Return values

`kStatus_FTFx_Success` – API was executed successfully.

`struct _flash_prefetch_speculation_status`

`#include <fsl_ftfx_cache.h>` FTFx prefetch speculation status.

#### Public Members

`bool instructionOff`

Instruction speculation.

`bool dataOff`

Data speculation.

`union function_bit_operation_ptr_t`

`#include <fsl_ftfx_cache.h>`

#### Public Members

`uint32_t commadAddr`

`void (*callFlashCommand)(volatile uint32_t *base, uint32_t bitMask, uint32_t bitShift, uint32_t bitValue)`

`struct _ftfx_cache_config`

`#include <fsl_ftfx_cache.h>` FTFx cache driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

## Public Members

uint8\_t flashMemoryIndex

0 - primary flash; 1 - secondary flash

*function\_bit\_operation\_ptr\_t* bitOperFuncAddr

An buffer point to the flash execute-in-RAM function.

## 2.21 ftfx controller

FTFx driver status codes.

*Values:*

enumerator kStatus\_FTFx\_Success

API is executed successfully

enumerator kStatus\_FTFx\_InvalidArgument

Invalid argument

enumerator kStatus\_FTFx\_SizeError

Error size

enumerator kStatus\_FTFx\_AlignmentError

Parameter is not aligned with the specified baseline

enumerator kStatus\_FTFx\_AddressError

Address is out of range

enumerator kStatus\_FTFx\_AccessError

Invalid instruction codes and out-of bound addresses

enumerator kStatus\_FTFx\_ProtectionViolation

The program/erase operation is requested to execute on protected areas

enumerator kStatus\_FTFx\_CommandFailure

Run-time error during command execution.

enumerator kStatus\_FTFx\_UnknownProperty

Unknown property.

enumerator kStatus\_FTFx\_EraseKeyError

API erase key is invalid.

enumerator kStatus\_FTFx\_RegionExecuteOnly

The current region is execute-only.

enumerator kStatus\_FTFx\_ExecuteInRamFunctionNotReady

Execute-in-RAM function is not available.

enumerator kStatus\_FTFx\_PartitionStatusUpdateFailure

Failed to update partition status.

enumerator kStatus\_FTFx\_SetFlexramAsEepromError

Failed to set FlexRAM as EEPROM.

enumerator kStatus\_FTFx\_RecoverFlexramAsRamError

Failed to recover FlexRAM as RAM.

enumerator kStatus\_FTFx\_SetFlexramAsRamError  
Failed to set FlexRAM as RAM.

enumerator kStatus\_FTFx\_RecoverFlexramAsEepromError  
Failed to recover FlexRAM as EEPROM.

enumerator kStatus\_FTFx\_CommandNotSupported  
Flash API is not supported.

enumerator kStatus\_FTFx\_SwapSystemNotInUninitialized  
Swap system is not in an uninitialized state.

enumerator kStatus\_FTFx\_SwapIndicatorAddressError  
The swap indicator address is invalid.

enumerator kStatus\_FTFx\_ReadOnlyProperty  
The flash property is read-only.

enumerator kStatus\_FTFx\_InvalidPropertyValue  
The flash property value is out of range.

enumerator kStatus\_FTFx\_InvalidSpeculationOption  
The option of flash prefetch speculation is invalid.

enumerator kStatus\_FTFx\_CommandOperationInProgress  
The option of flash command is processing.

enum \_ftfx\_driver\_api\_keys

Enumeration for FTFx driver API keys.

---

**Note:** The resulting value is built with a byte order such that the string being readable in expected order when viewed in a hex editor, if the value is treated as a 32-bit little endian value.

---

*Values:*

enumerator kFTFx\_ApiEraseKey  
Key value used to validate all FTFx erase APIs.

void FTFx\_API\_Init(*ftfx\_config\_t* \*config)

Initializes the global flash properties structure members.

This function checks and initializes the Flash module for the other Flash APIs.

#### Parameters

- config – Pointer to the storage for the driver runtime state.

*status\_t* FTFx\_API\_UpdateFlexnvmPartitionStatus(*ftfx\_config\_t* \*config)

Updates FlexNVM memory partition status according to data flash 0 IFR.

This function updates FlexNVM memory partition status.

#### Parameters

- config – Pointer to the storage for the driver runtime state.

#### Return values

- kStatus\_FTFx\_Success – API was executed successfully.
- kStatus\_FTFx\_InvalidArgument – An invalid argument is provided.
- kStatus\_FTFx\_PartitionStatusUpdateFailure – Failed to update the partition status.

*status\_t* FTFx\_CMD\_Erase(*ftfx\_config\_t* \*config, uint32\_t start, uint32\_t lengthInBytes, uint32\_t key)

Erases the flash sectors encompassed by parameters passed into function.

This function erases the appropriate number of flash sectors based on the desired start address and length.

#### Parameters

- config – The pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words) to be erased. Must be word-aligned.
- key – The value used to validate all flash erase APIs.

#### Return values

- kStatus\_FTFx\_Success – API was executed successfully.
- kStatus\_FTFx\_InvalidArgument – An invalid argument is provided.
- kStatus\_FTFx\_AlignmentError – The parameter is not aligned with the specified baseline.
- kStatus\_FTFx\_AddressError – The address is out of range.
- kStatus\_FTFx\_EraseKeyError – The API erase key is invalid.
- kStatus\_FTFx\_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus\_FTFx\_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus\_FTFx\_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus\_FTFx\_CommandFailure – Run-time error during the command execution.

*status\_t* FTFx\_CMD\_EraseSectorNonBlocking(*ftfx\_config\_t* \*config, uint32\_t start, uint32\_t key)

Erases the flash sectors encompassed by parameters passed into function.

This function erases one flash sector size based on the start address.

#### Parameters

- config – The pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
- key – The value used to validate all flash erase APIs.

#### Return values

- kStatus\_FTFx\_Success – API was executed successfully.
- kStatus\_FTFx\_InvalidArgument – An invalid argument is provided.
- kStatus\_FTFx\_AlignmentError – The parameter is not aligned with the specified baseline.
- kStatus\_FTFx\_AddressError – The address is out of range.
- kStatus\_FTFx\_EraseKeyError – The API erase key is invalid.

- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.

`status_t FTFx_CMD_EraseAll(ftfx_config_t *config, uint32_t key)`

Erases entire flash.

#### Parameters

- `config` – Pointer to the storage for the driver runtime state.
- `key` – A value used to validate all flash erase APIs.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_EraseKeyError` – API erase key is invalid.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during command execution.
- `kStatus_FTFx_PartitionStatusUpdateFailure` – Failed to update the partition status.

`status_t FTFx_CMD_EraseAllUnsecure(ftfx_config_t *config, uint32_t key)`

Erases the entire flash, including protected sectors.

#### Parameters

- `config` – Pointer to the storage for the driver runtime state.
- `key` – A value used to validate all flash erase APIs.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_EraseKeyError` – API erase key is invalid.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during command execution.
- `kStatus_FTFx_PartitionStatusUpdateFailure` – Failed to update the partition status.

`status_t FTFx_CMD_EraseAllExecuteOnlySegments(ftfx_config_t *config, uint32_t key)`

Erases all program flash execute-only segments defined by the FXACC registers.

#### Parameters



- `config` – Pointer to the storage for the driver runtime state.
- `key` – A value used to validate all flash erase APIs.

**Return values**

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_EraseKeyError` – API erase key is invalid.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FTFx_CMD_Program(ftfx_config_t *config, uint32_t start, const uint8_t *src, uint32_t lengthInBytes)`

Programs flash with data at locations passed in through parameters.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

**Parameters**

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be programmed. Must be word-aligned.
- `src` – A pointer to the source buffer of data that is to be programmed into the flash.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

**Return values**

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – Parameter is not aligned with the specified baseline.
- `kStatus_FTFx_AddressError` – Address is out of range.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FTFx_CMD_ProgramOnce(ftfx_config_t *config, uint32_t index, const uint8_t *src, uint32_t lengthInBytes)`

Programs Program Once Field through parameters.

This function programs the Program Once Field with the desired data for a given flash area as determined by the index and length.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `index` – The index indicating which area of the Program Once Field to be programmed.
- `src` – A pointer to the source buffer of data that is to be programmed into the Program Once Field.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FTFx_CMD_ProgramSection(ftfx_config_t *config, uint32_t start, const uint8_t *src, uint32_t lengthInBytes)`

Programs flash with data at locations passed in through parameters via the Program Section command.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and length.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be programmed. Must be word-aligned.
- `src` – A pointer to the source buffer of data that is to be programmed into the flash.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – Parameter is not aligned with specified baseline.
- `kStatus_FTFx_AddressError` – Address is out of range.
- `kStatus_FTFx_SetFlexramAsRamError` – Failed to set flexram as RAM.

- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during command execution.
- `kStatus_FTFx_RecoverFlexramAsEepromError` – Failed to recover FlexRAM as EEPROM.

`status_t FTFx_CMD_ProgramPartition(ftfx_config_t *config, ftfx_partition_flexram_load_opt_t option, uint32_t eepromDataSizeCode, uint32_t flexnvmPartitionCode, uint8_t CSEcKeySize, uint8_t CFE)`

Prepares the FlexNVM block for use as data flash, EEPROM backup, or a combination of both and initializes the FlexRAM.

#### Parameters

- `config` – Pointer to storage for the driver runtime state.
- `option` – The option used to set FlexRAM load behavior during reset.
- `eepromDataSizeCode` – Determines the amount of FlexRAM used in each of the available EEPROM subsystems.
- `flexnvmPartitionCode` – Specifies how to split the FlexNVM block between data flash memory and EEPROM backup memory supporting EEPROM functions.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidArgument` – Invalid argument is provided.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during command execution.

`status_t FTFx_CMD_ReadOnce(ftfx_config_t *config, uint32_t index, uint8_t *dst, uint32_t lengthInBytes)`

Reads the Program Once Field through parameters.

This function reads the read once feild with given index and length.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `index` – The index indicating the area of program once field to be read.
- `dst` – A pointer to the destination buffer of data that is used to store data to be read.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

**Return values**

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FTFx_CMD_ReadResource(ftfx_config_t *config, uint32_t start, uint8_t *dst, uint32_t lengthInBytes, ftfx_read_resource_opt_t option)`

Reads the resource with data at locations passed in through parameters.

This function reads the flash memory with the desired location for a given flash area as determined by the start address and length.

**Parameters**

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be programmed. Must be word-aligned.
- `dst` – A pointer to the destination buffer of data that is used to store data to be read.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be read. Must be word-aligned.
- `option` – The resource option which indicates which area should be read back.

**Return values**

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – Parameter is not aligned with the specified baseline.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FTFx_CMD_VerifyErase(ftfx_config_t *config, uint32_t start, uint32_t lengthInBytes, ftfx_margin_value_t margin)`

Verifies an erasure of the desired flash area at a specified margin level.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

**Parameters**

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- `margin` – Read margin choice.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – Parameter is not aligned with specified baseline.
- `kStatus_FTFx_AddressError` – Address is out of range.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FTFx_CMD_VerifyEraseAll(ftfx_config_t *config, ftfx_margin_value_t margin)`

Verifies erasure of the entire flash at a specified margin level.

This function checks whether the flash is erased to the specified read margin level.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `margin` – Read margin choice.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FTFx_CMD_VerifyEraseAllExecuteOnlySegments(ftfx_config_t *config, ftfx_margin_value_t margin)`

Verifies whether the program flash execute-only segments have been erased to the specified read margin level.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `margin` – Read margin choice.

**Return values**

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FTFx_CMD_VerifyProgram(ftfx_config_t *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, ftfx_margin_value_t margin, uint32_t *failedAddress, uint32_t *failedData)`

Verifies programming of the desired flash area at a specified margin level.

This function verifies the data programed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

**Parameters**

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be verified. Must be word-aligned.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- `expectedData` – A pointer to the expected data that is to be verified against.
- `margin` – Read margin choice.
- `failedAddress` – A pointer to the returned failing address.
- `failedData` – A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

**Return values**

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – Parameter is not aligned with specified baseline.
- `kStatus_FTFx_AddressError` – Address is out of range.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

*status\_t* FTFx\_REG\_GetSecurityState(*ftfx\_config\_t* \*config, *ftfx\_security\_state\_t* \*state)

Returns the security state via the pointer passed into the function.

This function retrieves the current flash security status, including the security enabling state and the backdoor key enabling state.

#### Parameters

- config – A pointer to storage for the driver runtime state.
- state – A pointer to the value returned for the current security status code:

#### Return values

- kStatus\_FTFx\_Success – API was executed successfully.
- kStatus\_FTFx\_InvalidArgument – An invalid argument is provided.

*status\_t* FTFx\_CMD\_SecurityBypass(*ftfx\_config\_t* \*config, const uint8\_t \*backdoorKey)

Allows users to bypass security with a backdoor key.

If the MCU is in secured state, this function unsecures the MCU by comparing the provided backdoor key with ones in the flash configuration field.

#### Parameters

- config – A pointer to the storage for the driver runtime state.
- backdoorKey – A pointer to the user buffer containing the backdoor key.

#### Return values

- kStatus\_FTFx\_Success – API was executed successfully.
- kStatus\_FTFx\_InvalidArgument – An invalid argument is provided.
- kStatus\_FTFx\_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus\_FTFx\_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus\_FTFx\_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus\_FTFx\_CommandFailure – Run-time error during the command execution.

*status\_t* FTFx\_CMD\_SetFlexramFunction(*ftfx\_config\_t* \*config, *ftfx\_flexram\_func\_opt\_t* option)

Sets the FlexRAM function command.

#### Parameters

- config – A pointer to the storage for the driver runtime state.
- option – The option used to set the work mode of FlexRAM.

#### Return values

- kStatus\_FTFx\_Success – API was executed successfully.
- kStatus\_FTFx\_InvalidArgument – An invalid argument is provided.
- kStatus\_FTFx\_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus\_FTFx\_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus\_FTFx\_ProtectionViolation – The program/erase operation is requested to execute on protected areas.

- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FTFx_CMD_SwapControl(ftfx_config_t *config, uint32_t address,  
                                ftfx_swap_control_opt_t option, ftfx_swap_state_config_t  
                                *returnInfo)`

Configures the Swap function or checks the swap state of the Flash module.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `address` – Address used to configure the flash Swap function.
- `option` – The possible option used to configure Flash Swap function or check the flash Swap status
- `returnInfo` – A pointer to the data which is used to return the information of flash Swap.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – Parameter is not aligned with specified baseline.
- `kStatus_FTFx_SwapIndicatorAddressError` – Swap indicator address is invalid.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`enum _ftfx_partition_flexram_load_option`

Enumeration for the FlexRAM load during reset option.

*Values:*

enumerator `kFTFx_PartitionFlexramLoadOptLoadedWithValidEepromData`

FlexRAM is loaded with valid EEPROM data during reset sequence.

enumerator `kFTFx_PartitionFlexramLoadOptNotLoaded`

FlexRAM is not loaded during reset sequence.

`enum _ftfx_read_resource_opt`

Enumeration for the two possible options of flash read resource command.

*Values:*

enumerator `kFTFx_ResourceOptionFlashIfR`

Select code for Program flash 0 IFR, Program flash swap 0 IFR, Data flash 0 IFR

enumerator `kFTFx_ResourceOptionVersionId`

Select code for the version ID



enum \_ftfx\_margin\_value

Enumeration for supported FTFx margin levels.

*Values:*

enumerator kFTFx\_MarginValueNormal

Use the 'normal' read level for 1s.

enumerator kFTFx\_MarginValueUser

Apply the 'User' margin to the normal read-1 level.

enumerator kFTFx\_MarginValueFactory

Apply the 'Factory' margin to the normal read-1 level.

enumerator kFTFx\_MarginValueInvalid

Not real margin level, Used to determine the range of valid margin level.

enum \_ftfx\_security\_state

Enumeration for the three possible FTFx security states.

*Values:*

enumerator kFTFx\_SecurityStateNotSecure

Flash is not secure.

enumerator kFTFx\_SecurityStateBackdoorEnabled

Flash backdoor is enabled.

enumerator kFTFx\_SecurityStateBackdoorDisabled

Flash backdoor is disabled.

enum \_ftfx\_flexram\_function\_option

Enumeration for the two possible options of set FlexRAM function command.

*Values:*

enumerator kFTFx\_FlexramFuncOptAvailableAsRam

An option used to make FlexRAM available as RAM

enumerator kFTFx\_FlexramFuncOptAvailableForEeprom

An option used to make FlexRAM available for EEPROM

enum \_flash\_acceleration\_ram\_property

Enumeration for acceleration ram property.

*Values:*

enumerator kFLASH\_AccelerationRamSize

enum \_ftfx\_swap\_control\_option

Enumeration for the possible options of Swap control commands.

*Values:*

enumerator kFTFx\_SwapControlOptionInitializeSystem

An option used to initialize the Swap system

enumerator kFTFx\_SwapControlOptionSetInUpdateState

An option used to set the Swap in an update state

enumerator kFTFx\_SwapControlOptionSetInCompleteState

An option used to set the Swap in a complete state

enumerator kFTFx\_SwapControlOptionReportStatus

An option used to report the Swap status

enumerator kFTFx\_SwapControlOptionDisableSystem

An option used to disable the Swap status

enum \_ftfx\_swap\_state

Enumeration for the possible flash Swap status.

*Values:*

enumerator kFTFx\_SwapStateUninitialized

Flash Swap system is in an uninitialized state.

enumerator kFTFx\_SwapStateReady

Flash Swap system is in a ready state.

enumerator kFTFx\_SwapStateUpdate

Flash Swap system is in an update state.

enumerator kFTFx\_SwapStateUpdateErased

Flash Swap system is in an updateErased state.

enumerator kFTFx\_SwapStateComplete

Flash Swap system is in a complete state.

enumerator kFTFx\_SwapStateDisabled

Flash Swap system is in a disabled state.

enum \_ftfx\_swap\_block\_status

Enumeration for the possible flash Swap block status.

*Values:*

enumerator kFTFx\_SwapBlockStatusLowerHalfProgramBlocksAtZero

Swap block status is that lower half program block at zero.

enumerator kFTFx\_SwapBlockStatusUpperHalfProgramBlocksAtZero

Swap block status is that upper half program block at zero.

enum \_ftfx\_memory\_type

Enumeration for FTFx memory type.

*Values:*

enumerator kFTFx\_MemTypePflash

enumerator kFTFx\_MemTypeFlexnvm

typedef enum \_ftfx\_partition\_flexram\_load\_option ftfx\_partition\_flexram\_load\_opt\_t

Enumeration for the FlexRAM load during reset option.

typedef enum \_ftfx\_read\_resource\_opt ftfx\_read\_resource\_opt\_t

Enumeration for the two possible options of flash read resource command.

typedef enum \_ftfx\_margin\_value ftfx\_margin\_value\_t

Enumeration for supported FTFx margin levels.

typedef enum \_ftfx\_security\_state ftfx\_security\_state\_t

Enumeration for the three possible FTFx security states.

typedef enum \_ftfx\_flexram\_function\_option ftfx\_flexram\_func\_opt\_t

Enumeration for the two possible options of set FlexRAM function command.

typedef enum \_ftfx\_swap\_control\_option ftfx\_swap\_control\_opt\_t

Enumeration for the possible options of Swap control commands.

```
typedef enum _ftfx_swap_state ftfx_swap_state_t
```

Enumeration for the possible flash Swap status.

```
typedef enum _ftfx_swap_block_status ftfx_swap_block_status_t
```

Enumeration for the possible flash Swap block status.

```
typedef struct _ftfx_swap_state_config ftfx_swap_state_config_t
```

Flash Swap information.

```
typedef struct _ftfx_special_mem ftfx_spec_mem_t
```

ftfx special memory access information.

```
typedef struct _ftfx_mem_descriptor ftfx_mem_desc_t
```

Flash memory descriptor.

```
typedef struct _ftfx_ops_config ftfx_ops_config_t
```

Active FTFx information for the current operation.

```
typedef struct _ftfx_ifr_descriptor ftfx_ifr_desc_t
```

Flash IFR memory descriptor.

```
typedef struct _ftfx_config ftfx_config_t
```

Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

```
struct _ftfx_swap_state_config
```

*#include <fsl\_ftfx\_controller.h>* Flash Swap information.

### Public Members

*ftfx\_swap\_state\_t* flashSwapState

The current Swap system status.

*ftfx\_swap\_block\_status\_t* currentSwapBlockStatus

The current Swap block status.

*ftfx\_swap\_block\_status\_t* nextSwapBlockStatus

The next Swap block status.

```
struct _ftfx_special_mem
```

*#include <fsl\_ftfx\_controller.h>* ftfx special memory access information.

### Public Members

uint32\_t base

Base address of flash special memory.

uint32\_t size

size of flash special memory.

uint32\_t count

flash special memory count.

```
struct _ftfx_mem_descriptor
```

*#include <fsl\_ftfx\_controller.h>* Flash memory descriptor.

### Public Members

uint32\_t blockSize

A base address of the flash block

uint32\_t aliasBlockSize

A base address of the alias flash block

uint32\_t totalSize

The size of the flash block.

uint32\_t sectorSize

The size in bytes of a sector of flash.

uint32\_t blockCount

A number of flash blocks.

struct \_ftfx\_ops\_config

*#include <fsl\_ftfx\_controller.h>* Active FTFx information for the current operation.

### Public Members

uint32\_t convertedAddress

A converted address for the current flash type.

struct \_ftfx\_ifr\_descriptor

*#include <fsl\_ftfx\_controller.h>* Flash IFR memory descriptor.

union function\_ptr\_t

*#include <fsl\_ftfx\_controller.h>*

### Public Members

uint32\_t commandAddr

void (\*callFlashCommand)(volatile uint8\_t \*FTMRx\_fstat)

struct \_ftfx\_config

*#include <fsl\_ftfx\_controller.h>* Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

### Public Members

uint32\_t flexramBlockSize

The base address of the FlexRAM/acceleration RAM

uint32\_t flexramTotalSize

The size of the FlexRAM/acceleration RAM

uint16\_t eepromTotalSize

The size of EEPROM area which was partitioned from FlexRAM

*function\_ptr\_t* runCmdFuncAddr

An buffer point to the flash execute-in-RAM function.

struct \_\_unnamed12\_\_

### Public Members

uint8\_t type

Type of flash block.

uint8\_t index

Index of flash block.

struct feature

struct addrAligment

struct feature

struct resRange

### Public Members

uint8\_t versionIdStart

Version ID start address

uint32\_t pflashIfrStart

Program Flash 0 IFR start address

uint32\_t dflashIfrStart

Data Flash 0 IFR start address

uint32\_t pflashSwapIfrStart

Program Flash Swap IFR start address

struct idxInfo

## 2.22 ftfx feature

FTFX\_DRIVER\_IS\_FLASH\_RESIDENT

Flash driver location.

Used for the flash resident application.

FTFX\_DRIVER\_IS\_EXPORTED

Flash Driver Export option.

Used for the MCUXpresso SDK application.

FTFX\_FLASH1\_HAS\_PROT\_CONTROL

Indicates whether the secondary flash has its own protection register in flash module.

FTFX\_FLASH1\_HAS\_XACC\_CONTROL

Indicates whether the secondary flash has its own Execute-Only access register in flash module.

FTFX\_DRIVER\_HAS\_FLASH1\_SUPPORT

Indicates whether the secondary flash is supported in the Flash driver.

FTFX\_FLASH\_COUNT

FTFX\_FLASH1\_IS\_INDEPENDENT\_BLOCK

## 2.23 Ftfxx FLASH Driver

*status\_t* FLASH\_Init(*flash\_config\_t* \*config)

Initializes the global flash properties structure members.

This function checks and initializes the Flash module for the other Flash APIs.

### Parameters

- config – Pointer to the storage for the driver runtime state.

### Return values

- kStatus\_FTFx\_Success – API was executed successfully.
- kStatus\_FTFx\_InvalidArgument – An invalid argument is provided.
- kStatus\_FTFx\_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus\_FTFx\_PartitionStatusUpdateFailure – Failed to update the partition status.

*status\_t* FLASH\_Erase(*flash\_config\_t* \*config, uint32\_t start, uint32\_t lengthInBytes, uint32\_t key)

Erases the Dflash sectors encompassed by parameters passed into function.

This function erases the appropriate number of flash sectors based on the desired start address and length.

### Parameters

- config – The pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words) to be erased. Must be word-aligned.
- key – The value used to validate all flash erase APIs.

### Return values

- kStatus\_FTFx\_Success – API was executed successfully; the appropriate number of flash sectors based on the desired start address and length were erased successfully.
- kStatus\_FTFx\_InvalidArgument – An invalid argument is provided.
- kStatus\_FTFx\_AlignmentError – The parameter is not aligned with the specified baseline.
- kStatus\_FTFx\_AddressError – The address is out of range.
- kStatus\_FTFx\_EraseKeyError – The API erase key is invalid.
- kStatus\_FTFx\_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus\_FTFx\_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus\_FTFx\_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus\_FTFx\_CommandFailure – Run-time error during the command execution.

*status\_t* FLASH\_EraseSectorNonBlocking(*flash\_config\_t* \*config, uint32\_t start, uint32\_t key)

Erases the Dflash sectors encompassed by parameters passed into function.

This function erases one flash sector size based on the start address, and it is executed asynchronously.

NOTE: This function can only erase one flash sector at a time, and the other commands can be executed after the previous command has been completed.

#### Parameters

- config – The pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
- key – The value used to validate all flash erase APIs.

#### Return values

- kStatus\_FTFx\_Success – API was executed successfully.
- kStatus\_FTFx\_InvalidArgument – An invalid argument is provided.
- kStatus\_FTFx\_AlignmentError – The parameter is not aligned with the specified baseline.
- kStatus\_FTFx\_AddressError – The address is out of range.
- kStatus\_FTFx\_EraseKeyError – The API erase key is invalid.

*status\_t* FLASH\_EraseAll(*flash\_config\_t* \*config, uint32\_t key)

Erases entire flexnvm.

#### Parameters

- config – Pointer to the storage for the driver runtime state.
- key – A value used to validate all flash erase APIs.

#### Return values

- kStatus\_FTFx\_Success – API was executed successfully; the all pflash and flexnvm were erased successfully, the swap and eeprom have been reset to unconfigured state.
- kStatus\_FTFx\_InvalidArgument – An invalid argument is provided.
- kStatus\_FTFx\_EraseKeyError – API erase key is invalid.
- kStatus\_FTFx\_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus\_FTFx\_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus\_FTFx\_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus\_FTFx\_CommandFailure – Run-time error during command execution.
- kStatus\_FTFx\_PartitionStatusUpdateFailure – Failed to update the partition status.

*status\_t* FLASH\_EraseAllUnsecure(*flash\_config\_t* \*config, uint32\_t key)

Erases the entire flexnvm, including protected sectors.

#### Parameters

- config – Pointer to the storage for the driver runtime state.

- `key` – A value used to validate all flash erase APIs.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the protected sectors of flash were reset to unprotected status.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_EraseKeyError` – API erase key is invalid.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during command execution.
- `kStatus_FTFx_PartitionStatusUpdateFailure` – Failed to update the partition status.

`status_t FLASH_Program(flash_config_t *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)`

Programs flash with data at locations passed in through parameters.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be programmed. Must be word-aligned.
- `src` – A pointer to the source buffer of data that is to be programmed into the flash.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the desired data were programmed successfully into flash based on desired start address and length.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – Parameter is not aligned with the specified baseline.
- `kStatus_FTFx_AddressError` – Address is out of range.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.



*status\_t* FLASH\_ProgramOnce(*flash\_config\_t* \*config, uint32\_t index, uint8\_t \*src, uint32\_t lengthInBytes)

Program the Program-Once-Field through parameters.

This function Program the Program-once-field with given index and length.

#### Parameters

- config – A pointer to the storage for the driver runtime state.
- index – The index indicating the area of program once field to be read.
- src – A pointer to the source buffer of data that is used to store data to be write.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

#### Return values

- kStatus\_FTFx\_Success – API was executed successfully; The index indicating the area of program once field was programmed successfully.
- kStatus\_FTFx\_InvalidArgument – An invalid argument is provided.
- kStatus\_FTFx\_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus\_FTFx\_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus\_FTFx\_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus\_FTFx\_CommandFailure – Run-time error during the command execution.

*status\_t* FLASH\_ProgramSection(*flash\_config\_t* \*config, uint32\_t start, uint8\_t \*src, uint32\_t lengthInBytes)

Programs flash with data at locations passed in through parameters via the Program Section command.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and length.

#### Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- src – A pointer to the source buffer of data that is to be programmed into the flash.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

#### Return values

- kStatus\_FTFx\_Success – API was executed successfully; the desired data have been programmed successfully into flash based on start address and length.
- kStatus\_FTFx\_InvalidArgument – An invalid argument is provided.
- kStatus\_FTFx\_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus\_FTFx\_AddressError – Address is out of range.

- `kStatus_FTFx_SetFlexramAsRamError` – Failed to set flexram as RAM.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during command execution.
- `kStatus_FTFx_RecoverFlexramAsEepromError` – Failed to recover FlexRAM as EEPROM.

`status_t` FLASH\_ReadResource(*flash\_config\_t* \*config, uint32\_t start, uint8\_t \*dst, uint32\_t lengthInBytes, *ftfx\_read\_resource\_opt\_t* option)

Reads the resource with data at locations passed in through parameters.

This function reads the flash memory with the desired location for a given flash area as determined by the start address and length.

#### Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- dst – A pointer to the destination buffer of data that is used to store data to be read.
- lengthInBytes – The length, given in bytes (not words or long-words), to be read. Must be word-aligned.
- option – The resource option which indicates which area should be read back.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the data have been read successfully from program flash IFR, data flash IFR space, and the Version ID field.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – Parameter is not aligned with the specified baseline.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t` FLASH\_ReadOnce(*flash\_config\_t* \*config, uint32\_t index, uint8\_t \*dst, uint32\_t lengthInBytes)

Reads the Program Once Field through parameters.

This function reads the read once field with given index and length.

**Parameters**

- `config` – A pointer to the storage for the driver runtime state.
- `index` – The index indicating the area of program once field to be read.
- `dst` – A pointer to the destination buffer of data that is used to store data to be read.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

**Return values**

- `kStatus_FTFx_Success` – API was executed successfully; the data have been successfully read from Program flash0 IFR map and Program Once field based on index and length.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FLASH_VerifyErase(flash_config_t *config, uint32_t start, uint32_t lengthInBytes, ftfx_margin_value_t margin)`

Verifies an erasure of the desired flash area at a specified margin level.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

**Parameters**

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- `margin` – Read margin choice.

**Return values**

- `kStatus_FTFx_Success` – API was executed successfully; the specified FLASH region has been erased.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – Parameter is not aligned with specified baseline.
- `kStatus_FTFx_AddressError` – Address is out of range.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.

- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FLASH_VerifyEraseAll(flash_config_t *config, ftfx_margin_value_t margin)`

Verifies erasure of the entire flash at a specified margin level.

This function checks whether the flash is erased to the specified read margin level.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `margin` – Read margin choice.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; all program flash and flexnvm were in erased state.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FLASH_VerifyProgram(flash_config_t *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, ftfx_margin_value_t margin, uint32_t *failedAddress, uint32_t *failedData)`

Verifies programming of the desired flash area at a specified margin level.

This function verifies the data programmed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be verified. Must be word-aligned.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- `expectedData` – A pointer to the expected data that is to be verified against.
- `margin` – Read margin choice.
- `failedAddress` – A pointer to the returned failing address.
- `failedData` – A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the desired data have been successfully programed into specified FLASH region.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – Parameter is not aligned with specified baseline.

- `kStatus_FTFx_AddressError` – Address is out of range.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FLASH_GetSecurityState(flash_config_t *config, ftfx_security_state_t *state)`

Returns the security state via the pointer passed into the function.

This function retrieves the current flash security status, including the security enabling state and the backdoor key enabling state.

#### Parameters

- `config` – A pointer to storage for the driver runtime state.
- `state` – A pointer to the value returned for the current security status code:

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the security state of flash was stored to state.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.

`status_t FLASH_SecurityBypass(flash_config_t *config, const uint8_t *backdoorKey)`

Allows users to bypass security with a backdoor key.

If the MCU is in secured state, this function unsecures the MCU by comparing the provided backdoor key with ones in the flash configuration field.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `backdoorKey` – A pointer to the user buffer containing the backdoor key.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FLASH_SetFlexramFunction(flash_config_t *config, ftfx_flexram_func_opt_t option)`

Sets the FlexRAM function command.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `option` – The option used to set the work mode of FlexRAM.

**Return values**

- `kStatus_FTFx_Success` – API was executed successfully; the FlexRAM has been successfully configured as RAM or EEPROM.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t` FLASH\_Swap(*flash\_config\_t* \*config, uint32\_t address, bool isSetEnable)

Swaps the lower half flash with the higher half flash.

**Parameters**

- config – A pointer to the storage for the driver runtime state.
- address – Address used to configure the flash swap function
- isSetEnable – The possible option used to configure the Flash Swap function or check the flash Swap status.

**Return values**

- `kStatus_FTFx_Success` – API was executed successfully; the lower half flash and higher half flash have been swapped.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – Parameter is not aligned with specified baseline.
- `kStatus_FTFx_SwapIndicatorAddressError` – Swap indicator address is invalid.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during command execution.
- `kStatus_FTFx_SwapSystemNotInUninitialized` – Swap system is not in an uninitialized state.

`status_t` FLASH\_IsProtected(*flash\_config\_t* \*config, uint32\_t start, uint32\_t lengthInBytes, *flash\_prot\_state\_t* \*protection\_state)

Returns the protection state of the desired flash area via the pointer passed into the function.

This function retrieves the current flash protect status for a given flash area as determined by the start address and length.

**Parameters**

- config – A pointer to the storage for the driver runtime state.

- `start` – The start address of the desired flash memory to be checked. Must be word-aligned.
- `lengthInBytes` – The length, given in bytes (not words or long-words) to be checked. Must be word-aligned.
- `protection_state` – A pointer to the value returned for the current protection status code for the desired flash area.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the protection state of specified FLASH region was stored to `protection_state`.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – Parameter is not aligned with specified baseline.
- `kStatus_FTFx_AddressError` – The address is out of range.

`status_t` FLASH\_IsExecuteOnly(*flash\_config\_t* \*config, uint32\_t start, uint32\_t lengthInBytes, *flash\_xacc\_state\_t* \*access\_state)

Returns the access state of the desired flash area via the pointer passed into the function.

This function retrieves the current flash access status for a given flash area as determined by the start address and length.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be checked. Must be word-aligned.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be checked. Must be word-aligned.
- `access_state` – A pointer to the value returned for the current access status code for the desired flash area.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the executeOnly state of specified FLASH region was stored to `access_state`.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – The parameter is not aligned to the specified baseline.
- `kStatus_FTFx_AddressError` – The address is out of range.

`status_t` FLASH\_PflashSetProtection(*flash\_config\_t* \*config, *pflash\_prot\_status\_t* \*protectStatus)

Sets the PFlash Protection to the intended protection status.

#### Parameters

- `config` – A pointer to storage for the driver runtime state.
- `protectStatus` – The expected protect status to set to the PFlash protection register. Each bit is corresponding to protection of 1/32(64) of the total PFlash. The least significant bit is corresponding to the lowest address area of PFlash. The most significant bit is corresponding to the highest address area of PFlash. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the specified FLASH region is protected.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_CommandFailure` – Run-time error during command execution.

`status_t` FLASH\_PflashGetProtection(*flash\_config\_t* \*config, *pflash\_prot\_status\_t* \*protectStatus)

Gets the PFlash protection status.

#### Parameters

- config – A pointer to the storage for the driver runtime state.
- protectStatus – Protect status returned by the PFlash IP. Each bit is corresponding to the protection of 1/32(64) of the total PFlash. The least significant bit corresponds to the lowest address area of the PFlash. The most significant bit corresponds to the highest address area of PFlash. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the Protection state was stored to protectStatus;
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.

`status_t` FLASH\_GetProperty(*flash\_config\_t* \*config, *flash\_property\_tag\_t* whichProperty, *uint32\_t* \*value)

Returns the desired flash property.

#### Parameters

- config – A pointer to the storage for the driver runtime state.
- whichProperty – The desired property from the list of properties in enum `flash_property_tag_t`
- value – A pointer to the value returned for the desired flash property.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the flash property was stored to value.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_UnknownProperty` – An unknown property tag.

`status_t` FLASH\_GetCommandState(*void*)

Get previous command status.

This function is used to obtain the execution status of the previous command.

#### Return values

- `kStatus_FTFx_Success` – The previous command is executed successfully.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.



- kStatus\_FTFx\_CommandFailure – Run-time error during the command execution.

FSL\_FLASH\_DRIVER\_VERSION

Flash driver version for SDK.

Version 3.1.3.

FSL\_FLASH\_DRIVER\_VERSION\_ROM

Flash driver version for ROM.

Version 3.0.0.

enum \_flash\_protection\_state

Enumeration for the three possible flash protection levels.

*Values:*

enumerator kFLASH\_ProtectionStateUnprotected

Flash region is not protected.

enumerator kFLASH\_ProtectionStateProtected

Flash region is protected.

enumerator kFLASH\_ProtectionStateMixed

Flash is mixed with protected and unprotected region.

enum \_flash\_execute\_only\_access\_state

Enumeration for the three possible flash execute access levels.

*Values:*

enumerator kFLASH\_AccessStateUnLimited

Flash region is unlimited.

enumerator kFLASH\_AccessStateExecuteOnly

Flash region is execute only.

enumerator kFLASH\_AccessStateMixed

Flash is mixed with unlimited and execute only region.

enum \_flash\_property\_tag

Enumeration for various flash properties.

*Values:*

enumerator kFLASH\_PropertyPflash0SectorSize

Pflash sector size property.

enumerator kFLASH\_PropertyPflash0TotalSize

Pflash total size property.

enumerator kFLASH\_PropertyPflash0BlockSize

Pflash block size property.

enumerator kFLASH\_PropertyPflash0BlockCount

Pflash block count property.

enumerator kFLASH\_PropertyPflash0BlockBaseAddr

Pflash block base address property.

enumerator kFLASH\_PropertyPflash0FacSupport

Pflash fac support property.

enumerator kFLASH\_PropertyPflash0AccessSegmentSize

Pflash access segment size property.

enumerator kFLASH\_PropertyPflash0AccessSegmentCount

Pflash access segment count property.

enumerator kFLASH\_PropertyPflash1SectorSize

Pflash sector size property.

enumerator kFLASH\_PropertyPflash1TotalSize

Pflash total size property.

enumerator kFLASH\_PropertyPflash1BlockSize

Pflash block size property.

enumerator kFLASH\_PropertyPflash1BlockCount

Pflash block count property.

enumerator kFLASH\_PropertyPflash1BlockBaseAddr

Pflash block base address property.

enumerator kFLASH\_PropertyPflash1FacSupport

Pflash fac support property.

enumerator kFLASH\_PropertyPflash1AccessSegmentSize

Pflash access segment size property.

enumerator kFLASH\_PropertyPflash1AccessSegmentCount

Pflash access segment count property.

enumerator kFLASH\_PropertyFlexRamBlockBaseAddr

FlexRam block base address property.

enumerator kFLASH\_PropertyFlexRamTotalSize

FlexRam total size property.

typedef enum *\_flash\_protection\_state* flash\_prot\_state\_t

Enumeration for the three possible flash protection levels.

typedef union *\_pflash\_protection\_status* pflash\_prot\_status\_t

PFlash protection status.

typedef enum *\_flash\_execute\_only\_access\_state* flash\_xacc\_state\_t

Enumeration for the three possible flash execute access levels.

typedef enum *\_flash\_property\_tag* flash\_property\_tag\_t

Enumeration for various flash properties.

typedef struct *\_flash\_config* flash\_config\_t

Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

kStatus\_FLASH\_Success

kFLASH\_ApiEraseKey

union *\_pflash\_protection\_status*

*#include <fsl\_ftfx\_flash.h>* PFlash protection status.

**Public Members**

uint32\_t protl  
 PROT[31:0] .

uint32\_t proth  
 PROT[63:32].

uint8\_t protsl  
 PROTS[7:0] .

uint8\_t protsh  
 PROTS[15:8] .

uint8\_t reserved[2]

struct \_flash\_config

*#include <fsl\_ftfx\_flash.h>* Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

## 2.24 Fftfx FLEXNVM Driver

*status\_t* FLEXNVM\_Init(*flexnvm\_config\_t* \*config)

Initializes the global flash properties structure members.

This function checks and initializes the Flash module for the other Flash APIs.

**Parameters**

- config – Pointer to the storage for the driver runtime state.

**Return values**

- kStatus\_FTFx\_Success – API was executed successfully.
- kStatus\_FTFx\_InvalidArgument – An invalid argument is provided.
- kStatus\_FTFx\_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus\_FTFx\_PartitionStatusUpdateFailure – Failed to update the partition status.

*status\_t* FLEXNVM\_DflashErase(*flexnvm\_config\_t* \*config, uint32\_t start, uint32\_t lengthInBytes, uint32\_t key)

Erases the Dflash sectors encompassed by parameters passed into function.

This function erases the appropriate number of flash sectors based on the desired start address and length.

**Parameters**

- config – The pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words) to be erased. Must be word-aligned.
- key – The value used to validate all flash erase APIs.

**Return values**

- `kStatus_FTFx_Success` – API was executed successfully; the appropriate number of data flash sectors based on the desired start address and length were erased successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – The parameter is not aligned with the specified baseline.
- `kStatus_FTFx_AddressError` – The address is out of range.
- `kStatus_FTFx_EraseKeyError` – The API erase key is invalid.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t` FLEXNVM\_EraseAll(*flexnvm\_config\_t* \*config, uint32\_t key)

Erases entire flexnvm.

#### Parameters

- config – Pointer to the storage for the driver runtime state.
- key – A value used to validate all flash erase APIs.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the entire flexnvm has been erased successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_EraseKeyError` – API erase key is invalid.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during command execution.
- `kStatus_FTFx_PartitionStatusUpdateFailure` – Failed to update the partition status.

`status_t` FLEXNVM\_EraseAllUnsecure(*flexnvm\_config\_t* \*config, uint32\_t key)

Erases the entire flexnvm, including protected sectors.

#### Parameters

- config – Pointer to the storage for the driver runtime state.
- key – A value used to validate all flash erase APIs.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the flexnvm is not in security state.

- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_EraseKeyError` – API erase key is invalid.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during command execution.
- `kStatus_FTFx_PartitionStatusUpdateFailure` – Failed to update the partition status.

`status_t FLEXNVM_DflashProgram(flexnvm_config_t *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)`

Programs flash with data at locations passed in through parameters.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be programmed. Must be word-aligned.
- `src` – A pointer to the source buffer of data that is to be programmed into the flash.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the desired data have been successfully programmed into specified data flash region.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – Parameter is not aligned with the specified baseline.
- `kStatus_FTFx_AddressError` – Address is out of range.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FLEXNVM_DflashProgramSection(flexnvm_config_t *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)`

Programs flash with data at locations passed in through parameters via the Program Section command.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and length.

### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be programmed. Must be word-aligned.
- `src` – A pointer to the source buffer of data that is to be programmed into the flash.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the desired data have been successfully programmed into specified data flash area.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – Parameter is not aligned with specified baseline.
- `kStatus_FTFx_AddressError` – Address is out of range.
- `kStatus_FTFx_SetFlexramAsRamError` – Failed to set flexram as RAM.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during command execution.
- `kStatus_FTFx_RecoverFlexramAsEepromError` – Failed to recover FlexRAM as EEPROM.

`status_t` FLEXNVM\_ProgramPartition(*flexnvm\_config\_t* \*config,  
  *ftfx\_partition\_flexram\_load\_opt\_t* option, *uint32\_t*  
  eepromDataSizeCode, *uint32\_t* flexnvmPartitionCode)

Prepares the FlexNVM block for use as data flash, EEPROM backup, or a combination of both and initializes the FlexRAM.

### Parameters

- `config` – Pointer to storage for the driver runtime state.
- `option` – The option used to set FlexRAM load behavior during reset.
- `eepromDataSizeCode` – Determines the amount of FlexRAM used in each of the available EEPROM subsystems.
- `flexnvmPartitionCode` – Specifies how to split the FlexNVM block between data flash memory and EEPROM backup memory supporting EEPROM functions.

### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the FlexNVM block for use as data flash, EEPROM backup, or a combination of both have been Prepared.
- `kStatus_FTFx_InvalidArgument` – Invalid argument is provided.

- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during command execution.

```
status_t FLEXNVM_ProgramPartition_CSE(flexnvm_config_t *config,
                                       ftfx_partition_flexram_load_opt_t option, uint32_t
                                       eeepromDataSizeCode, uint32_t
                                       flexnvmPartitionCode, uint8_t CSEcKeySize, uint8_t
                                       SFE)
```

Prepares the FlexNVM block for use as data flash, EEPROM backup, or a combination of both and initializes the FlexRAM. This is the CSE enabled version for IP's like FTFC.

#### Parameters

- `config` – Pointer to storage for the driver runtime state.
- `option` – The option used to set FlexRAM load behavior during reset.
- `eeepromDataSizeCode` – Determines the amount of FlexRAM used in each of the available EEPROM subsystems.
- `flexnvmPartitionCode` – Specifies how to split the FlexNVM block between data flash memory and EEPROM backup memory supporting EEPROM functions.
- `CSEcKeySize` – CSEc/SHE key size, see RM for details and possible values
- `SFE` – Security Flag Extension (SFE), see RM for details and possible values

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the FlexNVM block for use as data flash, EEPROM backup, or a combination of both have been Prepared.
- `kStatus_FTFx_InvalidArgument` – Invalid argument is provided.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during command execution.

```
status_t FLEXNVM_ReadResource(flexnvm_config_t *config, uint32_t start, uint8_t *dst, uint32_t
                              lengthInBytes, ftfx_read_resource_opt_t option)
```

Reads the resource with data at locations passed in through parameters.

This function reads the flash memory with the desired location for a given flash area as determined by the start address and length.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.

- `start` – The start address of the desired flash memory to be programmed. Must be word-aligned.
- `dst` – A pointer to the destination buffer of data that is used to store data to be read.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be read. Must be word-aligned.
- `option` – The resource option which indicates which area should be read back.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the data have been read successfully from program flash IFR, data flash IFR space, and the Version ID field
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – Parameter is not aligned with the specified baseline.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FLEXNVM_DflashVerifyErase(flexnvm_config_t *config, uint32_t start, uint32_t lengthInBytes, ftfx_margin_value_t margin)`

Verifies an erasure of the desired flash area at a specified margin level.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- `margin` – Read margin choice.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the specified data flash region is in erased state.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – Parameter is not aligned with specified baseline.
- `kStatus_FTFx_AddressError` – Address is out of range.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.



- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FLEXNVM_VerifyEraseAll(flexnm_config_t *config, ftfx_margin_value_t margin)`

Verifies erasure of the entire flash at a specified margin level.

This function checks whether the flash is erased to the specified read margin level.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `margin` – Read margin choice.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the entire flexnm region is in erased state.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FLEXNVM_DflashVerifyProgram(flexnm_config_t *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, ftfx_margin_value_t margin, uint32_t *failedAddress, uint32_t *failedData)`

Verifies programming of the desired flash area at a specified margin level.

This function verifies the data programmed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be verified. Must be word-aligned.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- `expectedData` – A pointer to the expected data that is to be verified against.
- `margin` – Read margin choice.
- `failedAddress` – A pointer to the returned failing address.
- `failedData` – A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the desired data have been programmed successfully into specified data flash region.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – Parameter is not aligned with specified baseline.
- `kStatus_FTFx_AddressError` – Address is out of range.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t` FLEXNVM\_GetSecurityState(*flexnvm\_config\_t* \*config, *ftfx\_security\_state\_t* \*state)

Returns the security state via the pointer passed into the function.

This function retrieves the current flash security status, including the security enabling state and the backdoor key enabling state.

#### Parameters

- `config` – A pointer to storage for the driver runtime state.
- `state` – A pointer to the value returned for the current security status code:

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the security state of flexnvm was stored to state.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.

`status_t` FLEXNVM\_SecurityBypass(*flexnvm\_config\_t* \*config, `const uint8_t` \*backdoorKey)

Allows users to bypass security with a backdoor key.

If the MCU is in secured state, this function unsecures the MCU by comparing the provided backdoor key with ones in the flash configuration field.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `backdoorKey` – A pointer to the user buffer containing the backdoor key.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

*status\_t* FLEXNVM\_SetFlexramFunction(*flexnvm\_config\_t* \*config, *ftfx\_flexram\_func\_opt\_t* option)

Sets the FlexRAM function command.

#### Parameters

- config – A pointer to the storage for the driver runtime state.
- option – The option used to set the work mode of FlexRAM.

#### Return values

- kStatus\_FTFx\_Success – API was executed successfully; the FlexRAM has been successfully configured as RAM or EEPROM
- kStatus\_FTFx\_InvalidArgument – An invalid argument is provided.
- kStatus\_FTFx\_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus\_FTFx\_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus\_FTFx\_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus\_FTFx\_CommandFailure – Run-time error during the command execution.

*status\_t* FLEXNVM\_DflashSetProtection(*flexnvm\_config\_t* \*config, *uint8\_t* protectStatus)

Sets the DFlash protection to the intended protection status.

#### Parameters

- config – A pointer to the storage for the driver runtime state.
- protectStatus – The expected protect status to set to the DFlash protection register. Each bit corresponds to the protection of the 1/8 of the total DFlash. The least significant bit corresponds to the lowest address area of the DFlash. The most significant bit corresponds to the highest address area of the DFlash. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

#### Return values

- kStatus\_FTFx\_Success – API was executed successfully; the specified DFlash region is protected.
- kStatus\_FTFx\_InvalidArgument – An invalid argument is provided.
- kStatus\_FTFx\_CommandNotSupported – Flash API is not supported.
- kStatus\_FTFx\_CommandFailure – Run-time error during command execution.

*status\_t* FLEXNVM\_DflashGetProtection(*flexnvm\_config\_t* \*config, *uint8\_t* \*protectStatus)

Gets the DFlash protection status.

#### Parameters

- config – A pointer to the storage for the driver runtime state.
- protectStatus – DFlash Protect status returned by the PFlash IP. Each bit corresponds to the protection of the 1/8 of the total DFlash. The least significant bit corresponds to the lowest address area of the DFlash. The most significant bit corresponds to the highest address area of the DFlash, and so on. There are two possible cases as below: 0: this area is protected. 1: this area is unprotected.

**Return values**

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_CommandNotSupported` – Flash API is not supported.

`status_t` FLEXNVM\_EepromSetProtection(*flexnvm\_config\_t* \*config, `uint8_t` protectStatus)

Sets the EEPROM protection to the intended protection status.

**Parameters**

- `config` – A pointer to the storage for the driver runtime state.
- `protectStatus` – The expected protect status to set to the EEPROM protection register. Each bit corresponds to the protection of the 1/8 of the total EEPROM. The least significant bit corresponds to the lowest address area of the EEPROM. The most significant bit corresponds to the highest address area of EEPROM, and so on. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

**Return values**

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_CommandNotSupported` – Flash API is not supported.
- `kStatus_FTFx_CommandFailure` – Run-time error during command execution.

`status_t` FLEXNVM\_EepromGetProtection(*flexnvm\_config\_t* \*config, `uint8_t` \*protectStatus)

Gets the EEPROM protection status.

**Parameters**

- `config` – A pointer to the storage for the driver runtime state.
- `protectStatus` – DFlash Protect status returned by the PFlash IP. Each bit corresponds to the protection of the 1/8 of the total EEPROM. The least significant bit corresponds to the lowest address area of the EEPROM. The most significant bit corresponds to the highest address area of the EEPROM. There are two possible cases as below: 0: this area is protected. 1: this area is unprotected.

**Return values**

- `kStatus_FTFx_Success` – API was executed successfully.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_CommandNotSupported` – Flash API is not supported.

`status_t` FLEXNVM\_GetProperty(*flexnvm\_config\_t* \*config, *flexnvm\_property\_tag\_t* whichProperty, `uint32_t` \*value)

Returns the desired flexnvm property.

**Parameters**

- `config` – A pointer to the storage for the driver runtime state.
- `whichProperty` – The desired property from the list of properties in enum `flexnvm_property_tag_t`
- `value` – A pointer to the value returned for the desired flexnvm property.

**Return values**

- `kStatus_FTFx_Success` – API was executed successfully.

- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_UnknownProperty` – An unknown property tag.

`enum _flexnvm_property_tag`

Enumeration for various flexnvm properties.

*Values:*

enumerator `kFLEXNVM_PropertyDflashSectorSize`

Dflash sector size property.

enumerator `kFLEXNVM_PropertyDflashTotalSize`

Dflash total size property.

enumerator `kFLEXNVM_PropertyDflashBlockSize`

Dflash block size property.

enumerator `kFLEXNVM_PropertyDflashBlockCount`

Dflash block count property.

enumerator `kFLEXNVM_PropertyDflashBlockBaseAddr`

Dflash block base address property.

enumerator `kFLEXNVM_PropertyAliasDflashBlockBaseAddr`

Dflash block base address Alias property.

enumerator `kFLEXNVM_PropertyFlexRamBlockBaseAddr`

FlexRam block base address property.

enumerator `kFLEXNVM_PropertyFlexRamTotalSize`

FlexRam total size property.

enumerator `kFLEXNVM_PropertyEepromTotalSize`

EEPROM total size property.

`typedef enum _flexnvm_property_tag flexnvm_property_tag_t`

Enumeration for various flexnvm properties.

`typedef struct _flexnvm_config flexnvm_config_t`

Flexnvm driver state information.

An instance of this structure is allocated by the user of the Flexnvm driver and passed into each of the driver APIs.

`status_t FLEXNVM_EepromWrite(flexnvm_config_t *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)`

Programs the EEPROM with data at locations passed in through parameters.

This function programs the emulated EEPROM with the desired data for a given flash area as determined by the start address and length.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be programmed. Must be word-aligned.
- `src` – A pointer to the source buffer of data that is to be programmed into the flash.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the desires data have been successfully programed into specified eeprom region.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AddressError` – Address is out of range.
- `kStatus_FTFx_SetFlexramAsEepromError` – Failed to set flexram as eeprom.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_RecoverFlexramAsRamError` – Failed to recover the FlexRAM as RAM.

`struct __flexnvm_config`

*#include <fsl\_ftfx\_flexnvm.h>* Flexnvm driver state information.

An instance of this structure is allocated by the user of the Flexnvm driver and passed into each of the driver APIs.

## 2.25 ftfx utilities

`ALIGN_DOWN(x, a)`

Alignment(down) utility.

`ALIGN_UP(x, a)`

Alignment(up) utility.

`MAKE_VERSION(major, minor, bugfix)`

Constructs the version number for drivers.

`MAKE_STATUS(group, code)`

Constructs a status code value from a group and a code number.

`FOUR_CHAR_CODE(a, b, c, d)`

Constructs the four character code for the Flash driver API key.

`B1P4(b)`

bytes2word utility.

`B1P3(b)`

`B1P2(b)`

`B1P1(b)`

`B2P3(b)`

`B2P2(b)`

`B2P1(b)`

`B3P2(b)`

`B3P1(b)`

`BYTE2WORD_1_3(x, y)`

`BYTE2WORD_2_2(x, y)`

`BYTE2WORD_3_1(x, y)`

BYTE2WORD\_1\_1\_2(x, y, z)

BYTE2WORD\_1\_2\_1(x, y, z)

BYTE2WORD\_2\_1\_1(x, y, z)

BYTE2WORD\_1\_1\_1\_1(x, y, z, w)

## 2.26 GPIO: General-Purpose Input/Output Driver

FSL\_GPIO\_DRIVER\_VERSION

GPIO driver version.

enum \_gpio\_pin\_direction

GPIO direction definition.

*Values:*

enumerator kGPIO\_DigitalInput

Set current pin as digital input

enumerator kGPIO\_DigitalOutput

Set current pin as digital output

enum \_gpio\_checker\_attribute

GPIO checker attribute.

*Values:*

enumerator kGPIO\_UsernonsecureRWUsersecureRWPrivilegedsecureRW

User nonsecure:Read+Write; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO\_UsernonsecureRUsersecureRWPrivilegedsecureRW

User nonsecure:Read; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO\_UsernonsecureNUsersecureRWPrivilegedsecureRW

User nonsecure:None; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO\_UsernonsecureRUsersecureRPrivilegedsecureRW

User nonsecure:Read; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO\_UsernonsecureNUsersecureRPrivilegedsecureRW

User nonsecure:None; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO\_UsernonsecureNUsersecureNPrivilegedsecureRW

User nonsecure:None; User Secure:None; Privileged Secure:Read+Write

enumerator kGPIO\_UsernonsecureNUsersecureNPrivilegedsecureR

User nonsecure:None; User Secure:None; Privileged Secure:Read

enumerator kGPIO\_UsernonsecureNUsersecureNPrivilegedsecureN

User nonsecure:None; User Secure:None; Privileged Secure:None

enumerator kGPIO\_IgnoreAttributeCheck

Ignores the attribute check

typedef enum \_gpio\_pin\_direction gpio\_pin\_direction\_t

GPIO direction definition.

typedef enum \_gpio\_checker\_attribute gpio\_checker\_attribute\_t

GPIO checker attribute.

```
typedef struct _gpio_pin_config gpio_pin_config_t
```

The GPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT\_SetPinConfig().

```
GPIO_FIT_REG(value)
```

```
struct _gpio_pin_config
```

*#include <fsl\_gpio.h>* The GPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT\_SetPinConfig().

### Public Members

*gpio\_pin\_direction\_t* pinDirection

GPIO direction, input or output

uint8\_t outputLogic

Set a default output logic, which has no use in input

## 2.27 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

static inline void GPIO\_PinWrite(GPIO\_Type \*base, uint32\_t pin, uint8\_t output)

Sets the output level of the multiple GPIO pins to the logic 1 or 0.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO pin number
- output – GPIO pin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

static inline void GPIO\_PortSet(GPIO\_Type \*base, uint32\_t mask)

Sets the output level of the multiple GPIO pins to the logic 1.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO\_PortClear(GPIO\_Type \*base, uint32\_t mask)

Sets the output level of the multiple GPIO pins to the logic 0.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO\_PortToggle(GPIO\_Type \*base, uint32\_t mask)

Reverses the current output logic of the multiple GPIO pins.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline uint32\_t GPIO\_PinRead(GPIO\_Type \*base, uint32\_t pin)

Reads the current input value of the GPIO port.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO pin number

**Return values**

GPIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

uint32\_t GPIO\_PortGetInterruptFlags(GPIO\_Type \*base)

Reads the GPIO port interrupt status flag.

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains

set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)

**Return values**

The – current GPIO port interrupt status flag, for example, 0x00010001 means the pin 0 and 17 have the interrupt.

```
void GPIO_PortClearInterruptFlags(GPIO_Type *base, uint32_t mask)
```

Clears multiple GPIO pin interrupt status flags.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
void GPIO_CheckAttributeBytes(GPIO_Type *base, gpio_checker_attribute_t attribute)
```

brief The GPIO module supports a device-specific number of data ports, organized as 32-bit words/8-bit Bytes. Each 32-bit/8-bit data port includes a GACR register, which defines the byte-level attributes required for a successful access to the GPIO programming model. If the GPIO module's GACR register organized as 32-bit words, the attribute controls for the 4 data bytes in the GACR follow a standard little endian data convention.

**Parameters**

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- attribute – GPIO checker attribute

## 2.28 INTMUX: Interrupt Multiplexer Driver

```
void INTMUX_Init(INTMUX_Type *base)
```

Initializes the INTMUX module.

This function enables the clock gate for the specified INTMUX. It then resets all channels, so that no interrupt sources are routed and the logic mode is set to default of kINTMUX\_ChannelLogicOR. Finally, the NVIC vectors for all the INTMUX output channels are enabled.

**Parameters**

- base – INTMUX peripheral base address.

```
void INTMUX_Deinit(INTMUX_Type *base)
```

Deinitializes an INTMUX instance for operation.

The clock gate for the specified INTMUX is disabled and the NVIC vectors for all channels are disabled.

**Parameters**

- base – INTMUX peripheral base address.

```
static inline void INTMUX_ResetChannel(INTMUX_Type *base, uint32_t channel)
```

Resets an INTMUX channel.

Sets all register values in the specified channel to their reset value. This function disables all interrupt sources for the channel.

**Parameters**

- base – INTMUX peripheral base address.

- `channel` – The INTMUX channel number.

```
static inline void INTMUX_SetChannelMode(INTMUX_Type *base, uint32_t channel,  
                                         intmux_channel_logic_mode_t logic)
```

Sets the logic mode for an INTMUX channel.

INTMUX channels can be configured to use one of the two logic modes that control how pending interrupt sources on the channel trigger the output interrupt.

- `kINTMUX_ChannelLogicOR` means any source pending triggers the output interrupt.
- `kINTMUX_ChannelLogicAND` means all selected sources on the channel must be pending before the channel output interrupt triggers.

#### Parameters

- `base` – INTMUX peripheral base address.
- `channel` – The INTMUX channel number.
- `logic` – The INTMUX channel logic mode.

```
static inline void INTMUX_EnableInterrupt(INTMUX_Type *base, uint32_t channel, IRQn_Type  
                                         irq)
```

Enables an interrupt source on an INTMUX channel.

#### Parameters

- `base` – INTMUX peripheral base address.
- `channel` – Index of the INTMUX channel on which the specified interrupt is enabled.
- `irq` – Interrupt to route to the specified INTMUX channel. The interrupt must be an INTMUX source.

```
static inline void INTMUX_DisableInterrupt(INTMUX_Type *base, uint32_t channel, IRQn_Type  
                                         irq)
```

Disables an interrupt source on an INTMUX channel.

#### Parameters

- `base` – INTMUX peripheral base address.
- `channel` – Index of the INTMUX channel on which the specified interrupt is disabled.
- `irq` – Interrupt number. The interrupt must be an INTMUX source.

```
static inline uint32_t INTMUX_GetChannelPendingSources(INTMUX_Type *base, uint32_t  
                                                       channel)
```

Gets INTMUX pending interrupt sources for a specific channel.

#### Parameters

- `base` – INTMUX peripheral base address.
- `channel` – The INTMUX channel number.

#### Returns

The mask of pending interrupt bits. Bit[n] set means INTMUX source n is pending.

FSL\_INTMUX\_DRIVER\_VERSION

```
enum _intmux_channel_logic_mode  
    INTMUX channel logic mode.
```

*Values:*

enumerator kINTMUX\_ChannelLogicOR  
Logic OR all enabled interrupt inputs

enumerator kINTMUX\_ChannelLogicAND  
Logic AND all enabled interrupt inputs

typedef enum *\_intmux\_channel\_logic\_mode* intmux\_channel\_logic\_mode\_t  
INTMUX channel logic mode.

## 2.29 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 USB CDC.

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\_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*.

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 SDSPIC 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\_HAL\_GPIO  
Group number for HAL GPIO status codes.

enumerator `kStatusGroup_HAL_UART`  
Group number for HAL UART status codes.

enumerator `kStatusGroup_HAL_TIMER`  
Group number for HAL TIMER status codes.

enumerator `kStatusGroup_HAL_SPI`  
Group number for HAL SPI status codes.

enumerator `kStatusGroup_HAL_I2C`  
Group number for HAL I2C status codes.

enumerator `kStatusGroup_HAL_FLASH`  
Group number for HAL FLASH status codes.

enumerator `kStatusGroup_HAL_PWM`  
Group number for HAL PWM status codes.

enumerator `kStatusGroup_HAL_RNG`  
Group number for HAL RNG status codes.

enumerator `kStatusGroup_HAL_I2S`  
Group number for HAL I2S status codes.

enumerator `kStatusGroup_HAL_ADC_SENSOR`  
Group number for HAL ADC SENSOR status codes.

enumerator `kStatusGroup_TIMERMANAGER`  
Group number for TiMER MANAGER status codes.

enumerator `kStatusGroup_SERIALMANAGER`  
Group number for SERIAL MANAGER status codes.

enumerator `kStatusGroup_LED`  
Group number for LED status codes.

enumerator `kStatusGroup_BUTTON`  
Group number for BUTTON status codes.

enumerator `kStatusGroup_EXTERN_EEPROM`  
Group number for EXTERN EEPROM status codes.

enumerator `kStatusGroup_SHELL`  
Group number for SHELL status codes.

enumerator `kStatusGroup_MEM_MANAGER`  
Group number for MEM MANAGER status codes.

enumerator `kStatusGroup_LIST`  
Group number for List status codes.

enumerator `kStatusGroup_OSA`  
Group number for OSA status codes.

enumerator `kStatusGroup_COMMON_TASK`  
Group number for Common task status codes.

enumerator `kStatusGroup_MSG`  
Group number for messaging status codes.

enumerator `kStatusGroup_SDK_OCOTP`  
Group number for OCOTP status codes.

enumerator kStatusGroup\_SDK\_FLEXSPINOR  
Group number for FLEXSPINOR status codes.

enumerator kStatusGroup\_CODEC  
Group number for codec status codes.

enumerator kStatusGroup\_ASRC  
Group number for codec status ASRC.

enumerator kStatusGroup\_OTFAD  
Group number for codec status codes.

enumerator kStatusGroup\_SDIO SLV  
Group number for SDIO SLV 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\_QUEUE DSPI  
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_INVALID_IRQ_HANDLER`

Invalid IRQ handler address.

## 2.30 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 returns `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

## 2.31 LLWU: Low-Leakage Wakeup Unit Driver

static inline void LLWU\_GetVersionId(LLWU\_Type \*base, *llwu\_version\_id\_t* \*versionId)

Gets the LLWU version ID.

This function gets the LLWU version ID, including the major version number, the minor version number, and the feature specification number.

### Parameters

- base – LLWU peripheral base address.
- versionId – A pointer to the version ID structure.

static inline void LLWU\_GetParam(LLWU\_Type \*base, *llwu\_param\_t* \*param)

Gets the LLWU parameter.

This function gets the LLWU parameter, including a wakeup pin number, a module number, a DMA number, and a pin filter number.

### Parameters

- base – LLWU peripheral base address.
- param – A pointer to the LLWU parameter structure.

```
void LLWU_SetExternalWakeupPinMode(LLWU_Type *base, uint32_t pinIndex,  
                                   llwu_external_pin_mode_t pinMode)
```

Sets the external input pin source mode.

This function sets the external input pin source mode that is used as a wake up source.

#### Parameters

- base – LLWU peripheral base address.
- pinIndex – A pin index to be enabled as an external wakeup source starting from 1.
- pinMode – A pin configuration mode defined in the llwu\_external\_pin\_modes\_t.

```
bool LLWU_GetExternalWakeupPinFlag(LLWU_Type *base, uint32_t pinIndex)
```

Gets the external wakeup source flag.

This function checks the external pin flag to detect whether the MCU is woken up by the specific pin.

#### Parameters

- base – LLWU peripheral base address.
- pinIndex – A pin index, which starts from 1.

#### Returns

True if the specific pin is a wakeup source.

```
void LLWU_ClearExternalWakeupPinFlag(LLWU_Type *base, uint32_t pinIndex)
```

Clears the external wakeup source flag.

This function clears the external wakeup source flag for a specific pin.

#### Parameters

- base – LLWU peripheral base address.
- pinIndex – A pin index, which starts from 1.

```
static inline void LLWU_EnableInternalModuleInterruptWakup(LLWU_Type *base, uint32_t  
                                                            moduleIndex, bool enable)
```

Enables/disables the internal module source.

This function enables/disables the internal module source mode that is used as a wake up source.

#### Parameters

- base – LLWU peripheral base address.
- moduleIndex – A module index to be enabled as an internal wakeup source starting from 1.
- enable – An enable or a disable setting

```
static inline void LLWU_EnableInternalModuleDmaRequestWakup(LLWU_Type *base, uint32_t  
                                                            moduleIndex, bool enable)
```

Enables/disables the internal module DMA wakeup source.

This function enables/disables the internal DMA that is used as a wake up source.

#### Parameters

- base – LLWU peripheral base address.
- moduleIndex – An internal module index which is used as a DMA request source, starting from 1.



- enable – Enable or disable the DMA request source

```
void LLWU_SetPinFilterMode(LLWU_Type *base, uint32_t filterIndex,  
                          llwu_external_pin_filter_mode_t filterMode)
```

Sets the pin filter configuration.

This function sets the pin filter configuration.

#### Parameters

- base – LLWU peripheral base address.
- filterIndex – A pin filter index used to enable/disable the digital filter, starting from 1.
- filterMode – A filter mode configuration

```
bool LLWU_GetPinFilterFlag(LLWU_Type *base, uint32_t filterIndex)
```

Gets the pin filter configuration.

This function gets the pin filter flag.

#### Parameters

- base – LLWU peripheral base address.
- filterIndex – A pin filter index, which starts from 1.

#### Returns

True if the flag is a source of the existing low-leakage power mode.

```
void LLWU_ClearPinFilterFlag(LLWU_Type *base, uint32_t filterIndex)
```

Clears the pin filter configuration.

This function clears the pin filter flag.

#### Parameters

- base – LLWU peripheral base address.
- filterIndex – A pin filter index to clear the flag, starting from 1.

```
void LLWU_SetResetPinMode(LLWU_Type *base, bool pinEnable, bool pinFilterEnable)
```

Sets the reset pin mode.

This function determines how the reset pin is used as a low leakage mode exit source.

#### Parameters

- base – LLWU peripheral base address.
- pinEnable – Enable reset the pin filter
- pinFilterEnable – Specify whether the pin filter is enabled in Low-Leakage power mode.

```
FSL_LLWU_DRIVER_VERSION
```

LLWU driver version.

```
enum _llwu_external_pin_mode
```

External input pin control modes.

*Values:*

```
enumerator kLLWU_ExternalPinDisable
```

Pin disabled as a wakeup input.

```
enumerator kLLWU_ExternalPinRisingEdge
```

Pin enabled with the rising edge detection.

enumerator `kLLWU_ExternalPinFallingEdge`  
Pin enabled with the falling edge detection.

enumerator `kLLWU_ExternalPinAnyEdge`  
Pin enabled with any change detection.

enum `_llwu_pin_filter_mode`  
Digital filter control modes.

*Values:*

enumerator `kLLWU_PinFilterDisable`  
Filter disabled.

enumerator `kLLWU_PinFilterRisingEdge`  
Filter positive edge detection.

enumerator `kLLWU_PinFilterFallingEdge`  
Filter negative edge detection.

enumerator `kLLWU_PinFilterAnyEdge`  
Filter any edge detection.

typedef enum `_llwu_external_pin_mode` `llwu_external_pin_mode_t`  
External input pin control modes.

typedef enum `_llwu_pin_filter_mode` `llwu_pin_filter_mode_t`  
Digital filter control modes.

typedef struct `_llwu_version_id` `llwu_version_id_t`  
IP version ID definition.

typedef struct `_llwu_param` `llwu_param_t`  
IP parameter definition.

typedef struct `_llwu_external_pin_filter_mode` `llwu_external_pin_filter_mode_t`  
An external input pin filter control structure.

`LLWU_REG_VAL(x)`

struct `_llwu_version_id`  
*#include <fsl\_llwu.h>* IP version ID definition.

### Public Members

`uint16_t` `feature`  
A feature specification number.

`uint8_t` `minor`  
The minor version number.

`uint8_t` `major`  
The major version number.

struct `_llwu_param`  
*#include <fsl\_llwu.h>* IP parameter definition.

**Public Members**

uint8\_t filters

A number of the pin filter.

uint8\_t dmas

A number of the wakeup DMA.

uint8\_t modules

A number of the wakeup module.

uint8\_t pins

A number of the wake up pin.

struct \_llwu\_external\_pin\_filter\_mode

*#include <fsl\_llwu.h>* An external input pin filter control structure.

**Public Members**

uint32\_t pinIndex

A pin number

llwu\_pin\_filter\_mode\_t filterMode

Filter mode

## 2.32 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.

## 2.33 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.

## 2.34 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.

## 2.35 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.

## 2.36 LPIT: Low-Power Interrupt Timer

enum \_lpit\_chnl  
List of LPIT channels.

---

**Note:** Actual number of available channels is SoC-dependent

---

*Values:*

enumerator kLPIT\_Chnl\_0  
LPIT channel number 0

enumerator kLPIT\_Chnl\_1  
LPIT channel number 1

enumerator kLPIT\_Chnl\_2  
LPIT channel number 2

enumerator kLPIT\_Chnl\_3  
LPIT channel number 3

enum \_lpit\_timer\_modes

Mode options available for the LPIT timer.

*Values:*

enumerator kLPIT\_PeriodicCounter  
Use the all 32-bits, counter loads and decrements to zero

enumerator kLPIT\_DualPeriodicCounter  
Counter loads, lower 16-bits decrement to zero, then upper 16-bits decrement

enumerator kLPIT\_TriggerAccumulator  
Counter loads on first trigger and decrements on each trigger

enumerator kLPIT\_InputCapture  
Counter loads with 0xFFFFFFFF, decrements to zero. It stores the inverse of the current value when a input trigger is detected

enum \_lpit\_trigger\_select

Trigger options available.

This is used for both internal and external trigger sources. The actual trigger options available is SoC-specific, user should refer to the reference manual.

*Values:*

enumerator kLPIT\_Trigger\_TimerChn0  
Channel 0 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn1  
Channel 1 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn2  
Channel 2 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn3  
Channel 3 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn4  
Channel 4 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn5  
Channel 5 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn6  
Channel 6 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn7  
Channel 7 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn8  
Channel 8 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn9  
Channel 9 is selected as a trigger source



enumerator kLPIT\_Trigger\_TimerChn10

Channel 10 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn11

Channel 11 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn12

Channel 12 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn13

Channel 13 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn14

Channel 14 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn15

Channel 15 is selected as a trigger source

enum \_lpit\_trigger\_source

Trigger source options available.

*Values:*

enumerator kLPIT\_TriggerSource\_External

Use external trigger input

enumerator kLPIT\_TriggerSource\_Internal

Use internal trigger

enum \_lpit\_interrupt\_enable

List of LPIT interrupts.

---

**Note:** Number of timer channels are SoC-specific. See the SoC Reference Manual.

---

*Values:*

enumerator kLPIT\_Channel0TimerInterruptEnable

Channel 0 Timer interrupt

enumerator kLPIT\_Channel1TimerInterruptEnable

Channel 1 Timer interrupt

enumerator kLPIT\_Channel2TimerInterruptEnable

Channel 2 Timer interrupt

enumerator kLPIT\_Channel3TimerInterruptEnable

Channel 3 Timer interrupt

enum \_lpit\_status\_flags

List of LPIT status flags.

---

**Note:** Number of timer channels are SoC-specific. See the SoC Reference Manual.

---

*Values:*

enumerator kLPIT\_Channel0TimerFlag

Channel 0 Timer interrupt flag

enumerator kLPIT\_Channel1TimerFlag

Channel 1 Timer interrupt flag

enumerator kLPIT\_Channel2TimerFlag

Channel 2 Timer interrupt flag

enumerator kLPIT\_Channel3TimerFlag

Channel 3 Timer interrupt flag

typedef enum *\_lpit\_chnl* lpit\_chnl\_t

List of LPIT channels.

---

**Note:** Actual number of available channels is SoC-dependent

---

typedef enum *\_lpit\_timer\_modes* lpit\_timer\_modes\_t

Mode options available for the LPIT timer.

typedef enum *\_lpit\_trigger\_select* lpit\_trigger\_select\_t

Trigger options available.

This is used for both internal and external trigger sources. The actual trigger options available is SoC-specific, user should refer to the reference manual.

typedef enum *\_lpit\_trigger\_source* lpit\_trigger\_source\_t

Trigger source options available.

typedef enum *\_lpit\_interrupt\_enable* lpit\_interrupt\_enable\_t

List of LPIT interrupts.

---

**Note:** Number of timer channels are SoC-specific. See the SoC Reference Manual.

---

typedef enum *\_lpit\_status\_flags* lpit\_status\_flags\_t

List of LPIT status flags.

---

**Note:** Number of timer channels are SoC-specific. See the SoC Reference Manual.

---

typedef struct *\_lpit\_chnl\_params* lpit\_chnl\_params\_t

Structure to configure the channel timer.

typedef struct *\_lpit\_config* lpit\_config\_t

LPIT configuration structure.

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

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

FSL\_LPIT\_DRIVER\_VERSION

Version 2.1.1

LPIT\_RESET\_STATE\_DELAY

Delay used in LPIT\_Reset.

The macro value should be larger than  $4 * \text{core clock} / \text{LPIT peripheral clock}$ .

void LPIT\_Init(LPIT\_Type \*base, const *lpit\_config\_t* \*config)

Ungates the LPIT clock and configures the peripheral for a basic operation.

This function issues a software reset to reset all channels and registers except the Module Control register.

---

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

---

### Parameters

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

`void LPIT_Deinit(LPIT_Type *base)`

Disables the module and gates the LPIT clock.

### Parameters

- `base` – LPIT peripheral base address.

`void LPIT_GetDefaultConfig(lpit_config_t *config)`

Fills in the LPIT configuration structure with default settings.

The default values are:

```
config->enableRunInDebug = false;
config->enableRunInDoze = false;
```

### Parameters

- `config` – Pointer to the user configuration structure.

`status_t LPIT_SetupChannel(LPIT_Type *base, lpit_chnl_t channel, const lpit_chnl_params_t *chnlSetup)`

Sets up an LPIT channel based on the user's preference.

This function sets up the operation mode to one of the options available in the enumeration `lpit_timer_modes_t`. It sets the trigger source as either internal or external, trigger selection and the timers behaviour when a timeout occurs. It also chains the timer if a prior timer if requested by the user.

### Parameters

- `base` – LPIT peripheral base address.
- `channel` – Channel that is being configured.
- `chnlSetup` – Configuration parameters.

`static inline void LPIT_EnableInterrupts(LPIT_Type *base, uint32_t mask)`

Enables the selected PIT interrupts.

### Parameters

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

`static inline void LPIT_DisableInterrupts(LPIT_Type *base, uint32_t mask)`

Disables the selected PIT interrupts.

### Parameters

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

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

Gets the enabled LPIT interrupts.

**Parameters**

- base – LPIT peripheral base address.

**Returns**

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

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

Gets the LPIT status flags.

**Parameters**

- base – LPIT peripheral base address.

**Returns**

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

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

Clears the LPIT status flags.

**Parameters**

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

static inline void LPIT\_SetTimerPeriod(LPIT\_Type \*base, *lpit\_chnl\_t* channel, uint32\_t ticks)

Sets the timer period in units of count.

Timers begin counting down from the value set by this function until it reaches 0, at which point it generates an interrupt and loads this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

---

**Note:** User can call the utility macros provided in `fsl_common.h` to convert to ticks.

---

**Parameters**

- base – LPIT peripheral base address.
- channel – Timer channel number.
- ticks – Timer period in units of ticks.

static inline void LPIT\_SetTimerValue(LPIT\_Type \*base, *lpit\_chnl\_t* channel, uint32\_t ticks)

Sets the timer period in units of count.

In the Dual 16-bit Periodic Counter mode, the counter will load and then the lower 16-bits will decrement down to zero, which will assert the output pre-trigger. The upper 16-bits will then decrement down to zero, which will negate the output pre-trigger and set the timer interrupt flag.

---

**Note:** Set TVAL register to 0 or 1 is invalid in compare mode.

---

**Parameters**

- base – LPIT peripheral base address.
- channel – Timer channel number.

- ticks – Timer period in units of ticks.

static inline uint32\_t LPIT\_GetCurrentTimerCount(LPIT\_Type \*base, *lpit\_chnl\_t* channel)

Reads the current timer counting value.

This function returns the real-time timer counting value, in a range from 0 to a timer period.

---

**Note:** User can call the utility macros provided in `fsl_common.h` to convert ticks to microseconds or milliseconds.

---

#### Parameters

- base – LPIT peripheral base address.
- channel – Timer channel number.

#### Returns

Current timer counting value in ticks.

static inline void LPIT\_StartTimer(LPIT\_Type \*base, *lpit\_chnl\_t* channel)

Starts the timer counting.

After calling this function, timers load the period value and count down to 0. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

#### Parameters

- base – LPIT peripheral base address.
- channel – Timer channel number.

static inline void LPIT\_StopTimer(LPIT\_Type \*base, *lpit\_chnl\_t* channel)

Stops the timer counting.

#### Parameters

- base – LPIT peripheral base address.
- channel – Timer channel number.

static void LPIT\_ResetStateDelay(void)

Short wait for LPIT state reset.

After clear or set LPIT\_EN, there should be delay longer than 4 LPIT functional clock.

#### Parameters

- count – Delay count.

static inline void LPIT\_Reset(LPIT\_Type \*base)

Performs a software reset on the LPIT module.

This resets all channels and registers except the Module Control Register.

#### Parameters

- base – LPIT peripheral base address.

struct *\_\_lpit\_chnl\_params*

*#include <fsl\_lpit.h>* Structure to configure the channel timer.

#### Public Members

bool chainChannel

true: Timer chained to previous timer; false: Timer not chained

*lpit\_timer\_modes\_t* timerMode

Timers mode of operation.

*lpit\_trigger\_select\_t* triggerSelect

Trigger selection for the timer

*lpit\_trigger\_source\_t* triggerSource

Decides if we use external or internal trigger.

bool enableReloadOnTrigger

true: Timer reloads when a trigger is detected; false: No effect

bool enableStopOnTimeout

true: Timer will stop after timeout; false: does not stop after timeout

bool enableStartOnTrigger

true: Timer starts when a trigger is detected; false: decrement immediately

struct *\_lpit\_config*

*#include <fsl\_lpit.h>* LPIT configuration structure.

This structure holds the configuration settings for the LPIT peripheral. To initialize this structure to reasonable defaults, call the LPIT\_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 enableRunInDebug

true: Timers run in debug mode; false: Timers stop in debug mode

bool enableRunInDoze

true: Timers run in doze mode; false: Timers stop in doze mode

## 2.37 LPSPI: Low Power Serial Peripheral Interface

### 2.38 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(LPSPI_Type *base, lpspi_master_handle_t *handle,  
                                         lpspi_transfer_t *transfer)
```

LPSPI master transfer data using an interrupt method.

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

#### Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_master_handle_t` structure which stores the transfer state.

- transfer – pointer to `lpspi_transfer_t` structure.

**Returns**

status of `status_t`.

`status_t` LPSPI\_MasterTransferGetCount(LPSPI\_Type \*base, *lpspi\_master\_handle\_t* \*handle, size\_t \*count)

Gets the master transfer remaining bytes.

This function gets the master transfer remaining bytes.

**Parameters**

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_master_handle_t` structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

**Returns**

status of `status_t`.

void LPSPI\_MasterTransferAbort(LPSPI\_Type \*base, *lpspi\_master\_handle\_t* \*handle)

LPSPI master abort transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

**Parameters**

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_master_handle_t` structure which stores the transfer state.

void LPSPI\_MasterTransferHandleIRQ(LPSPI\_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 kLPSPI_ClockPolarityActiveHigh
        CPOL=0. Active-high LPSPI clock (idles low)
    enumerator kLPSPI_ClockPolarityActiveLow
        CPOL=1. Active-low LPSPI clock (idles high)
enum __lpspi_clock_phase
    LPSPI clock phase configuration.
    Values:
    enumerator kLPSPI_ClockPhaseFirstEdge
        CPHA=0. Data is captured on the leading edge of the SCK and changed on the following
        edge.
    enumerator kLPSPI_ClockPhaseSecondEdge
        CPHA=1. Data is changed on the leading edge of the SCK and captured on the following
        edge.
```

enum `_lpspi_shift_direction`

LPSPi data shifter direction options.

*Values:*

enumerator `kLPSPi_MsbFirst`

Data transfers start with most significant bit.

enumerator `kLPSPi_LsbFirst`

Data transfers start with least significant bit.

enum `_lpspi_host_request_select`

LPSPi Host Request select configuration.

*Values:*

enumerator `kLPSPi_HostReqExtPin`

Host Request is an ext pin.

enumerator `kLPSPi_HostReqInternalTrigger`

Host Request is an internal trigger.

enum `_lpspi_match_config`

LPSPi Match configuration options.

*Values:*

enumerator `kLPSPi_MatchDisabled`

LPSPi Match Disabled.

enumerator `kLPSPi_1stWordEqualsM0orM1`

LPSPi Match Enabled.

enumerator `kLPSPi_AnyWordEqualsM0orM1`

LPSPi Match Enabled.

enumerator `kLPSPi_1stWordEqualsM0and2ndWordEqualsM1`

LPSPi Match Enabled.

enumerator `kLPSPi_AnyWordEqualsM0andNxtWordEqualsM1`

LPSPi Match Enabled.

enumerator `kLPSPi_1stWordAndM1EqualsM0andM1`

LPSPi Match Enabled.

enumerator `kLPSPi_AnyWordAndM1EqualsM0andM1`

LPSPi Match Enabled.

enum `_lpspi_pin_config`

LPSPi pin (SDO and SDI) configuration.

*Values:*

enumerator `kLPSPi_SdiInSdoOut`

LPSPi SDI input, SDO output.

enumerator `kLPSPi_SdiInSdiOut`

LPSPi SDI input, SDI output.

enumerator `kLPSPi_SdoInSdoOut`

LPSPi SDO input, SDO output.

enumerator `kLPSPi_SdoInSdiOut`

LPSPi SDO input, SDI output.

enum \_lpspi\_data\_out\_config

LPSPI data output configuration.

*Values:*

enumerator kLpspiDataOutRetained

Data out retains last value when chip select is de-asserted

enumerator kLpspiDataOutTristate

Data out is tristated when chip select is de-asserted

enum \_lpspi\_pcs\_function\_config

LPSPI cs function configuration.

*Values:*

enumerator kLPSPi\_PcsAsCs

PCS pin select as cs function

enumerator kLPSPi\_PcsAsData

PCS pin select as data function

enum \_lpspi\_transfer\_width

LPSPI transfer width configuration.

*Values:*

enumerator kLPSPi\_SingleBitXfer

1-bit shift at a time, data out on SDO, in on SDI (normal mode)

enumerator kLPSPi\_TwoBitXfer

2-bits shift out on SDO/SDI and in on SDO/SDI

enumerator kLPSPi\_FourBitXfer

4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

enum \_lpspi\_delay\_type

LPSPI delay type selection.

*Values:*

enumerator kLPSPi\_PcsToSck

PCS-to-SCK delay.

enumerator kLPSPi\_LastSckToPcs

Last SCK edge to PCS delay.

enumerator kLPSPi\_BetweenTransfer

Delay between transfers.

enum \_lpspi\_transfer\_config\_flag\_for\_master

Use this enumeration for LPSPi master transfer configFlags.

*Values:*

enumerator kLPSPi\_MasterPcs0

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS0 signal

enumerator kLPSPi\_MasterPcs1

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS1 signal

enumerator kLPSPi\_MasterPcs2

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS2 signal

enumerator kLPSPi\_MasterPcs3

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS3 signal

enumerator kLPSPi\_MasterWidth1

LPSPi master width shift macro, internal used LPSPi master transfer 1bit

enumerator kLPSPi\_MasterWidth2

LPSPi master width shift macro, internal used LPSPi master transfer 2bit

enumerator kLPSPi\_MasterWidth4

LPSPi master width shift macro, internal used LPSPi master transfer 4bit

enumerator kLPSPi\_MasterPcsContinuous

Is PCS signal continuous

enumerator kLPSPi\_MasterByteSwap

Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set `lpspi_shift_direction_t` to MSB).

- i. If you set `bitPerFrame = 8` , no matter the `kLPSPi_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_pcs_function_config` `lpspi_pcs_function_config_t`

LPSPI cs function configuration.

typedef enum `_lpspi_transfer_width` `lpspi_transfer_width_t`

LPSPI transfer width configuration.

typedef enum `_lpspi_delay_type` `lpspi_delay_type_t`

LPSPI delay type selection.

typedef struct `_lpspi_master_config` `lpspi_master_config_t`

LPSPI master configuration structure.

typedef struct `_lpspi_slave_config` `lpspi_slave_config_t`

LPSPI slave configuration structure.

typedef struct `_lpspi_master_handle` `lpspi_master_handle_t`

Forward declaration of the `_lpspi_master_handle` typedefs.

```
typedef struct _lpspi_slave_handle lpspi_slave_handle_t
```

Forward declaration of the *\_lpspi\_slave\_handle* typedefs.

```
typedef void (*lpspi_master_transfer_callback_t)(LPSPI_Type *base, lpspi_master_handle_t *handle, status_t status, void *userData)
```

Master completion callback function pointer type.

**Param base**

LPSPI peripheral address.

**Param handle**

Pointer to the handle for the LPSPI master.

**Param status**

Success or error code describing whether the transfer is completed.

**Param userData**

Arbitrary pointer-dataSized value passed from the application.

```
typedef void (*lpspi_slave_transfer_callback_t)(LPSPI_Type *base, lpspi_slave_handle_t *handle, status_t status, void *userData)
```

Slave completion callback function pointer type.

**Param base**

LPSPI peripheral address.

**Param handle**

Pointer to the handle for the LPSPI slave.

**Param status**

Success or error code describing whether the transfer is completed.

**Param userData**

Arbitrary pointer-dataSized value passed from the application.

```
typedef struct _lpspi_transfer lpspi_transfer_t
```

LPSPI master/slave transfer structure.

```
volatile uint8_t g_lpspiDummyData[]
```

Global variable for dummy data value setting.

```
LPSPI_DUMMY_DATA
```

LPSPI dummy data if no Tx data.

Dummy data used for tx if there is not txData.

```
SPI_RETRY_TIMES
```

Retry times for waiting flag.

```
LPSPI_MASTER_PCS_SHIFT
```

LPSPI master PCS shift macro , internal used.

```
LPSPI_MASTER_PCS_MASK
```

LPSPI master PCS shift macro , internal used.

```
LPSPI_MASTER_WIDTH_SHIFT
```

LPSPI master width shift macro, internal used

```
LPSPI_MASTER_WIDTH_MASK
```

LPSPI master width shift mask, internal used

```
LPSPI_SLAVE_PCS_SHIFT
```

LPSPI slave PCS shift macro , internal used.



LPSPI\_SLAVE\_PCS\_MASK

LPSPI slave PCS shift macro , internal used.

struct \_\_lpspi\_master\_config

*#include <fsl\_lpspi.h>* LPSPI master configuration structure.

### Public Members

uint32\_t baudRate

Baud Rate for LPSPI.

uint32\_t bitsPerFrame

Bits per frame, minimum 8, maximum 4096.

*lpspi\_clock\_polarity\_t* cpol

Clock polarity.

*lpspi\_clock\_phase\_t* cpha

Clock phase.

*lpspi\_shift\_direction\_t* direction

MSB or LSB data shift direction.

uint32\_t pcsToSckDelayInNanoSec

PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32\_t lastSckToPcsDelayInNanoSec

Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32\_t betweenTransferDelayInNanoSec

After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

*lpspi\_which\_pcs\_t* whichPcs

Desired Peripheral Chip Select (PCS).

*lpspi\_pcs\_polarity\_config\_t* pcsActiveHighOrLow

Desired PCS active high or low

*lpspi\_pin\_config\_t* pinCfg

Configures which pins are used for input and output data during single bit transfers.

*lpspi\_pcs\_function\_config\_t* pcsFunc

Configures cs pins function.

*lpspi\_data\_out\_config\_t* dataOutConfig

Configures if the output data is tristated between accesses (LPSPI\_PCS is negated).

bool enableInputDelay

Enable master to sample the input data on a delayed SCK. This can help improve slave setup time. Refer to device data sheet for specific time length.

struct \_\_lpspi\_slave\_config

*#include <fsl\_lpspi.h>* LPSPI slave configuration structure.

**Public Members**

uint32\_t bitsPerFrame

Bits per frame, minimum 8, maximum 4096.

lpspi\_clock\_polarity\_t cpol

Clock polarity.

lpspi\_clock\_phase\_t cpha

Clock phase.

lpspi\_shift\_direction\_t direction

MSB or LSB data shift direction.

lpspi\_which\_pcs\_t whichPcs

Desired Peripheral Chip Select (pcs)

lpspi\_pcs\_polarity\_config\_t pcsActiveHighOrLow

Desired PCS active high or low

lpspi\_pin\_config\_t pinCfg

Configures which pins are used for input and output data during single bit transfers.

lpspi\_data\_out\_config\_t dataOutConfig

Configures if the output data is tristated between accesses (LPSPi\_PCS is negated).

struct \_lpspi\_transfer

#include <fsl\_lpspi.h> LPSPi master/slave transfer structure.

**Public Members**

const uint8\_t \*txData

Send buffer.

uint8\_t \*rxData

Receive buffer.

volatile size\_t dataSize

Transfer bytes.

uint32\_t configFlags

Transfer transfer configuration flags. Set from \_lpspi\_transfer\_config\_flag\_for\_master if the transfer is used for master or \_lpspi\_transfer\_config\_flag\_for\_slave enumeration if the transfer is used for slave.

struct \_lpspi\_master\_handle

#include <fsl\_lpspi.h> LPSPi master transfer handle structure used for transactional API.

**Public Members**

volatile bool isPcsContinuous

Is PCS continuous in transfer.

volatile bool writeTcrInIsr

A flag that whether should write TCR in ISR.

volatile bool isByteSwap

A flag that whether should byte swap.

`volatile bool` `isTxMask`  
A flag that whether TCR[TXMSK] is set.

`volatile uint16_t` `bytesPerFrame`  
Number of bytes in each frame

`volatile 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.

## 2.39 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(LPSPIType \*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(LPSPIType \*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(LPSPIType \*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

## 2.40 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, which should be equal or greater than 1.

```
static inline uint32_t LPTMR_GetCurrentTimerCount(LPTMR_Type *base)
```

Reads the current timer counting value.

This function returns the real-time timer counting value in a range from 0 to a timer period.

---

**Note:** Call the utility macros provided in the `fsl_common.h` to convert ticks to usec or msec.

---

**Parameters**

- base – LPTMR peripheral base address

**Returns**

The current counter value in ticks

```
static inline void LPTMR_StartTimer(LPTMR_Type *base)
```

Starts the timer.

After calling this function, the timer counts up to the CMR register value. Each time the timer reaches the CMR value and then increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

**Parameters**

- base – LPTMR peripheral base address

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

Stops the timer.

This function stops the timer and resets the timer's counter register.

**Parameters**

- base – LPTMR peripheral base address

FSL\_LPTMR\_DRIVER\_VERSION

Driver Version

enum \_lptmr\_pin\_select

LPTMR pin selection used in pulse counter mode.

*Values:*

enumerator kLPTMR\_PinSelectInput\_0

Pulse counter input 0 is selected

enumerator kLPTMR\_PinSelectInput\_1

Pulse counter input 1 is selected

enumerator kLPTMR\_PinSelectInput\_2

Pulse counter input 2 is selected

enumerator kLPTMR\_PinSelectInput\_3

Pulse counter input 3 is selected

enum \_lptmr\_pin\_polarity

LPTMR pin polarity used in pulse counter mode.

*Values:*

enumerator kLPTMR\_PinPolarityActiveHigh

Pulse Counter input source is active-high

enumerator kLPTMR\_PinPolarityActiveLow

Pulse Counter input source is active-low

enum \_lptmr\_timer\_mode

LPTMR timer mode selection.

*Values:*

enumerator kLPTMR\_TimerModeTimeCounter

Time Counter mode

enumerator kLPTMR\_TimerModePulseCounter

Pulse Counter mode

enum \_lptmr\_prescaler\_glitch\_value

LPTMR prescaler/glitch filter values.

*Values:*

enumerator kLPTMR\_Prescale\_Glitch\_0

Prescaler divide 2, glitch filter does not support this setting

enumerator kLPTMR\_Prescale\_Glitch\_1

Prescaler divide 4, glitch filter 2

enumerator kLPTMR\_Prescale\_Glitch\_2

Prescaler divide 8, glitch filter 4

enumerator kLPTMR\_Prescale\_Glitch\_3  
Prescaler divide 16, glitch filter 8

enumerator kLPTMR\_Prescale\_Glitch\_4  
Prescaler divide 32, glitch filter 16

enumerator kLPTMR\_Prescale\_Glitch\_5  
Prescaler divide 64, glitch filter 32

enumerator kLPTMR\_Prescale\_Glitch\_6  
Prescaler divide 128, glitch filter 64

enumerator kLPTMR\_Prescale\_Glitch\_7  
Prescaler divide 256, glitch filter 128

enumerator kLPTMR\_Prescale\_Glitch\_8  
Prescaler divide 512, glitch filter 256

enumerator kLPTMR\_Prescale\_Glitch\_9  
Prescaler divide 1024, glitch filter 512

enumerator kLPTMR\_Prescale\_Glitch\_10  
Prescaler divide 2048 glitch filter 1024

enumerator kLPTMR\_Prescale\_Glitch\_11  
Prescaler divide 4096, glitch filter 2048

enumerator kLPTMR\_Prescale\_Glitch\_12  
Prescaler divide 8192, glitch filter 4096

enumerator kLPTMR\_Prescale\_Glitch\_13  
Prescaler divide 16384, glitch filter 8192

enumerator kLPTMR\_Prescale\_Glitch\_14  
Prescaler divide 32768, glitch filter 16384

enumerator kLPTMR\_Prescale\_Glitch\_15  
Prescaler divide 65536, glitch filter 32768

enum \_lptmr\_prescaler\_clock\_select  
LPTMR prescaler/glitch filter clock select.

---

**Note:** Clock connections are SoC-specific

---

*Values:*

enumerator kLPTMR\_PrescalerClock\_0  
Prescaler/glitch filter clock 0 selected.

enumerator kLPTMR\_PrescalerClock\_1  
Prescaler/glitch filter clock 1 selected.

enumerator kLPTMR\_PrescalerClock\_2  
Prescaler/glitch filter clock 2 selected.

enumerator kLPTMR\_PrescalerClock\_3  
Prescaler/glitch filter clock 3 selected.

enum \_lptmr\_interrupt\_enable  
List of the LPTMR interrupts.

*Values:*

enumerator kLPTMR\_TimerInterruptEnable

Timer interrupt enable

enum \_lptmr\_status\_flags

List of the LPTMR status flags.

*Values:*

enumerator kLPTMR\_TimerCompareFlag

Timer compare flag

typedef enum \_lptmr\_pin\_select lptmr\_pin\_select\_t

LPTMR pin selection used in pulse counter mode.

typedef enum \_lptmr\_pin\_polarity lptmr\_pin\_polarity\_t

LPTMR pin polarity used in pulse counter mode.

typedef enum \_lptmr\_timer\_mode lptmr\_timer\_mode\_t

LPTMR timer mode selection.

typedef enum \_lptmr\_prescaler\_glitch\_value lptmr\_prescaler\_glitch\_value\_t

LPTMR prescaler/glitch filter values.

typedef enum \_lptmr\_prescaler\_clock\_select lptmr\_prescaler\_clock\_select\_t

LPTMR prescaler/glitch filter clock select.

---

**Note:** Clock connections are SoC-specific

---

typedef enum \_lptmr\_interrupt\_enable lptmr\_interrupt\_enable\_t

List of the LPTMR interrupts.

typedef enum \_lptmr\_status\_flags lptmr\_status\_flags\_t

List of the LPTMR status flags.

typedef struct \_lptmr\_config lptmr\_config\_t

LPTMR config structure.

This structure holds the configuration settings for the LPTMR peripheral. To initialize this structure to reasonable defaults, call the LPTMR\_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration struct can be made constant so it resides in flash.

static inline void LPTMR\_EnableTimerDMA(LPTMR\_Type \*base, bool enable)

Enable or disable timer DMA request.

#### 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

## 2.41 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

### 2.42 LPUART Driver

**static inline void** LPUART\_SoftwareReset(LPUART\_Type \*base)

Resets the LPUART using software.

This function resets all internal logic and registers except the Global Register. Remains set until cleared by software.

**Parameters**

- base – LPUART peripheral base address.

**status\_t** LPUART\_Init(LPUART\_Type \*base, const *lpuart\_config\_t* \*config, uint32\_t srcClock\_Hz)

Initializes an LPUART instance with the user configuration structure and the peripheral clock.

This function configures the LPUART module with user-defined settings. Call the LPUART\_GetDefaultConfig() function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
lpuart_config_t lpuartConfig;
lpuartConfig.baudRate_Bps = 115200U;
lpuartConfig.parityMode = kLPUART_ParityDisabled;
lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
lpuartConfig.isMsb = false;
lpuartConfig.stopBitCount = kLPUART_OneStopBit;
lpuartConfig.txFifoWatermark = 0;
lpuartConfig.rxFifoWatermark = 1;
LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
```

**Parameters**

- `base` – LPUART peripheral base address.
- `config` – Pointer to a user-defined configuration structure.
- `srcClock_Hz` – LPUART clock source frequency in HZ.

#### Return values

- `kStatus_LPUART_BaudrateNotSupport` – Baudrate is not support in current clock source.
- `kStatus_Success` – LPUART initialize succeed

`void LPUART_Deinit(LPUART_Type *base)`

Deinitializes a LPUART instance.

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

#### Parameters

- `base` – LPUART peripheral base address.

`void LPUART_GetDefaultConfig(lpuart_config_t *config)`

Gets the default configuration structure.

This function initializes the LPUART configuration structure to a default value. The default values are: `lpuartConfig->baudRate_Bps = 115200U`; `lpuartConfig->parityMode = kLPUART_ParityDisabled`; `lpuartConfig->dataBitsCount = kLPUART_EightDataBits`; `lpuartConfig->isMsb = false`; `lpuartConfig->stopBitCount = kLPUART_OneStopBit`; `lpuartConfig->txFifoWatermark = 0`; `lpuartConfig->rxFifoWatermark = 1`; `lpuartConfig->rxIdleType = kLPUART_IdleTypeStartBit`; `lpuartConfig->rxIdleConfig = kLPUART_IdleCharacter1`; `lpuartConfig->enableTx = false`; `lpuartConfig->enableRx = false`;

#### Parameters

- `config` – Pointer to a configuration structure.

`status_t LPUART_SetBaudRate(LPUART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)`

Sets the LPUART instance baudrate.

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the `LPUART_Init`.

```
LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
```

#### Parameters

- `base` – LPUART peripheral base address.
- `baudRate_Bps` – LPUART baudrate to be set.
- `srcClock_Hz` – LPUART clock source frequency in HZ.

#### Return values

- `kStatus_LPUART_BaudrateNotSupport` – Baudrate is not supported in the current clock source.
- `kStatus_Success` – Set baudrate succeeded.

`void LPUART_Enable9bitMode(LPUART_Type *base, bool enable)`

Enable 9-bit data mode for LPUART.

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

#### Parameters

- base – LPUART peripheral base address.
- enable – true to enable, 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 examples shows how to enable TX empty interrupt and RX full interrupt:

```
LPUART_EnableInterrupts(LPUART1, kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↳ RxDataRegFullInterruptEnable);
```

### Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to enable. Logical OR of `_lpuart_interrupt_enable`.

`void LPUART_DisableInterrupts(LPUART_Type *base, uint32_t mask)`

Disables LPUART interrupts according to a provided mask.

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See `_lpuart_interrupt_enable`. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
LPUART_DisableInterrupts(LPUART1, kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↳ RxDataRegFullInterruptEnable);
```

### Parameters

- `base` – LPUART peripheral base address.
- `mask` – The interrupts to disable. Logical OR of `_lpuart_interrupt_enable`.

`uint32_t LPUART_GetEnabledInterrupts(LPUART_Type *base)`

Gets enabled LPUART interrupts.

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators `_lpuart_interrupt_enable`. To check a specific interrupt enable status, compare the return value with enumerators in `_lpuart_interrupt_enable`. For example, to check whether the TX empty interrupt is enabled:

```
uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);

if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
{
    ...
}
```

### Parameters

- `base` – LPUART peripheral base address.

### Returns

LPUART interrupt flags which are logical OR of the enumerators in `_lpuart_interrupt_enable`.

`static inline uintptr_t LPUART_GetDataRegisterAddress(LPUART_Type *base)`

Gets the LPUART data register address.

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

### Parameters

- `base` – LPUART peripheral base address.

### Returns

LPUART data register addresses which are used both by the transmitter and receiver.

`static inline void LPUART_EnableTxDMA(LPUART_Type *base, bool enable)`

Enables or disables the LPUART transmitter DMA request.

This function enables or disables the transmit data register empty flag, `STAT[TDRE]`, to generate DMA requests.

### Parameters

- `base` – LPUART peripheral base address.
- `enable` – True to enable, false to disable.

```
static inline void LPUART_EnableRxDMA(LPUART_Type *base, bool enable)
```

Enables or disables the LPUART receiver DMA.

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

**Parameters**

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

```
uint32_t LPUART_GetInstance(LPUART_Type *base)
```

Get the LPUART instance from peripheral base address.

**Parameters**

- base – LPUART peripheral base address.

**Returns**

LPUART instance.

```
static inline void LPUART_EnableTx(LPUART_Type *base, bool enable)
```

Enables or disables the LPUART transmitter.

This function enables or disables the LPUART transmitter.

**Parameters**

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

```
static inline void LPUART_EnableRx(LPUART_Type *base, bool enable)
```

Enables or disables the LPUART receiver.

This function enables or disables the LPUART receiver.

**Parameters**

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

```
static inline void LPUART_WriteByte(LPUART_Type *base, uint8_t data)
```

Writes to the transmitter register.

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

**Parameters**

- base – LPUART peripheral base address.
- data – Data write to the TX register.

```
static inline uint8_t LPUART_ReadByte(LPUART_Type *base)
```

Reads the receiver register.

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

**Parameters**

- base – LPUART peripheral base address.

**Returns**

Data read from data register.

static inline uint8\_t LPUART\_GetRxFifoCount(LPUART\_Type \*base)

Gets the rx FIFO data count.

**Parameters**

- base – LPUART peripheral base address.

**Returns**

rx FIFO data count.

static inline uint8\_t LPUART\_GetTxFifoCount(LPUART\_Type \*base)

Gets the tx FIFO data count.

**Parameters**

- base – LPUART peripheral base address.

**Returns**

tx FIFO data count.

void LPUART\_SendAddress(LPUART\_Type \*base, uint8\_t address)

Transmit an address frame in 9-bit data mode.

**Parameters**

- base – LPUART peripheral base address.
- address – LPUART slave address.

status\_t LPUART\_WriteBlocking(LPUART\_Type \*base, const uint8\_t \*data, size\_t length)

Writes to the transmitter register using a blocking method.

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the 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 TX register.

#### Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data.
- length – Size of the buffer.

#### Return values

- kStatus\_LPUART\_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus\_LPUART\_NoiseError – Noise error happened while receiving data.
- kStatus\_LPUART\_FramingError – Framing error happened while receiving data.
- kStatus\_LPUART\_ParityError – Parity error happened while receiving data.
- kStatus\_LPUART\_Timeout – Transmission timed out and was aborted.
- kStatus\_Success – Successfully received all data.

*status\_t* LPUART\_ReadBlocking16bit(LPUART\_Type \*base, uint16\_t \*data, size\_t length)

Reads the receiver data register in 9bit or 10bit mode.

---

**Note:** This function only support 9bit or 10bit transfer.

---

#### Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data by 16bit, only 10bit is valid.
- length – Size of the buffer.

#### Return values

- kStatus\_LPUART\_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus\_LPUART\_NoiseError – Noise error happened while receiving data.
- kStatus\_LPUART\_FramingError – Framing error happened while receiving data.
- kStatus\_LPUART\_ParityError – Parity error happened while receiving data.
- kStatus\_LPUART\_Timeout – Transmission timed out and was aborted.
- kStatus\_Success – Successfully received all data.

void LPUART\_TransferCreateHandle(LPUART\_Type \*base, *lpuart\_handle\_t* \*handle,  
*lpuart\_transfer\_callback\_t* callback, void \*userData)

Initializes the LPUART handle.



This function initializes the LPUART handle, which can be used for other LPUART transactional APIs. Usually, for a specified LPUART instance, call this API once to get the initialized handle.

The LPUART driver supports the “background” receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn’t call the LPUART\_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as ringBuffer.

#### Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- callback – Callback function.
- userData – User data.

*status\_t* LPUART\_TransferSendNonBlocking(LPUART\_Type \*base, *lpuart\_handle\_t* \*handle, *lpuart\_transfer\_t* \*xfer)

Transmits a buffer of data using the interrupt method.

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the kStatus\_LPUART\_TxIdle as status parameter.

---

**Note:** The kStatus\_LPUART\_TxIdle is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the TX, check the kLPUART\_TransmissionCompleteFlag to ensure that the transmit is finished.

---

#### Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see *lpuart\_transfer\_t*.

#### Return values

- kStatus\_Success – Successfully start the data transmission.
- kStatus\_LPUART\_TxBusy – Previous transmission still not finished, data not all written to the TX register.
- kStatus\_InvalidArgument – Invalid argument.

void LPUART\_TransferStartRingBuffer(LPUART\_Type \*base, *lpuart\_handle\_t* \*handle, uint8\_t \*ringBuffer, size\_t ringBufferSize)

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn’t call the UART\_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

---

**Note:** When using RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, then only 31 bytes are used for saving data.

---

**Parameters**

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – size of the ring buffer.

void LPUART\_\_TransferStopRingBuffer(LPUART\_Type \*base, *lpuart\_handle\_t* \*handle)

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

**Parameters**

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

size\_t LPUART\_\_TransferGetRxRingBufferLength(LPUART\_Type \*base, *lpuart\_handle\_t* \*handle)

Get the length of received data in RX ring buffer.

**Parameters**

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

**Returns**

Length of received data in RX ring buffer.

void LPUART\_\_TransferAbortSend(LPUART\_Type \*base, *lpuart\_handle\_t* \*handle)

Aborts the interrupt-driven data transmit.

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are not sent out.

**Parameters**

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

status\_t LPUART\_\_TransferGetSendCount(LPUART\_Type \*base, *lpuart\_handle\_t* \*handle, uint32\_t \*count)

Gets the number of bytes that have been sent out to bus.

This function gets the number of bytes that have been sent out to bus by an interrupt method.

**Parameters**

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Send bytes count.

**Return values**

- kStatus\_NoTransferInProgress – No send in progress.
- kStatus\_InvalidArgument – Parameter is invalid.
- kStatus\_Success – Get successfully through the parameter count;

```
status_t LPUART_TransferReceiveNonBlocking(LPUART_Type *base, lpuart_handle_t *handle,
                                           lpuart_transfer_t *xfer, size_t *receivedBytes)
```

Receives a buffer of data using the interrupt method.

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter `kStatus_UART_RxIdle`. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to `xfer->data`, which returns with the parameter `receivedBytes` set to 5. For the remaining 5 bytes, the newly arrived data is saved from `xfer->data[5]`. When 5 bytes are received, the LPUART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to `xfer->data`. When all data is received, the upper layer is notified.

#### Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.
- `xfer` – LPUART transfer structure, see `uart_transfer_t`.
- `receivedBytes` – Bytes received from the ring buffer directly.

#### Return values

- `kStatus_Success` – Successfully queue the transfer into the transmit queue.
- `kStatus_LPUART_RxBusy` – Previous receive request is not finished.
- `kStatus_InvalidArgument` – Invalid argument.

```
void LPUART_TransferAbortReceive(LPUART_Type *base, lpuart_handle_t *handle)
```

Aborts the interrupt-driven data receiving.

This function aborts the interrupt-driven data receiving. The user can get the `remainBytes` to find out how many bytes not received yet.

#### Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.

```
status_t LPUART_TransferGetReceiveCount(LPUART_Type *base, lpuart_handle_t *handle,
                                         uint32_t *count)
```

Gets the number of bytes that have been received.

This function gets the number of bytes that have been received.

#### Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.
- `count` – Receive bytes count.

#### Return values

- `kStatus_NoTransferInProgress` – No receive in progress.
- `kStatus_InvalidArgument` – Parameter is invalid.
- `kStatus_Success` – Get successfully through the parameter `count`;

`void LPUART__TransferHandleIRQ(LPUART_Type *base, void *irqHandle)`

LPUART IRQ handle function.

This function handles the LPUART transmit and receive IRQ request.

**Parameters**

- `base` – LPUART peripheral base address.
- `irqHandle` – LPUART handle pointer.

`void LPUART__TransferHandleErrorIRQ(LPUART_Type *base, void *irqHandle)`

LPUART Error IRQ handle function.

This function handles the LPUART error IRQ request.

**Parameters**

- `base` – LPUART peripheral base address.
- `irqHandle` – LPUART handle pointer.

`void LPUART__DriverIRQHandler(uint32_t instance)`

LPUART driver IRQ handler common entry.

This function provides the common IRQ request entry for LPUART.

**Parameters**

- `instance` – LPUART instance.

`FSL_LPUART_DRIVER_VERSION`

LPUART driver version.

Error codes for the LPUART driver.

*Values:*

enumerator `kStatus_LPUART_TxBusy`

TX busy

enumerator `kStatus_LPUART_RxBusy`

RX busy

enumerator `kStatus_LPUART_TxIdle`

LPUART transmitter is idle.

enumerator `kStatus_LPUART_RxIdle`

LPUART receiver is idle.

enumerator `kStatus_LPUART_TxWatermarkTooLarge`

TX FIFO watermark too large

enumerator `kStatus_LPUART_RxWatermarkTooLarge`

RX FIFO watermark too large

enumerator `kStatus_LPUART_FlagCannotClearManually`

Some flag can't manually clear

enumerator `kStatus_LPUART_Error`

Error happens on LPUART.

enumerator `kStatus_LPUART_RxRingBufferOverrun`

LPUART RX software ring buffer overrun.

enumerator kStatus\_LPUART\_RxHardwareOverrun  
LPUART RX receiver overrun.

enumerator kStatus\_LPUART\_NoiseError  
LPUART noise error.

enumerator kStatus\_LPUART\_FramingError  
LPUART framing error.

enumerator kStatus\_LPUART\_ParityError  
LPUART parity error.

enumerator kStatus\_LPUART\_BaudrateNotSupport  
Baudrate is not support in current clock source

enumerator kStatus\_LPUART\_IdleLineDetected  
IDLE flag.

enumerator kStatus\_LPUART\_Timeout  
LPUART times out.

enum \_lpuart\_parity\_mode  
LPUART parity mode.

*Values:*

enumerator kLPUART\_ParityDisabled  
Parity disabled

enumerator kLPUART\_ParityEven  
Parity enabled, type even, bit setting: PE|PT = 10

enumerator kLPUART\_ParityOdd  
Parity enabled, type odd, bit setting: PE|PT = 11

enum \_lpuart\_data\_bits  
LPUART data bits count.

*Values:*

enumerator kLPUART\_EightDataBits  
Eight data bit

enumerator kLPUART\_SevenDataBits  
Seven data bit

enum \_lpuart\_stop\_bit\_count  
LPUART stop bit count.

*Values:*

enumerator kLPUART\_OneStopBit  
One stop bit

enumerator kLPUART\_TwoStopBit  
Two stop bits

enum \_lpuart\_transmit\_cts\_source  
LPUART transmit CTS source.

*Values:*

enumerator kLPUART\_CtsSourcePin  
CTS resource is the LPUART\_CTS pin.

enumerator kLPUART\_CtsSourceMatchResult

CTS resource is the match result.

enum \_lpuart\_transmit\_cts\_config

LPUART transmit CTS configure.

*Values:*

enumerator kLPUART\_CtsSampleAtStart

CTS input is sampled at the start of each character.

enumerator kLPUART\_CtsSampleAtIdle

CTS input is sampled when the transmitter is idle

enum \_lpuart\_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 \_\_unnamed24\_\_

### 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 \_\_unnamed26\_\_

### 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 \_\_unnamed28\_\_

### Public Members

uint8\_t \*volatile rxData  
Address of remaining data to receive.  
uint16\_t \*volatile rxData16  
Address of remaining data to receive.  
union \_\_unnamed30\_\_

### Public Members

uint8\_t \*rxRingBuffer  
Start address of the receiver ring buffer.  
uint16\_t \*rxRingBuffer16  
Start address of the receiver ring buffer.

## 2.43 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

## 2.44 MCM: Miscellaneous Control Module

FSL\_MCM\_DRIVER\_VERSION

MCM driver version.

Enum \_mcm\_interrupt\_flag. Interrupt status flag mask. .

*Values:*

enumerator kMCM\_CacheWriteBuffer  
Cache Write Buffer Error Enable.

enumerator kMCM\_ParityError  
Cache Parity Error Enable.

enumerator kMCM\_FPUInvalidOperation  
FPU Invalid Operation Interrupt Enable.

enumerator kMCM\_FPUDivideByZero  
FPU Divide-by-zero Interrupt Enable.

enumerator kMCM\_FPUOverflow  
FPU Overflow Interrupt Enable.

enumerator kMCM\_FPUUnderflow  
FPU Underflow Interrupt Enable.

enumerator kMCM\_FPUInexact  
FPU Inexact Interrupt Enable.

enumerator kMCM\_FPUInputDenormalInterrupt  
FPU Input Denormal Interrupt Enable.

typedef union *\_mcm\_buffer\_fault\_attribute* mcm\_buffer\_fault\_attribute\_t  
The union of buffer fault attribute.

typedef union *\_mcm\_lmem\_fault\_attribute* mcm\_lmem\_fault\_attribute\_t  
The union of LMEM fault attribute.

static inline void MCM\_EnableCrossbarRoundRobin(MCM\_Type \*base, bool enable)  
Enables/Disables crossbar round robin.

### Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable crossbar round robin.
  - **true** Enable crossbar round robin.
  - **false** disable crossbar round robin.

static inline void MCM\_EnableInterruptStatus(MCM\_Type \*base, uint32\_t mask)  
Enables the interrupt.

### Parameters

- base – MCM peripheral base address.
- mask – Interrupt status flags mask(\_mcm\_interrupt\_flag).

static inline void MCM\_DisableInterruptStatus(MCM\_Type \*base, uint32\_t mask)

Disables the interrupt.

**Parameters**

- base – MCM peripheral base address.
- mask – Interrupt status flags mask(`mcm_interrupt_flag`).

static inline uint16\_t MCM\_GetInterruptStatus(MCM\_Type \*base)

Gets the Interrupt status .

**Parameters**

- base – MCM peripheral base address.

static inline void MCM\_ClearCacheWriteBufferErrorStatus(MCM\_Type \*base)

Clears the Interrupt status .

**Parameters**

- base – MCM peripheral base address.

static inline uint32\_t MCM\_GetBufferFaultAddress(MCM\_Type \*base)

Gets buffer fault address.

**Parameters**

- base – MCM peripheral base address.

static inline void MCM\_GetBufferFaultAttribute(MCM\_Type \*base, *mcm\_buffer\_fault\_attribute\_t* \*bufferfault)

Gets buffer fault attributes.

**Parameters**

- base – MCM peripheral base address.

static inline uint32\_t MCM\_GetBufferFaultData(MCM\_Type \*base)

Gets buffer fault data.

**Parameters**

- base – MCM peripheral base address.

static inline void MCM\_LimitCodeCachePeripheralWriteBuffering(MCM\_Type \*base, bool enable)

Limit code cache peripheral write buffering.

**Parameters**

- base – MCM peripheral base address.
- enable – Used to enable/disable limit code cache peripheral write buffering.
  - **true** Enable limit code cache peripheral write buffering.
  - **false** disable limit code cache peripheral write buffering.

static inline void MCM\_BypassFixedCodeCacheMap(MCM\_Type \*base, bool enable)

Bypass fixed code cache map.

**Parameters**

- base – MCM peripheral base address.
- enable – Used to enable/disable bypass fixed code cache map.
  - **true** Enable bypass fixed code cache map.
  - **false** disable bypass fixed code cache map.



static inline void MCM\_EnableCodeBusCache(MCM\_Type \*base, bool enable)

Enables/Disables code bus cache.

**Parameters**

- base – MCM peripheral base address.
- enable – Used to disable/enable code bus cache.
  - **true** Enable code bus cache.
  - **false** disable code bus cache.

static inline void MCM\_ForceCodeCacheToNoAllocation(MCM\_Type \*base, bool enable)

Force code cache to no allocation.

**Parameters**

- base – MCM peripheral base address.
- enable – Used to force code cache to allocation or no allocation.
  - **true** Force code cache to no allocation.
  - **false** Force code cache to allocation.

static inline void MCM\_EnableCodeCacheWriteBuffer(MCM\_Type \*base, bool enable)

Enables/Disables code cache write buffer.

**Parameters**

- base – MCM peripheral base address.
- enable – Used to enable/disable code cache write buffer.
  - **true** Enable code cache write buffer.
  - **false** Disable code cache write buffer.

static inline void MCM\_ClearCodeBusCache(MCM\_Type \*base)

Clear code bus cache.

**Parameters**

- base – MCM peripheral base address.

static inline void MCM\_EnablePcParityFaultReport(MCM\_Type \*base, bool enable)

Enables/Disables PC Parity Fault Report.

**Parameters**

- base – MCM peripheral base address.
- enable – Used to enable/disable PC Parity Fault Report.
  - **true** Enable PC Parity Fault Report.
  - **false** disable PC Parity Fault Report.

static inline void MCM\_EnablePcParity(MCM\_Type \*base, bool enable)

Enables/Disables PC Parity.

**Parameters**

- base – MCM peripheral base address.
- enable – Used to enable/disable PC Parity.
  - **true** Enable PC Parity.
  - **false** disable PC Parity.

```
static inline void MCM_LockConfigState(MCM_Type *base)
```

Lock the configuration state.

#### Parameters

- base – MCM peripheral base address.

```
static inline void MCM_EnableCacheParityReporting(MCM_Type *base, bool enable)
```

Enables/Disables cache parity reporting.

#### Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable cache parity reporting.
  - **true** Enable cache parity reporting.
  - **false** disable cache parity reporting.

```
static inline uint32_t MCM_GetLmemFaultAddress(MCM_Type *base)
```

Gets LMEM fault address.

#### Parameters

- base – MCM peripheral base address.

```
static inline void MCM_GetLmemFaultAttribute(MCM_Type *base, mcm_lmem_fault_attribute_t  
*lmemFault)
```

Get LMEM fault attributes.

#### Parameters

- base – MCM peripheral base address.

```
static inline uint64_t MCM_GetLmemFaultData(MCM_Type *base)
```

Gets LMEM fault data.

#### Parameters

- base – MCM peripheral base address.

MCM\_LMFATR\_TYPE\_MASK

MCM\_LMFATR\_MODE\_MASK

MCM\_LMFATR\_BUFF\_MASK

MCM\_LMFATR\_CACH\_MASK

MCM\_ISCR\_STAT\_MASK

FSL\_COMPONENT\_ID

```
union _mcm_buffer_fault_attribute
```

*#include <fsl\_mcm.h>* The union of buffer fault attribute.

#### Public Members

```
uint32_t attribute
```

Indicates the faulting attributes, when a properly-enabled cache write buffer error interrupt event is detected.

```
struct _mcm_buffer_fault_attribute._mcm_buffer_fault_attribut attribute_memory
```

```
struct _mcm_buffer_fault_attribut
```

*#include <fsl\_mcm.h>*

### Public Members

uint32\_t busErrorDataAccessType

Indicates the type of cache write buffer access.

uint32\_t busErrorPrivilegeLevel

Indicates the privilege level of the cache write buffer access.

uint32\_t busErrorSize

Indicates the size of the cache write buffer access.

uint32\_t busErrorAccess

Indicates the type of system bus access.

uint32\_t busErrorMasterID

Indicates the crossbar switch bus master number of the captured cache write buffer bus error.

uint32\_t busErrorOverrun

Indicates if another cache write buffer bus error is detected.

union \_mcm\_lmem\_fault\_attribute

*#include <fsl\_mcm.h>* The union of LMEM fault attribute.

### Public Members

uint32\_t attribute

Indicates the attributes of the LMEM fault detected.

struct \_mcm\_lmem\_fault\_attribute.\_mcm\_lmem\_fault\_attribut attribute\_\_memory

struct \_mcm\_lmem\_fault\_attribut

*#include <fsl\_mcm.h>*

### Public Members

uint32\_t parityFaultProtectionSignal

Indicates the features of parity fault protection signal.

uint32\_t parityFaultMasterSize

Indicates the parity fault master size.

uint32\_t parityFaultWrite

Indicates the parity fault is caused by read or write.

uint32\_t backdoorAccess

Indicates the LMEM access fault is initiated by core access or backdoor access.

uint32\_t parityFaultSyndrome

Indicates the parity fault syndrome.

uint32\_t overrun

Indicates the number of faultss.

## 2.45 MMDVSQ: Memory-Mapped Divide and Square Root

```
int32_t MMDVSQ_GetDivideRemainder(MMDVSQ_Type *base, int32_t dividend, int32_t divisor,  
                                  bool isUnsigned)
```

Performs the MMDVSQ division operation and returns the remainder.

#### Parameters

- base – MMDVSQ peripheral address
- dividend – Dividend value
- divisor – Divisor value
- isUnsigned – Mode of unsigned divide
  - true unsigned divide
  - false signed divide

```
int32_t MMDVSQ_GetDivideQuotient(MMDVSQ_Type *base, int32_t dividend, int32_t divisor,  
                                 bool isUnsigned)
```

Performs the MMDVSQ division operation and returns the quotient.

#### Parameters

- base – MMDVSQ peripheral address
- dividend – Dividend value
- divisor – Divisor value
- isUnsigned – Mode of unsigned divide
  - true unsigned divide
  - false signed divide

```
uint16_t MMDVSQ_Sqrt(MMDVSQ_Type *base, uint32_t radicand)
```

Performs the MMDVSQ square root operation.

This function performs the MMDVSQ square root operation and returns the square root result of a given radicand value.

#### Parameters

- base – MMDVSQ peripheral address
- radicand – Radicand value

```
static inline mmdvsq_execution_status_t MMDVSQ_GetExecutionStatus(MMDVSQ_Type *base)
```

Gets the MMDVSQ execution status.

This function checks the current MMDVSQ execution status of the combined CSR[BUSY, DIV, Sqrt] indicators.

#### Parameters

- base – MMDVSQ peripheral address

#### Returns

Current MMDVSQ execution status

```
static inline void MMDVSQ_SetFastStartConfig(MMDVSQ_Type *base,  
                                              mmdvsq_fast_start_select_t mode)
```

Configures MMDVSQ fast start mode.

This function sets the MMDVSQ division fast start. The MMDVSQ supports two mechanisms for initiating a division operation. The default mechanism is a “fast start” where a write to the DSOR register begins the division. Alternatively, the start mechanism can begin after a write to the CSR register with CSR[SRT] set.

#### Parameters

- base – MMDVSQ peripheral address
- mode – Mode of Divide-Fast-Start
  - kMmdvsqDivideFastStart = 0
  - kMmdvsqDivideNormalStart = 1

static inline void MMDVSQ\_SetDivideByZeroConfig(MMDVSQ\_Type \*base, bool isDivByZero)

Configures the MMDVSQ divide-by-zero mode.

This function configures the MMDVSQ response to divide-by-zero calculations. If both CSR[DZ] and CSR[DZE] are set, then a subsequent read of the RES register is error-terminated to signal the processor of the attempted divide-by-zero. Otherwise, the register contents are returned.

#### Parameters

- base – MMDVSQ peripheral address
- isDivByZero – Mode of Divide-By-Zero
  - kMmdvsqDivideByZeroDis = 0
  - kMmdvsqDivideByZeroEn = 1

FSL\_MMSVSQ\_DRIVER\_VERSION

Version 2.0.4.

enum \_mmdvsq\_execution\_status

MMDVSQ execution status.

*Values:*

enumerator kMMDVSQ\_IdleSquareRoot

MMDVSQ is idle; the last calculation was a square root

enumerator kMMDVSQ\_IdleDivide

MMDVSQ is idle; the last calculation was division

enumerator kMMDVSQ\_BusySquareRoot

MMDVSQ is busy processing a square root calculation

enumerator kMMDVSQ\_BusyDivide

MMDVSQ is busy processing a division calculation

enum \_mmdvsq\_fast\_start\_select

MMDVSQ divide fast start select.

*Values:*

enumerator kMMDVSQ\_EnableFastStart

Division operation is initiated by a write to the DSOR register

enumerator kMMDVSQ\_DisableFastStart

Division operation is initiated by a write to CSR[SRT] = 1; normal start instead fast start

typedef enum \_mmdvsq\_execution\_status mmdvsq\_execution\_status\_t

MMDVSQ execution status.

typedef enum \_mmdvsq\_fast\_start\_select mmdvsq\_fast\_start\_select\_t

MMDVSQ divide fast start select.

## 2.46 MSCM: Miscellaneous System Control

FSL\_MSCM\_DRIVER\_VERSION

MSCM driver version 2.0.0.

```
typedef struct _mscm_uid mscm_uid_t
```

```
static inline void MSCM_GetUID(MSCM_Type *base, mscm_uid_t *uid)
```

Get MSCM UID.

### Parameters

- base – MSCM peripheral base address.
- uid – Pointer to an uid struct.

```
static inline void MSCM_SetSecureIrqParameter(MSCM_Type *base, const uint32_t parameter)
```

Set MSCM Secure Irq.

### Parameters

- base – MSCM peripheral base address.
- parameter – Value to be write to SECURE\_IRQ.

```
static inline uint32_t MSCM_GetSecureIrq(MSCM_Type *base)
```

Get MSCM Secure Irq.

### Parameters

- base – MSCM peripheral base address.

### Returns

MSCM Secure Irq.

FSL\_COMPONENT\_ID

```
struct _mscm_uid
```

```
#include <fsl_mscm.h>
```

## 2.47 PMC: Power Management Controller

```
static inline void PMC_GetVersionId(PMC_Type *base, pmc_version_id_t *versionId)
```

Gets the PMC version ID.

This function gets the PMC version ID, including major version number, minor version number, and a feature specification number.

### Parameters

- base – PMC peripheral base address.
- versionId – Pointer to version ID structure.

```
void PMC_GetParam(PMC_Type *base, pmc_param_t *param)
```

Gets the PMC parameter.

This function gets the PMC parameter including the VLPO enable and the HVD enable.

### Parameters

- base – PMC peripheral base address.
- param – Pointer to PMC param structure.

```
void PMC_ConfigureLowVoltDetect(PMC_Type *base, const pmc_low_volt_detect_config_t
                                *config)
```

Configures the low-voltage detect setting.

This function configures the low-voltage detect setting, including the trip point voltage setting, enables or disables the interrupt, enables or disables the system reset.

#### Parameters

- base – PMC peripheral base address.
- config – Low-voltage detect configuration structure.

```
static inline bool PMC_GetLowVoltDetectFlag(PMC_Type *base)
```

Gets the Low-voltage Detect Flag status.

This function reads the current LVDF status. If it returns 1, a low-voltage event is detected.

#### Parameters

- base – PMC peripheral base address.

#### Returns

Current low-voltage detect flag

- true: Low-voltage detected
- false: Low-voltage not detected

```
static inline void PMC_ClearLowVoltDetectFlag(PMC_Type *base)
```

Acknowledges clearing the Low-voltage Detect flag.

This function acknowledges the low-voltage detection errors (write 1 to clear LVDF).

#### Parameters

- base – PMC peripheral base address.

```
void PMC_ConfigureLowVoltWarning(PMC_Type *base, const pmc_low_volt_warning_config_t
                                  *config)
```

Configures the low-voltage warning setting.

This function configures the low-voltage warning setting, including the trip point voltage setting and enabling or disabling the interrupt.

#### Parameters

- base – PMC peripheral base address.
- config – Low-voltage warning configuration structure.

```
static inline bool PMC_GetLowVoltWarningFlag(PMC_Type *base)
```

Gets the Low-voltage Warning Flag status.

This function polls the current LVWF status. When 1 is returned, it indicates a low-voltage warning event. LVWF is set when V Supply transitions below the trip point or after reset and V Supply is already below the V LVW.

#### Parameters

- base – PMC peripheral base address.

#### Returns

Current LVWF status

- true: Low-voltage Warning Flag is set.
- false: the Low-voltage Warning does not happen.

static inline void PMC\_ClearLowVoltWarningFlag(PMC\_Type \*base)

Acknowledges the Low-voltage Warning flag.

This function acknowledges the low voltage warning errors (write 1 to clear LVWF).

#### Parameters

- base – PMC peripheral base address.

void PMC\_ConfigureHighVoltDetect(PMC\_Type \*base, const *pmc\_high\_volt\_detect\_config\_t* \*config)

Configures the high-voltage detect setting.

This function configures the high-voltage detect setting, including the trip point voltage setting, enabling or disabling the interrupt, enabling or disabling the system reset.

#### Parameters

- base – PMC peripheral base address.
- config – High-voltage detect configuration structure.

static inline bool PMC\_GetHighVoltDetectFlag(PMC\_Type \*base)

Gets the High-voltage Detect Flag status.

This function reads the current HVDF status. If it returns 1, a low voltage event is detected.

#### Parameters

- base – PMC peripheral base address.

#### Returns

Current high-voltage detect flag

- true: High-voltage detected
- false: High-voltage not detected

static inline void PMC\_ClearHighVoltDetectFlag(PMC\_Type \*base)

Acknowledges clearing the High-voltage Detect flag.

This function acknowledges the high-voltage detection errors (write 1 to clear HVDF).

#### Parameters

- base – PMC peripheral base address.

void PMC\_ConfigureBandgapBuffer(PMC\_Type \*base, const *pmc\_bandgap\_buffer\_config\_t* \*config)

Configures the PMC bandgap.

This function configures the PMC bandgap, including the drive select and behavior in low-power mode.

#### Parameters

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

static inline bool PMC\_GetPeriphIOIsolationFlag(PMC\_Type \*base)

Gets the acknowledge Peripherals and I/O pads isolation flag.

This function reads the Acknowledge Isolation setting that indicates whether certain peripherals and the I/O pads are in a latched state as a result of having been in the VLLS mode.

#### Parameters

- base – PMC peripheral base address.



- base – Base address for current PMC instance.

**Returns**

ACK isolation 0 - Peripherals and I/O pads are in a normal run state. 1 - Certain peripherals and I/O pads are in an isolated and latched state.

static inline void PMC\_ClearPeriphIOIsolationFlag(PMC\_Type \*base)

Acknowledges the isolation flag to Peripherals and I/O pads.

This function clears the ACK Isolation flag. Writing one to this setting when it is set releases the I/O pads and certain peripherals to their normal run mode state.

**Parameters**

- base – PMC peripheral base address.

static inline bool PMC\_IsRegulatorInRunRegulation(PMC\_Type \*base)

Gets the regulator regulation status.

This function returns the regulator to run a regulation status. It provides the current status of the internal voltage regulator.

**Parameters**

- base – PMC peripheral base address.
- base – Base address for current PMC instance.

**Returns**

Regulation status 0 - Regulator is in a stop regulation or in transition to/from the regulation. 1 - Regulator is in a run regulation.

FSL\_PMC\_DRIVER\_VERSION

PMC driver version.

Version 2.0.3.

enum \_pmc\_low\_volt\_detect\_volt\_select

Low-voltage Detect Voltage Select.

*Values:*

enumerator kPMC\_LowVoltDetectLowTrip

Low-trip point selected (VLVD = VLVDL )

enumerator kPMC\_LowVoltDetectHighTrip

High-trip point selected (VLVD = VLVDH )

enum \_pmc\_low\_volt\_warning\_volt\_select

Low-voltage Warning Voltage Select.

*Values:*

enumerator kPMC\_LowVoltWarningLowTrip

Low-trip point selected (VLVW = VLVW1)

enumerator kPMC\_LowVoltWarningMid1Trip

Mid 1 trip point selected (VLVW = VLVW2)

enumerator kPMC\_LowVoltWarningMid2Trip

Mid 2 trip point selected (VLVW = VLVW3)

enumerator kPMC\_LowVoltWarningHighTrip

High-trip point selected (VLVW = VLVW4)

enum `_pmc_high_volt_detect_volt_select`

High-voltage Detect Voltage Select.

*Values:*

enumerator `kPMC_HighVoltDetectLowTrip`

Low-trip point selected (VHVD = VHVDL )

enumerator `kPMC_HighVoltDetectHighTrip`

High-trip point selected (VHVD = VHVDH )

enum `_pmc_bandgap_buffer_drive_select`

Bandgap Buffer Drive Select.

*Values:*

enumerator `kPMC_BandgapBufferDriveLow`

Low-drive.

enumerator `kPMC_BandgapBufferDriveHigh`

High-drive.

enum `_pmc_vlp_freq_option`

VLPx Option.

*Values:*

enumerator `kPMC_FreqRestrict`

Frequency is restricted in VLPx mode.

enumerator `kPMC_FreqUnrestrict`

Frequency is unrestricted in VLPx mode.

typedef enum `_pmc_low_volt_detect_volt_select` `pmc_low_volt_detect_volt_select_t`

Low-voltage Detect Voltage Select.

typedef enum `_pmc_low_volt_warning_volt_select` `pmc_low_volt_warning_volt_select_t`

Low-voltage Warning Voltage Select.

typedef enum `_pmc_high_volt_detect_volt_select` `pmc_high_volt_detect_volt_select_t`

High-voltage Detect Voltage Select.

typedef enum `_pmc_bandgap_buffer_drive_select` `pmc_bandgap_buffer_drive_select_t`

Bandgap Buffer Drive Select.

typedef enum `_pmc_vlp_freq_option` `pmc_vlp_freq_mode_t`

VLPx Option.

typedef struct `_pmc_version_id` `pmc_version_id_t`

IP version ID definition.

typedef struct `_pmc_param` `pmc_param_t`

IP parameter definition.

typedef struct `_pmc_low_volt_detect_config` `pmc_low_volt_detect_config_t`

Low-voltage Detect Configuration Structure.

typedef struct `_pmc_low_volt_warning_config` `pmc_low_volt_warning_config_t`

Low-voltage Warning Configuration Structure.

typedef struct `_pmc_high_volt_detect_config` `pmc_high_volt_detect_config_t`

High-voltage Detect Configuration Structure.

```
typedef struct _pmc_bandgap_buffer_config pmc_bandgap_buffer_config_t
    Bandgap Buffer configuration.

struct _pmc_version_id
    #include <fsl_pmc.h> IP version ID definition.
```

### Public Members

```
uint16_t feature
    Feature Specification Number.

uint8_t minor
    Minor version number.

uint8_t major
    Major version number.

struct _pmc_param
    #include <fsl_pmc.h> IP parameter definition.
```

### Public Members

```
bool vlpoEnable
    VLPO enable.

bool hvdEnable
    HVD enable.

struct _pmc_low_volt_detect_config
    #include <fsl_pmc.h> Low-voltage Detect Configuration Structure.
```

### Public Members

```
bool enableInt
    Enable interrupt when Low-voltage detect

bool enableReset
    Enable system reset when Low-voltage detect

pmc_low_volt_detect_volt_select_t voltSelect
    Low-voltage detect trip point voltage selection

struct _pmc_low_volt_warning_config
    #include <fsl_pmc.h> Low-voltage Warning Configuration Structure.
```

### Public Members

```
bool enableInt
    Enable interrupt when low-voltage warning

pmc_low_volt_warning_volt_select_t voltSelect
    Low-voltage warning trip point voltage selection

struct _pmc_high_volt_detect_config
    #include <fsl_pmc.h> High-voltage Detect Configuration Structure.
```

### Public Members

`bool enableInt`  
Enable interrupt when high-voltage detect

`bool enableReset`  
Enable system reset when high-voltage detect

`pmc_high_volt_detect_volt_select_t voltSelect`  
High-voltage detect trip point voltage selection

`struct __pmc_bandgap_buffer_config`  
*#include <fsl\_pmc.h>* Bandgap Buffer configuration.

### Public Members

`bool enable`  
Enable bandgap buffer.

`bool enableInLowPowerMode`  
Enable bandgap buffer in low-power mode.

`pmc_bandgap_buffer_drive_select_t drive`  
Bandgap buffer drive select.

## 2.48 PORT: Port Control and Interrupts

`static inline void PORT_SetPinConfig(PORT_Type *base, uint32_t pin, const port_pin_config_t *config)`

Sets the port PCR register.

This is an example to define an input pin or output pin PCR configuration.

```
// Define a digital input pin PCR configuration
port_pin_config_t config = {
    kPORT_PullUp,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainDisable,
    kPORT_LowDriveStrength,
    kPORT_MuxAsGpio,
    kPORT_UnLockRegister,
};
```

### Parameters

- `base` – PORT peripheral base pointer.
- `pin` – PORT pin number.
- `config` – PORT PCR register configuration structure.

`static inline void PORT_SetMultiplePinsConfig(PORT_Type *base, uint32_t mask, const port_pin_config_t *config)`

Sets the port PCR register for multiple pins.

This is an example to define input pins or output pins PCR configuration.

```

Define a digital input pin PCR configuration
port_pin_config_t config = {
    kPORT_PullUp ,
    kPORT_PullEnable,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainDisable,
    kPORT_LowDriveStrength,
    kPORT_MuxAsGpio,
    kPORT_UnlockRegister,
};

```

### Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- config – PORT PCR register configuration structure.

```
static inline void PORT_SetMultipleInterruptPinsConfig(PORT_Type *base, uint32_t mask,
                                                    port_interrupt_t config)
```

Sets the port interrupt configuration in PCR register for multiple pins.

### Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- config – PORT pin interrupt configuration.
  - kPORT\_InterruptOrDMADisabled: Interrupt/DMA request disabled.
  - kPORT\_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).
  - kPORT\_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit).
  - kPORT\_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).
  - kPORT\_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).
  - kPORT\_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).
  - kPORT\_FlagEitherEdge : Flag sets on either edge(if the Flag states exit).
  - kPORT\_InterruptLogicZero : Interrupt when logic zero.
  - kPORT\_InterruptRisingEdge : Interrupt on rising edge.
  - kPORT\_InterruptFallingEdge: Interrupt on falling edge.
  - kPORT\_InterruptEitherEdge : Interrupt on either edge.
  - kPORT\_InterruptLogicOne : Interrupt when logic one.
  - kPORT\_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit).
  - kPORT\_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit)..

static inline void PORT\_SetPinMux(PORT\_Type \*base, uint32\_t pin, *port\_mux\_t* mux)

Configures the pin muxing.

---

**Note:** : This function is NOT recommended to use together with the PORT\_SetPinsConfig, because the PORT\_SetPinsConfig need to configure the pin mux anyway (Otherwise the pin mux is reset to zero : kPORT\_PinDisabledOrAnalog). This function is recommended to use to reset the pin mux

---

#### Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- mux – pin muxing slot selection.
  - kPORT\_PinDisabledOrAnalog: Pin disabled or work in analog function.
  - kPORT\_MuxAsGpio : Set as GPIO.
  - kPORT\_MuxAlt2 : chip-specific.
  - kPORT\_MuxAlt3 : chip-specific.
  - kPORT\_MuxAlt4 : chip-specific.
  - kPORT\_MuxAlt5 : chip-specific.
  - kPORT\_MuxAlt6 : chip-specific.
  - kPORT\_MuxAlt7 : chip-specific.

static inline void PORT\_EnablePinsDigitalFilter(PORT\_Type \*base, uint32\_t mask, bool enable)

Enables the digital filter in one port, each bit of the 32-bit register represents one pin.

#### Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- enable – PORT digital filter configuration.

static inline void PORT\_SetDigitalFilterConfig(PORT\_Type \*base, const  
*port\_digital\_filter\_config\_t* \*config)

Sets the digital filter in one port, each bit of the 32-bit register represents one pin.

#### Parameters

- base – PORT peripheral base pointer.
- config – PORT digital filter configuration structure.

static inline void PORT\_SetPinInterruptConfig(PORT\_Type \*base, uint32\_t pin, *port\_interrupt\_t*  
config)

Configures the port pin interrupt/DMA request.

#### Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- config – PORT pin interrupt configuration.
  - kPORT\_InterruptOrDMADisabled: Interrupt/DMA request disabled.
  - kPORT\_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).

- kPORT\_DMALFallingEdge: DMA request on falling edge(if the DMA requests exit).
- kPORT\_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).
- kPORT\_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).
- kPORT\_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).
- kPORT\_FlagEitherEdge : Flag sets on either edge(if the Flag states exit).
- kPORT\_InterruptLogicZero : Interrupt when logic zero.
- kPORT\_InterruptRisingEdge : Interrupt on rising edge.
- kPORT\_InterruptFallingEdge: Interrupt on falling edge.
- kPORT\_InterruptEitherEdge : Interrupt on either edge.
- kPORT\_InterruptLogicOne : Interrupt when logic one.
- kPORT\_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit).
- kPORT\_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit).

static inline void PORT\_SetPinDriveStrength(PORT\_Type \*base, uint32\_t pin, uint8\_t strength)

Configures the port pin drive strength.

#### Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- strength – PORT pin drive strength
  - kPORT\_LowDriveStrength = 0U - Low-drive strength is configured.
  - kPORT\_HighDriveStrength = 1U - High-drive strength is configured.

static inline uint32\_t PORT\_GetPinsInterruptFlags(PORT\_Type \*base)

Reads the whole port status flag.

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

#### Parameters

- base – PORT peripheral base pointer.

#### Returns

Current port interrupt status flags, for example, 0x00010001 means the pin 0 and 16 have the interrupt.

static inline void PORT\_ClearPinsInterruptFlags(PORT\_Type \*base, uint32\_t mask)

Clears the multiple pin interrupt status flag.

#### Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.

FSL\_PORT\_DRIVER\_VERSION

PORT driver version.

enum `_port_pull`

Internal resistor pull feature selection.

*Values:*

enumerator `kPORT_PullDisable`

Internal pull-up/down resistor is disabled.

enumerator `kPORT_PullDown`

Internal pull-down resistor is enabled.

enumerator `kPORT_PullUp`

Internal pull-up resistor is enabled.

enum `_port_slew_rate`

Slew rate selection.

*Values:*

enumerator `kPORT_FastSlewRate`

Fast slew rate is configured.

enumerator `kPORT_SlowSlewRate`

Slow slew rate is configured.

enum `_port_open_drain_enable`

Open Drain feature enable/disable.

*Values:*

enumerator `kPORT_OpenDrainDisable`

Open drain output is disabled.

enumerator `kPORT_OpenDrainEnable`

Open drain output is enabled.

enum `_port_passive_filter_enable`

Passive filter feature enable/disable.

*Values:*

enumerator `kPORT_PassiveFilterDisable`

Passive input filter is disabled.

enumerator `kPORT_PassiveFilterEnable`

Passive input filter is enabled.

enum `_port_drive_strength`

Configures the drive strength.

*Values:*

enumerator `kPORT_LowDriveStrength`

Low-drive strength is configured.

enumerator `kPORT_HighDriveStrength`

High-drive strength is configured.

enum `_port_lock_register`

Unlock/lock the pin control register field[15:0].

*Values:*

enumerator `kPORT_UnlockRegister`

Pin Control Register fields [15:0] are not locked.



enumerator kPORT\_LockRegister

Pin Control Register fields [15:0] are locked.

enum \_\_port\_mux

Pin mux selection.

*Values:*

enumerator kPORT\_PinDisabledOrAnalog

Corresponding pin is disabled, but is used as an analog pin.

enumerator kPORT\_MuxAsGpio

Corresponding pin is configured as GPIO.

enumerator kPORT\_MuxAlt0

Chip-specific

enumerator kPORT\_MuxAlt1

Chip-specific

enumerator kPORT\_MuxAlt2

Chip-specific

enumerator kPORT\_MuxAlt3

Chip-specific

enumerator kPORT\_MuxAlt4

Chip-specific

enumerator kPORT\_MuxAlt5

Chip-specific

enumerator kPORT\_MuxAlt6

Chip-specific

enumerator kPORT\_MuxAlt7

Chip-specific

enumerator kPORT\_MuxAlt8

Chip-specific

enumerator kPORT\_MuxAlt9

Chip-specific

enumerator kPORT\_MuxAlt10

Chip-specific

enumerator kPORT\_MuxAlt11

Chip-specific

enumerator kPORT\_MuxAlt12

Chip-specific

enumerator kPORT\_MuxAlt13

Chip-specific

enumerator kPORT\_MuxAlt14

Chip-specific

enumerator kPORT\_MuxAlt15

Chip-specific

enum `_port_interrupt`

Configures the interrupt generation condition.

*Values:*

enumerator `kPORT_InterruptOrDMADisabled`

Interrupt/DMA request is disabled.

enumerator `kPORT_DMARisingEdge`

DMA request on rising edge.

enumerator `kPORT_DMAFallingEdge`

DMA request on falling edge.

enumerator `kPORT_DMAEitherEdge`

DMA request on either edge.

enumerator `kPORT_FlagRisingEdge`

Flag sets on rising edge.

enumerator `kPORT_FlagFallingEdge`

Flag sets on falling edge.

enumerator `kPORT_FlagEitherEdge`

Flag sets on either edge.

enumerator `kPORT_InterruptLogicZero`

Interrupt when logic zero.

enumerator `kPORT_InterruptRisingEdge`

Interrupt on rising edge.

enumerator `kPORT_InterruptFallingEdge`

Interrupt on falling edge.

enumerator `kPORT_InterruptEitherEdge`

Interrupt on either edge.

enumerator `kPORT_InterruptLogicOne`

Interrupt when logic one.

enumerator `kPORT_ActiveHighTriggerOutputEnable`

Enable active high-trigger output.

enumerator `kPORT_ActiveLowTriggerOutputEnable`

Enable active low-trigger output.

enum `_port_digital_filter_clock_source`

Digital filter clock source selection.

*Values:*

enumerator `kPORT_BusClock`

Digital filters are clocked by the bus clock.

enumerator `kPORT_LpoClock`

Digital filters are clocked by the 1 kHz LPO clock.

typedef enum `_port_mux` `port_mux_t`

Pin mux selection.

typedef enum `_port_interrupt` `port_interrupt_t`

Configures the interrupt generation condition.

```
typedef enum _port_digital_filter_clock_source port_digital_filter_clock_source_t
```

Digital filter clock source selection.

```
typedef struct _port_digital_filter_config port_digital_filter_config_t
```

PORT digital filter feature configuration definition.

```
typedef struct _port_pin_config port_pin_config_t
```

PORT pin configuration structure.

FSL\_COMPONENT\_ID

```
struct _port_digital_filter_config
```

*#include <fsl\_port.h>* PORT digital filter feature configuration definition.

### Public Members

uint32\_t digitalFilterWidth

Set digital filter width

*port\_digital\_filter\_clock\_source\_t* clockSource

Set digital filter clockSource

```
struct _port_pin_config
```

*#include <fsl\_port.h>* PORT pin configuration structure.

### Public Members

uint16\_t pullSelect

No-pull/pull-down/pull-up select

uint16\_t slewRate

Fast/slow slew rate Configure

uint16\_t passiveFilterEnable

Passive filter enable/disable

uint16\_t openDrainEnable

Open drain enable/disable

uint16\_t driveStrength

Fast/slow drive strength configure

uint16\_t lockRegister

Lock/unlock the PCR field[15:0]

## 2.49 RCM: Reset Control Module Driver

```
static inline void RCM_GetVersionId(RCM_Type *base, rcm_version_id_t *versionId)
```

Gets the RCM version ID.

This function gets the RCM version ID including the major version number, the minor version number, and the feature specification number.

### Parameters

- base – RCM peripheral base address.
- versionId – Pointer to the version ID structure.

```
static inline uint32_t RCM_GetResetSourceImplementedStatus(RCM_Type *base)
```

Gets the reset source implemented status.

This function gets the RCM parameter that indicates whether the corresponding reset source is implemented. Use source masks defined in the `rcm_reset_source_t` to get the desired source status.

This is an example.

```
uint32_t status;
```

To test whether the MCU is reset using Watchdog.

```
status = RCM_GetResetSourceImplementedStatus(RCM) & (kRCM_SourceWdog | kRCM_SourcePin);
```

### Parameters

- base – RCM peripheral base address.

### Returns

All reset source implemented status bit map.

```
static inline uint32_t RCM_GetPreviousResetSources(RCM_Type *base)
```

Gets the reset source status which caused a previous reset.

This function gets the current reset source status. Use source masks defined in the `rcm_reset_source_t` to get the desired source status.

This is an example.

```
uint32_t resetStatus;
```

To get all reset source statuses.

```
resetStatus = RCM_GetPreviousResetSources(RCM) & kRCM_SourceAll;
```

To test whether the MCU is reset using Watchdog.

```
resetStatus = RCM_GetPreviousResetSources(RCM) & kRCM_SourceWdog;
```

To test multiple reset sources.

```
resetStatus = RCM_GetPreviousResetSources(RCM) & (kRCM_SourceWdog | kRCM_SourcePin);
```

### Parameters

- base – RCM peripheral base address.

### Returns

All reset source status bit map.

```
static inline uint32_t RCM_GetStickyResetSources(RCM_Type *base)
```

Gets the sticky reset source status.

This function gets the current reset source status that has not been cleared by software for a specific source.

This is an example.

```
uint32_t resetStatus;
```

To get all reset source statuses.

```
resetStatus = RCM_GetStickyResetSources(RCM) & kRCM_SourceAll;
```

To test whether the MCU is reset using Watchdog.

```
resetStatus = RCM_GetStickyResetSources(RCM) & kRCM_SourceWdog;
```

To test multiple reset sources.

```
resetStatus = RCM_GetStickyResetSources(RCM) & (kRCM_SourceWdog | kRCM_SourcePin);
```

**Parameters**

- base – RCM peripheral base address.

**Returns**

All reset source status bit map.

```
static inline void RCM_ClearStickyResetSources(RCM_Type *base, uint32_t sourceMasks)
```

Clears the sticky reset source status.

This function clears the sticky system reset flags indicated by source masks.

This is an example.

Clears multiple reset sources.

```
RCM_ClearStickyResetSources(kRCM_SourceWdog | kRCM_SourcePin);
```

**Parameters**

- base – RCM peripheral base address.
- sourceMasks – reset source status bit map

```
void RCM_ConfigureResetPinFilter(RCM_Type *base, const rcm_reset_pin_filter_config_t *config)
```

Configures the reset pin filter.

This function sets the reset pin filter including the filter source, filter width, and so on.

**Parameters**

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

```
static inline bool RCM_GetEasyPortModePinStatus(RCM_Type *base)
```

Gets the EZP\_MS\_B pin assert status.

This function gets the easy port mode status (EZP\_MS\_B) pin assert status.

**Parameters**

- base – RCM peripheral base address.

**Returns**

status true - asserted, false - reasserted

```
static inline rcm_boot_rom_config_t RCM_GetBootRomSource(RCM_Type *base)
```

Gets the ROM boot source.

This function gets the ROM boot source during the last chip reset.

**Parameters**

- base – RCM peripheral base address.

**Returns**

The ROM boot source.

```
static inline void RCM_ClearBootRomSource(RCM_Type *base)
```

Clears the ROM boot source flag.

This function clears the ROM boot source flag.

**Parameters**

- base – Register base address of RCM

```
void RCM_SetForceBootRomSource(RCM_Type *base, rcm_boot_rom_config_t config)
```

Forces the boot from ROM.

This function forces booting from ROM during all subsequent system resets.

#### Parameters

- base – RCM peripheral base address.
- config – Boot configuration.

```
static inline void RCM_SetSystemResetInterruptConfig(RCM_Type *base, uint32_t intMask,  
                                                    rcm_reset_delay_t delay)
```

Sets the system reset interrupt configuration.

For a graceful shut down, the RCM supports delaying the assertion of the system reset for a period of time when the reset interrupt is generated. This function can be used to enable the interrupt and the delay period. The interrupts are passed in as bit mask. See `rcm_int_t` for details. For example, to delay a reset for 512 LPO cycles after the WDOG timeout or loss-of-clock occurs, configure as follows: `RCM_SetSystemResetInterruptConfig(kRCM_IntWatchDog | kRCM_IntLossOfClk, kRCM_ResetDelay512Lpo);`

#### Parameters

- base – RCM peripheral base address.
- intMask – Bit mask of the system reset interrupts to enable. See `rcm_interrupt_enable_t` for details.
- delay – Bit mask of the system reset interrupts to enable.

```
FSL_RCM_DRIVER_VERSION
```

RCM driver version 2.0.4.

```
enum _rcm_reset_source
```

System Reset Source Name definitions.

*Values:*

```
enumerator kRCM_SourceWakeup
```

Low-leakage wakeup reset

```
enumerator kRCM_SourceLvd
```

Low-voltage detect reset

```
enumerator kRCM_SourceLoc
```

Loss of clock reset

```
enumerator kRCM_SourceLol
```

Loss of lock reset

```
enumerator kRCM_SourceWdog
```

Watchdog reset

```
enumerator kRCM_SourcePin
```

External pin reset

```
enumerator kRCM_SourcePor
```

Power on reset

```
enumerator kRCM_SourceJtag
```

JTAG generated reset

```
enumerator kRCM_SourceLockup
```

Core lock up reset

enumerator kRCM\_SourceSw  
Software reset

enumerator kRCM\_SourceMdmap  
MDM-AP system reset

enumerator kRCM\_SourceEzpt  
EzPort reset

enumerator kRCM\_SourceSackerr  
Parameter could get all reset flags

enumerator kRCM\_SourceAll

enum \_rcm\_run\_wait\_filter\_mode  
Reset pin filter select in Run and Wait modes.

*Values:*

enumerator kRCM\_FilterDisable  
All filtering disabled

enumerator kRCM\_FilterBusClock  
Bus clock filter enabled

enumerator kRCM\_FilterLpoClock  
LPO clock filter enabled

enum \_rcm\_boot\_rom\_config  
Boot from ROM configuration.

*Values:*

enumerator kRCM\_BootFlash  
Boot from flash

enumerator kRCM\_BootRomCfg0  
Boot from boot ROM due to BOOTCFG0

enumerator kRCM\_BootRomFopt  
Boot from boot ROM due to FOPT[7]

enumerator kRCM\_BootRomBoth  
Boot from boot ROM due to both BOOTCFG0 and FOPT[7]

enum \_rcm\_reset\_delay  
Maximum delay time from interrupt asserts to system reset.

*Values:*

enumerator kRCM\_ResetDelay8Lpo  
Delay 8 LPO cycles.

enumerator kRCM\_ResetDelay32Lpo  
Delay 32 LPO cycles.

enumerator kRCM\_ResetDelay128Lpo  
Delay 128 LPO cycles.

enumerator kRCM\_ResetDelay512Lpo  
Delay 512 LPO cycles.

enum *\_rcm\_interrupt\_enable*

System reset interrupt enable bit definitions.

*Values:*

enumerator *kRCM\_IntNone*

No interrupt enabled.

enumerator *kRCM\_IntLossOfClk*

Loss of clock interrupt.

enumerator *kRCM\_IntLossOfLock*

Loss of lock interrupt.

enumerator *kRCM\_IntWatchDog*

Watch dog interrupt.

enumerator *kRCM\_IntExternalPin*

External pin interrupt.

enumerator *kRCM\_IntGlobal*

Global interrupts.

enumerator *kRCM\_IntCoreLockup*

Core lock up interrupt

enumerator *kRCM\_IntSoftware*

software interrupt

enumerator *kRCM\_IntStopModeAckErr*

Stop mode ACK error interrupt.

enumerator *kRCM\_IntCore1*

Core 1 interrupt.

enumerator *kRCM\_IntAll*

Enable all interrupts.

typedef enum *\_rcm\_reset\_source* *rcm\_reset\_source\_t*

System Reset Source Name definitions.

typedef enum *\_rcm\_run\_wait\_filter\_mode* *rcm\_run\_wait\_filter\_mode\_t*

Reset pin filter select in Run and Wait modes.

typedef enum *\_rcm\_boot\_rom\_config* *rcm\_boot\_rom\_config\_t*

Boot from ROM configuration.

typedef enum *\_rcm\_reset\_delay* *rcm\_reset\_delay\_t*

Maximum delay time from interrupt asserts to system reset.

typedef enum *\_rcm\_interrupt\_enable* *rcm\_interrupt\_enable\_t*

System reset interrupt enable bit definitions.

typedef struct *\_rcm\_version\_id* *rcm\_version\_id\_t*

IP version ID definition.

typedef struct *\_rcm\_reset\_pin\_filter\_config* *rcm\_reset\_pin\_filter\_config\_t*

Reset pin filter configuration.

struct *\_rcm\_version\_id*

*#include <fsl\_rcm.h>* IP version ID definition.



**Public Members**

uint16\_t feature  
Feature Specification Number.

uint8\_t minor  
Minor version number.

uint8\_t major  
Major version number.

struct \_rcm\_reset\_pin\_filter\_config  
#include <fsl\_rcm.h> Reset pin filter configuration.

**Public Members**

bool enableFilterInStop  
Reset pin filter select in stop mode.

rcm\_run\_wait\_filter\_mode\_t filterInRunWait  
Reset pin filter in run/wait mode.

uint8\_t busClockFilterCount  
Reset pin bus clock filter width.

## 2.50 RTC: Real Time Clock

void RTC\_Init(RTC\_Type \*base, const rtc\_config\_t \*config)  
Ungates the RTC clock and configures the peripheral for basic operation.  
This function issues a software reset if the timer invalid flag is set.

---

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

---

**Parameters**

- base – RTC peripheral base address
- config – Pointer to the user's RTC configuration structure.

static inline void RTC\_Deinit(RTC\_Type \*base)  
Stops the timer and gate the RTC clock.

**Parameters**

- base – RTC peripheral base address

void RTC\_GetDefaultConfig(rtc\_config\_t \*config)  
Fills in the RTC config struct with the default settings.  
The default values are as follows.

```
config->wakeupSelect = false;
config->updateMode = false;
config->supervisorAccess = false;
config->compensationInterval = 0;
config->compensationTime = 0;
```

**Parameters**

- `config` – Pointer to the user's RTC configuration structure.

`status_t RTC_SetDatetime(RTC_Type *base, const rtc_datetime_t *datetime)`

Sets the RTC date and time according to the given time structure.

The RTC counter must be stopped prior to calling this function because writes to the RTC seconds register fail if the RTC counter is running.

#### Parameters

- `base` – RTC peripheral base address
- `datetime` – Pointer to the structure where the date and time details are stored.

#### Returns

`kStatus_Success`: Success in setting the time and starting the RTC  
`kStatus_InvalidArgument`: Error because the datetime format is incorrect

`void RTC_GetDatetime(RTC_Type *base, rtc_datetime_t *datetime)`

Gets the RTC time and stores it in the given time structure.

#### Parameters

- `base` – RTC peripheral base address
- `datetime` – Pointer to the structure where the date and time details are stored.

`status_t RTC_SetAlarm(RTC_Type *base, const rtc_datetime_t *alarmTime)`

Sets the RTC alarm time.

The function checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

#### Parameters

- `base` – RTC peripheral base address
- `alarmTime` – Pointer to the structure where the alarm time is stored.

#### Returns

`kStatus_Success`: success in setting the RTC alarm  
`kStatus_InvalidArgument`: Error because the alarm datetime format is incorrect  
`kStatus_Fail`: Error because the alarm time has already passed

`void RTC_GetAlarm(RTC_Type *base, rtc_datetime_t *datetime)`

Returns the RTC alarm time.

#### Parameters

- `base` – RTC peripheral base address
- `datetime` – Pointer to the structure where the alarm date and time details are stored.

`void RTC_EnableInterrupts(RTC_Type *base, uint32_t mask)`

Enables the selected RTC interrupts.

#### Parameters

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

`void RTC_DisableInterrupts(RTC_Type *base, uint32_t mask)`

Disables the selected RTC interrupts.

**Parameters**

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

`uint32_t RTC_GetEnabledInterrupts(RTC_Type *base)`

Gets the enabled RTC interrupts.

**Parameters**

- `base` – RTC peripheral base address

**Returns**

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

`uint32_t RTC_GetStatusFlags(RTC_Type *base)`

Gets the RTC status flags.

**Parameters**

- `base` – RTC peripheral base address

**Returns**

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

`void RTC_ClearStatusFlags(RTC_Type *base, uint32_t mask)`

Clears the RTC status flags.

**Parameters**

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

`static inline void RTC_EnableOscillatorClock(RTC_Type *base, bool enable)`

Enable/Disable RTC 32kHz Oscillator clock.

---

**Note:** After setting this bit, wait the oscillator startup time before enabling the time counter to allow the 32.768 kHz clock time to stabilize.

---

**Parameters**

- `base` – RTC peripheral base address
- `enable` – Enable/Disable RTC 32.768 kHz clock

`static inline void RTC_SetClockSource(RTC_Type *base)`

Set RTC clock source.

*Deprecated:*

Do not use this function. It has been superceded by `RTC_EnableOscillatorClock`

---

**Note:** After setting this bit, wait the oscillator startup time before enabling the time counter to allow the 32.768 kHz clock time to stabilize.

---

**Parameters**

- base – RTC peripheral base address

static inline void RTC\_EnableLPOClock(RTC\_Type \*base, bool enable)

Enable/Disable RTC 1kHz LPO clock.

---

**Note:** After setting this bit, RTC prescaler increments using the LPO 1kHz clock and not the RTC 32kHz crystal clock.

---

**Parameters**

- base – RTC peripheral base address
- enable – Enable/Disable RTC 1kHz LPO clock

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

Starts the RTC time counter.

After calling this function, the timer counter increments once a second provided SR[TOF] or SR[TIF] are not set.

**Parameters**

- base – RTC peripheral base address

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

Stops the RTC time counter.

RTC's seconds register can be written to only when the timer is stopped.

**Parameters**

- base – RTC peripheral base address

void RTC\_GetMonotonicCounter(RTC\_Type \*base, uint64\_t \*counter)

Reads the values of the Monotonic Counter High and Monotonic Counter Low and returns them as a single value.

**Parameters**

- base – RTC peripheral base address
- counter – Pointer to variable where the value is stored.

void RTC\_SetMonotonicCounter(RTC\_Type \*base, uint64\_t counter)

Writes values Monotonic Counter High and Monotonic Counter Low by decomposing the given single value. The Monotonic Overflow Flag in RTC\_SR is cleared due to the API.

**Parameters**

- base – RTC peripheral base address
- counter – Counter value

status\_t RTC\_IncrementMonotonicCounter(RTC\_Type \*base)

Increments the Monotonic Counter by one.

Increments the Monotonic Counter (registers RTC\_MCLR and RTC\_MCHR accordingly) by setting the monotonic counter enable (MER[MCE]) and then writing to the RTC\_MCLR register. A write to the monotonic counter low that causes it to overflow also increments the monotonic counter high.

**Parameters**

- base – RTC peripheral base address

**Returns**

kStatus\_Success: success kStatus\_Fail: error occurred, either time invalid or monotonic overflow flag was found

FSL\_RTC\_DRIVER\_VERSION

Version 2.3.3

enum \_\_rtc\_interrupt\_enable

List of RTC interrupts.

*Values:*

enumerator kRTC\_TimeInvalidInterruptEnable

Time invalid interrupt.

enumerator kRTC\_TimeOverflowInterruptEnable

Time overflow interrupt.

enumerator kRTC\_AlarmInterruptEnable

Alarm interrupt.

enumerator kRTC\_MonotonicOverflowInterruptEnable

Monotonic Overflow Interrupt Enable

enumerator kRTC\_SecondsInterruptEnable

Seconds interrupt.

enumerator kRTC\_TestModeInterruptEnable

enumerator kRTC\_FlashSecurityInterruptEnable

enumerator kRTC\_TamperPinInterruptEnable

enumerator kRTC\_SecurityModuleInterruptEnable

enumerator kRTC\_LossOfClockInterruptEnable

enum \_\_rtc\_status\_flags

List of RTC flags.

*Values:*

enumerator kRTC\_TimeInvalidFlag

Time invalid flag

enumerator kRTC\_TimeOverflowFlag

Time overflow flag

enumerator kRTC\_AlarmFlag

Alarm flag

enumerator kRTC\_MonotonicOverflowFlag

Monotonic Overflow Flag

enumerator kRTC\_TamperInterruptDetectFlag

Tamper interrupt detect flag

enumerator kRTC\_TestModeFlag

enumerator kRTC\_FlashSecurityFlag

enumerator kRTC\_TamperPinFlag

enumerator kRTC\_SecurityTamperFlag

enumerator kRTC\_LossOfClockTamperFlag

enum \_rtc\_osc\_cap\_load

List of RTC Oscillator capacitor load settings.

*Values:*

enumerator kRTC\_Capacitor\_2p

2 pF capacitor load

enumerator kRTC\_Capacitor\_4p

4 pF capacitor load

enumerator kRTC\_Capacitor\_8p

8 pF capacitor load

enumerator kRTC\_Capacitor\_16p

16 pF capacitor load

typedef enum \_rtc\_interrupt\_enable rtc\_interrupt\_enable\_t

List of RTC interrupts.

typedef enum \_rtc\_status\_flags rtc\_status\_flags\_t

List of RTC flags.

typedef enum \_rtc\_osc\_cap\_load rtc\_osc\_cap\_load\_t

List of RTC Oscillator capacitor load settings.

typedef struct \_rtc\_datetime rtc\_datetime\_t

Structure is used to hold the date and time.

typedef struct \_rtc\_pin\_config rtc\_pin\_config\_t

RTC pin config structure.

typedef struct \_rtc\_config rtc\_config\_t

RTC config structure.

This structure holds the configuration settings for the RTC peripheral. To initialize this structure to reasonable defaults, call the RTC\_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

static inline uint32\_t RTC\_GetTamperTimeSeconds(RTC\_Type \*base)

Get the RTC tamper time seconds.

#### Parameters

- base – RTC peripheral base address

static inline void RTC\_SetOscCapLoad(RTC\_Type \*base, uint32\_t capLoad)

This function sets the specified capacitor configuration for the RTC oscillator.

#### Parameters

- base – RTC peripheral base address
- capLoad – Oscillator loads to enable. This is a logical OR of members of the enumeration rtc\_osc\_cap\_load\_t

static inline void RTC\_Reset(RTC\_Type \*base)

Performs a software reset on the RTC module.

This resets all RTC registers except for the SWR bit and the RTC\_WAR and RTC\_RAR registers. The SWR bit is cleared by software explicitly clearing it.

#### Parameters

- base – RTC peripheral base address

static inline void RTC\_EnableWakeUpPin(RTC\_Type \*base, bool enable)

Enables or disables the RTC Wakeup Pin Operation.

This function enable or disable RTC Wakeup Pin. The wakeup pin is optional and not available on all devices.

#### Parameters

- base – RTC\_Type base pointer.
- enable – true to enable, false to disable.

struct \_rtc\_datetime

*#include <fsl\_rtc.h>* Structure is used to hold the date and time.

#### Public Members

uint16\_t year

Range from 1970 to 2099.

uint8\_t month

Range from 1 to 12.

uint8\_t day

Range from 1 to 31 (depending on month).

uint8\_t hour

Range from 0 to 23.

uint8\_t minute

Range from 0 to 59.

uint8\_t second

Range from 0 to 59.

struct \_rtc\_pin\_config

*#include <fsl\_rtc.h>* RTC pin config structure.

#### Public Members

bool inputLogic

true: Tamper pin input data is logic one. false: Tamper pin input data is logic zero.

bool pinActiveLow

true: Tamper pin is active low. false: Tamper pin is active high.

bool filterEnable

true: Input filter is enabled on the tamper pin. false: Input filter is disabled on the tamper pin.

bool pullSelectNegate

true: Tamper pin pull resistor direction will negate the tamper pin. false: Tamper pin pull resistor direction will assert the tamper pin.

bool pullEnable

true: Pull resistor is enabled on tamper pin. false: Pull resistor is disabled on tamper pin.

struct `_rtc_config`

*#include <fsl\_rtc.h>* RTC config structure.

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

The config struct can be made const so it resides in flash

### Public Members

bool `wakeupSelect`

true: Wakeup pin outputs the 32 KHz clock; false: Wakeup pin used to wakeup the chip

bool `updateMode`

true: Registers can be written even when locked under certain conditions, false: No writes allowed when registers are locked

bool `supervisorAccess`

true: Non-supervisor accesses are allowed; false: Non-supervisor accesses are not supported

uint32\_t `compensationInterval`

Compensation interval that is written to the CIR field in RTC TCR Register

uint32\_t `compensationTime`

Compensation time that is written to the TCR field in RTC TCR Register

## 2.51 SIM: System Integration Module Driver

FSL\_SIM\_DRIVER\_VERSION

Driver version.

enum `_sim_usb_volt_reg_enable_mode`

USB voltage regulator enable setting.

*Values:*

enumerator `kSIM_UsbVoltRegEnable`

Enable voltage regulator.

enumerator `kSIM_UsbVoltRegEnableInLowPower`

Enable voltage regulator in VLPR/VLPW modes.

enumerator `kSIM_UsbVoltRegEnableInStop`

Enable voltage regulator in STOP/VLPS/LLS/VLLS modes.

enumerator `kSIM_UsbVoltRegEnableInAllModes`

Enable voltage regulator in all power modes.

enum `_sim_flash_mode`

Flash enable mode.

*Values:*

enumerator `kSIM_FlashDisableInWait`

Disable flash in wait mode.

enumerator `kSIM_FlashDisable`

Disable flash in normal mode.



```
typedef struct _sim_uid sim_uid_t
```

Unique ID.

```
void SIM_SetUsbVoltRegulatorEnableMode(uint32_t mask)
```

Sets the USB voltage regulator setting.

This function configures whether the USB voltage regulator is enabled in normal RUN mode, STOP/VLPS/LLS/VLLS modes, and VLPR/VLPW modes. The configurations are passed in as mask value of `_sim_usb_volt_reg_enable_mode`. For example, to enable USB voltage regulator in RUN/VLPR/VLPW modes and disable in STOP/VLPS/LLS/VLLS mode, use:

```
SIM_SetUsbVoltRegulatorEnableMode(kSIM_UsbVoltRegEnable  
kSIM_UsbVoltRegEnableInLowPower);
```

#### Parameters

- `mask` – USB voltage regulator enable setting.

```
void SIM_GetUniqueId(sim_uid_t *uid)
```

Gets the unique identification register value.

#### Parameters

- `uid` – Pointer to the structure to save the UID value.

```
static inline void SIM_SetFlashMode(uint8_t mode)
```

Sets the flash enable mode.

#### Parameters

- `mode` – The mode to set; see `_sim_flash_mode` for mode details.

```
struct _sim_uid
```

*#include <fsl\_sim.h>* Unique ID.

#### Public Members

```
uint32_t H
```

UIDH.

```
uint32_t M
```

SIM\_UIDM.

```
uint32_t L
```

UIDL.

## 2.52 Smart Card

```
FSL_SMARTCARD_DRIVER_VERSION
```

Smart card driver version 2.3.0.

Smart card Error codes.

*Values:*

```
enumerator kStatus_SMARTCARD_Success
```

Transfer ends successfully

```
enumerator kStatus_SMARTCARD_TxBusy
```

Transmit in progress

enumerator kStatus\_SMARTCARD\_RxBusy

Receiving in progress

enumerator kStatus\_SMARTCARD\_NoTransferInProgress

No transfer in progress

enumerator kStatus\_SMARTCARD\_Timeout

Transfer ends with time-out

enumerator kStatus\_SMARTCARD\_Initialized

Smart card driver is already initialized

enumerator kStatus\_SMARTCARD\_PhyInitialized

Smart card PHY drive is already initialized

enumerator kStatus\_SMARTCARD\_CardNotActivated

Smart card is not activated

enumerator kStatus\_SMARTCARD\_InvalidInput

Function called with invalid input arguments

enumerator kStatus\_SMARTCARD\_OtherError

Some other error occur

enum \_smartcard\_control

Control codes for the Smart card protocol timers and misc.

*Values:*

enumerator kSMARTCARD\_EnableADT

enumerator kSMARTCARD\_DisableADT

enumerator kSMARTCARD\_EnableGTV

enumerator kSMARTCARD\_DisableGTV

enumerator kSMARTCARD\_ResetWWT

enumerator kSMARTCARD\_EnableWWT

enumerator kSMARTCARD\_DisableWWT

enumerator kSMARTCARD\_ResetCWT

enumerator kSMARTCARD\_EnableCWT

enumerator kSMARTCARD\_DisableCWT

enumerator kSMARTCARD\_ResetBWT

enumerator kSMARTCARD\_EnableBWT

enumerator kSMARTCARD\_DisableBWT

enumerator kSMARTCARD\_EnableInitDetect

enumerator kSMARTCARD\_EnableAnack

enumerator kSMARTCARD\_DisableAnack

enumerator kSMARTCARD\_ConfigureBaudrate

enumerator kSMARTCARD\_SetupATRMMode

enumerator kSMARTCARD\_SetupT0Mode  
enumerator kSMARTCARD\_SetupT1Mode  
enumerator kSMARTCARD\_EnableReceiverMode  
enumerator kSMARTCARD\_DisableReceiverMode  
enumerator kSMARTCARD\_EnableTransmitterMode  
enumerator kSMARTCARD\_DisableTransmitterMode  
enumerator kSMARTCARD\_ResetWaitTimeMultiplier

enum \_smartcard\_card\_voltage\_class

Defines Smart card interface voltage class values.

*Values:*

enumerator kSMARTCARD\_VoltageClassUnknown  
enumerator kSMARTCARD\_VoltageClassA5\_0V  
enumerator kSMARTCARD\_VoltageClassB3\_3V  
enumerator kSMARTCARD\_VoltageClassC1\_8V

enum \_smartcard\_transfer\_state

Defines Smart card I/O transfer states.

*Values:*

enumerator kSMARTCARD\_IdleState  
enumerator kSMARTCARD\_WaitingForTSSState  
enumerator kSMARTCARD\_InvalidTSDetectedState  
enumerator kSMARTCARD\_ReceivingState  
enumerator kSMARTCARD\_TransmittingState

enum \_smartcard\_reset\_type

Defines Smart card reset types.

*Values:*

enumerator kSMARTCARD\_ColdReset  
enumerator kSMARTCARD\_WarmReset  
enumerator kSMARTCARD\_NoColdReset  
enumerator kSMARTCARD\_NoWarmReset

enum \_smartcard\_transport\_type

Defines Smart card transport protocol types.

*Values:*

enumerator kSMARTCARD\_T0Transport  
enumerator kSMARTCARD\_T1Transport

enum *\_smartcard\_parity\_type*

Defines Smart card data parity types.

*Values:*

enumerator kSMARTCARD\_EvenParity

enumerator kSMARTCARD\_OddParity

enum *\_smartcard\_card\_convention*

Defines data Convention format.

*Values:*

enumerator kSMARTCARD\_DirectConvention

enumerator kSMARTCARD\_InverseConvention

enum *\_smartcard\_interface\_control*

Defines Smart card interface IC control types.

*Values:*

enumerator kSMARTCARD\_InterfaceSetVcc

enumerator kSMARTCARD\_InterfaceSetClockToResetDelay

enumerator kSMARTCARD\_InterfaceReadStatus

enum *\_smartcard\_direction*

Defines transfer direction.

*Values:*

enumerator kSMARTCARD\_Receive

enumerator kSMARTCARD\_Transmit

typedef enum *\_smartcard\_control* smartcard\_control\_t

Control codes for the Smart card protocol timers and misc.

typedef enum *\_smartcard\_card\_voltage\_class* smartcard\_card\_voltage\_class\_t

Defines Smart card interface voltage class values.

typedef enum *\_smartcard\_transfer\_state* smartcard\_transfer\_state\_t

Defines Smart card I/O transfer states.

typedef enum *\_smartcard\_reset\_type* smartcard\_reset\_type\_t

Defines Smart card reset types.

typedef enum *\_smartcard\_transport\_type* smartcard\_transport\_type\_t

Defines Smart card transport protocol types.

typedef enum *\_smartcard\_parity\_type* smartcard\_parity\_type\_t

Defines Smart card data parity types.

typedef enum *\_smartcard\_card\_convention* smartcard\_card\_convention\_t

Defines data Convention format.

typedef enum *\_smartcard\_interface\_control* smartcard\_interface\_control\_t

Defines Smart card interface IC control types.

typedef enum *\_smartcard\_direction* smartcard\_direction\_t

Defines transfer direction.

`typedef void (*smartcard_interface_callback_t)(void *smartcardContext, void *param)`  
Smart card interface interrupt callback function type.

`typedef void (*smartcard_transfer_callback_t)(void *smartcardContext, void *param)`  
Smart card transfer interrupt callback function type.

`typedef void (*smartcard_time_delay_t)(uint32_t us)`  
Time Delay function used to passive waiting using RTOS [us].

`typedef struct _smartcard_card_params smartcard_card_params_t`  
Defines card-specific parameters for Smart card driver.

`typedef struct _smartcard_timers_state smartcard_timers_state_t`  
Smart card defines the state of the EMV timers in the Smart card driver.

`typedef struct _smartcard_interface_config smartcard_interface_config_t`  
Defines user specified configuration of Smart card interface.

`typedef struct _smartcard_xfer smartcard_xfer_t`  
Defines user transfer structure used to initialize transfer.

`typedef struct _smartcard_context smartcard_context_t`  
Runtime state of the Smart card driver.

`SMARTCARD_INIT_DELAY_CLOCK_CYCLES`  
Smart card global define which specify number of clock cycles until initial 'TS' character has to be received.

`SMARTCARD_EMV_ATR_DURATION_ETU`  
Smart card global define which specify number of clock cycles during which ATR string has to be received.

`SMARTCARD_TS_DIRECT_CONVENTION`  
Smart card specification initial TS character definition of direct convention.

`SMARTCARD_TS_INVERSE_CONVENTION`  
Smart card specification initial TS character definition of inverse convention.

`struct _smartcard_card_params`  
`#include <fsl_smartcard.h>` Defines card-specific parameters for Smart card driver.

### Public Members

`uint16_t Fi`  
4 bits Fi - clock rate conversion integer

`uint8_t fMax`  
Maximum Smart card frequency in MHz

`uint8_t WI`  
8 bits WI - work wait time integer

`uint8_t Di`  
4 bits DI - baud rate divisor

`uint8_t BWI`  
4 bits BWI - block wait time integer

`uint8_t CWI`  
4 bits CWI - character wait time integer

uint8\_t BGI

4 bits BGI - block guard time integer

uint8\_t GTN

8 bits GTN - extended guard time integer

uint8\_t IFSC

Indicates IFSC value of the card

uint8\_t modeNegotiable

Indicates if the card acts in negotiable or a specific mode.

uint8\_t currentD

4 bits DI - current baud rate divisor

uint8\_t status

Indicates smart card status

bool t0Indicated

Indicates if T=0 indicated in TD1 byte

bool t1Indicated

Indicates if T=1 indicated in TD2 byte

bool atrComplete

Indicates whether the ATR received from the card was complete or not

bool atrValid

Indicates whether the ATR received from the card was valid or not

bool present

Indicates if a smart card is present

bool active

Indicates if the smart card is activated

bool faulty

Indicates whether smart card/interface is faulty

*smartcard\_card\_convention\_t* convention

Card convention, kSMARTCARD\_DirectConvention for direct convention, kSMARTCARD\_InverseConvention for inverse convention

struct \_\_smartcard\_timers\_state

*#include <fsl\_smartcard.h>* Smart card defines the state of the EMV timers in the Smart card driver.

### Public Members

volatile bool adtExpired

Indicates whether ADT timer expired

volatile bool wwtExpired

Indicates whether WWT timer expired

volatile bool cwtExpired

Indicates whether CWT timer expired

volatile bool bwtExpired

Indicates whether BWT timer expired

volatile bool initCharTimerExpired

Indicates whether reception timer for initialization character (TS) after the RST has expired

struct \_\_smartcard\_\_interface\_\_config

*#include <fsl\_smartcard.h>* Defines user specified configuration of Smart card interface.

### Public Members

uint32\_t smartCardClock

Smart card interface clock [Hz]

uint32\_t clockToResetDelay

Indicates clock to RST apply delay [smart card clock cycles]

uint8\_t clockModule

Smart card clock module number

uint8\_t clockModuleChannel

Smart card clock module channel number

uint8\_t clockModuleSourceClock

Smart card clock module source clock [e.g., BusClk]

*smartcard\_card\_voltage\_class\_t* vcc

Smart card voltage class

uint8\_t controlPort

Smart card PHY control port instance

uint8\_t controlPin

Smart card PHY control pin instance

uint8\_t irqPort

Smart card PHY Interrupt port instance

uint8\_t irqPin

Smart card PHY Interrupt pin instance

uint8\_t resetPort

Smart card reset port instance

uint8\_t resetPin

Smart card reset pin instance

uint8\_t vsel0Port

Smart card PHY Vsel0 control port instance

uint8\_t vsel0Pin

Smart card PHY Vsel0 control pin instance

uint8\_t vsel1Port

Smart card PHY Vsel1 control port instance

uint8\_t vsel1Pin

Smart card PHY Vsel1 control pin instance

uint8\_t dataPort

Smart card PHY data port instance

uint8\_t dataPin

Smart card PHY data pin instance

uint8\_t dataPinMux

Smart card PHY data pin mux option

uint8\_t tsTimerId

Numerical identifier of the External HW timer for Initial character detection

struct \_\_smartcard\_xfer

*#include <fsl\_smartcard.h>* Defines user transfer structure used to initialize transfer.

### Public Members

*smartcard\_direction\_t* direction

Direction of communication. (RX/TX)

uint8\_t \*buff

The buffer of data.

size\_t size

The number of transferred units.

struct \_\_smartcard\_context

*#include <fsl\_smartcard.h>* Runtime state of the Smart card driver.

### Public Members

void \*base

Smart card module base address

*smartcard\_direction\_t* direction

Direction of communication. (RX/TX)

uint8\_t \*xBuff

The buffer of data being transferred.

volatile size\_t xSize

The number of bytes to be transferred.

volatile bool xIsBusy

True if there is an active transfer.

uint8\_t txFifoEntryCount

Number of data word entries in transmit FIFO.

uint8\_t rxFifoThreshold

The max value of the receiver FIFO threshold.

*smartcard\_interface\_callback\_t* interfaceCallback

Callback to invoke after interface IC raised interrupt.

*smartcard\_transfer\_callback\_t* transferCallback

Callback to invoke after transfer event occur.

void \*interfaceCallbackParam

Interface callback parameter pointer.

void \*transferCallbackParam

Transfer callback parameter pointer.



*smartcard\_time\_delay\_t* timeDelay

Function which handles time delay defined by user or RTOS.

*smartcard\_reset\_type\_t* resetType

Indicates whether a Cold reset or Warm reset was requested.

*smartcard\_transport\_type\_t* tType

Indicates current transfer protocol (T0 or T1)

volatile *smartcard\_transfer\_state\_t* transferState

Indicates the current transfer state

*smartcard\_timers\_state\_t* timersState

Indicates the state of different protocol timers used in driver

*smartcard\_card\_params\_t* cardParams

Smart card parameters(ATR and current) and interface slots states(ATR and current)

uint8\_t IFSD

Indicates the terminal IFSD

*smartcard\_parity\_type\_t* parity

Indicates current parity even/odd

volatile bool rxtCrossed

Indicates whether RXT thresholds has been crossed

volatile bool txtCrossed

Indicates whether TXT thresholds has been crossed

volatile bool wtxRequested

Indicates whether WTX has been requested or not

volatile bool parityError

Indicates whether a parity error has been detected

uint8\_t statusBytes[2]

Used to store Status bytes SW1, SW2 of the last executed card command response

*smartcard\_interface\_config\_t* interfaceConfig

Smart card interface configuration structure

bool abortTransfer

Used to abort transfer.

## 2.53 Smart Card EMV SIM Driver

void SMARTCARD\_EMVSIM\_GetDefaultConfig(*smartcard\_card\_params\_t* \*cardParams)

Fills in the *smartcard\_card\_params* structure with default values according to the EMV 4.3 specification.

### Parameters

- *cardParams* – The configuration structure of type *smartcard\_interface\_config\_t*. Function fill in members: *Fi* = 372; *Di* = 1; *currentD* = 1; *WI* = 0x0A; *GTN* = 0x00; with default values.

```
status_t SMARTCARD_EMVSIM_Init(EMVSIM_Type *base, smartcard_context_t *context,  
                                uint32_t srcClock_Hz)
```

Initializes an EMVSIM peripheral for the Smart card/ISO-7816 operation.

This function un-gates the EMVSIM clock, initializes the module to EMV default settings, configures the IRQ, enables the module-level interrupt to the core and, initializes the driver context.

#### Parameters

- base – The EMVSIM peripheral base address.
- context – A pointer to the smart card driver context structure.
- srcClock\_Hz – Smart card clock generation module source clock.

#### Returns

An error code or `kStatus_SMARTCARD_Success`.

```
void SMARTCARD_EMVSIM_Deinit(EMVSIM_Type *base)
```

This function disables the EMVSIM interrupts, disables the transmitter and receiver, flushes the FIFOs, and gates EMVSIM clock in SIM.

#### Parameters

- base – The EMVSIM module base address.

```
int32_t SMARTCARD_EMVSIM_GetTransferRemainingBytes(EMVSIM_Type *base,  
                                                    smartcard_context_t *context)
```

Returns whether the previous EMVSIM transfer has finished.

When performing an async transfer, call this function to ascertain the context of the current transfer: in progress (or busy) or complete (success). If the transfer is still in progress, the user can obtain the number of words that have not been transferred.

#### Parameters

- base – The EMVSIM module base address.
- context – A pointer to a smart card driver context structure.

#### Returns

The number of bytes not transferred.

```
status_t SMARTCARD_EMVSIM_AbortTransfer(EMVSIM_Type *base, smartcard_context_t  
                                         *context)
```

Terminates an asynchronous EMVSIM transfer early.

During an async EMVSIM transfer, the user can terminate the transfer early if the transfer is still in progress.

#### Parameters

- base – The EMVSIM peripheral address.
- context – A pointer to a smart card driver context structure.

#### Return values

- `kStatus_SMARTCARD_Success` – The transmit abort was successful.
- `kStatus_SMARTCARD_NoTransmitInProgress` – No transmission is currently in progress.

```
status_t SMARTCARD_EMVSIM_TransferNonBlocking(EMVSIM_Type *base,  
                                              smartcard_context_t *context,  
                                              smartcard_xfer_t *xfer)
```

Transfer data using interrupts.

A non-blocking (also known as asynchronous) function means that the function returns immediately after initiating the transfer function. The application has to get the transfer status to see when the transfer is complete. In other words, after calling the non-blocking (asynchronous) transfer function, the application must get the transfer status to check if the transmit is completed or not.

#### Parameters

- `base` – The EMVSIM peripheral base address.
- `context` – A pointer to a smart card driver context structure.
- `xfer` – A pointer to the smart card transfer structure where the linked buffers and sizes are stored.

#### Returns

An error code or `kStatus_SMARTCARD_Success`.

```
status_t SMARTCARD__EMVSIM__Control(EMVSIM_Type *base, smartcard_context_t *context,
                                     smartcard_control_t control, uint32_t param)
```

Controls the EMVSIM module per different user request.

return `kStatus_SMARTCARD_Success` in success. return `kStatus_SMARTCARD_OtherError` in case of error.

#### Parameters

- `base` – The EMVSIM peripheral base address.
- `context` – A pointer to a smart card driver context structure.
- `control` – Control type.
- `param` – Integer value of specific to control command.

```
void SMARTCARD__EMVSIM__IRQHandler(EMVSIM_Type *base, smartcard_context_t *context)
```

Handles EMVSIM module interrupts.

#### Parameters

- `base` – The EMVSIM peripheral base address.
- `context` – A pointer to a smart card driver context structure.

```
enum __emvsim_gpc_clock_select
```

General Purpose Counter clock selections.

*Values:*

```
enumerator kEMVSIM__GPCClockDisable
    Disabled
```

```
enumerator kEMVSIM__GPCCardClock
    Card clock
```

```
enumerator kEMVSIM__GPCRxClock
    Receive clock
```

```
enumerator kEMVSIM__GPCTxClock
    Transmit ETU clock
```

enum `_presence_detect_edge`

EMVSIM card presence detection edge control.

*Values:*

enumerator `kEMVSIM_DetectOnFallingEdge`

Presence detected on the falling edge

enumerator `kEMVSIM_DetectOnRisingEdge`

Presence detected on the rising edge

enum `_presence_detect_status`

EMVSIM card presence detection status.

*Values:*

enumerator `kEMVSIM_DetectPinIsLow`

Presence detected pin is logic low

enumerator `kEMVSIM_DetectPinIsHigh`

Presence detected pin is logic high

typedef enum `_emvsim_gpc_clock_select` `emvsim_gpc_clock_select_t`

General Purpose Counter clock selections.

typedef enum `_presence_detect_edge` `emvsim_presence_detect_edge_t`

EMVSIM card presence detection edge control.

typedef enum `_presence_detect_status` `emvsim_presence_detect_status_t`

EMVSIM card presence detection status.

`SMARTCARD_EMV_RX_NACK_THRESHOLD`

EMV RX NACK interrupt generation threshold.

`SMARTCARD_EMV_TX_NACK_THRESHOLD`

EMV TX NACK interrupt generation threshold.

`SMARTCARD_WWT_ADJUSTMENT`

Smart card Word Wait Timer adjustment value.

`SMARTCARD_CWT_ADJUSTMENT`

Smart card Character Wait Timer adjustment value.

## 2.54 Smart Card PHY Driver

void `SMARTCARD_PHY_GetDefaultConfig`(`smartcard_interface_config_t` \*config)

Fills in the configuration structure with default values.

### Parameters

- `config` – The Smart card user configuration structure which contains configuration structure of type `smartcard_interface_config_t`. Function fill in members: `clockToResetDelay` = 42000, `vcc` = `kSmartcardVoltageClassB3_3V`, with default values.

`status_t` `SMARTCARD_PHY_Init`(void \*base, `smartcard_interface_config_t` const \*config, `uint32_t` srcClock\_Hz)

Initializes a Smart card interface instance.

### Parameters

- `base` – The Smart card peripheral base address.

- `config` – The user configuration structure of type `smartcard_interface_config_t`. Call the function `SMARTCARD_PHY_GetDefaultConfig()` to fill the configuration structure.
- `srcClock_Hz` – Smart card clock generation module source clock.

**Return values**

`kStatus_SMARTCARD_Success` – or `kStatus_SMARTCARD_OtherError` in case of error.

`void SMARTCARD_PHY_Deinit(void *base, smartcard_interface_config_t const *config)`

De-initializes a Smart card interface, stops the Smart card clock, and disables the VCC.

**Parameters**

- `base` – The Smart card peripheral module base address.
- `config` – The user configuration structure of type `smartcard_interface_config_t`.

`status_t SMARTCARD_PHY_Activate(void *base, smartcard_context_t *context, smartcard_reset_type_t resetType)`

Activates the Smart card IC.

**Parameters**

- `base` – The Smart card peripheral module base address.
- `context` – A pointer to a Smart card driver context structure.
- `resetType` – type of reset to be performed, possible values = `kSmartcardColdReset`, `kSmartcardWarmReset`

**Return values**

`kStatus_SMARTCARD_Success` – or `kStatus_SMARTCARD_OtherError` in case of error.

`status_t SMARTCARD_PHY_Deactivate(void *base, smartcard_context_t *context)`

De-activates the Smart card IC.

**Parameters**

- `base` – The Smart card peripheral module base address.
- `context` – A pointer to a Smart card driver context structure.

**Return values**

`kStatus_SMARTCARD_Success` – or `kStatus_SMARTCARD_OtherError` in case of error.

`status_t SMARTCARD_PHY_Control(void *base, smartcard_context_t *context, smartcard_interface_control_t control, uint32_t param)`

Controls the Smart card interface IC.

**Parameters**

- `base` – The Smart card peripheral module base address.
- `context` – A pointer to a Smart card driver context structure.
- `control` – A interface command type.
- `param` – Integer value specific to control type

**Return values**

`kStatus_SMARTCARD_Success` – or `kStatus_SMARTCARD_OtherError` in case of error.

SMARTCARD\_ATR\_DURATION\_ADJUSTMENT

Smart card definition which specifies the adjustment number of clock cycles during which an ATR string has to be received.

SMARTCARD\_INIT\_DELAY\_CLOCK\_CYCLES\_ADJUSTMENT

Smart card definition which specifies the adjustment number of clock cycles until an initial 'TS' character has to be received.

## 2.55 Smart Card PHY EMVSIM Driver

## 2.56 Smart Card PHY TDA8035 Driver

## 2.57 SMC: System Mode Controller Driver

```
static inline void SMC_GetVersionId(SMC_Type *base, smc_version_id_t *versionId)
```

Gets the SMC version ID.

This function gets the SMC version ID, including major version number, minor version number, and feature specification number.

### Parameters

- base – SMC peripheral base address.
- versionId – Pointer to the version ID structure.

```
void SMC_GetParam(SMC_Type *base, smc_param_t *param)
```

Gets the SMC parameter.

This function gets the SMC parameter including the enabled power modes.

### Parameters

- base – SMC peripheral base address.
- param – Pointer to the SMC param structure.

```
static inline void SMC_SetPowerModeProtection(SMC_Type *base, uint8_t allowedModes)
```

Configures all power mode protection settings.

This function configures the power mode protection settings for supported power modes in the specified chip family. The available power modes are defined in the `smc_power_mode_protection_t`. This should be done at an early system level initialization stage. See the reference manual for details. This register can only write once after the power reset.

The allowed modes are passed as bit map. For example, to allow LLS and VLLS, use `SMC_SetPowerModeProtection(kSMC_AllowPowerModeVlls | kSMC_AllowPowerModeVlps)`. To allow all modes, use `SMC_SetPowerModeProtection(kSMC_AllowPowerModeAll)`.

### Parameters

- base – SMC peripheral base address.
- allowedModes – Bitmap of the allowed power modes.

static inline *smc\_power\_state\_t* SMC\_GetPowerModeState(SMC\_Type \*base)

Gets the current power mode status.

This function returns the current power mode status. After the application switches the power mode, it should always check the status to check whether it runs into the specified mode or not. The application should check this mode before switching to a different mode. The system requires that only certain modes can switch to other specific modes. See the reference manual for details and the *smc\_power\_state\_t* for information about the power status.

#### Parameters

- *base* – SMC peripheral base address.

#### Returns

Current power mode status.

void SMC\_PreEnterStopModes(void)

Prepares to enter stop modes.

This function should be called before entering STOP/VLPS/LLS/VLLS modes.

void SMC\_PostExitStopModes(void)

Recovers after wake up from stop modes.

This function should be called after wake up from STOP/VLPS/LLS/VLLS modes. It is used with SMC\_PreEnterStopModes.

void SMC\_PreEnterWaitModes(void)

Prepares to enter wait modes.

This function should be called before entering WAIT/VLPW modes.

void SMC\_PostExitWaitModes(void)

Recovers after wake up from stop modes.

This function should be called after wake up from WAIT/VLPW modes. It is used with SMC\_PreEnterWaitModes.

*status\_t* SMC\_SetPowerModeRun(SMC\_Type \*base)

Configures the system to RUN power mode.

#### Parameters

- *base* – SMC peripheral base address.

#### Returns

SMC configuration error code.

*status\_t* SMC\_SetPowerModeHsrn(SMC\_Type \*base)

Configures the system to HSRUN power mode.

#### Parameters

- *base* – SMC peripheral base address.

#### Returns

SMC configuration error code.

*status\_t* SMC\_SetPowerModeWait(SMC\_Type \*base)

Configures the system to WAIT power mode.

#### Parameters

- *base* – SMC peripheral base address.

#### Returns

SMC configuration error code.

*status\_t* SMC\_SetPowerModeStop(SMC\_Type \*base, *smc\_partial\_stop\_option\_t* option)

Configures the system to Stop power mode.

**Parameters**

- base – SMC peripheral base address.
- option – Partial Stop mode option.

**Returns**

SMC configuration error code.

*status\_t* SMC\_SetPowerModeVlpr(SMC\_Type \*base, bool wakeupMode)

Configures the system to VLPR power mode.

**Parameters**

- base – SMC peripheral base address.
- wakeupMode – Enter Normal Run mode if true, else stay in VLPR mode.

**Returns**

SMC configuration error code.

*status\_t* SMC\_SetPowerModeVlpr(SMC\_Type \*base)

Configures the system to VLPW power mode.

**Parameters**

- base – SMC peripheral base address.

**Returns**

SMC configuration error code.

*status\_t* SMC\_SetPowerModeVlps(SMC\_Type \*base)

Configures the system to VLPS power mode.

**Parameters**

- base – SMC peripheral base address.

**Returns**

SMC configuration error code.

*status\_t* SMC\_SetPowerModeLls(SMC\_Type \*base, const *smc\_power\_mode\_lls\_config\_t* \*config)

Configures the system to LLS power mode.

**Parameters**

- base – SMC peripheral base address.
- config – The LLS power mode configuration structure

**Returns**

SMC configuration error code.

*status\_t* SMC\_SetPowerModeVlls(SMC\_Type \*base, const *smc\_power\_mode\_vlls\_config\_t* \*config)

Configures the system to VLLS power mode.

**Parameters**

- base – SMC peripheral base address.
- config – The VLLS power mode configuration structure.

**Returns**

SMC configuration error code.

FSL\_SMC\_DRIVER\_VERSION

SMC driver version.



enum \_smc\_power\_mode\_protection

Power Modes Protection.

*Values:*

enumerator kSMC\_\_AllowPowerModeVlls

Allow Very-low-leakage Stop Mode.

enumerator kSMC\_\_AllowPowerModeLls

Allow Low-leakage Stop Mode.

enumerator kSMC\_\_AllowPowerModeVlp

Allow Very-Low-power Mode.

enumerator kSMC\_\_AllowPowerModeHsrn

Allow High-speed Run mode.

enumerator kSMC\_\_AllowPowerModeAll

Allow all power mode.

enum \_smc\_power\_state

Power Modes in PMSTAT.

*Values:*

enumerator kSMC\_\_PowerStateRun

0000\_0001 - Current power mode is RUN

enumerator kSMC\_\_PowerStateStop

0000\_0010 - Current power mode is STOP

enumerator kSMC\_\_PowerStateVlpr

0000\_0100 - Current power mode is VLPR

enumerator kSMC\_\_PowerStateVlpw

0000\_1000 - Current power mode is VLPW

enumerator kSMC\_\_PowerStateVlps

0001\_0000 - Current power mode is VLPS

enumerator kSMC\_\_PowerStateLls

0010\_0000 - Current power mode is LLS

enumerator kSMC\_\_PowerStateVlls

0100\_0000 - Current power mode is VLLS

enumerator kSMC\_\_PowerStateHsrn

1000\_0000 - Current power mode is HSRUN

enum \_smc\_run\_mode

Run mode definition.

*Values:*

enumerator kSMC\_\_RunNormal

Normal RUN mode.

enumerator kSMC\_\_RunVlpr

Very-low-power RUN mode.

enumerator kSMC\_\_Hsrn

High-speed Run mode (HSRUN).

enum `_smc_stop_mode`

Stop mode definition.

*Values:*

enumerator `kSMC_StopNormal`

Normal STOP mode.

enumerator `kSMC_StopVlps`

Very-low-power STOP mode.

enumerator `kSMC_StopLls`

Low-leakage Stop mode.

enumerator `kSMC_StopVlls`

Very-low-leakage Stop mode.

enum `_smc_stop_submode`

VLLS/LLS stop sub mode definition.

*Values:*

enumerator `kSMC_StopSub0`

Stop submode 0, for VLLS0/LLS0.

enumerator `kSMC_StopSub1`

Stop submode 1, for VLLS1/LLS1.

enumerator `kSMC_StopSub2`

Stop submode 2, for VLLS2/LLS2.

enumerator `kSMC_StopSub3`

Stop submode 3, for VLLS3/LLS3.

enum `_smc_partial_stop_mode`

Partial STOP option.

*Values:*

enumerator `kSMC_PartialStop`

STOP - Normal Stop mode

enumerator `kSMC_PartialStop1`

Partial Stop with both system and bus clocks disabled

enumerator `kSMC_PartialStop2`

Partial Stop with system clock disabled and bus clock enabled

`_smc_status`, SMC configuration status.

*Values:*

enumerator `kStatus_SMC_StopAbort`

Entering Stop mode is abort

typedef enum `_smc_power_mode_protection` `smc_power_mode_protection_t`

Power Modes Protection.

typedef enum `_smc_power_state` `smc_power_state_t`

Power Modes in PMSTAT.

typedef enum `_smc_run_mode` `smc_run_mode_t`

Run mode definition.

typedef enum *\_smc\_stop\_mode* smc\_stop\_mode\_t  
Stop mode definition.

typedef enum *\_smc\_stop\_submode* smc\_stop\_submode\_t  
VLLS/LLS stop sub mode definition.

typedef enum *\_smc\_partial\_stop\_mode* smc\_partial\_stop\_option\_t  
Partial STOP option.

typedef struct *\_smc\_version\_id* smc\_version\_id\_t  
IP version ID definition.

typedef struct *\_smc\_param* smc\_param\_t  
IP parameter definition.

typedef struct *\_smc\_power\_mode\_lls\_config* smc\_power\_mode\_lls\_config\_t  
SMC Low-Leakage Stop power mode configuration.

typedef struct *\_smc\_power\_mode\_vlls\_config* smc\_power\_mode\_vlls\_config\_t  
SMC Very Low-Leakage Stop power mode configuration.

struct *\_smc\_version\_id*  
*#include <fsl\_smc.h>* IP version ID definition.

### Public Members

uint16\_t feature  
Feature Specification Number.

uint8\_t minor  
Minor version number.

uint8\_t major  
Major version number.

struct *\_smc\_param*  
*#include <fsl\_smc.h>* IP parameter definition.

### Public Members

bool hsruntimeEnable  
HSRUN mode enable.

bool llsEnable  
LLS mode enable.

bool lls2Enable  
LLS2 mode enable.

bool vlls0Enable  
VLLS0 mode enable.

struct *\_smc\_power\_mode\_lls\_config*  
*#include <fsl\_smc.h>* SMC Low-Leakage Stop power mode configuration.

**Public Members**

*smc\_stop\_submode\_t* subMode  
Low-leakage Stop sub-mode

bool enableLpoClock  
Enable LPO clock in LLS mode

struct \_\_smc\_power\_mode\_vlls\_config  
*#include <fsl\_smc.h>* SMC Very Low-Leakage Stop power mode configuration.

**Public Members**

*smc\_stop\_submode\_t* subMode  
Very Low-leakage Stop sub-mode

bool enablePorDetectInVlls0  
Enable Power on reset detect in VLLS mode

bool enableRam2InVlls2  
Enable RAM2 power in VLLS2

bool enableLpoClock  
Enable LPO clock in VLLS mode

## 2.58 TPM: Timer PWM Module

uint32\_t TPM\_GetInstance(TPM\_Type \*base)  
Gets the instance from the base address.

**Parameters**

- base – TPM peripheral base address

**Returns**

The TPM instance

void TPM\_Init(TPM\_Type \*base, const *tpm\_config\_t* \*config)  
Ungates the TPM clock and configures the peripheral for basic operation.

---

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

---

**Parameters**

- base – TPM peripheral base address
- config – Pointer to user's TPM config structure.

void TPM\_Deinit(TPM\_Type \*base)  
Stops the counter and gates the TPM clock.

**Parameters**

- base – TPM peripheral base address

void TPM\_GetDefaultConfig(*tpm\_config\_t* \*config)  
Fill in the TPM config struct with the default settings.  
The default values are:

```

config->prescale = kTPM_Prescale_Divide_1;
config->useGlobalTimeBase = false;
config->syncGlobalTimeBase = false;
config->dozeEnable = false;
config->dbgMode = false;
config->enableReloadOnTrigger = false;
config->enableStopOnOverflow = false;
config->enableStartOnTrigger = false;
#if FSL_FEATURE_TPM_HAS_PAUSE_COUNTER_ON_TRIGGER
config->enablePauseOnTrigger = false;
#endif
config->triggerSelect = kTPM_Trigger_Select_0;
#if FSL_FEATURE_TPM_HAS_EXTERNAL_TRIGGER_SELECTION
config->triggerSource = kTPM_TriggerSource_External;
config->extTriggerPolarity = kTPM_ExtTrigger_Active_High;
#endif
#if defined(FSL_FEATURE_TPM_HAS_POL) && FSL_FEATURE_TPM_HAS_POL
config->chnlPolarity = 0U;
#endif

```

### Parameters

- config – Pointer to user's TPM config structure.

*tpm\_clock\_prescale\_t* TPM\_CalculateCounterClkDiv(TPM\_Type \*base, uint32\_t counterPeriod\_Hz, uint32\_t srcClock\_Hz)

Calculates the counter clock prescaler.

This function calculates the values for SC[PS].

return Calculated clock prescaler value.

### Parameters

- base – TPM peripheral base address
- counterPeriod\_Hz – The desired frequency in Hz which corresponding to the time when the counter reaches the mod value
- srcClock\_Hz – TPM counter clock in Hz

*status\_t* TPM\_SetupPwm(TPM\_Type \*base, const *tpm\_chnl\_pwm\_signal\_param\_t* \*chnlParams, uint8\_t numOfChnls, *tpm\_pwm\_mode\_t* mode, uint32\_t pwmFreq\_Hz, uint32\_t srcClock\_Hz)

Configures the PWM signal parameters.

User calls this function to configure the PWM signals period, mode, dutycycle and edge. Use this function to configure all the TPM channels that will be used to output a PWM signal

### Parameters

- base – TPM peripheral base address
- chnlParams – Array of PWM channel parameters to configure the channel(s)
- numOfChnls – Number of channels to configure, this should be the size of the array passed in
- mode – PWM operation mode, options available in enumeration *tpm\_pwm\_mode\_t*
- pwmFreq\_Hz – PWM signal frequency in Hz
- srcClock\_Hz – TPM counter clock in Hz

**Returns**

kStatus\_Success if the PWM setup was successful, kStatus\_Error on failure

*status\_t* TPM\_UpdatePwmDutycycle(TPM\_Type \*base, *tpm\_chnl\_t* chnlNumber,  
                                  *tpm\_pwm\_mode\_t* currentPwmMode, uint8\_t  
                                  dutyCyclePercent)

Update the duty cycle of an active PWM signal.

**Parameters**

- base – TPM peripheral base address
- chnlNumber – The channel number. In combined mode, this represents the channel pair number
- currentPwmMode – 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)

**Returns**

kStatus\_Success if the PWM setup was successful, kStatus\_Error on failure

void TPM\_UpdateChnlEdgeLevelSelect(TPM\_Type \*base, *tpm\_chnl\_t* chnlNumber, uint8\_t level)

Update the edge level selection for a channel.

---

**Note:** When the TPM has PWM pause level select feature (FSL\_FEATURE\_TPM\_HAS\_PAUSE\_LEVEL\_SELECT = 1), the PWM output cannot be turned off by selecting the output level. In this case, must use TPM\_DisableChannel API to close the PWM output.

---

**Parameters**

- base – TPM peripheral base address
- chnlNumber – The channel number
- level – The level to be set to the ELSnB:ELSnA field; valid values are 00, 01, 10, 11. See the appropriate SoC reference manual for details about this field.

static inline uint8\_t TPM\_GetChannelControlBits(TPM\_Type \*base, *tpm\_chnl\_t* chnlNumber)

Get the channel control bits value (mode, edge and level bit fields).

This function disable the channel by clear all mode and level control bits.

**Parameters**

- base – TPM peripheral base address
- chnlNumber – The channel number

**Returns**

The control bits value. This is the logical OR of members of the enumeration *tpm\_chnl\_control\_bit\_mask\_t*.

static inline void TPM\_DisableChannel(TPM\_Type \*base, *tpm\_chnl\_t* chnlNumber)

Disable the channel.

This function disable the channel by clear all mode and level control bits.

**Parameters**

- base – TPM peripheral base address
- chnlNumber – The channel number

```
static inline void TPM_EnableChannel(TPM_Type *base, tpm_chnl_t chnlNumber, uint8_t control)
```

Enable the channel according to mode and level configs.

This function enable the channel output according to input mode/level config parameters.

#### Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- control – The control bits value. This is the logical OR of members of the enumeration *tpm\_chnl\_control\_bit\_mask\_t*.

```
void TPM_SetupInputCapture(TPM_Type *base, tpm_chnl_t chnlNumber, tpm_input_capture_edge_t captureMode)
```

Enables capturing an input signal on the channel using the function parameters.

When the edge specified in the captureMode argument occurs on the channel, the TPM counter is captured into the CnV register. The user has to read the CnV register separately to get this value.

#### Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- captureMode – Specifies which edge to capture

```
void TPM_SetupOutputCompare(TPM_Type *base, tpm_chnl_t chnlNumber, tpm_output_compare_mode_t compareMode, uint32_t compareValue)
```

Configures the TPM to generate timed pulses.

When the TPM counter matches the value of compareVal argument (this is written into CnV reg), the channel output is changed based on what is specified in the compareMode argument.

#### Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- compareMode – Action to take on the channel output when the compare condition is met
- compareValue – Value to be programmed in the CnV register.

```
void TPM_SetupDualEdgeCapture(TPM_Type *base, tpm_chnl_t chnlPairNumber, const tpm_dual_edge_capture_param_t *edgeParam, uint32_t filterValue)
```

Configures the dual edge capture mode of the TPM.

This function allows to measure a pulse width of the signal on the input of channel of a channel pair. The filter function is disabled if the filterVal argument passed is zero.

#### Parameters

- base – TPM peripheral base address
- chnlPairNumber – The TPM channel pair number; options are 0, 1, 2, 3
- edgeParam – Sets up the dual edge capture function
- filterValue – Filter value, specify 0 to disable filter.

```
void TPM_SetupQuadDecode(TPM_Type *base, const tpm_phase_params_t *phaseAParams,  
                        const tpm_phase_params_t *phaseBParams,  
                        tpm_quad_decode_mode_t quadMode)
```

Configures the parameters and activates the quadrature decode mode.

#### Parameters

- base – TPM peripheral base address
- phaseAParams – Phase A configuration parameters
- phaseBParams – Phase B configuration parameters
- quadMode – Selects encoding mode used in quadrature decoder mode

```
static inline void TPM_SetChannelPolarity(TPM_Type *base, tpm_chnl_t chnlNumber, bool  
                                         enable)
```

Set the input and output polarity of each of the channels.

#### Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- enable – true: Set the channel polarity to active high; false: Set the channel polarity to active low;

```
static inline void TPM_EnableChannelExtTrigger(TPM_Type *base, tpm_chnl_t chnlNumber, bool  
                                              enable)
```

Enable external trigger input to be used by channel.

In input capture mode, configures the trigger input that is used by the channel to capture the counter value. In output compare or PWM mode, configures the trigger input used to modulate the channel output. When modulating the output, the output is forced to the channel initial value whenever the trigger is not asserted.

---

**Note:** No matter how many external trigger sources there are, only input trigger 0 and 1 are used. The even numbered channels share the input trigger 0 and the odd numbered channels share the second input trigger 1.

---

#### Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- enable – true: Configures trigger input 0 or 1 to be used by channel; false: Trigger input has no effect on the channel

```
void TPM_EnableInterrupts(TPM_Type *base, uint32_t mask)
```

Enables the selected TPM interrupts.

#### Parameters

- base – TPM peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *tpm\_interrupt\_enable\_t*

```
void TPM_DisableInterrupts(TPM_Type *base, uint32_t mask)
```

Disables the selected TPM interrupts.

#### Parameters

- base – TPM peripheral base address



- `mask` – The interrupts to disable. This is a logical OR of members of the enumeration `tpm_interrupt_enable_t`

`uint32_t TPM_GetEnabledInterrupts(TPM_Type *base)`

Gets the enabled TPM interrupts.

#### Parameters

- `base` – TPM peripheral base address

#### Returns

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

`void TPM_RegisterCallBack(TPM_Type *base, tpm_callback_t callback)`

Register callback.

If channel or overflow interrupt is enabled by the user, then a callback can be registered which will be invoked when the interrupt is triggered.

#### Parameters

- `base` – TPM peripheral base address
- `callback` – Callback function

`static inline uint32_t TPM_GetChannelValue(TPM_Type *base, tpm_chnl_t chnlNumber)`

Gets the TPM channel value.

---

**Note:** The TPM channel value contain the captured TPM counter value for the input modes or the match value for the output modes.

---

#### Parameters

- `base` – TPM peripheral base address
- `chnlNumber` – The channel number

#### Returns

The channle CnV regisyer value.

`static inline uint32_t TPM_GetStatusFlags(TPM_Type *base)`

Gets the TPM status flags.

#### Parameters

- `base` – TPM peripheral base address

#### Returns

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

`static inline void TPM_ClearStatusFlags(TPM_Type *base, uint32_t mask)`

Clears the TPM status flags.

#### Parameters

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

`static inline void TPM_SetTimerPeriod(TPM_Type *base, uint32_t ticks)`

Sets the timer period in units of ticks.

Timers counts from 0 until it equals the count value set here. The count value is written to the MOD register.

---

**Note:**

- a. This API allows the user to use the TPM module as a timer. Do not mix usage of this API with TPM's PWM setup API's.
  - b. Call the utility macros provided in the `fsl_common.h` to convert usec or msec to ticks.
- 

**Parameters**

- `base` – TPM peripheral base address
- `ticks` – A timer period in units of ticks, which should be equal or greater than 1.

`static inline uint32_t TPM_GetCurrentTimerCount(TPM_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` – TPM peripheral base address

**Returns**

The current counter value in ticks

`static inline void TPM_StartTimer(TPM_Type *base, tpm_clock_source_t clockSource)`

Starts the TPM counter.

**Parameters**

- `base` – TPM peripheral base address
- `clockSource` – TPM clock source; once clock source is set the counter will start running

`static inline void TPM_StopTimer(TPM_Type *base)`

Stops the TPM counter.

**Parameters**

- `base` – TPM peripheral base address

`FSL_TPM_DRIVER_VERSION`

TPM driver version 2.3.5.

`enum __tpm_chnl`

List of TPM channels.

---

**Note:** Actual number of available channels is SoC dependent

---

*Values:*

enumerator `kTPM_Chnl_0`

TPM channel number 0

enumerator `kTPM_Chnl_1`

TPM channel number 1

enumerator kTPM\_Chnl\_2  
TPM channel number 2

enumerator kTPM\_Chnl\_3  
TPM channel number 3

enumerator kTPM\_Chnl\_4  
TPM channel number 4

enumerator kTPM\_Chnl\_5  
TPM channel number 5

enumerator kTPM\_Chnl\_6  
TPM channel number 6

enumerator kTPM\_Chnl\_7  
TPM channel number 7

enum \_tpm\_pwm\_mode  
TPM PWM operation modes.

*Values:*

enumerator kTPM\_EdgeAlignedPwm  
Edge aligned PWM

enumerator kTPM\_CenterAlignedPwm  
Center aligned PWM

enumerator kTPM\_CombinedPwm  
Combined PWM (Edge-aligned, center-aligned, or asymmetrical PWMs can be obtained in combined mode using different software configurations)

enum \_tpm\_pwm\_level\_select  
TPM PWM output pulse mode: high-true, low-true or no output.

---

**Note:** When the TPM has PWM pause level select feature, the PWM output cannot be turned off by selecting the output level. In this case, the channel must be closed to close the PWM output.

---

*Values:*

enumerator kTPM\_HighTrue  
High true pulses

enumerator kTPM\_LowTrue  
Low true pulses

enum \_tpm\_pwm\_pause\_level\_select  
TPM PWM output when first enabled or paused: set or clear.

*Values:*

enumerator kTPM\_ClearOnPause  
Clear Output when counter first enabled or paused.

enumerator kTPM\_SetOnPause  
Set Output when counter first enabled or paused.

enum \_tpm\_chnl\_control\_bit\_mask  
List of TPM channel modes and level control bit mask.

*Values:*

enumerator kTPM\_ChnlELSnAMask  
Channel ELSA bit mask.

enumerator kTPM\_ChnlELSnBMask  
Channel ELSB bit mask.

enumerator kTPM\_ChnlMSAMask  
Channel MSA bit mask.

enumerator kTPM\_ChnlMSBMask  
Channel MSB bit mask.

enum \_tpm\_trigger\_select  
Trigger sources available.

This is used for both internal & external trigger sources (external trigger sources available in certain SoC's)

---

**Note:** The actual trigger sources available is SoC-specific.

---

*Values:*

enumerator kTPM\_Trigger\_Select\_0

enumerator kTPM\_Trigger\_Select\_1

enumerator kTPM\_Trigger\_Select\_2

enumerator kTPM\_Trigger\_Select\_3

enumerator kTPM\_Trigger\_Select\_4

enumerator kTPM\_Trigger\_Select\_5

enumerator kTPM\_Trigger\_Select\_6

enumerator kTPM\_Trigger\_Select\_7

enumerator kTPM\_Trigger\_Select\_8

enumerator kTPM\_Trigger\_Select\_9

enumerator kTPM\_Trigger\_Select\_10

enumerator kTPM\_Trigger\_Select\_11

enumerator kTPM\_Trigger\_Select\_12

enumerator kTPM\_Trigger\_Select\_13

enumerator kTPM\_Trigger\_Select\_14

enumerator kTPM\_Trigger\_Select\_15

enum \_tpm\_trigger\_source  
Trigger source options available.

---

**Note:** This selection is available only on some SoC's. For SoC's without this selection, the only trigger source available is internal trigger.

---

*Values:*

enumerator kTPM\_TriggerSource\_External  
Use external trigger input

enumerator kTPM\_TriggerSource\_Internal  
Use internal trigger (channel pin input capture)

enum \_tpm\_ext\_trigger\_polarity  
External trigger source polarity.

---

**Note:** Selects the polarity of the external trigger source.

---

*Values:*

enumerator kTPM\_ExtTrigger\_Active\_High  
External trigger input is active high

enumerator kTPM\_ExtTrigger\_Active\_Low  
External trigger input is active low

enum \_tpm\_output\_compare\_mode  
TPM output compare modes.

*Values:*

enumerator kTPM\_NoOutputSignal  
No channel output when counter reaches CnV

enumerator kTPM\_ToggleOnMatch  
Toggle output

enumerator kTPM\_ClearOnMatch  
Clear output

enumerator kTPM\_SetOnMatch  
Set output

enumerator kTPM\_HighPulseOutput  
Pulse output high

enumerator kTPM\_LowPulseOutput  
Pulse output low

enum \_tpm\_input\_capture\_edge  
TPM input capture edge.

*Values:*

enumerator kTPM\_RisingEdge  
Capture on rising edge only

enumerator kTPM\_FallingEdge  
Capture on falling edge only

enumerator kTPM\_RiseAndFallEdge  
Capture on rising or falling edge

enum \_tpm\_quad\_decode\_mode  
TPM quadrature decode modes.

---

**Note:** This mode is available only on some SoC's.

---

*Values:*

enumerator kTPM\_QuadPhaseEncode  
Phase A and Phase B encoding mode

enumerator kTPM\_QuadCountAndDir  
Count and direction encoding mode

enum \_tpm\_phase\_polarity  
TPM quadrature phase polarities.

*Values:*

enumerator kTPM\_QuadPhaseNormal  
Phase input signal is not inverted

enumerator kTPM\_QuadPhaseInvert  
Phase input signal is inverted

enum \_tpm\_clock\_source  
TPM clock source selection.

*Values:*

enumerator kTPM\_SystemClock  
System clock

enumerator kTPM\_ExternalClock  
External TPM\_EXTCLK pin clock

enumerator kTPM\_ExternalInputTriggerClock  
Selected external input trigger clock

enum \_tpm\_clock\_prescale  
TPM prescale value selection for the clock source.

*Values:*

enumerator kTPM\_Prescale\_Divide\_1  
Divide by 1

enumerator kTPM\_Prescale\_Divide\_2  
Divide by 2

enumerator kTPM\_Prescale\_Divide\_4  
Divide by 4

enumerator kTPM\_Prescale\_Divide\_8  
Divide by 8

enumerator kTPM\_Prescale\_Divide\_16  
Divide by 16

enumerator kTPM\_Prescale\_Divide\_32  
Divide by 32

enumerator kTPM\_Prescale\_Divide\_64  
Divide by 64

enumerator kTPM\_Prescale\_Divide\_128  
Divide by 128

enum \_tpm\_interrupt\_enable  
List of TPM interrupts.

*Values:*

enumerator kTPM\_Chnl0InterruptEnable  
Channel 0 interrupt.

enumerator kTPM\_Chnl1InterruptEnable  
Channel 1 interrupt.

enumerator kTPM\_Chnl2InterruptEnable  
Channel 2 interrupt.

enumerator kTPM\_Chnl3InterruptEnable  
Channel 3 interrupt.

enumerator kTPM\_Chnl4InterruptEnable  
Channel 4 interrupt.

enumerator kTPM\_Chnl5InterruptEnable  
Channel 5 interrupt.

enumerator kTPM\_Chnl6InterruptEnable  
Channel 6 interrupt.

enumerator kTPM\_Chnl7InterruptEnable  
Channel 7 interrupt.

enumerator kTPM\_TimeOverflowInterruptEnable  
Time overflow interrupt.

enum \_tpm\_status\_flags  
List of TPM flags.

*Values:*

enumerator kTPM\_Chnl0Flag  
Channel 0 flag

enumerator kTPM\_Chnl1Flag  
Channel 1 flag

enumerator kTPM\_Chnl2Flag  
Channel 2 flag

enumerator kTPM\_Chnl3Flag  
Channel 3 flag

enumerator kTPM\_Chnl4Flag  
Channel 4 flag

enumerator kTPM\_Chnl5Flag  
Channel 5 flag

enumerator kTPM\_Chnl6Flag  
Channel 6 flag

enumerator kTPM\_Chnl7Flag  
Channel 7 flag

enumerator kTPM\_TimeOverflowFlag  
Time overflow flag

typedef enum \_tpm\_chnl tpm\_chnl\_t  
List of TPM channels.

---

**Note:** Actual number of available channels is SoC dependent

---

typedef enum *\_tpm\_pwm\_mode* tpm\_pwm\_mode\_t

TPM PWM operation modes.

typedef enum *\_tpm\_pwm\_level\_select* tpm\_pwm\_level\_select\_t

TPM PWM output pulse mode: high-true, low-true or no output.

---

**Note:** When the TPM has PWM pause level select feature, the PWM output cannot be turned off by selecting the output level. In this case, the channel must be closed to close the PWM output.

---

typedef enum *\_tpm\_pwm\_pause\_level\_select* tpm\_pwm\_pause\_level\_select\_t

TPM PWM output when first enabled or paused: set or clear.

typedef enum *\_tpm\_chnl\_control\_bit\_mask* tpm\_chnl\_control\_bit\_mask\_t

List of TPM channel modes and level control bit mask.

typedef struct *\_tpm\_chnl\_pwm\_signal\_param* tpm\_chnl\_pwm\_signal\_param\_t

Options to configure a TPM channel's PWM signal.

typedef enum *\_tpm\_trigger\_select* tpm\_trigger\_select\_t

Trigger sources available.

This is used for both internal & external trigger sources (external trigger sources available in certain SoC's)

---

**Note:** The actual trigger sources available is SoC-specific.

---

typedef enum *\_tpm\_trigger\_source* tpm\_trigger\_source\_t

Trigger source options available.

---

**Note:** This selection is available only on some SoC's. For SoC's without this selection, the only trigger source available is internal trigger.

---

typedef enum *\_tpm\_ext\_trigger\_polarity* tpm\_ext\_trigger\_polarity\_t

External trigger source polarity.

---

**Note:** Selects the polarity of the external trigger source.

---

typedef enum *\_tpm\_output\_compare\_mode* tpm\_output\_compare\_mode\_t

TPM output compare modes.

typedef enum *\_tpm\_input\_capture\_edge* tpm\_input\_capture\_edge\_t

TPM input capture edge.

typedef struct *\_tpm\_dual\_edge\_capture\_param* tpm\_dual\_edge\_capture\_param\_t

TPM dual edge capture parameters.

---

**Note:** This mode is available only on some SoC's.

---

typedef enum *\_tpm\_quad\_decode\_mode* tpm\_quad\_decode\_mode\_t

TPM quadrature decode modes.

---

**Note:** This mode is available only on some SoC's.

---



typedef enum *\_tpm\_phase\_polarity* tpm\_phase\_polarity\_t  
TPM quadrature phase polarities.

typedef struct *\_tpm\_phase\_param* tpm\_phase\_params\_t  
TPM quadrature decode phase parameters.

typedef enum *\_tpm\_clock\_source* tpm\_clock\_source\_t  
TPM clock source selection.

typedef enum *\_tpm\_clock\_prescale* tpm\_clock\_prescale\_t  
TPM prescale value selection for the clock source.

typedef struct *\_tpm\_config* tpm\_config\_t  
TPM config structure.

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

The config struct can be made const so it resides in flash

typedef enum *\_tpm\_interrupt\_enable* tpm\_interrupt\_enable\_t  
List of TPM interrupts.

typedef enum *\_tpm\_status\_flags* tpm\_status\_flags\_t  
List of TPM flags.

typedef void (\*tpm\_callback\_t)(TPM\_Type \*base)  
TPM callback function pointer.

#### **Param base**

TPM peripheral base address.

static inline void TPM\_Reset(TPM\_Type \*base)  
Performs a software reset on the TPM module.

Reset all internal logic and registers, except the Global Register. Remains set until cleared by software.

---

**Note:** TPM software reset is available on certain SoC's only

---

#### **Parameters**

- base – TPM peripheral base address

void TPM\_DriverIRQHandler(uint32\_t instance)  
TPM driver IRQ handler common entry.

This function provides the common IRQ request entry for TPM.

#### **Parameters**

- instance – TPM instance.

TPM\_MAX\_COUNTER\_VALUE(x)  
Help macro to get the max counter value.

struct *\_tpm\_chnl\_pwm\_signal\_param*  
*#include <fsl\_tpm.h>* Options to configure a TPM channel's PWM signal.

### Public Members

*tpm\_chnl\_t* chnlNumber

TPM channel to configure. In combined mode (available in some SoC's), this represents the channel pair number

*tpm\_pwm\_pause\_level\_select\_t* pauseLevel

PWM output level when counter first enabled or paused

*tpm\_pwm\_level\_select\_t* level

PWM output active level select

*uint8\_t* dutyCyclePercent

PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=always active signal (100% duty cycle)

*uint8\_t* firstEdgeDelayPercent

Used only in combined PWM mode to generate asymmetrical PWM. Specifies the delay to the first edge in a PWM period. If unsure, leave as 0. Should be specified as percentage of the PWM period, (dutyCyclePercent + firstEdgeDelayPercent) value should be not greater than 100.

*bool* enableComplementary

Used only in combined PWM mode. true: The combined channels output complementary signals; false: The combined channels output same signals;

*tpm\_pwm\_pause\_level\_select\_t* secPauseLevel

Used only in combined PWM mode. Define the second channel output level when counter first enabled or paused

*uint8\_t* deadTimeValue[2]

The dead time value for channel n and n+1 in combined complementary PWM mode. Deadtime insertion is disabled when this value is zero, otherwise deadtime insertion for channel n/n+1 is configured as (deadTimeValue \* 4) clock cycles. deadTimeValue's available range is 0 ~ 15.

*struct* *\_tpm\_dual\_edge\_capture\_param*

*#include <fsl\_tpm.h>* TPM dual edge capture parameters.

---

**Note:** This mode is available only on some SoC's.

---

### Public Members

*bool* enableSwap

true: Use channel n+1 input, channel n input is ignored; false: Use channel n input, channel n+1 input is ignored

*tpm\_input\_capture\_edge\_t* currChanEdgeMode

Input capture edge select for channel n

*tpm\_input\_capture\_edge\_t* nextChanEdgeMode

Input capture edge select for channel n+1

*struct* *\_tpm\_phase\_param*

*#include <fsl\_tpm.h>* TPM quadrature decode phase parameters.

### Public Members

uint32\_t phaseFilterVal

Filter value, filter is disabled when the value is zero

*tpm\_phase\_polarity\_t* phasePolarity

Phase polarity

struct *\_tpm\_config*

*#include <fsl\_tpm.h>* TPM config structure.

This structure holds the configuration settings for the TPM peripheral. To initialize this structure to reasonable defaults, call the *TPM\_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

*tpm\_clock\_prescale\_t* prescale

Select TPM clock prescale value

bool useGlobalTimeBase

true: The TPM channels use an external global time base (the local counter still use for generate overflow interrupt and DMA request); false: All TPM channels use the local counter as their timebase

bool syncGlobalTimeBase

true: The TPM counter is synchronized to the global time base; false: disabled

*tpm\_trigger\_select\_t* triggerSelect

Input trigger to use for controlling the counter operation

*tpm\_trigger\_source\_t* triggerSource

Decides if we use external or internal trigger.

*tpm\_ext\_trigger\_polarity\_t* extTriggerPolarity

when using external trigger source, need selects the polarity of it.

bool enableDoze

true: TPM counter is paused in doze mode; false: TPM counter continues in doze mode

bool enableDebugMode

true: TPM counter continues in debug mode; false: TPM counter is paused in debug mode

bool enableReloadOnTrigger

true: TPM counter is reloaded on trigger; false: TPM counter not reloaded

bool enableStopOnOverflow

true: TPM counter stops after overflow; false: TPM counter continues running after overflow

bool enableStartOnTrigger

true: TPM counter only starts when a trigger is detected; false: TPM counter starts immediately

bool enablePauseOnTrigger

true: TPM counter will pause while trigger remains asserted; false: TPM counter continues running

uint8\_t chnlPolarity

Defines the input/output polarity of the channels in POL register

## 2.59 TRGMUX: Trigger Mux Driver

`static inline void TRGMUX_LockRegister(TRGMUX_Type *base, uint32_t index)`

Sets the flag of the register which is used to mark writeable.

The function sets the flag of the register which is used to mark writeable. Example:

```
TRGMUX_LockRegister(TRGMUX0, kTRGMUX_Trgmux0Dmamux0);
```

### Parameters

- `base` – TRGMUX peripheral base address.
- `index` – The index of the TRGMUX register, see the enum `trgmux_device_t` defined in `<SOC>.h`.

`status_t TRGMUX_SetTriggerSource(TRGMUX_Type *base, uint32_t index, trgmux_trigger_input_t input, uint32_t trigger_src)`

Configures the trigger source of the appointed peripheral.

The function configures the trigger source of the appointed peripheral. Example:

```
TRGMUX_SetTriggerSource(TRGMUX0, kTRGMUX_Trgmux0Dmamux0, kTRGMUX_TriggerInput0,
↪ kTRGMUX_SourcePortPin);
```

### Parameters

- `base` – TRGMUX peripheral base address.
- `index` – The index of the TRGMUX register, see the enum `trgmux_device_t` defined in `<SOC>.h`.
- `input` – The MUX select for peripheral trigger input
- `trigger_src` – The trigger inputs for various peripherals. See the enum `trgmux_source_t` defined in `<SOC>.h`.

### Return values

- `kStatus_Success` – Configured successfully.
- `kStatus_TRGMUX_Locked` – Configuration failed because the register is locked.

`FSL_TRGMUX_DRIVER_VERSION`

TRGMUX driver version.

TRGMUX configure status.

*Values:*

enumerator `kStatus_TRGMUX_Locked`

Configure failed for register is locked

enum `_trgmux_trigger_input`

Defines the MUX select for peripheral trigger input.

*Values:*

enumerator `kTRGMUX_TriggerInput0`

The MUX select for peripheral trigger input 0

enumerator `kTRGMUX_TriggerInput1`

The MUX select for peripheral trigger input 1

enumerator kTRGMUX\_TriggerInput2

The MUX select for peripheral trigger input 2

enumerator kTRGMUX\_TriggerInput3

The MUX select for peripheral trigger input 3

typedef enum *trgmux\_trigger\_input* trgmux\_trigger\_input\_t

Defines the MUX select for peripheral trigger input.

## 2.60 TRNG: True Random Number Generator

FSL\_TRNG\_DRIVER\_VERSION

TRNG driver version 2.0.18.

Current version: 2.0.18

Change log:

- version 2.0.18
  - TRNG health checks now done in software on RT5xx and RT6xx.
- version 2.0.17
  - Added support for RT700.
- version 2.0.16
  - Added support for Dual oscillator mode.
- version 2.0.15
  - Changed TRNG\_USER\_CONFIG\_DEFAULT\_XXX values according to latest recommended by design team.
- version 2.0.14
  - add support for RW610 and RW612
- version 2.0.13
  - After deepsleep it might return error, added clearing bits in TRNG\_GetRandomData() and generating new entropy.
  - Modified reloading entropy in TRNG\_GetRandomData(), for some data length it doesn't reloading entropy correctly.
- version 2.0.12
  - For KW34A4\_SERIES, KW35A4\_SERIES, KW36A4\_SERIES set TRNG\_USER\_CONFIG\_DEFAULT\_OSC\_DIV to kTRNG\_RingOscDiv8.
- version 2.0.11
  - Add clearing pending errors in TRNG\_Init().
- version 2.0.10
  - Fixed doxygen issues.
- version 2.0.9
  - Fix HIS\_CCM metrics issues.
- version 2.0.8
  - For K32L2A41A\_SERIES set TRNG\_USER\_CONFIG\_DEFAULT\_OSC\_DIV to kTRNG\_RingOscDiv4.

- version 2.0.7
  - Fix MISRA 2004 issue rule 12.5.
- version 2.0.6
  - For KW35Z4\_SERIES set TRNG\_USER\_CONFIG\_DEFAULT\_OSC\_DIV to kTRNG\_RingOscDiv8.
- version 2.0.5
  - Add possibility to define default TRNG configuration by device specific preprocessor macros for FRQMIN, FRQMAX and OSCDIV.
- version 2.0.4
  - Fix MISRA-2012 issues.
- Version 2.0.3
  - update TRNG\_Init to restart entropy generation
- Version 2.0.2
  - fix MISRA issues
- Version 2.0.1
  - add support for KL8x and KL28Z
  - update default OSCDIV for K81 to divide by 2

enum \_trng\_sample\_mode

TRNG sample mode. Used by trng\_config\_t.

*Values:*

enumerator kTRNG\_SampleModeVonNeumann

Use von Neumann data in both Entropy shifter and Statistical Checker.

enumerator kTRNG\_SampleModeRaw

Use raw data into both Entropy shifter and Statistical Checker.

enumerator kTRNG\_SampleModeVonNeumannRaw

Use von Neumann data in Entropy shifter. Use raw data into Statistical Checker.

enum \_trng\_clock\_mode

TRNG clock mode. Used by trng\_config\_t.

*Values:*

enumerator kTRNG\_ClockModeRingOscillator

Ring oscillator is used to operate the TRNG (default).

enumerator kTRNG\_ClockModeSystem

System clock is used to operate the TRNG. This is for test use only, and indeterminate results may occur.

enum \_trng\_ring\_osc\_div

TRNG ring oscillator divide. Used by trng\_config\_t.

*Values:*

enumerator kTRNG\_RingOscDiv0

Ring oscillator with no divide

enumerator kTRNG\_RingOscDiv2

Ring oscillator divided-by-2.

enumerator kTRNG\_RingOscDiv4  
Ring oscillator divided-by-4.

enumerator kTRNG\_RingOscDiv8  
Ring oscillator divided-by-8.

typedef enum *\_trng\_sample\_mode* trng\_sample\_mode\_t  
TRNG sample mode. Used by trng\_config\_t.

typedef enum *\_trng\_clock\_mode* trng\_clock\_mode\_t  
TRNG clock mode. Used by trng\_config\_t.

typedef enum *\_trng\_ring\_osc\_div* trng\_ring\_osc\_div\_t  
TRNG ring oscillator divide. Used by trng\_config\_t.

typedef struct *\_trng\_statistical\_check\_limit* trng\_statistical\_check\_limit\_t  
Data structure for definition of statistical check limits. Used by trng\_config\_t.

typedef struct *\_trng\_user\_config* trng\_config\_t  
Data structure for the TRNG initialization.  
  
This structure initializes the TRNG by calling the TRNG\_Init() function. It contains all TRNG configurations.

*status\_t* TRNG\_GetDefaultConfig(*trng\_config\_t* \*userConfig)  
Initializes the user configuration structure to default values.  
  
This function initializes the configuration structure to default values. The default values are platform dependent.

#### Parameters

- userConfig – User configuration structure.

#### Returns

If successful, returns the kStatus\_TRNG\_Success. Otherwise, it returns an error.

*status\_t* TRNG\_Init(TRNG\_Type \*base, const *trng\_config\_t* \*userConfig)  
Initializes the TRNG.

This function initializes the TRNG. When called, the TRNG entropy generation starts immediately.

#### Parameters

- base – TRNG base address
- userConfig – Pointer to the initialization configuration structure.

#### Returns

If successful, returns the kStatus\_TRNG\_Success. Otherwise, it returns an error.

void TRNG\_Deinit(TRNG\_Type \*base)  
Shuts down the TRNG.

This function shuts down the TRNG.

#### Parameters

- base – TRNG base address.

*status\_t* TRNG\_GetRandomData(TRNG\_Type \*base, void \*data, size\_t dataSize)  
Gets random data.

This function gets random data from the TRNG.

### Parameters

- base – TRNG base address.
- data – Pointer address used to store random data.
- dataSize – Size of the buffer pointed by the data parameter.

### Returns

random data

struct `_trng_statistical_check_limit`

*#include <fsl\_trng.h>* Data structure for definition of statistical check limits. Used by `trng_config_t`.

### Public Members

uint32\_t maximum

Maximum limit.

int32\_t minimum

Minimum limit.

struct `_trng_user_config`

*#include <fsl\_trng.h>* Data structure for the TRNG initialization.

This structure initializes the TRNG by calling the `TRNG_Init()` function. It contains all TRNG configurations.

### Public Members

bool lock

Disable programmability of TRNG registers.

*trng\_clock\_mode\_t* clockMode

Clock mode used to operate TRNG.

*trng\_ring\_osc\_div\_t* ringOscDiv

Ring oscillator divide used by TRNG.

*trng\_sample\_mode\_t* sampleMode

Sample mode of the TRNG ring oscillator.

uint16\_t entropyDelay

Entropy Delay. Defines the length (in system clocks) of each Entropy sample taken.

uint16\_t sampleSize

Sample Size. Defines the total number of Entropy samples that will be taken during Entropy generation.

uint16\_t sparseBitLimit

Sparse Bit Limit which defines the maximum number of consecutive samples that may be discarded before an error is generated. This limit is used only for during von Neumann sampling (enabled by `TRNG_HAL_SetSampleMode()`). Samples are discarded if two consecutive raw samples are both 0 or both 1. If this discarding occurs for a long period of time, it indicates that there is insufficient Entropy.

uint8\_t retryCount

Retry count. It defines the number of times a statistical check may fails during the TRNG Entropy Generation before generating an error.



`uint8_t longRunMaxLimit`

Largest allowable number of consecutive samples of all 1, or all 0, that is allowed during the Entropy generation.

`trng_statistical_check_limit_t monobitLimit`

Maximum and minimum limits for statistical check of number of ones/zero detected during entropy generation.

`trng_statistical_check_limit_t runBit1Limit`

Maximum and minimum limits for statistical check of number of runs of length 1 detected during entropy generation.

`trng_statistical_check_limit_t runBit2Limit`

Maximum and minimum limits for statistical check of number of runs of length 2 detected during entropy generation.

`trng_statistical_check_limit_t runBit3Limit`

Maximum and minimum limits for statistical check of number of runs of length 3 detected during entropy generation.

`trng_statistical_check_limit_t runBit4Limit`

Maximum and minimum limits for statistical check of number of runs of length 4 detected during entropy generation.

`trng_statistical_check_limit_t runBit5Limit`

Maximum and minimum limits for statistical check of number of runs of length 5 detected during entropy generation.

`trng_statistical_check_limit_t runBit6PlusLimit`

Maximum and minimum limits for statistical check of number of runs of length 6 or more detected during entropy generation.

`trng_statistical_check_limit_t pokerLimit`

Maximum and minimum limits for statistical check of “Poker Test”.

`trng_statistical_check_limit_t frequencyCountLimit`

Maximum and minimum limits for statistical check of entropy sample frequency count.

## 2.61 TSI: Touch Sensing Input

`FSL_TSI_DRIVER_VERSION`

TSI driver version.

Version 2.1.3

`enum _tsi_n_consecutive_scans`

TSI number of scan intervals for each electrode.

These constants define the tsi number of consecutive scans in a TSI instance for each electrode.

*Values:*

enumerator `kTSI_ConsecutiveScansNumber_1time`

Once per electrode

enumerator `kTSI_ConsecutiveScansNumber_2time`

Twice per electrode

enumerator kTSI\_ConsecutiveScansNumber\_\_3time  
3 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_4time  
4 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_5time  
5 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_6time  
6 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_7time  
7 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_8time  
8 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_9time  
9 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_10time  
10 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_11time  
11 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_12time  
12 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_13time  
13 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_14time  
14 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_15time  
15 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_16time  
16 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_17time  
17 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_18time  
18 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_19time  
19 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_20time  
20 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_21time  
21 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_22time  
22 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_\_23time  
23 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_24time  
24 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_25time  
25 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_26time  
26 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_27time  
27 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_28time  
28 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_29time  
29 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_30time  
30 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_31time  
31 times consecutive scan

enumerator kTSI\_ConsecutiveScansNumber\_32time  
32 times consecutive scan

enum \_tsi\_electrode\_osc\_prescaler

TSI electrode oscillator prescaler.

These constants define the TSI electrode oscillator prescaler in a TSI instance.

*Values:*

enumerator kTSI\_ElecOscPrescaler\_1div  
Electrode oscillator frequency divided by 1

enumerator kTSI\_ElecOscPrescaler\_2div  
Electrode oscillator frequency divided by 2

enumerator kTSI\_ElecOscPrescaler\_4div  
Electrode oscillator frequency divided by 4

enumerator kTSI\_ElecOscPrescaler\_8div  
Electrode oscillator frequency divided by 8

enumerator kTSI\_ElecOscPrescaler\_16div  
Electrode oscillator frequency divided by 16

enumerator kTSI\_ElecOscPrescaler\_32div  
Electrode oscillator frequency divided by 32

enumerator kTSI\_ElecOscPrescaler\_64div  
Electrode oscillator frequency divided by 64

enumerator kTSI\_ElecOscPrescaler\_128div  
Electrode oscillator frequency divided by 128

enum \_tsi\_analog\_mode

TSI analog mode select.

Set up TSI analog modes in a TSI instance.

*Values:*

enumerator kTSI\_AnalogModeSel\_Capacitive

Active TSI capacitive sensing mode

enumerator kTSI\_AnalogModeSel\_NoiseNoFreqLim

Single threshold noise detection mode with no freq. limitation.

enumerator kTSI\_AnalogModeSel\_NoiseFreqLim

Single threshold noise detection mode with freq. limitation.

enumerator kTSI\_AnalogModeSel\_AutoNoise

Active TSI analog in automatic noise detection mode

enum \_tsi\_reference\_osc\_charge\_current

TSI Reference oscillator charge and discharge current select.

These constants define the TSI Reference oscillator charge current select in a TSI (REFCHRG) instance.

*Values:*

enumerator kTSI\_RefOscChargeCurrent\_500nA

Reference oscillator charge current is 500  $\mu$ A

enumerator kTSI\_RefOscChargeCurrent\_1uA

Reference oscillator charge current is 1  $\mu$ A

enumerator kTSI\_RefOscChargeCurrent\_2uA

Reference oscillator charge current is 2  $\mu$ A

enumerator kTSI\_RefOscChargeCurrent\_4uA

Reference oscillator charge current is 4  $\mu$ A

enumerator kTSI\_RefOscChargeCurrent\_8uA

Reference oscillator charge current is 8  $\mu$ A

enumerator kTSI\_RefOscChargeCurrent\_16uA

Reference oscillator charge current is 16  $\mu$ A

enumerator kTSI\_RefOscChargeCurrent\_32uA

Reference oscillator charge current is 32  $\mu$ A

enumerator kTSI\_RefOscChargeCurrent\_64uA

Reference oscillator charge current is 64  $\mu$ A

enum \_tsi\_osc\_voltage\_rails

TSI oscillator's voltage rails.

These bits indicate the oscillator's voltage rails.

*Values:*

enumerator kTSI\_OscVolRailsOption\_0

DVOLT value option 0, the value may differ on different platforms

enumerator kTSI\_OscVolRailsOption\_1

DVOLT value option 1, the value may differ on different platforms

enumerator kTSI\_OscVolRailsOption\_2

DVOLT value option 2, the value may differ on different platforms

enumerator kTSI\_OscVolRailsOption\_3

DVOLT value option 3, the value may differ on different platforms

enum \_tsi\_external\_osc\_charge\_current

TSI External oscillator charge and discharge current select.

These bits indicate the electrode oscillator charge and discharge current value in TSI (EXTCHRG) instance.

*Values:*

- enumerator kTSI\_ExtOscChargeCurrent\_500nA  
External oscillator charge current is 500  $\mu$ A
- enumerator kTSI\_ExtOscChargeCurrent\_1uA  
External oscillator charge current is 1  $\mu$ A
- enumerator kTSI\_ExtOscChargeCurrent\_2uA  
External oscillator charge current is 2  $\mu$ A
- enumerator kTSI\_ExtOscChargeCurrent\_4uA  
External oscillator charge current is 4  $\mu$ A
- enumerator kTSI\_ExtOscChargeCurrent\_8uA  
External oscillator charge current is 8  $\mu$ A
- enumerator kTSI\_ExtOscChargeCurrent\_16uA  
External oscillator charge current is 16  $\mu$ A
- enumerator kTSI\_ExtOscChargeCurrent\_32uA  
External oscillator charge current is 32  $\mu$ A
- enumerator kTSI\_ExtOscChargeCurrent\_64uA  
External oscillator charge current is 64  $\mu$ A

enum \_tsi\_series\_resistance

TSI series resistance RS value select.

These bits indicate the electrode RS series resistance for the noise mode in TSI (EXTCHRG) instance.

*Values:*

- enumerator kTSI\_SeriesResistance\_32k  
Series Resistance is 32 kilo ohms
- enumerator kTSI\_SeriesResistance\_187k  
Series Resistance is 18 7 kilo ohms

enum \_tsi\_filter\_bits

TSI series filter bits select.

These bits indicate the count of the filter bits in TSI noise mode EXTCHRG[2:1] bits

*Values:*

- enumerator kTSI\_FilterBits\_3  
3 filter bits, 8 peaks increments the cnt+1
- enumerator kTSI\_FilterBits\_2  
2 filter bits, 4 peaks increments the cnt+1
- enumerator kTSI\_FilterBits\_1  
1 filter bits, 2 peaks increments the cnt+1
- enumerator kTSI\_FilterBits\_0  
no filter bits,1 peak increments the cnt+1

enum `_tsi_status_flags`

TSI status flags.

*Values:*

enumerator `kTSI_EndOfScanFlag`

End-Of-Scan flag

enumerator `kTSI_OutOfRangeFlag`

Out-Of-Range flag

enum `_tsi_interrupt_enable`

TSI feature interrupt source.

*Values:*

enumerator `kTSI_GlobalInterruptEnable`

TSI module global interrupt

enumerator `kTSI_OutOfRangeInterruptEnable`

Out-Of-Range interrupt

enumerator `kTSI_EndOfScanInterruptEnable`

End-Of-Scan interrupt

typedef enum `_tsi_n_consecutive_scans` `tsi_n_consecutive_scans_t`

TSI number of scan intervals for each electrode.

These constants define the tsi number of consecutive scans in a TSI instance for each electrode.

typedef enum `_tsi_electrode_osc_prescaler` `tsi_electrode_osc_prescaler_t`

TSI electrode oscillator prescaler.

These constants define the TSI electrode oscillator prescaler in a TSI instance.

typedef enum `_tsi_analog_mode` `tsi_analog_mode_t`

TSI analog mode select.

Set up TSI analog modes in a TSI instance.

typedef enum `_tsi_reference_osc_charge_current` `tsi_reference_osc_charge_current_t`

TSI Reference oscillator charge and discharge current select.

These constants define the TSI Reference oscillator charge current select in a TSI (REFCHRG) instance.

typedef enum `_tsi_osc_voltage_rails` `tsi_osc_voltage_rails_t`

TSI oscillator's voltage rails.

These bits indicate the oscillator's voltage rails.

typedef enum `_tsi_external_osc_charge_current` `tsi_external_osc_charge_current_t`

TSI External oscillator charge and discharge current select.

These bits indicate the electrode oscillator charge and discharge current value in TSI (EXTCHRG) instance.

typedef enum `_tsi_series_resistance` `tsi_series_resistor_t`

TSI series resistance RS value select.

These bits indicate the electrode RS series resistance for the noise mode in TSI (EXTCHRG) instance.

```
typedef enum _tsi_filter_bits tsi_filter_bits_t
```

TSI series filter bits select.

These bits indicate the count of the filter bits in TSI noise mode EXTCHRG[2:1] bits

```
typedef enum _tsi_status_flags tsi_status_flags_t
```

TSI status flags.

```
typedef enum _tsi_interrupt_enable tsi_interrupt_enable_t
```

TSI feature interrupt source.

```
typedef struct _tsi_calibration_data tsi_calibration_data_t
```

TSI calibration data storage.

```
typedef struct _tsi_config tsi_config_t
```

TSI configuration structure.

This structure contains the settings for the most common TSI configurations including the TSI module charge currents, number of scans, thresholds, and so on.

```
void TSI_Init(TSI_Type *base, const tsi_config_t *config)
```

Initializes hardware.

Initializes the peripheral to the targeted state specified by parameter configuration, such as sets prescalers, number of scans, clocks, delta voltage series resistor, filter bits, reference, and electrode charge current and threshold.

#### Parameters

- base – TSI peripheral base address.
- config – Pointer to TSI module configuration structure.

#### Returns

none

```
void TSI_Deinit(TSI_Type *base)
```

De-initializes hardware.

De-initializes the peripheral to default state.

#### Parameters

- base – TSI peripheral base address.

#### Returns

none

```
void TSI_GetNormalModeDefaultConfig(tsi_config_t *userConfig)
```

Gets the TSI normal mode user configuration structure. This interface sets userConfig structure to a default value. The configuration structure only includes the settings for the whole TSI. The user configure is set to these values:

```
userConfig->prescaler = kTSI_ElecOscPrescaler_2div;
userConfig->extchrg = kTSI_ExtOscChargeCurrent_500nA;
userConfig->refchrg = kTSI_RefOscChargeCurrent_4uA;
userConfig->nscn = kTSI_ConsecutiveScansNumber_10time;
userConfig->mode = kTSI_AnalogModeSel_Capacitive;
userConfig->dvolt = kTSI_OscVolRailsOption_0;
userConfig->thresh = 0U;
userConfig->thresl = 0U;
```

#### Parameters

- userConfig – Pointer to the TSI user configuration structure.

void TSI\_GetLowPowerModeDefaultConfig(*tsi\_config\_t* \*userConfig)

Gets the TSI low power mode default user configuration structure. This interface sets userConfig structure to a default value. The configuration structure only includes the settings for the whole TSI. The user configure is set to these values:

```
userConfig->prescaler = kTSI_ElecOscPrescaler_2div;
userConfig->extchrg = kTSI_ExtOscChargeCurrent_500nA;
userConfig->refchrg = kTSI_RefOscChargeCurrent_4uA;
userConfig->nscn = kTSI_ConsecutiveScansNumber_10time;
userConfig->mode = kTSI_AnalogModeSel_Capacitive;
userConfig->dvolt = kTSI_OscVolRailsOption_0;
userConfig->thresh = 400U;
userConfig->thresl = 0U;
```

### Parameters

- userConfig – Pointer to the TSI user configuration structure.

void TSI\_Calibrate(TSI\_Type \*base, *tsi\_calibration\_data\_t* \*calBuff)

Hardware calibration.

Calibrates the peripheral to fetch the initial counter value of the enabled electrodes. This API is mostly used at initial application setup. Call this function after the TSI\_Init API and use the calibrated counter values to set up applications (such as to determine under which counter value we can confirm a touch event occurs).

### Parameters

- base – TSI peripheral base address.
- calBuff – Data buffer that store the calibrated counter value.

### Returns

none

void TSI\_EnableInterrupts(TSI\_Type \*base, uint32\_t mask)

Enables the TSI interrupt requests.

### Parameters

- base – TSI peripheral base address.
- mask – interrupt source The parameter can be combination of the following source if defined:
  - kTSI\_GlobalInterruptEnable
  - kTSI\_EndOfScanInterruptEnable
  - kTSI\_OutOfRangeInterruptEnable

void TSI\_DisableInterrupts(TSI\_Type \*base, uint32\_t mask)

Disables the TSI interrupt requests.

### Parameters

- base – TSI peripheral base address.
- mask – interrupt source The parameter can be combination of the following source if defined:
  - kTSI\_GlobalInterruptEnable
  - kTSI\_EndOfScanInterruptEnable
  - kTSI\_OutOfRangeInterruptEnable



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

Gets an interrupt flag. This function gets the TSI interrupt flags.

**Parameters**

- base – TSI peripheral base address.

**Returns**

The mask of these status flags combination.

void TSI\_ClearStatusFlags(TSI\_Type \*base, uint32\_t mask)

Clears the interrupt flag.

This function clears the TSI interrupt flag, automatically cleared flags can't be cleared by this function.

**Parameters**

- base – TSI peripheral base address.
- mask – The status flags to clear.

static inline uint32\_t TSI\_GetScanTriggerMode(TSI\_Type \*base)

Gets the TSI scan trigger mode.

**Parameters**

- base – TSI peripheral base address.

**Returns**

Scan trigger mode.

static inline bool TSI\_IsScanInProgress(TSI\_Type \*base)

Gets the scan in progress flag.

**Parameters**

- base – TSI peripheral base address.

**Returns**

True - scan is in progress. False - scan is not in progress.

static inline void TSI\_SetElectrodeOSCPrescaler(TSI\_Type \*base, *tsi\_electrode\_osc\_prescaler\_t* prescaler)

Sets the prescaler.

**Parameters**

- base – TSI peripheral base address.
- prescaler – Prescaler value.

**Returns**

none.

static inline void TSI\_SetNumberOfScans(TSI\_Type \*base, *tsi\_n\_consecutive\_scans\_t* number)

Sets the number of scans (NSCN).

**Parameters**

- base – TSI peripheral base address.
- number – Number of scans.

**Returns**

none.

static inline void TSI\_EnableModule(TSI\_Type \*base, bool enable)

Enables/disables the TSI module.

**Parameters**

- base – TSI peripheral base address.
- enable – Choose whether to enable or disable module;
  - true Enable TSI module;
  - false Disable TSI module;

**Returns**

none.

static inline void TSI\_EnableLowPower(TSI\_Type \*base, bool enable)

Sets the TSI low power STOP mode as enabled or disabled. This enables the TSI module function in low power modes.

**Parameters**

- base – TSI peripheral base address.
- enable – Choose to enable or disable STOP mode.
  - true Enable module in STOP mode;
  - false Disable module in STOP mode;

**Returns**

none.

static inline void TSI\_EnableHardwareTriggerScan(TSI\_Type \*base, bool enable)

Enables/disables the hardware trigger scan.

**Parameters**

- base – TSI peripheral base address.
- enable – Choose to enable hardware trigger or software trigger scan.
  - true Enable hardware trigger scan;
  - false Enable software trigger scan;

**Returns**

none.

static inline void TSI\_StartSoftwareTrigger(TSI\_Type \*base)

Starts a software trigger measurement (triggers a new measurement).

**Parameters**

- base – TSI peripheral base address.

**Returns**

none.

static inline void TSI\_SetMeasuredChannelNumber(TSI\_Type \*base, uint8\_t channel)

Sets the measured channel number.

**Parameters**

- base – TSI peripheral base address.
- channel – Channel number 0 ... 15.

**Returns**

none.

static inline uint8\_t TSI\_GetMeasuredChannelNumber(TSI\_Type \*base)

Gets the current measured channel number.

**Parameters**

- base – TSI peripheral base address.

**Returns**

uint8\_t Channel number 0 ... 15.

static inline void TSI\_EnableDmaTransfer(TSI\_Type \*base, bool enable)

Enables/disables the DMA transfer.

**Parameters**

- base – TSI peripheral base address.
- enable – Choose to enable DMA transfer or not.
  - true Enable DMA transfer;
  - false Disable DMA transfer;

**Returns**

none.

static inline void TSI\_EnableEndOfScanDmaTransferOnly(TSI\_Type \*base, bool enable)

Decides whether to enable end of scan DMA transfer request only.

**Parameters**

- base – TSI peripheral base address.
- enable – Choose whether to enable End of Scan DMA transfer request only.
  - true Enable End of Scan DMA transfer request only;
  - false Both End-of-Scan and Out-of-Range can generate DMA transfer request.

**Returns**

none.

static inline uint16\_t TSI\_GetCounter(TSI\_Type \*base)

Gets the conversion counter value.

**Parameters**

- base – TSI peripheral base address.

**Returns**

Accumulated scan counter value ticked by the reference clock.

static inline void TSI\_SetLowThreshold(TSI\_Type \*base, uint16\_t low\_threshold)

Sets the TSI wake-up channel low threshold.

**Parameters**

- base – TSI peripheral base address.
- low\_threshold – Low counter threshold.

**Returns**

none.

static inline void TSI\_SetHighThreshold(TSI\_Type \*base, uint16\_t high\_threshold)

Sets the TSI wake-up channel high threshold.

**Parameters**

- base – TSI peripheral base address.

- `high_threshold` – High counter threshold.

**Returns**

none.

static inline void TSI\_SetAnalogMode(TSI\_Type \*base, *tsi\_analog\_mode\_t* mode)

Sets the analog mode of the TSI module.

**Parameters**

- `base` – TSI peripheral base address.
- `mode` – Mode value.

**Returns**

none.

static inline uint8\_t TSI\_GetNoiseModeResult(TSI\_Type \*base)

Gets the noise mode result of the TSI module.

**Parameters**

- `base` – TSI peripheral base address.

**Returns**

Value of the GENCS[MODE] bit-fields.

static inline void TSI\_SetReferenceChargeCurrent(TSI\_Type \*base,  
*tsi\_reference\_osc\_charge\_current\_t* current)

Sets the reference oscillator charge current.

**Parameters**

- `base` – TSI peripheral base address.
- `current` – The reference oscillator charge current.

**Returns**

none.

static inline void TSI\_SetElectrodeChargeCurrent(TSI\_Type \*base,  
*tsi\_external\_osc\_charge\_current\_t* current)

Sets the external electrode charge current.

**Parameters**

- `base` – TSI peripheral base address.
- `current` – External electrode charge current.

**Returns**

none.

static inline void TSI\_SetOscVoltageRails(TSI\_Type \*base, *tsi\_osc\_voltage\_rails\_t* dvolt)

Sets the oscillator's voltage rails.

**Parameters**

- `base` – TSI peripheral base address.
- `dvolt` – The voltage rails.

**Returns**

none.

static inline void TSI\_SetElectrodeSeriesResistor(TSI\_Type \*base, *tsi\_series\_resistor\_t* resistor)

Sets the electrode series resistance value in EXTCHRG[0] bit.

**Parameters**

- `base` – TSI peripheral base address.

- resistor – Series resistance.

**Returns**

none.

static inline void TSI\_SetFilterBits(TSI\_Type \*base, *tsi\_filter\_bits\_t* filter)

Sets the electrode filter bits value in EXTCHRG[2:1] bits.

**Parameters**

- base – TSI peripheral base address.
- filter – Series resistance.

**Returns**

none.

ALL\_FLAGS\_MASK

TSI status flags macro collection.

TSI\_V4\_EXTCHRG\_RESISTOR\_BIT\_SHIFT

resistor bit shift in EXTCHRG bit-field

TSI\_V4\_EXTCHRG\_FILTER\_BITS\_SHIFT

filter bits shift in EXTCHRG bit-field

TSI\_V4\_EXTCHRG\_RESISTOR\_BIT\_CLEAR

macro of clearing the resistor bit in EXTCHRG bit-field

TSI\_V4\_EXTCHRG\_FILTER\_BITS\_CLEAR

macro of clearing the filter bits in EXTCHRG bit-field

struct *\_tsi\_calibration\_data*

*#include <fsl\_tsi\_v4.h>* TSI calibration data storage.

**Public Members**

uint16\_t calibratedData[1]

TSI calibration data storage buffer

struct *\_tsi\_config*

*#include <fsl\_tsi\_v4.h>* TSI configuration structure.

This structure contains the settings for the most common TSI configurations including the TSI module charge currents, number of scans, thresholds, and so on.

**Public Members**

uint16\_t thresh

High threshold.

uint16\_t thresl

Low threshold.

*tsi\_electrode\_osc\_prescaler\_t* prescaler

Prescaler

*tsi\_external\_osc\_charge\_current\_t* extchrg

Electrode charge current

*tsi\_reference\_osc\_charge\_current\_t* refchrg

Reference charge current

*tsi\_n\_consecutive\_scans\_t* nscn

Number of scans.

*tsi\_analog\_mode\_t* mode

TSI mode of operation.

*tsi\_osc\_voltage\_rails\_t* dvolt

Oscillator's voltage rails.

*tsi\_series\_resistor\_t* resistor

Series resistance value

*tsi\_filter\_bits\_t* filter

Noise mode filter bits

## 2.62 TSTMR: Timestamp Timer Driver

FSL\_TSTMR\_DRIVER\_VERSION

Version 2.0.2

static inline uint64\_t TSTMR\_ReadTimeStamp(TSTMR\_Type \*base)

Reads the time stamp.

This function reads the low and high registers and returns the 56-bit free running counter value. This can be read by software at any time to determine the software ticks. TSTMR registers can be read with 32-bit accesses only. The TSTMR LOW read should occur first, followed by the TSTMR HIGH read.

### Parameters

- base – TSTMR peripheral base address.

### Returns

The 56-bit time stamp value.

static inline void TSTMR\_DelayUs(TSTMR\_Type \*base, uint64\_t delayInUs)

Delays for a specified number of microseconds.

This function repeatedly reads the timestamp register and waits for the user-specified delay value.

### Parameters

- base – TSTMR peripheral base address.
- delayInUs – Delay value in microseconds.

FSL\_COMPONENT\_ID

## 2.63 VREF: Voltage Reference Driver

*status\_t* VREF\_Init(VREF\_Type \*base, const *vref\_config\_t* \*config)

Enables the clock gate and configures the VREF module according to the configuration structure.

This function must be called before calling all other VREF driver functions, read/write registers, and configurations with user-defined settings. The example below shows how to set up *vref\_config\_t* parameters and how to call the VREF\_Init function by passing in these parameters. This is an example.

```
vref_config_t vrefConfig;
vrefConfig.bufferMode = kVREF_ModeHighPowerBuffer;
vrefConfig.enableExternalVoltRef = false;
vrefConfig.enableLowRef = false;
VREF_Init(VREF, &vrefConfig);
```

### Parameters

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

### Return values

- kStatus\_Success – run success.
- kStatus\_Timeout – timeout occurs.

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

Stops and disables the clock for the VREF module.

This function should be called to shut down the module. This is an example.

```
vref_config_t vrefUserConfig;
VREF_Init(VREF);
VREF_GetDefaultConfig(&vrefUserConfig);
...
VREF_Deinit(VREF);
```

### Parameters

- base – VREF peripheral address.

**void VREF\_GetDefaultConfig(vref\_config\_t \*config)**

Initializes the VREF configuration structure.

This function initializes the VREF configuration structure to default values. This is an example.

```
vrefConfig->bufferMode = kVREF_ModeHighPowerBuffer;
vrefConfig->enableExternalVoltRef = false;
vrefConfig->enableLowRef = false;
```

### Parameters

- config – Pointer to the initialization structure.

**status\_t VREF\_SetTrimVal(VREF\_Type \*base, uint8\_t trimValue)**

Sets a TRIM value for the reference voltage.

This function sets a TRIM value for the reference voltage. Note that the TRIM value maximum is 0x3F.

### Parameters

- base – VREF peripheral address.
- trimValue – Value of the trim register to set the output reference voltage (maximum 0x3F (6-bit)).

### Return values

- kStatus\_Success – run success.
- kStatus\_Timeout – timeout occurs.

`static inline uint8_t VREF_GetTrimVal(VREF_Type *base)`

Reads the value of the TRIM meaning output voltage.

This function gets the TRIM value from the TRM register.

**Parameters**

- `base` – VREF peripheral address.

**Returns**

Six-bit value of trim setting.

`status_t VREF_SetTrim2V1Val(VREF_Type *base, uint8_t trimValue)`

Sets a TRIM value for the reference voltage (2V1).

This function sets a TRIM value for the reference voltage (2V1). Note that the TRIM value maximum is 0x3F.

**Parameters**

- `base` – VREF peripheral address.
- `trimValue` – Value of the trim register to set the output reference voltage (maximum 0x3F (6-bit)).

**Return values**

- `kStatus_Success` – run success.
- `kStatus_Timeout` – timeout occurs.

`static inline uint8_t VREF_GetTrim2V1Val(VREF_Type *base)`

Reads the value of the TRIM meaning output voltage (2V1).

This function gets the TRIM value from the VREF\_TRM4 register.

**Parameters**

- `base` – VREF peripheral address.

**Returns**

Six-bit value of trim setting.

`status_t VREF_SetLowReferenceTrimVal(VREF_Type *base, uint8_t trimValue)`

Sets the TRIM value for the low voltage reference.

This function sets the TRIM value for low reference voltage. Note the following.

- The TRIM value maximum is 0x05U
- The values 111b and 110b are not valid/allowed.

**Parameters**

- `base` – VREF peripheral address.
- `trimValue` – Value of the trim register to set output low reference voltage (maximum 0x05U (3-bit)).

**Return values**

- `kStatus_Success` – run success.
- `kStatus_Timeout` – timeout occurs.

`static inline uint8_t VREF_GetLowReferenceTrimVal(VREF_Type *base)`

Reads the value of the TRIM meaning output voltage.

This function gets the TRIM value from the VREFL\_TRM register.

**Parameters**



- base – VREF peripheral address.

**Returns**

Three-bit value of the trim setting.

FSL\_VREF\_DRIVER\_VERSION

Version 2.1.3.

VREF\_INTERNAL\_VOLTAGE\_STABLE\_TIMEOUT

Max loops to wait for VREF internal voltage stable.

This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

enum \_vref\_buffer\_mode

VREF modes.

*Values:*

enumerator kVREF\_ModeBandgapOnly

Bandgap on only, for stabilization and startup

enumerator kVREF\_ModeHighPowerBuffer

High-power buffer mode enabled

enumerator kVREF\_ModeLowPowerBuffer

Low-power buffer mode enabled

typedef enum \_vref\_buffer\_mode vref\_buffer\_mode\_t

VREF modes.

typedef struct \_vref\_config vref\_config\_t

The description structure for the VREF module.

VREF\_SC\_MODE\_LV

VREF\_SC\_REGEN

VREF\_SC\_VREFEN

VREF\_SC\_ICOMPEN

VREF\_SC\_REGEN\_MASK

VREF\_SC\_VREFST\_MASK

VREF\_SC\_VREFEN\_MASK

VREF\_SC\_MODE\_LV\_MASK

VREF\_SC\_ICOMPEN\_MASK

TRM

VREF\_TRM\_TRIM

VREF\_TRM\_CHOPEN\_MASK

VREF\_TRM\_TRIM\_MASK

VREF\_TRM\_CHOPEN\_SHIFT

VREF\_TRM\_TRIM\_SHIFT

VREF\_SC\_MODE\_LV\_SHIFT

VREF\_SC\_REGEN\_SHIFT

VREF\_SC\_VREFST\_SHIFT

VREF\_SC\_ICOMPEN\_SHIFT

struct \_\_vref\_config

*#include <fsl\_vref.h>* The description structure for the VREF module.

### Public Members

*vref\_buffer\_mode\_t* bufferMode

Buffer mode selection

bool enableLowRef

Set VREFL (0.4 V) reference buffer enable or disable

bool enableExternalVoltRef

Select external voltage reference or not (internal)

bool enable2V1VoltRef

Enable Internal Voltage Reference (2.1V)

## 2.64 WDOG32: 32-bit Watchdog Timer

void WDOG32\_GetDefaultConfig(*wdog32\_config\_t* \*config)

Initializes the WDOG32 configuration structure.

This function initializes the WDOG32 configuration structure to default values. The default values are:

```
wdog32Config->enableWdog32 = true;
wdog32Config->clockSource = kWDOG32_ClockSource1;
wdog32Config->prescaler = kWDOG32_ClockPrescalerDivide1;
wdog32Config->workMode.enableWait = true;
wdog32Config->workMode.enableStop = false;
wdog32Config->workMode.enableDebug = false;
wdog32Config->testMode = kWDOG32_TestModeDisabled;
wdog32Config->enableUpdate = true;
wdog32Config->enableInterrupt = false;
wdog32Config->enableWindowMode = false;
wdog32Config->windowValue = 0U;
wdog32Config->timeoutValue = 0xFFFFU;
```

### See also:

*wdog32\_config\_t*

### Parameters

- config – Pointer to the WDOG32 configuration structure.

AT\_QUICKACCESS\_SECTION\_CODE (status\_t WDOG32\_Init(WDOG\_Type \*base,  
const *wdog32\_config\_t* \*config))

Initializes the WDOG32 module.

This function initializes the WDOG32. To reconfigure the WDOG32 without forcing a reset first, enableUpdate must be set to true in the configuration.

Example:

```
wdog32_config_t config;
WDOG32_GetDefaultConfig(&config);
config.timeoutValue = 0x7ffU;
config.enableUpdate = true;
WDOG32_Init(wdog_base,&config);
```

**Note:** If there is errata ERR010536 (FSL\_FEATURE\_WDOG\_HAS\_ERRATA\_010536 defined as 1), then after calling this function, user need delay at least 4 LPO clock cycles before accessing other WDOG32 registers.

### Parameters

- base – WDOG32 peripheral base address.
- config – The configuration of the WDOG32.

### Return values

- kStatus\_Success – The initialization was successful
- kStatus\_Timeout – The initialization timed out

*status\_t* WDOG32\_Deinit(WDOG\_Type \*base)

De-initializes the WDOG32 module.

This function shuts down the WDOG32. Ensure that the WDOG\_CS.UPDATE is 1, which means that the register update is enabled.

### Parameters

- base – WDOG32 peripheral base address.

### Return values

- kStatus\_Success – The de-initialization was successful
- kStatus\_Timeout – The de-initialization timed out

AT\_QUICKACCESS\_SECTION\_CODE (*status\_t* WDOG32\_Unlock(WDOG\_Type \*base))

Unlocks the WDOG32 register written.

This function unlocks the WDOG32 register written.

Before starting the unlock sequence and following the configuration, disable the global interrupts. Otherwise, an interrupt could effectively invalidate the unlock sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

### Parameters

- base – WDOG32 peripheral base address

### Return values

- kStatus\_Success – The unlock sequence was successful
- kStatus\_Timeout – The unlock sequence timed out

AT\_QUICKACCESS\_SECTION\_CODE (*void* WDOG32\_Enable(WDOG\_Type \*base))

Enables the WDOG32 module.

Disables the WDOG32 module.

This function writes a value into the WDOG\_CS register to enable the WDOG32. The WDOG\_CS register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32\_Init to do wdog initialize. Before call the re-configuration APIs, ensure

that the WCT window is still open and this register has not been written in this WCT while the function is called.

This function writes a value into the WDOG\_CS register to disable the WDOG32. The WDOG\_CS register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32\_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

#### Parameters

- base – WDOG32 peripheral base address.
- base – WDOG32 peripheral base address

AT\_QUICKACCESS\_SECTION\_CODE (void WDOG32\_EnableInterrupts(WDOG\_Type \*base, uint32\_t mask))

Enables the WDOG32 interrupt.

Clears the WDOG32 flag.

Disables the WDOG32 interrupt.

This function writes a value into the WDOG\_CS register to enable the WDOG32 interrupt. The WDOG\_CS register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32\_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Example to clear an interrupt flag:

```
WDOG32_ClearStatusFlags(wdog_base, kWDOG32_InterruptFlag);
```

#### Parameters

- base – WDOG32 peripheral base address.
- mask – The interrupts to enable. The parameter can be a combination of the following source if defined:
  - kWDOG32\_InterruptEnable

This function writes a value into the WDOG\_CS register to disable the WDOG32 interrupt. The WDOG\_CS register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32\_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

- base – WDOG32 peripheral base address.
- mask – The interrupts to disabled. The parameter can be a combination of the following source if defined:
  - kWDOG32\_InterruptEnable

This function clears the WDOG32 status flag.

- base – WDOG32 peripheral base address.
- mask – The status flags to clear. The parameter can be any combination of the following values:
  - kWDOG32\_InterruptFlag

```
static inline uint32_t WDOG32_GetStatusFlags(WDOG_Type *base)
```

Gets the WDOG32 all status flags.

This function gets all status flags.

Example to get the running flag:

```
uint32_t status;
status = WDOG32_GetStatusFlags(wdog_base) & kWDOG32_RunningFlag;
```

#### See also:

`_wdog32_status_flags_t`

- true: related status flag has been set.
- false: related status flag is not set.

#### Parameters

- base – WDOG32 peripheral base address

#### Returns

State of the status flag: asserted (true) or not-asserted (false).

```
AT_QUICKACCESS_SECTION_CODE (void WDOG32_SetTimeoutValue(WDOG_Type *base,
uint16_t timeoutCount))
```

Sets the WDOG32 timeout value.

This function writes a timeout value into the WDOG\_TOVAL register. The WDOG\_TOVAL register is a write-once register. To ensure the reconfiguration fits the timing of WCT, unlock function will be called inline.

#### Parameters

- base – WDOG32 peripheral base address
- timeoutCount – WDOG32 timeout value, count of WDOG32 clock ticks.

```
AT_QUICKACCESS_SECTION_CODE (void WDOG32_SetWindowValue(WDOG_Type *base,
uint16_t windowValue))
```

Sets the WDOG32 window value.

This function writes a window value into the WDOG\_WIN register. The WDOG\_WIN register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32\_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

#### Parameters

- base – WDOG32 peripheral base address.
- windowValue – WDOG32 window value.

```
static inline void WDOG32_Refresh(WDOG_Type *base)
```

Refreshes the WDOG32 timer.

This function feeds the WDOG32. This function should be called before the Watchdog timer is in timeout. Otherwise, a reset is asserted.

#### Parameters

- base – WDOG32 peripheral base address

```
static inline uint16_t WDOG32_GetCounterValue(WDOG_Type *base)
```

Gets the WDOG32 counter value.

This function gets the WDOG32 counter value.

**Parameters**

- base – WDOG32 peripheral base address.

**Returns**

Current WDOG32 counter value.

WDOG\_FIRST\_WORD\_OF\_UNLOCK

First word of unlock sequence

WDOG\_SECOND\_WORD\_OF\_UNLOCK

Second word of unlock sequence

WDOG\_FIRST\_WORD\_OF\_REFRESH

First word of refresh sequence

WDOG\_SECOND\_WORD\_OF\_REFRESH

Second word of refresh sequence

FSL\_WDOG32\_DRIVER\_VERSION

WDOG32 driver version.

enum \_\_wdog32\_clock\_source

Max loops to wait for WDOG32 unlock sequence complete.

This is the maximum number of loops to wait for the wdog32 unlock sequence to complete. If set to 0, it will wait indefinitely until the unlock sequence is complete.

Max loops to wait for WDOG32 reconfiguration complete.

This is the maximum number of loops to wait for the wdog32 reconfiguration to complete. If set to 0, it will wait indefinitely until the reconfiguration is complete.

Describes WDOG32 clock source.

*Values:*

enumerator kWDOG32\_ClockSource0

Clock source 0

enumerator kWDOG32\_ClockSource1

Clock source 1

enumerator kWDOG32\_ClockSource2

Clock source 2

enumerator kWDOG32\_ClockSource3

Clock source 3

enum \_\_wdog32\_clock\_prescaler

Describes the selection of the clock prescaler.

*Values:*

enumerator kWDOG32\_ClockPrescalerDivide1

Divided by 1

enumerator kWDOG32\_ClockPrescalerDivide256

Divided by 256

enum \_\_wdog32\_test\_mode

Describes WDOG32 test mode.

*Values:*

enumerator kWDOG32\_TestModeDisabled

Test Mode disabled

enumerator kWDOG32\_UserModeEnabled

User Mode enabled

enumerator kWDOG32\_LowByteTest

Test Mode enabled, only low byte is used

enumerator kWDOG32\_HighByteTest

Test Mode enabled, only high byte is used

enum \_\_wdog32\_interrupt\_enable\_t

WDOG32 interrupt configuration structure.

This structure contains the settings for all of the WDOG32 interrupt configurations.

*Values:*

enumerator kWDOG32\_InterruptEnable

Interrupt is generated before forcing a reset

enum \_\_wdog32\_status\_flags\_t

WDOG32 status flags.

This structure contains the WDOG32 status flags for use in the WDOG32 functions.

*Values:*

enumerator kWDOG32\_RunningFlag

Running flag, set when WDOG32 is enabled

enumerator kWDOG32\_InterruptFlag

Interrupt flag, set when interrupt occurs

typedef enum \_\_wdog32\_clock\_source wdog32\_clock\_source\_t

Max loops to wait for WDOG32 unlock sequence complete.

This is the maximum number of loops to wait for the wdog32 unlock sequence to complete. If set to 0, it will wait indefinitely until the unlock sequence is complete.

Max loops to wait for WDOG32 reconfiguration complete.

This is the maximum number of loops to wait for the wdog32 reconfiguration to complete. If set to 0, it will wait indefinitely until the reconfiguration is complete.

Describes WDOG32 clock source.

typedef enum \_\_wdog32\_clock\_prescaler wdog32\_clock\_prescaler\_t

Describes the selection of the clock prescaler.

typedef struct \_\_wdog32\_work\_mode wdog32\_work\_mode\_t

Defines WDOG32 work mode.

typedef enum \_\_wdog32\_test\_mode wdog32\_test\_mode\_t

Describes WDOG32 test mode.

typedef struct \_\_wdog32\_config wdog32\_config\_t

Describes WDOG32 configuration structure.

struct \_\_wdog32\_work\_mode

*#include <fsl\_wdog32.h>* Defines WDOG32 work mode.

### Public Members

`bool enableWait`  
Enables or disables WDOG32 in wait mode

`bool enableStop`  
Enables or disables WDOG32 in stop mode

`bool enableDebug`  
Enables or disables WDOG32 in debug mode

`struct __wdog32_config`  
*#include <fsl\_wdog32.h>* Describes WDOG32 configuration structure.

### Public Members

`bool enableWdog32`  
Enables or disables WDOG32

`wdog32_clock_source_t clockSource`  
Clock source select

`wdog32_clock_prescaler_t prescaler`  
Clock prescaler value

`wdog32_work_mode_t workMode`  
Configures WDOG32 work mode in debug stop and wait mode

`wdog32_test_mode_t testMode`  
Configures WDOG32 test mode

`bool enableUpdate`  
Update write-once register enable

`bool enableInterrupt`  
Enables or disables WDOG32 interrupt

`bool enableWindowMode`  
Enables or disables WDOG32 window mode

`uint16_t windowValue`  
Window value

`uint16_t timeoutValue`  
Timeout value



# Chapter 3

## Middleware

### 3.1 Motor Control

#### 3.1.1 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\_RQUEUE\_SIZE

```
#define FMSTR_COMM_RQUEUE_SIZE [number]
```

**Value Type** Value in range 0...255

**Description** Specify the size of the FIFO receiver queue used to quickly receive and store characters in the FMSTR\_SHORT\_INTR interrupt mode. The default value is 32 B.

### FMSTR\_SERIAL\_SINGLEWIRE

```
#define FMSTR_SERIAL_SINGLEWIRE [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Set to non-zero to enable the “True” single-wire mode which uses a single MCU pin to communicate. The low-level driver enables the pin direction switching when the MCU peripheral supports it.

**CAN Bus transport** This section describes configuration parameters used when CAN transport is used:

```
#define FMSTR_TRANSPORT FMSTR_CAN
```

**FMSTR\_CAN\_DRV** Select what low-level driver interface will be used when implementing the CAN communication.

```
#define FMSTR_CAN_DRV [identifier]
```

**Value Type** Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing CAN driver instances.

**Description** When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/can implementation*):

- **FMSTR\_CAN\_MCUX\_FLEXCAN** - FlexCAN driver
- **FMSTR\_CAN\_MCUX\_MCAN** - MCAN driver
- **FMSTR\_CAN\_MCUX\_MSCAN** - msCAN driver
- **FMSTR\_CAN\_MCUX\_DSCFLEXCAN** - DSC FlexCAN driver
- **FMSTR\_CAN\_MCUX\_DSCMSCAN** - DSC msCAN driver

Other SDKs or BSPs may define the custom low-level driver interface structure which may be used as FMSTR\_CAN\_DRV.

### FMSTR\_CAN\_BASE

```
#define FMSTR_CAN_BASE [address|symbol]
```

**Value Type** Optional address value (numeric or symbolic)

**Description** Specify the base address of the FlexCAN, msCAN, or other CAN peripheral module to be used for the communication. This value is not defined by default. User application should call `FMSTR_SetCanBaseAddress()` to select the peripheral module.

#### FMSTR\_CAN\_CMDID

```
#define FMSTR_CAN_CMDID [number]
```

**Value Type** CAN identifier (11-bit or 29-bit number)

**Description** CAN message identifier used for FreeMASTER commands (direction from PC Host tool to target application). When declaring 29-bit identifier, combine the numeric value with `FMSTR_CAN_EXTID` bit. Default value is 0x7AA.

#### FMSTR\_CAN\_RSPID

```
#define FMSTR_CAN_RSPID [number]
```

**Value Type** CAN identifier (11-bit or 29-bit number)

**Description** CAN message identifier used for responding messages (direction from target application to PC Host tool). When declaring 29-bit identifier, combine the numeric value with `FMSTR_CAN_EXTID` bit. Note that both *CMDID* and *RSPID* values may be the same. Default value is 0x7AA.

#### FMSTR\_FLEXCAN\_TXMB

```
#define FMSTR_FLEXCAN_TXMB [number]
```

**Value Type** Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

**Description** Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame transmission. Default value is 0.

#### FMSTR\_FLEXCAN\_RXMB

```
#define FMSTR_FLEXCAN_RXMB [number]
```

**Value Type** Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

**Description** Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame reception. Note that the FreeMASTER driver may also operate with a common message buffer used by both TX and RX directions. Default value is 1.

**Network transport** This section describes configuration parameters used when Network transport is used:

```
#define FMSTR_TRANSPORT FMSTR_NET
```

**FMSTR\_NET\_DRV** Select network interface implementation.

```
#define FMSTR_NET_DRV [identifier]
```

**Value Type** Identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing NET driver instances.

**Description** When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/network implementation*):

- **FMSTR\_NET\_LWIP\_TCP** - TCP communication using lwIP stack
- **FMSTR\_NET\_LWIP\_UDP** - UDP communication using lwIP stack
- **FMSTR\_NET\_SEGGER\_RTT** - Communication using SEGGER J-Link RTT interface

Other SDKs or BSPs may define the custom networking interface which may be used as FMSTR\_CAN\_DRV.

Add another row below:

#### FMSTR\_NET\_PORT

```
#define FMSTR_NET_PORT [number]
```

**Value Type** TCP or UDP port number (short integer)

**Description** Specifies the server port number used by TCP or UDP protocols.

#### FMSTR\_NET\_BLOCKING\_TIMEOUT

```
#define FMSTR_NET_BLOCKING_TIMEOUT [number]
```

**Value Type** Timeout as number of milliseconds

**Description** This value specifies a timeout in milliseconds for which the network socket operations may block the execution inside *FMSTR\_Poll*. This may be set high (e.g. 250) when a dedicated RTOS task is used to handle FreeMASTER protocol polling. Set to a lower value when the polling task is also responsible for other operations. Set to 0 to attempt to use non-blocking socket operations.

**FMSTR\_NET\_AUTODISCOVERY**

```
#define FMSTR_NET_AUTODISCOVERY [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** This option enables the FreeMASTER driver to use a separate UDP socket to broadcast auto-discovery messages to network. This helps the FreeMASTER tool to discover the target device address, port and protocol options.

**Debugging options****FMSTR\_DISABLE**

```
#define FMSTR_DISABLE [0|1]
```

**Value Type** boolean (0 or 1)

**Description** Define as non-zero to disable all FreeMASTER features, exclude the driver code from build, and compile all its API functions empty. This may be useful to remove FreeMASTER without modifying any application source code. Default value is 0 (false).

**FMSTR\_DEBUG\_TX**

```
#define FMSTR_DEBUG_TX [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to enable the driver to periodically transmit test frames out on the selected communication interface (SCI or CAN). With the debug transmission enabled, it is simpler to detect problems in the baudrate or other communication configuration settings.

The test frames are transmitted until the first valid command frame is received from the PC Host tool. The test frame is a valid error status frame, as defined by the protocol format. On the serial line, the test frame consists of three printable characters (+©W) which are easy to capture using the serial terminal tools.

This feature requires the FMSTR\_Poll() function to be called periodically. Default value is 0 (false).

**FMSTR\_APPLICATION\_STR**

```
#define FMSTR_APPLICATION_STR
```

**Value Type** String.

**Description** Name of the application visible in FreeMASTER host application.

**Memory access**

**FMSTR\_USE\_READMEM**

```
#define FMSTR_USE_READMEM [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the Memory Read command and enable FreeMASTER to have read access to memory and variables. The access can be further restricted by using a TSA feature.  
Default value is 1 (true).

**FMSTR\_USE\_WRITEMEM**

```
#define FMSTR_USE_WRITEMEM [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the Memory Write command.  
The default value is 1 (true).

**Oscilloscope options****FMSTR\_USE\_SCOPE**

```
#define FMSTR_USE_SCOPE [number]
```

**Value Type** Integer number.

**Description** Number of Oscilloscope instances to be supported. Set to 0 to disable the Oscilloscope feature.  
Default value is 0.

**FMSTR\_MAX\_SCOPE\_VARS**

```
#define FMSTR_MAX_SCOPE_VARS [number]
```

**Value Type** Integer number larger than 2.

**Description** Number of variables to be supported by each Oscilloscope instance.  
Default value is 8.

**Recorder options****FMSTR\_USE\_RECORDER**

```
#define FMSTR_USE_RECORDER [number]
```



**Value Type** Integer number.

**Description** Number of Recorder instances to be supported. Set to 0 to disable the Recorder feature.  
Default value is 0.

#### FMSTR\_REC\_BUFF\_SIZE

```
#define FMSTR_REC_BUFF_SIZE [number]
```

**Value Type** Integer number larger than 2.

**Description** Defines the size of the memory buffer used by the Recorder instance #0.  
Default: not defined, user shall call 'FMSTR\_RecorderCreate()' API function to specify this parameter in run time.

#### FMSTR\_REC\_TIMEBASE

```
#define FMSTR_REC_TIMEBASE [time specification]
```

**Value Type** Number (nanoseconds time).

**Description** Defines the base sampling rate in nanoseconds (sampling speed) Recorder instance #0.

Use one of the following macros:

- FMSTR\_REC\_BASE\_SECONDS(x)
- FMSTR\_REC\_BASE\_MILLISEC(x)
- FMSTR\_REC\_BASE\_MICROSEC(x)
- FMSTR\_REC\_BASE\_NANOSEC(x)

Default: not defined, user shall call 'FMSTR\_RecorderCreate()' API function to specify this parameter in run time.

#### FMSTR\_REC\_FLOAT\_TRIG

```
#define FMSTR_REC_FLOAT_TRIG [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the floating-point triggering. Be aware that floating-point triggering may grow the code size by linking the floating-point standard library.  
Default value is 0 (false).

### Application Commands options

**FMSTR\_USE\_APPCMD**

```
#define FMSTR_USE_APPCMD [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the Application Commands feature. Default value is 0 (false).

**FMSTR\_APPCMD\_BUFF\_SIZE**

```
#define FMSTR_APPCMD_BUFF_SIZE [size]
```

**Value Type** Numeric buffer size in range 1..255

**Description** The size of the Application Command data buffer allocated by the driver. The buffer stores the (optional) parameters of the Application Command which waits to be processed.

**FMSTR\_MAX\_APPCMD\_CALLS**

```
#define FMSTR_MAX_APPCMD_CALLS [number]
```

**Value Type** Number in range 0..255

**Description** The number of different Application Commands that can be assigned a callback handler function using FMSTR\_RegisterAppCmdCall(). Default value is 0.

**TSA options****FMSTR\_USE\_TSA**

```
#define FMSTR_USE_TSA [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable the FreeMASTER TSA feature to be used. With this option enabled, the TSA tables defined in the applications are made available to the FreeMASTER host tool. Default value is 0 (false).

**FMSTR\_USE\_TSA\_SAFETY**

```
#define FMSTR_USE_TSA_SAFETY [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable the memory access validation in the FreeMASTER driver. With this option, the host tool is not able to access the memory which is not described by at least one TSA descriptor. Also a write access is denied for objects defined as read-only in TSA tables. Default value is 0 (false).

#### FMSTR\_USE\_TSA\_INROM

```
#define FMSTR_USE_TSA_INROM [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Declare all TSA descriptors as *const*, which enables the linker to put the data into the flash memory. The actual result depends on linker settings or the linker commands used in the project. Default value is 0 (false).

#### FMSTR\_USE\_TSA\_DYNAMIC

```
#define FMSTR_USE_TSA_DYNAMIC [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable runtime-defined TSA entries to be added to the TSA table by the FMSTR\_SetUpTsaBuff() and FMSTR\_TsaAddVar() functions. Default value is 0 (false).

### Pipes options

#### FMSTR\_USE\_PIPES

```
#define FMSTR_USE_PIPES [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable the FreeMASTER Pipes feature to be used. Default value is 0 (false).

#### FMSTR\_MAX\_PIPES\_COUNT

```
#define FMSTR_MAX_PIPES_COUNT [number]
```

**Value Type** Number in range 1..63.

**Description** The number of simultaneous pipe connections to support. The default value is 1.

**Driver interrupt modes** To implement the communication, the FreeMASTER driver handles the Serial or CAN module's receive and transmit requests. Use the *freemaster\_cfg.h* configuration file to select whether the driver processes the communication automatically in the interrupt service routine handler or if it only polls the status of the module (typically during the application idle time).

This section describes each of the interrupt mode in more details.

### Completely Interrupt-Driven operation Activated using:

```
#define FMSTR_LONG_INTR 1
```

In this mode, both the communication and the FreeMASTER protocol decoding is done in the *FMSTR\_SerialIsr*, *FMSTR\_CanIsr*, or other interrupt service routine. Because the protocol execution may be a lengthy task (especially with the TSA-Safety enabled) it is recommended to use this mode only if the interrupt prioritization scheme is possible in the application and the FreeMASTER interrupt is assigned to a lower (the lowest) priority.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR\_SerialIsr* or *FMSTR\_CanIsr* functions from that handler.

### Mixed Interrupt and Polling Modes Activated using:

```
#define FMSTR_SHORT_INTR 1
```

In this mode, the communication processing time is split between the interrupt routine and the main application loop or task. The raw communication is handled by the *FMSTR\_SerialIsr*, *FMSTR\_CanIsr*, or other interrupt service routine, while the protocol decoding and execution is handled by the *FMSTR\_Poll* routine. Call *FMSTR\_Poll* during the idle time in the application main loop.

The interrupt processing in this mode is relatively fast and deterministic. Upon a serial-receive event, the received character is only placed into a FIFO-like queue and it is not further processed. Upon a CAN receive event, the received frame is stored into a receive buffer. When transmitting, the characters are fetched from the prepared transmit buffer.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR\_SerialIsr* or *FMSTR\_CanIsr* functions from that handler.

When the serial interface is used as the serial communication interface, ensure that the *FMSTR\_Poll* function is called at least once per *N* character time periods. *N* is the length of the FreeMASTER FIFO queue (*FMSTR\_COMM\_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\_RQUEUE\_SIZE macro).  $N$  is 1 for the poll-driven mode.
- $Tchar$  is the character time, which is the time needed to transmit or receive a single byte over the SCI line.

**Note:** In the long interrupt mode, this function typically compiles as an empty function and can still be called. It is worthwhile to call this function regardless of the interrupt mode used in the application. This approach enables a convenient switching between the different interrupt modes only by changing the configuration macros in the *freemaster\_cfg.h* file.

## FMSTR\_SerialIsr / FMSTR\_CanIsr

### Prototype

```
void FMSTR_SerialIsr(void);
void FMSTR_CanIsr(void);
```

- Declaration: *freemaster.h*
- Implementation: *hw-specific low-level driver C file*

**Description** This function contains the interrupt-processing code of the FreeMASTER driver. In long or short interrupt modes (see [Driver interrupt modes](#)), this function must be called from the application interrupt service routine registered for the communication interrupt vector. On platforms where the communication module uses multiple interrupt vectors, the application should register a handler for all vectors and call this function at each interrupt.

**Note:** In a poll-driven mode, this function is compiled as an empty function and does not have to be used.

## Recorder API

### FMSTR\_RecorderCreate

#### Prototype

```
FMSTR_BOOL FMSTR_RecorderCreate(FMSTR_INDEX recIndex, FMSTR_REC_BUFF* buffCfg);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_rec.c*

**Description** This function registers a recorder instance and enables it to be used by the PC Host tool. Call this function for all recorder instances from 0 to the maximum number defined by the FMSTR\_USE\_RECORDER configuration option (minus one). An exception to this requirement is the recorder of instance 0 which may be automatically configured by FMSTR\_Init when the *freemaster\_cfg.h* configuration file defines the *FMSTR\_REC\_BUFF\_SIZE* and *FMSTR\_REC\_TIMEBASE* options.

For more information, see [Configurable items](#).

### FMSTR\_Recorder

#### Prototype

```
void FMSTR_Recorder(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_rec.c*

**Description** This function takes a sample of the variables being recorded using the FreeMASTER Recorder instance *recIndex*. If the selected Recorder is not active when the *FMSTR\_Recorder* function is being called, the function returns immediately. When the Recorder is active, the values of the variables being recorded are copied into the recorder buffer and the trigger conditions are evaluated.

If a trigger condition is satisfied, the Recorder enters the post-trigger mode, where it counts down the follow-up samples (number of *FMSTR\_Recorder* function calls) and de-activates the Recorder when the required post-trigger samples are finished.

The *FMSTR\_Recorder* function is typically called in the timer or PWM interrupt service routines. This function can also be called in the application main loop (for testing purposes).

### FMSTR\_RecorderTrigger

#### Prototype

```
void FMSTR_RecorderTrigger(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_rec.c*



**Description** This function forces the Recorder trigger condition to happen, which causes the Recorder to be automatically deactivated after the post-trigger samples are sampled. Use this function in the application code for programmatic control over the Recorder triggering. This can be useful when a more complex triggering conditions need to be used.

**Fast Recorder API** The Fast Recorder feature is not available in the FreeMASTER driver version 3. This feature was heavily dependent on the target platform and it was only available for the 56F8xxxx DSCs.

**TSA Tables** When the TSA is enabled in the FreeMASTER driver configuration file (by setting the FMSTR\_USE\_TSA macro to a non-zero value), it defines the so-called TSA tables in the application. This section describes the macros that must to be used to define the TSA tables.

There can be any number of TSA tables spread across the application source files. There must be always exactly one TSA Table List defined, which informs the FreeMASTER driver about the active TSA tables.

When there is at least one TSA table and one TSA Table List defined in the application, the TSA information automatically appears in the FreeMASTER symbols list. The symbols can then be used to create FreeMASTER variables for visualization or control.

**TSA table definition** The TSA table describes the static or global variables together with their address, size, type, and access-protection information. If the TSA-described variables are of a structure type, the TSA table may also describe this type and provide an access to the individual structure members of the variable.

The TSA table definition begins with the FMSTR\_TSA\_TABLE\_BEGIN macro with a *table\_id* identifying the table. The *table\_id* shall be a valid C-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 */
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 *fmrstr:* 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 */
```

(continues on next page)

(continued from previous page)

```

/* 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 coremqtt-agent**

The coreMQTT Agent library is a high level API that adds thread safety to the coreMQTT library.

## **Readme**

### **4.1.8 corepkcs11**

PKCS #11 key management library.

## **Readme**

### **4.1.9 freertos-plus-tcp**

Open source RTOS FreeRTOS Plus TCP.

## **Readme**