



# MCUXpresso SDK Documentation

Release 25.12.00-pvw2



NXP  
Nov 14, 2025



# Table of contents

|          |  |            |
|----------|--|------------|
| <b>1</b> | <b>FRDM-MCXC041</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  | 103        |
| 1.7      | Middleware Documentation   | 103        |
| 1.7.1    | FreeMASTER   | 103        |
| 1.7.2    | FreeRTOS   | 103        |
| 1.7.3    | File systemFatfs   | 103        |
| <b>2</b> | <b>MCXC041</b>   | <b>105</b> |
| 2.1      | ADC16: 16-bit SAR Analog-to-Digital Converter Driver                 | 105        |
| 2.2      | Clock Driver   | 114        |
| 2.3      | CMP: Analog Comparator Driver  | 123        |
| 2.4      | COP: Watchdog Driver   | 128        |
| 2.5      | FGPIO Driver   | 130        |
| 2.6      | C90TFS Flash Driver  | 132        |
| 2.7      | ftfx adapter   | 132        |
| 2.8      | Ftfx CACHE Driver  | 132        |
| 2.9      | ftfx controller  | 134        |
| 2.10     | ftfx feature   | 151        |
| 2.11     | Ftfx FLASH Driver  | 151        |
| 2.12     | Ftfx FLEXNVM Driver  | 165        |
| 2.13     | ftfx utilities   | 176        |
| 2.14     | GPIO: General-Purpose Input/Output Driver                            | 176        |
| 2.15     | GPIO Driver  | 178        |
| 2.16     | I2C: Inter-Integrated Circuit Driver                                 | 180        |
| 2.17     | I2C Driver   | 180        |
| 2.18     | Common Driver  | 194        |
| 2.19     | Lin_lpuart_driver  | 206        |
| 2.20     | LLWU: Low-Leakage Wakeup Unit Driver                                 | 214        |
| 2.21     | LPTMR: Low-Power Timer   | 218        |
| 2.22     | LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver | 223        |
| 2.23     | LPUART Driver  | 223        |
| 2.24     | MCM: Miscellaneous Control Module                                    | 242        |
| 2.25     | PMC: Power Management Controller                                     | 247        |
| 2.26     | PORT: Port Control and Interrupts                                    | 253        |
| 2.27     | RCM: Reset Control Module Driver                                     | 260        |
| 2.28     | RTC: Real Time Clock   | 266        |
| 2.29     | SIM: System Integration Module Driver                                | 274        |

|          |   |            |
|----------|---|------------|
| 2.30     | SMC: System Mode Controller Driver      | 275        |
| 2.31     | SPI: Serial Peripheral Interface Driver | 281        |
| 2.32     | SPI Driver                              | 281        |
| 2.33     | TPM: Timer PWM Module                   | 294        |
| 2.34     | VREF: Voltage Reference Driver          | 310        |
| <b>3</b> | <b>Middleware</b>                       | <b>315</b> |
| 3.1      | Motor Control                           | 315        |
| 3.1.1    | FreeMASTER                              | 315        |
| <b>4</b> | <b>RTOS</b>                             | <b>353</b> |
| 4.1      | FreeRTOS                                | 353        |
| 4.1.1    | FreeRTOS kernel                         | 353        |
| 4.1.2    | FreeRTOS drivers                        | 353        |
| 4.1.3    | backoffalgorithm                        | 353        |
| 4.1.4    | corehttp                                | 353        |
| 4.1.5    | corejson                                | 353        |
| 4.1.6    | coremqtt                                | 354        |
| 4.1.7    | corepkcs11                              | 354        |
| 4.1.8    | freertos-plus-tcp                       | 354        |

This documentation contains information specific to the frdm-mxc041 board.

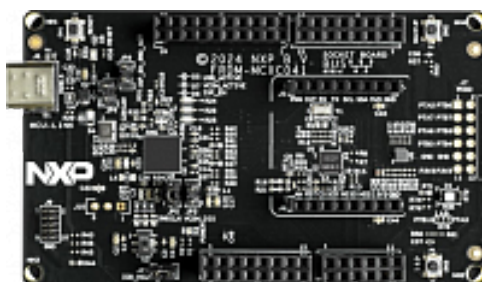


# Chapter 1

## FRDM-MCXC041

### 1.1 Overview

The FRDM-MCXC041 is supported by a range of NXP and third-party development software.



MCU device and part on board is shown below:

- Device: MCXC041
- PartNumber: MCXC041VFK

### 1.2 Getting Started with MCUXpresso SDK Package

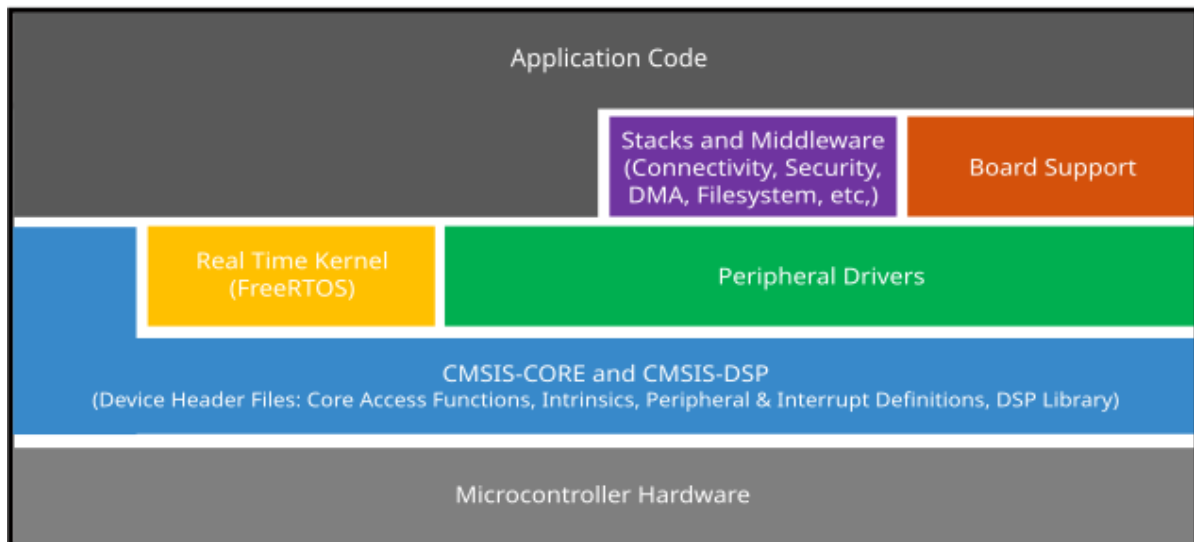
#### 1.2.1 Getting Started with MCUXpresso SDK Package

##### Overview

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

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

For more details about MCUXpresso SDK, see [MCUXpresso Software Development Kit \(SDK\)](#).



### MCUXpresso SDK board support package folders

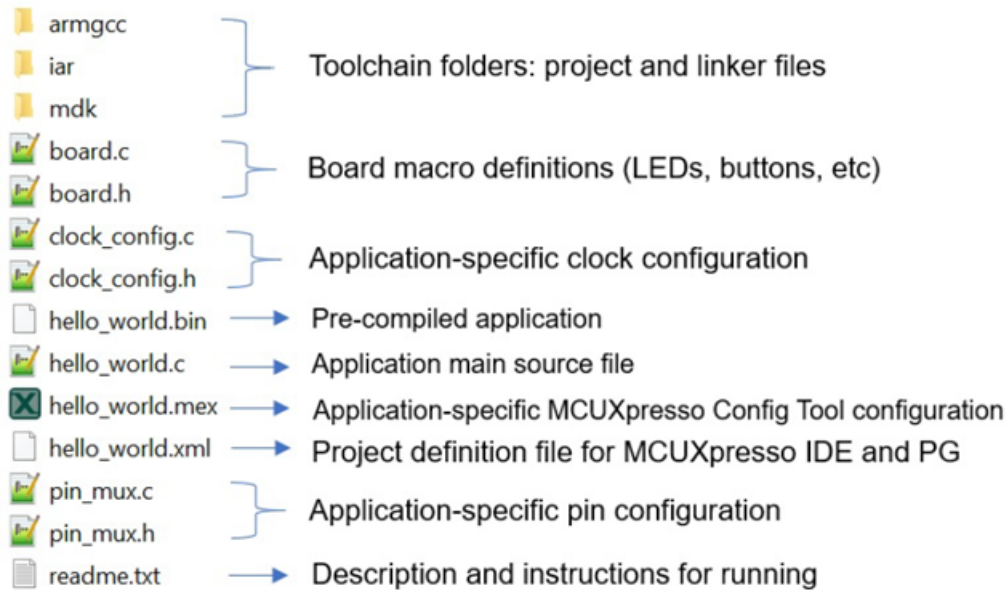
MCUXpresso SDK board support package provides example applications for NXP development and evaluation boards for Arm Cortex-M cores including Freedom, Tower System, and LPCXpresso boards. Board support packages are found inside the top-level boards folder and each supported board has its own folder (an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

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

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

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

In the `hello_world` application folder you see the following contents:



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

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

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

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

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

## Run a demo using MCUXpresso IDE

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

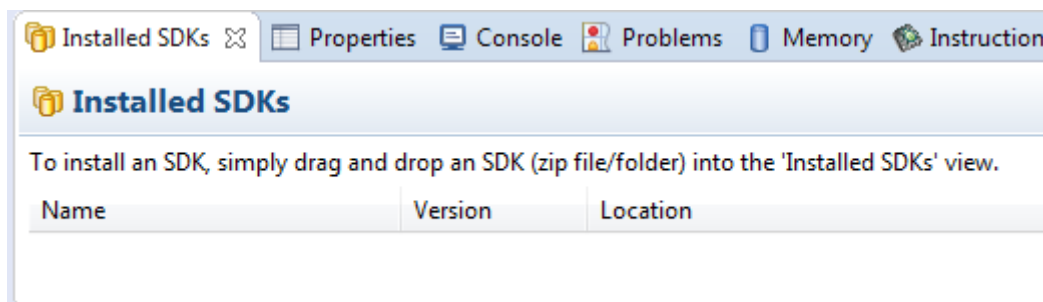
This section describes the steps required to configure MCUXpresso IDE to build, run, and debug example applications. The `hello_world` demo application targeted for the hardware platform is

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

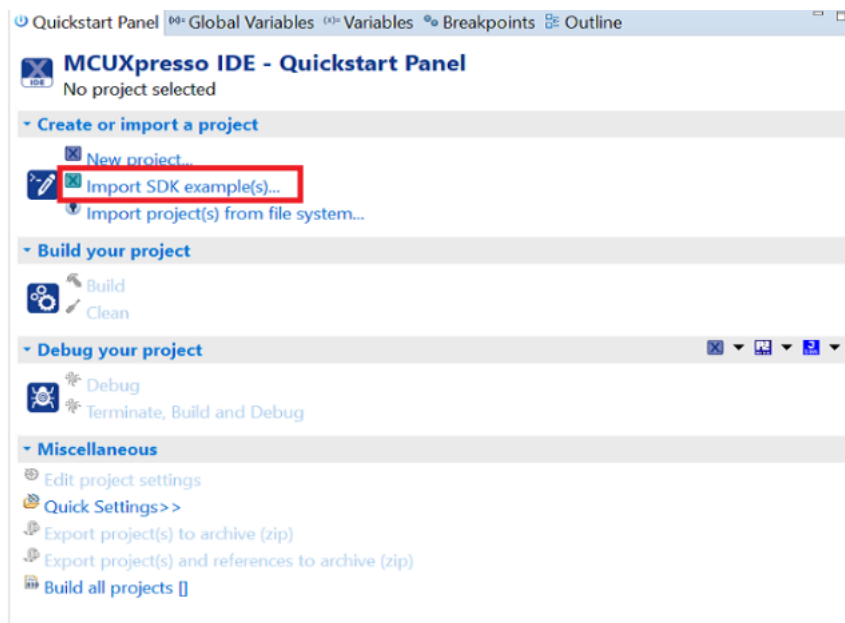
**Select the workspace location** Every time MCUXpresso IDE launches, it prompts the user to select a workspace location. MCUXpresso IDE is built on top of Eclipse which uses workspace to store information about its current configuration, and in some use cases, source files for the projects are in the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

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

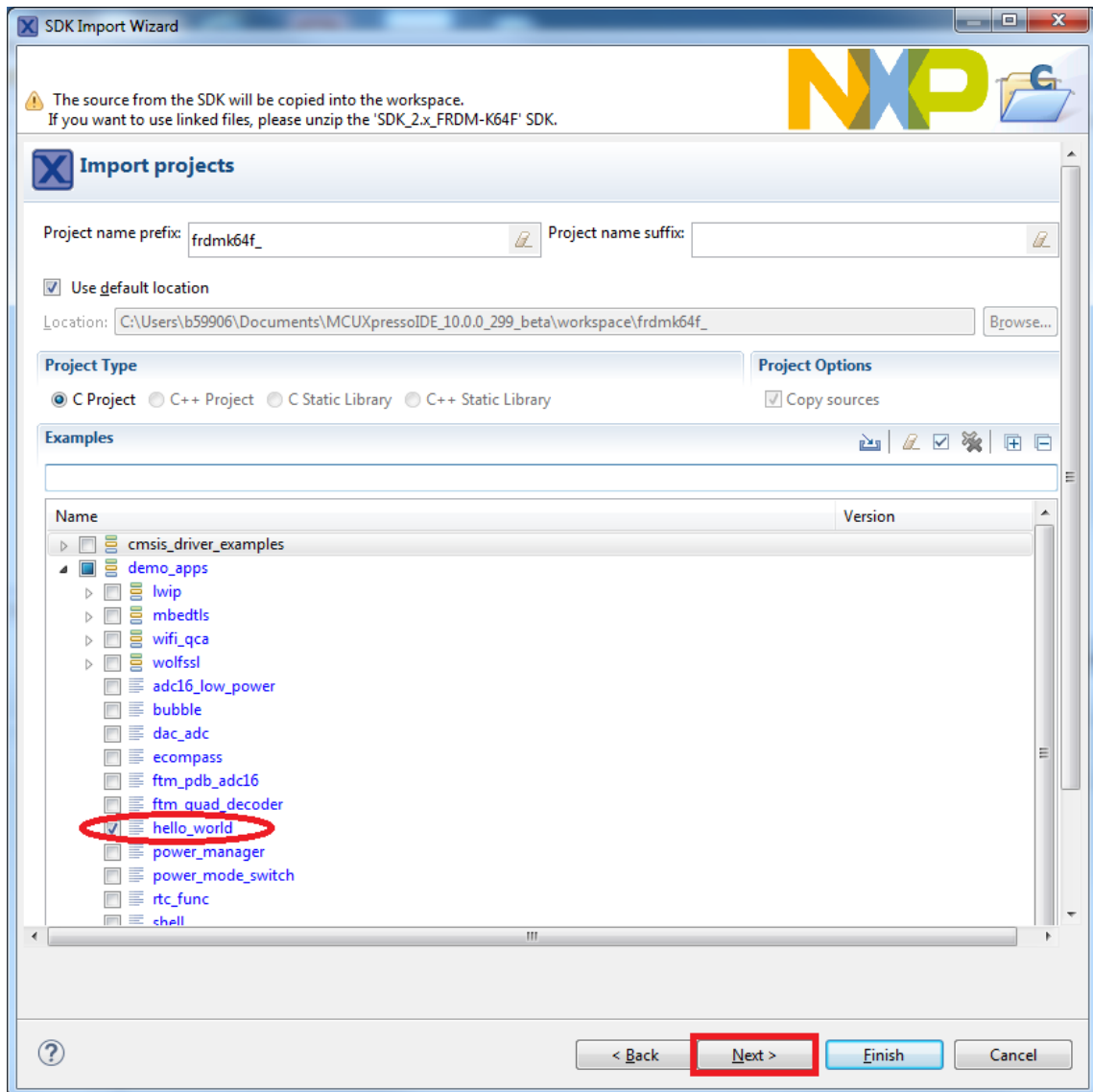
1. Drag and drop the SDK zip file into the **Installed SDKs** view to install an SDK. In the window that appears, click **OK** and wait until the import has finished.



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



3. Expand the demo\_apps folder and select hello\_world.
4. Click **Next**.



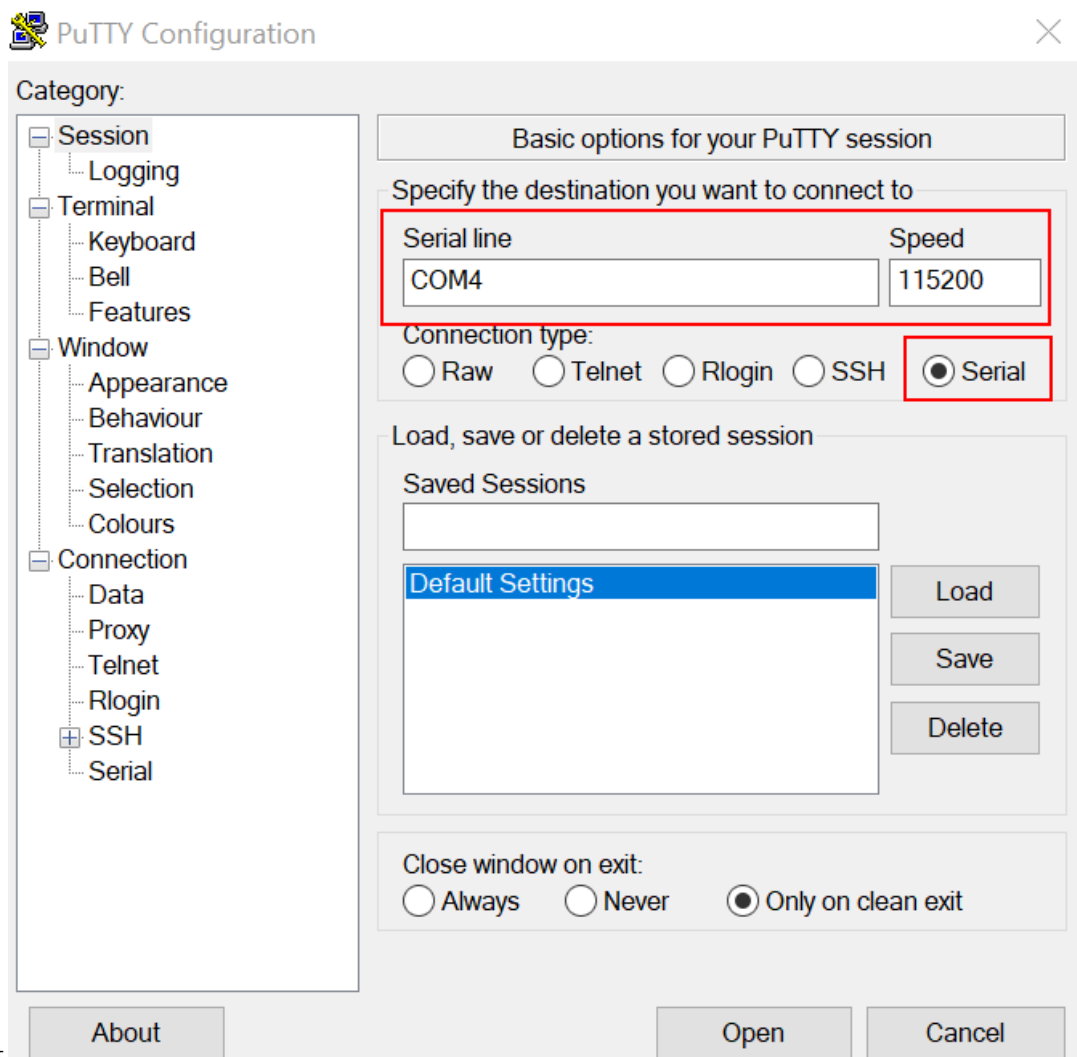
5. Ensure **Redlib: Use floating-point version of printf** is selected if the example prints floating-point numbers on the terminal for demo applications such as `adc_basic`, `adc_burst`, `adc_dma`, and `adc_interrupt`. Otherwise, it is not necessary to select this option. Then, click **Finish**.

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

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

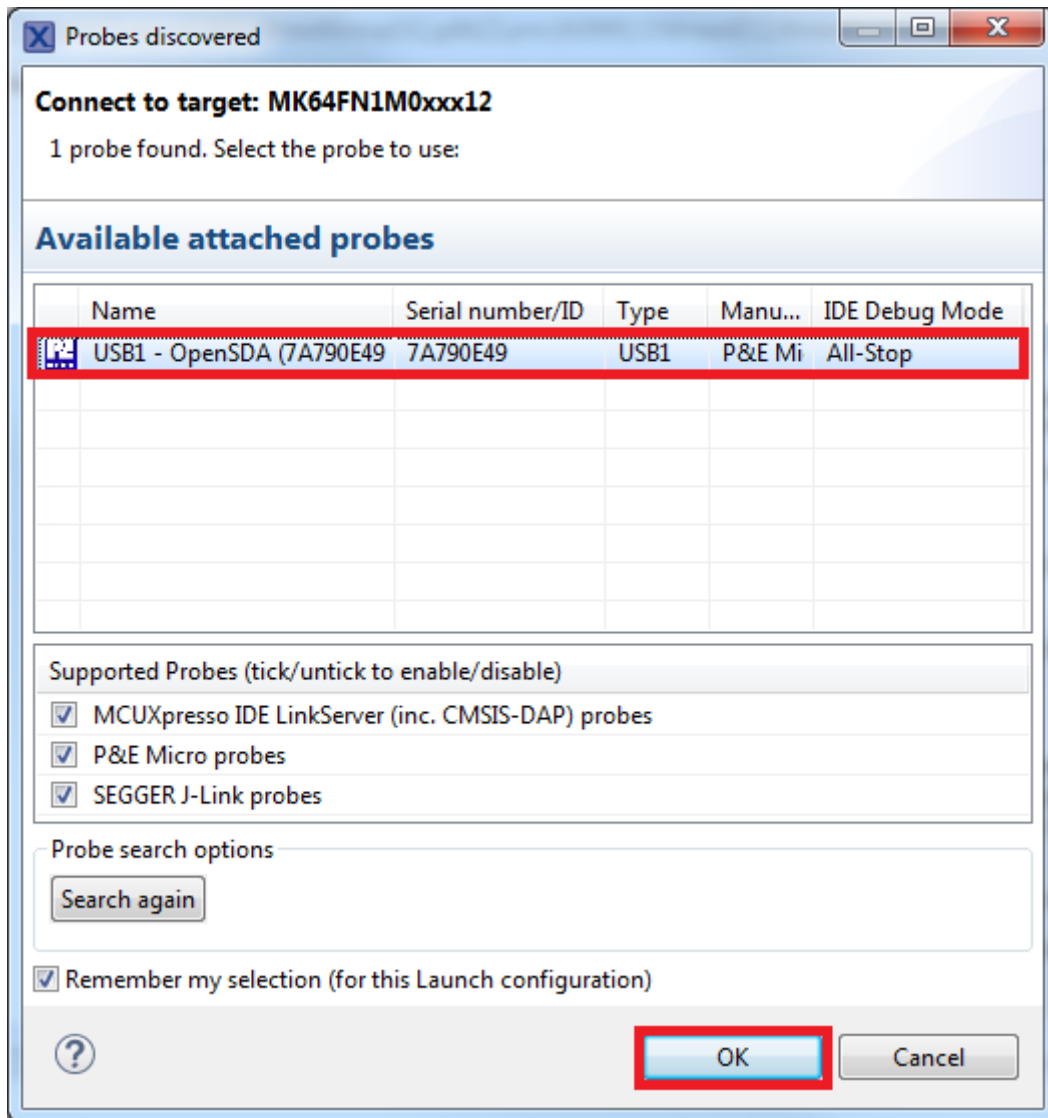
1. Ensure the host driver for the debugger firmware has been installed. See [On-board debugger](#).
2. Connect the development platform to your PC via a USB cable.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see [How to determine COM port](#)). Configure the terminal with these settings:
  1. 115200 or 9600 baud rate, depending on your board (reference `BOARD_DEBUG_UART_BAUDRATE` variable in `board.h` file)
  2. No parity

3. 8 data bits



4. 1 stop bit

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



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

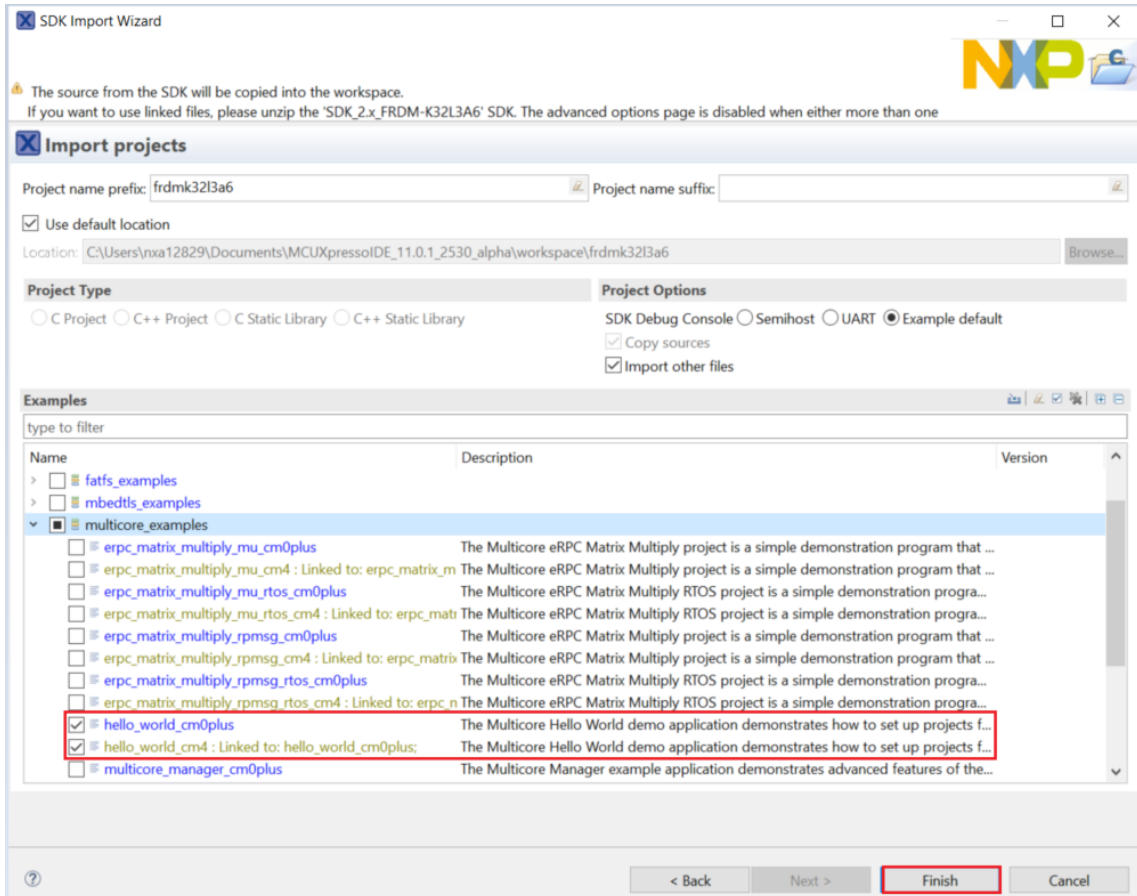


The `hello_world` application is now running and a banner is displayed on the terminal. If not, check your terminal settings and connections.

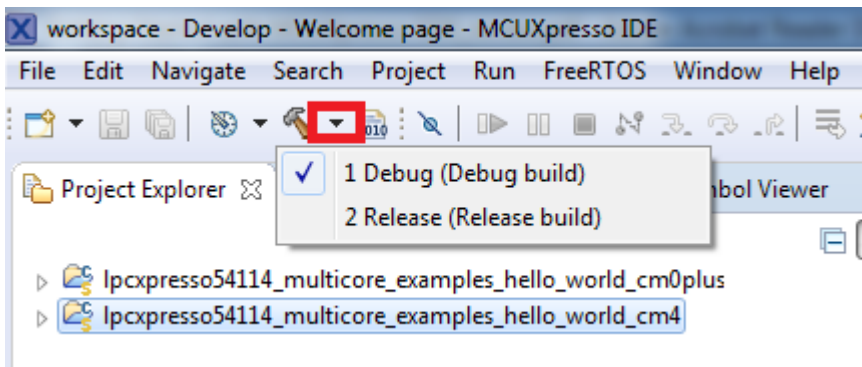


**Build a multicore example application** This section describes the steps required to configure MCUXpresso IDE to build, run, and debug multicore example applications. The following steps can be applied to any multicore example application in the MCUXpresso SDK. Here, the dual-core version of hello\_world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

1. Multicore examples are imported into the workspace in a similar way as single core applications, explained in **Build an example application**. When the SDK zip package for LPCXpresso54114 is installed and available in the **Installed SDKs** view, click **Import SDK example(s)...** on the Quickstart Panel. In the window that appears, expand the **LPCxx** folder and select **LPC54114J256**. Then, select **lpcxpresso54114** and click **Next**.
2. Expand the multicore\_examples/hello\_world folder and select **cm4**. The cm0plus counterpart project is automatically imported with the cm4 project, because the multicore examples are linked together and there is no need to select it explicitly. Click **Finish**.

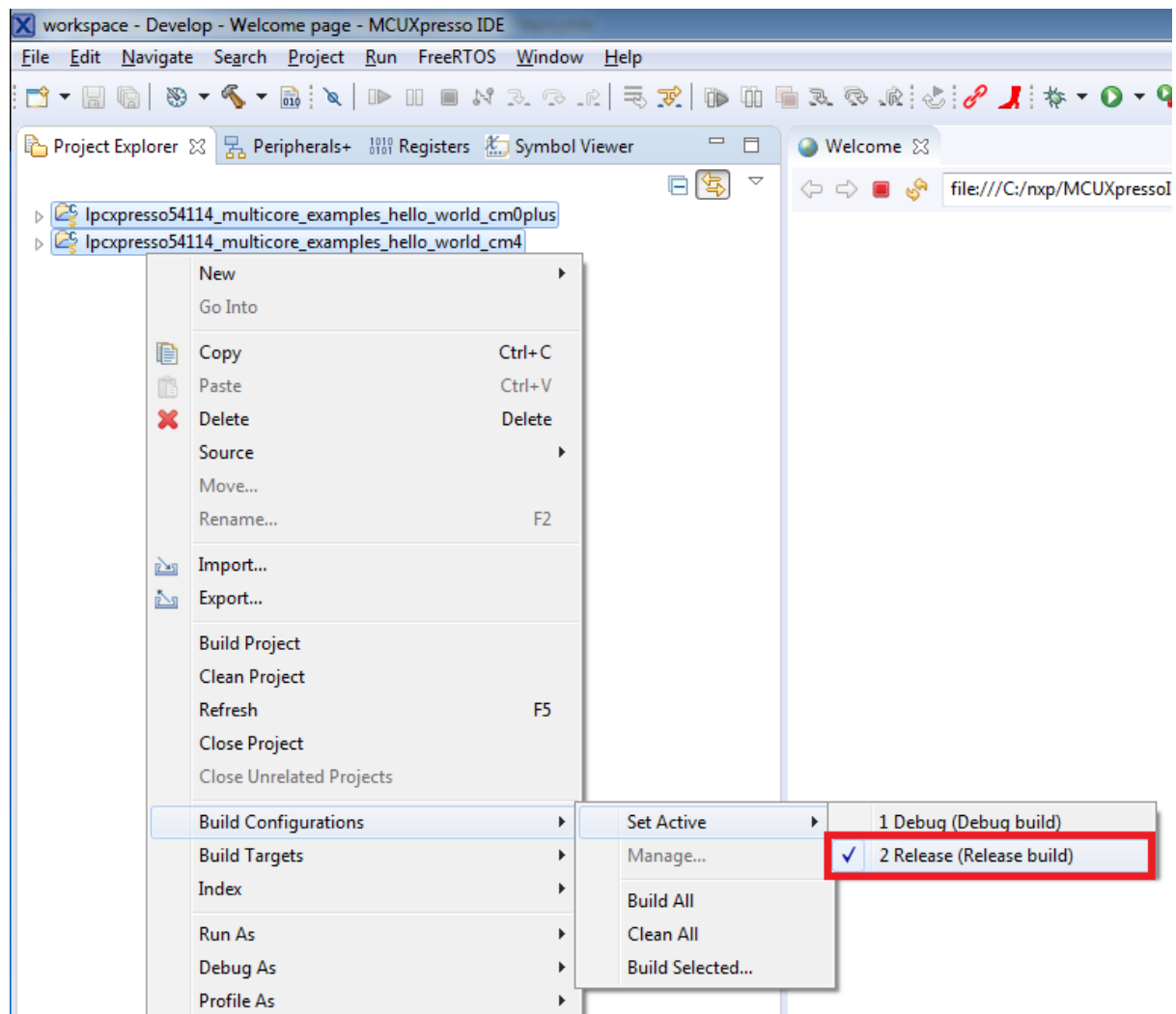


3. Now, two projects should be imported into the workspace. To start building the multicore application, highlight the `lpcxpresso54114_multicore_examples_hello_world_cm4` project (multicore master project) in the Project Explorer. Then choose the appropriate build target, **Debug** or **Release**, by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

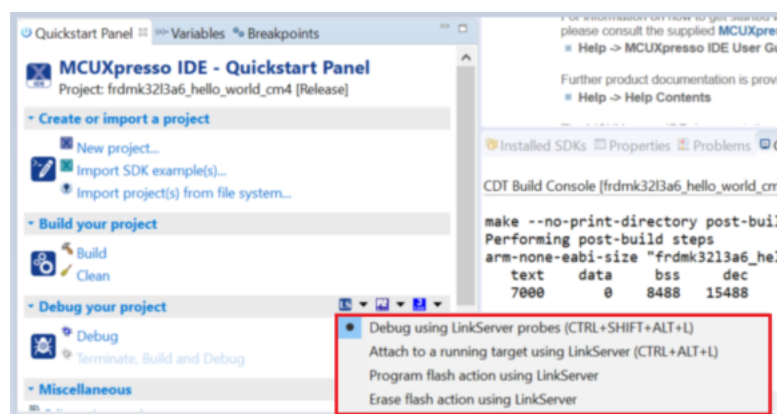


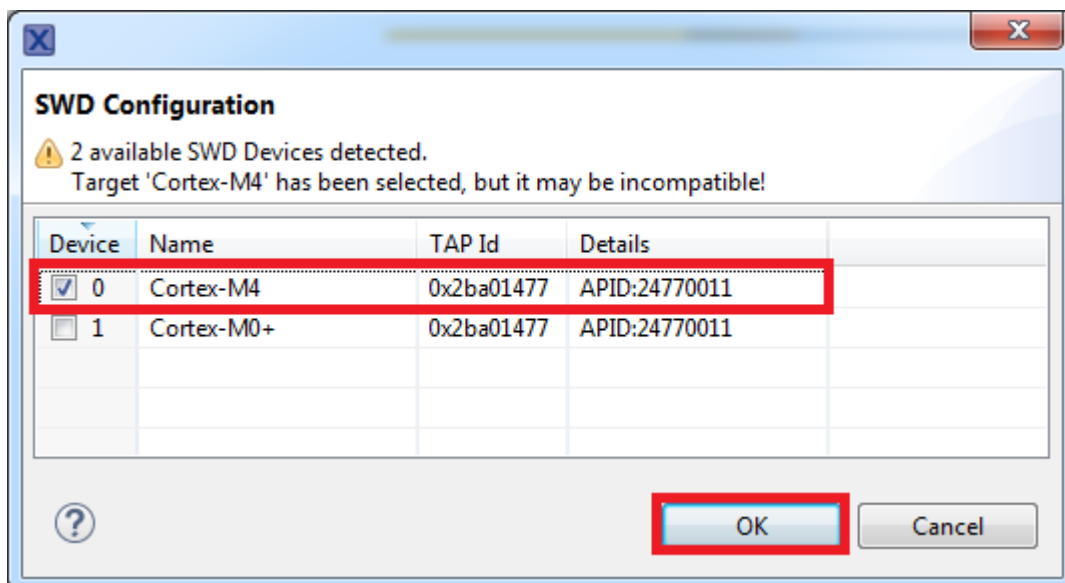
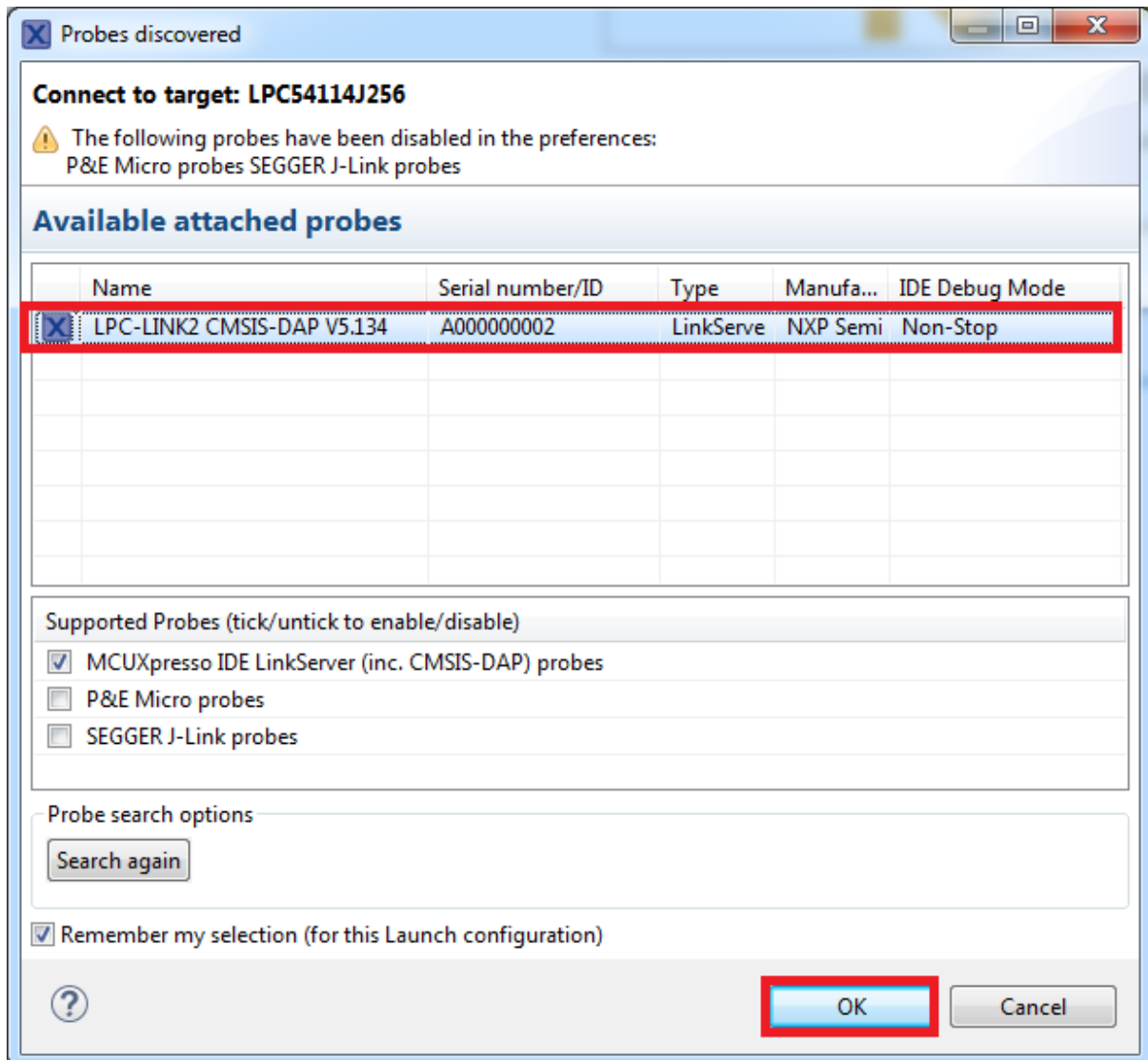
The project starts building after the build target is selected. Because of the project reference settings in multicore projects, triggering the build of the primary core application (cm4) also causes the referenced auxiliary core application (cm0plus) to build.

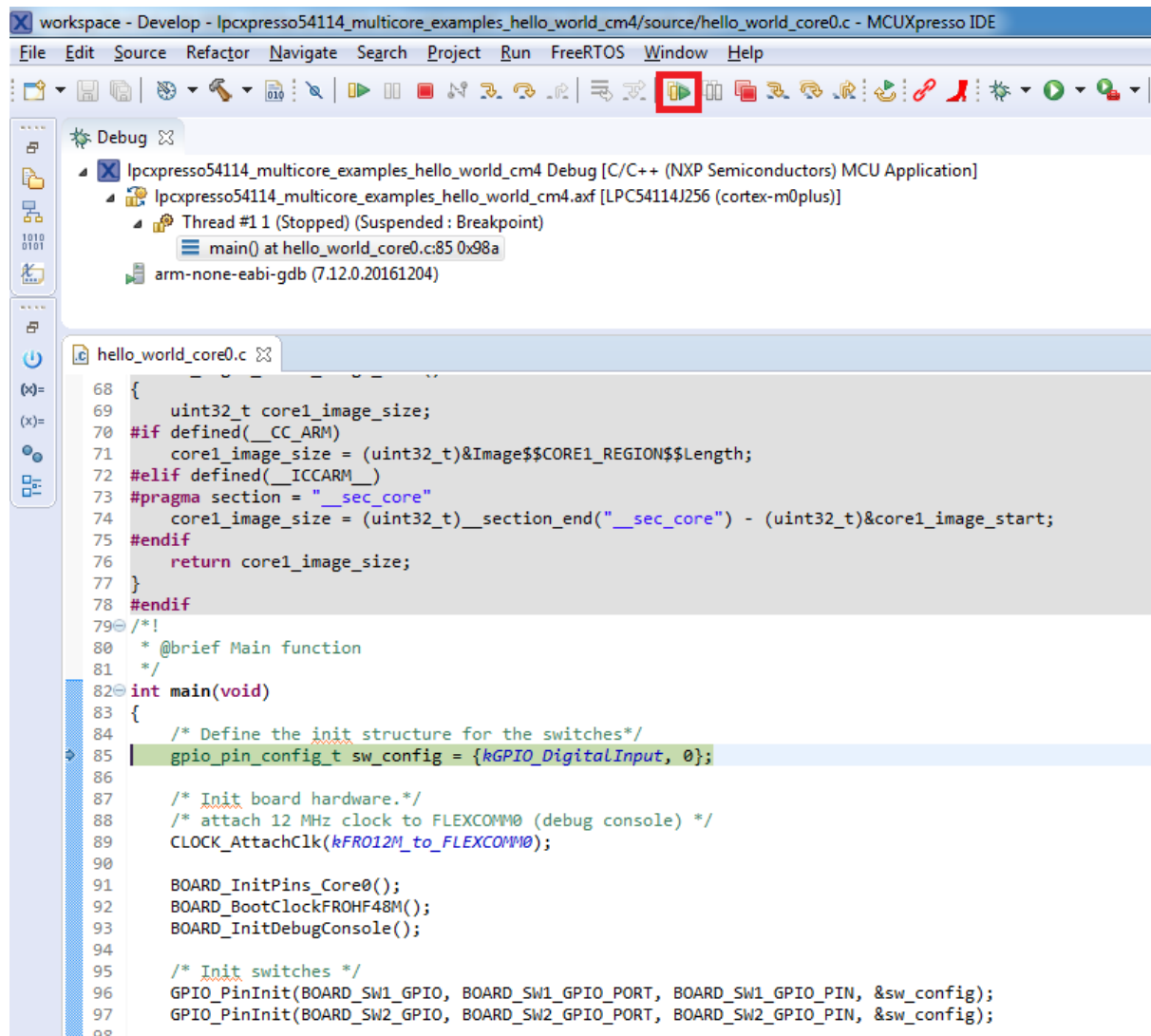
**Note:** When the **Release** build is requested, it is necessary to change the build configuration of both the primary and auxiliary core application projects first. To do this, select both projects in the Project Explorer view and then right click which displays the context-sensitive menu. Select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



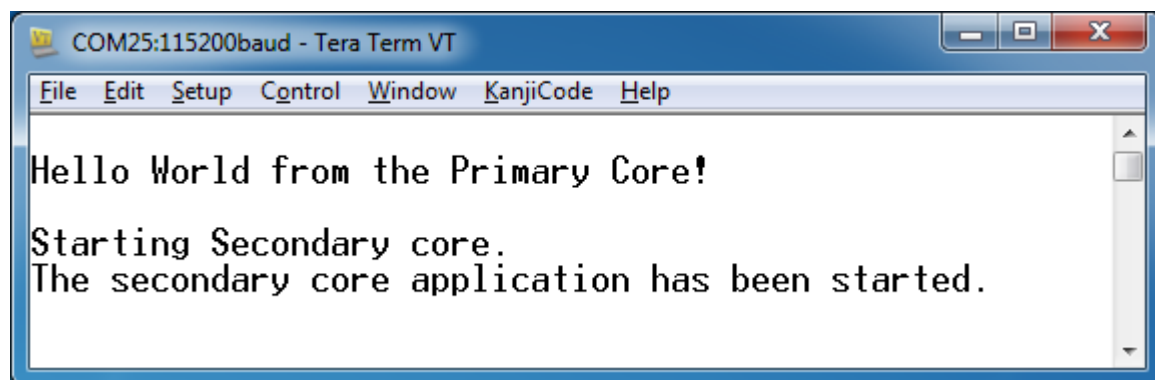
**Run a multicore example application** The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform all steps as described in **Run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





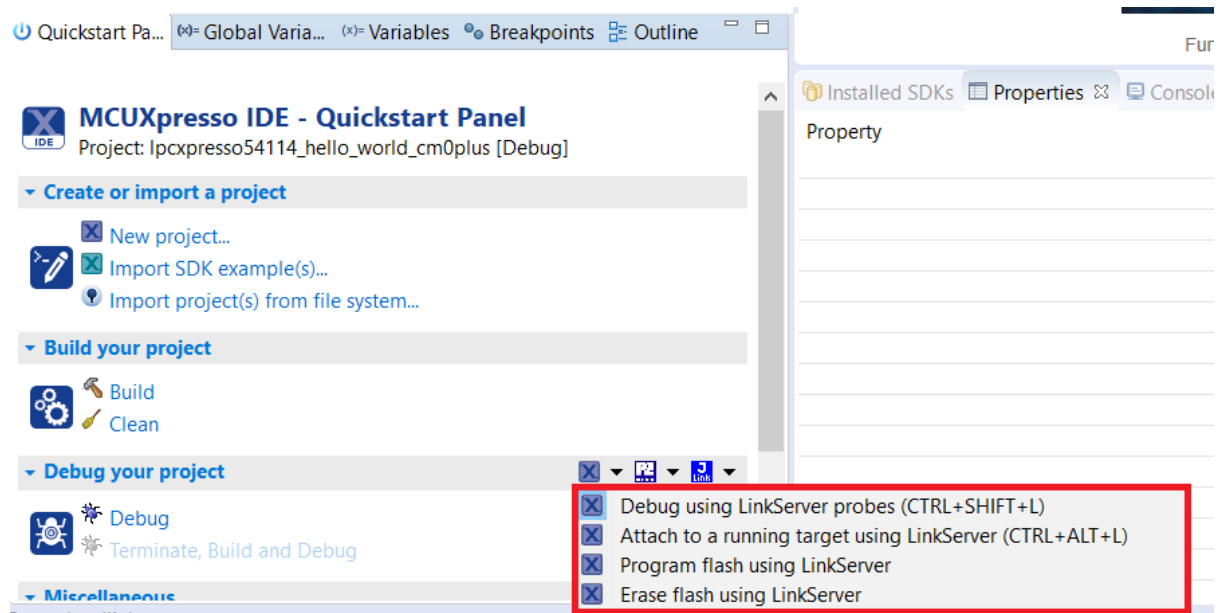


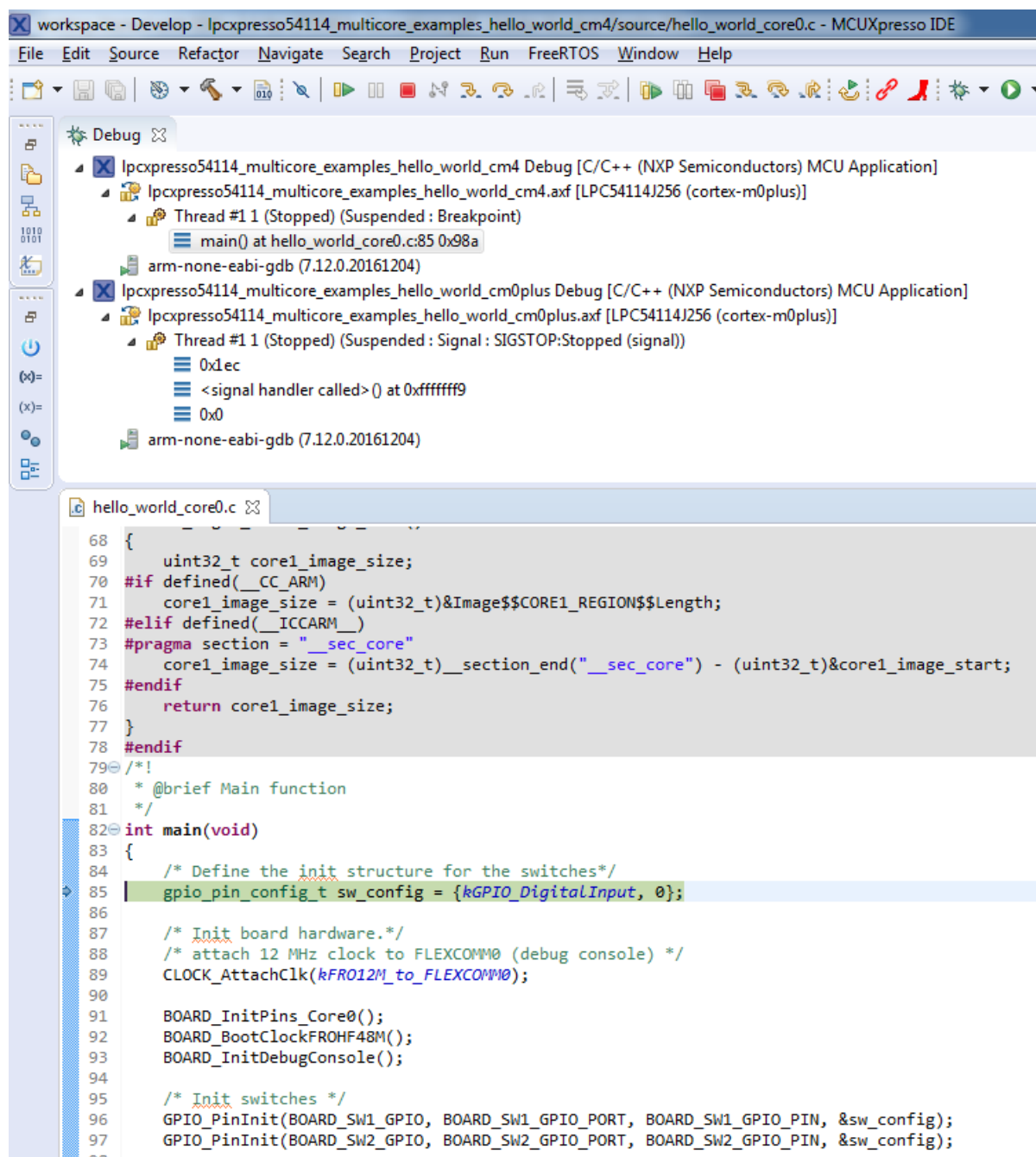
After clicking the “Resume All Debug sessions” button, the hello\_world multicore application runs and a banner is displayed on the terminal. If this is not the case, check your terminal settings and connections.



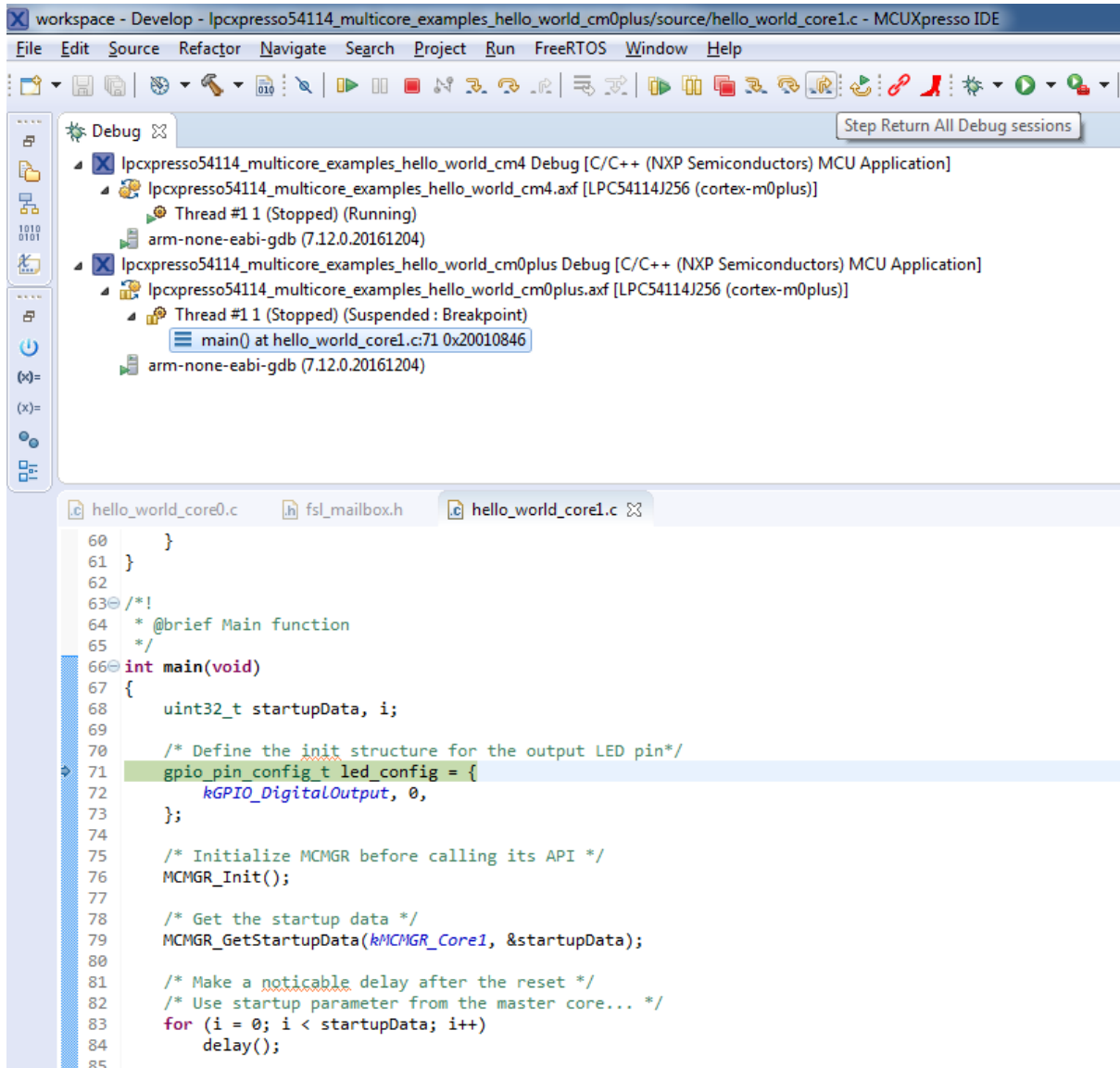
An LED controlled by the auxiliary core starts flashing, indicating that the auxiliary core has been released from the reset and running correctly. It is also possible to debug both sides of the multicore application in parallel. After creating the debug session for the primary core, perform same steps also for the auxiliary core application. Highlight the lpcxpresso54114\_multicore\_examples\_hello\_world\_cm0plus project (multicore slave project) in the Project Explorer. On the Quickstart Panel, click “Debug ‘lpcxpresso54114\_multicore\_examples\_hello\_world\_cm0plus’ [Debug]” to launch the second debug

session.

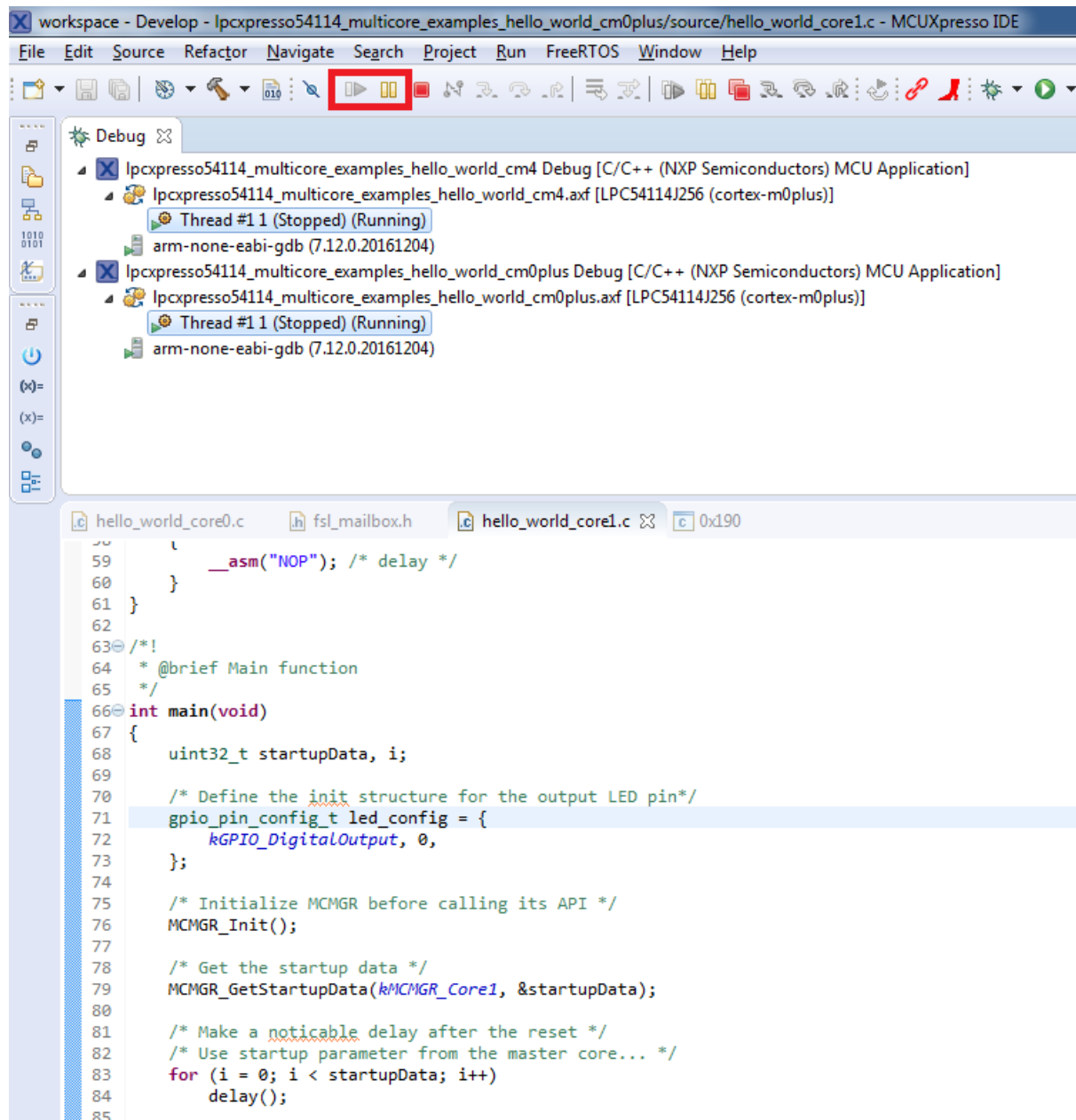


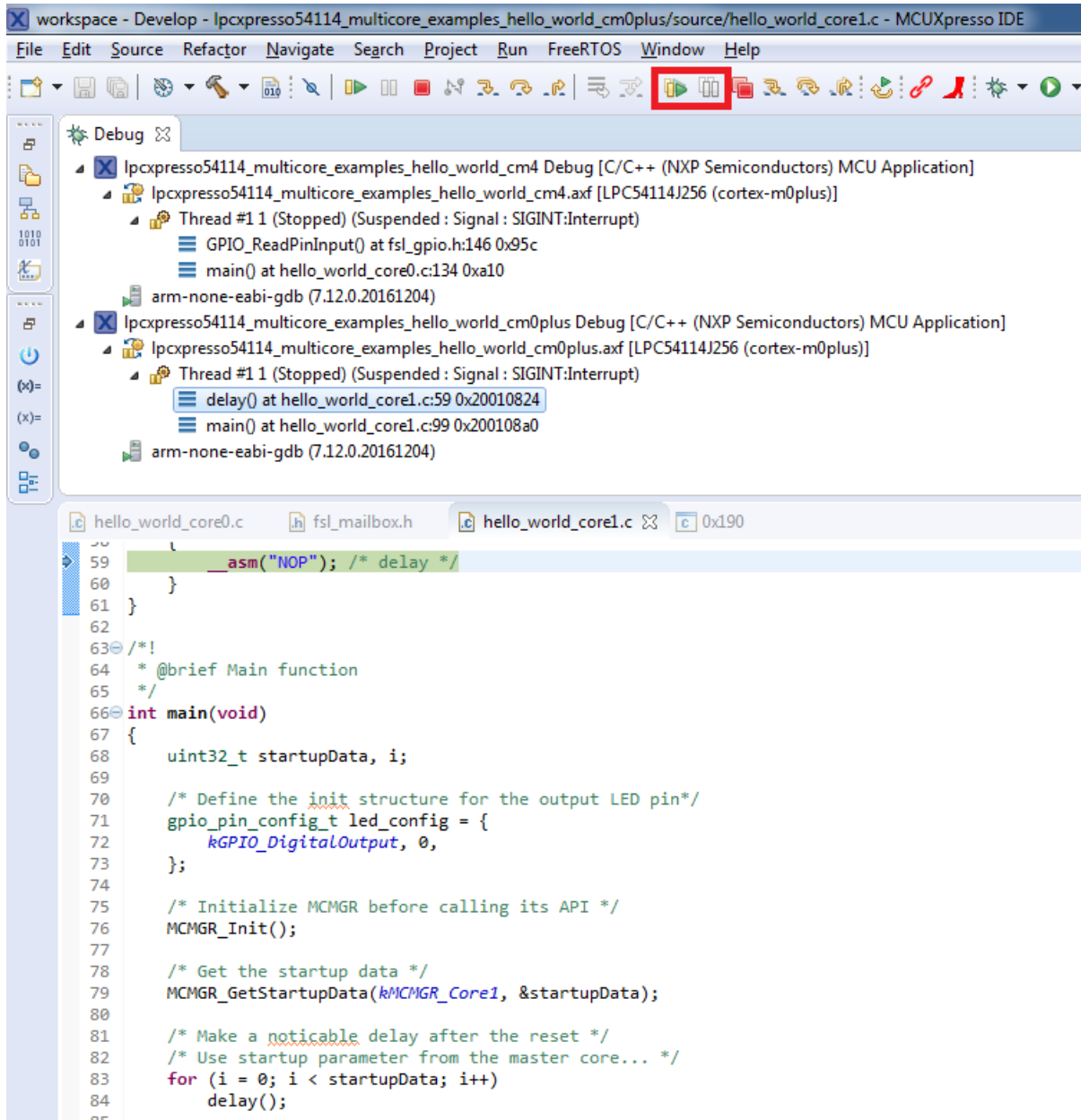


Now, the two debug sessions should be opened, and the debug controls can be used for both debug sessions depending on the debug session selection. Keep the primary core debug session selected by clicking the “Resume” button. The hello\_world multicore application then starts running. The primary core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



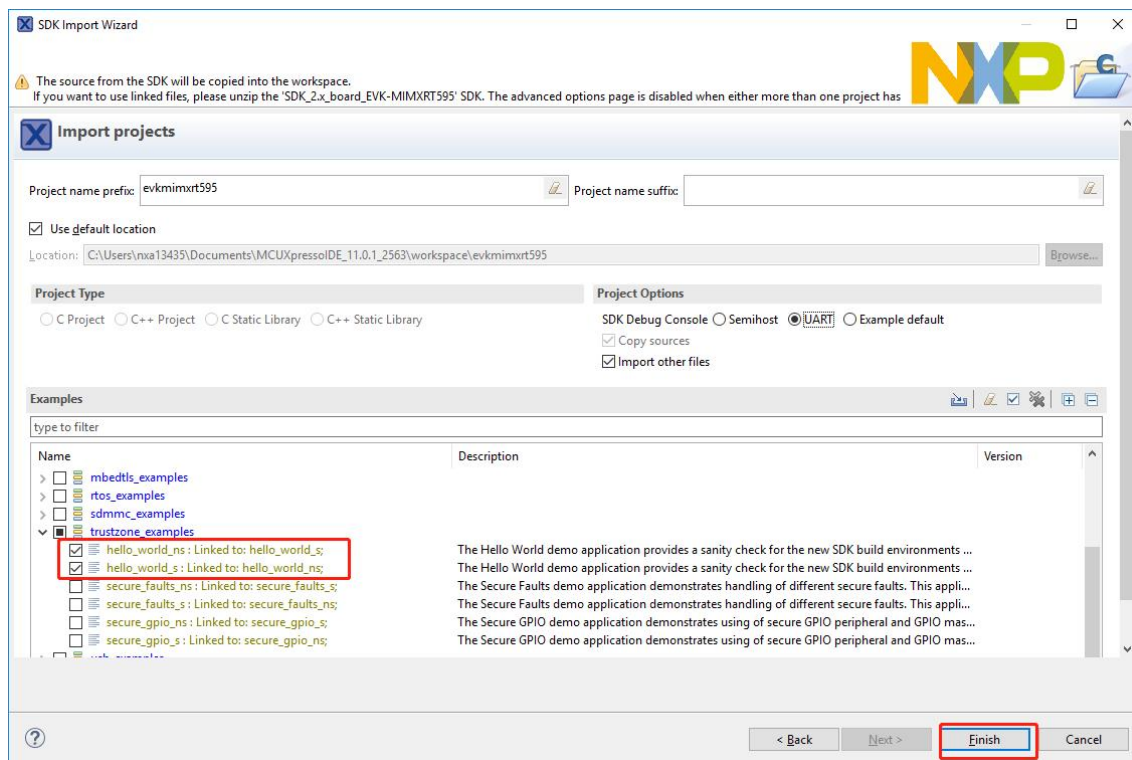
At this point, it is possible to suspend and resume individual cores independently. It is also possible to make synchronous suspension and resumption of both the cores. This is done either by selecting both opened debug sessions (multiple selections) and clicking the “Suspend” / “Resume” control button,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



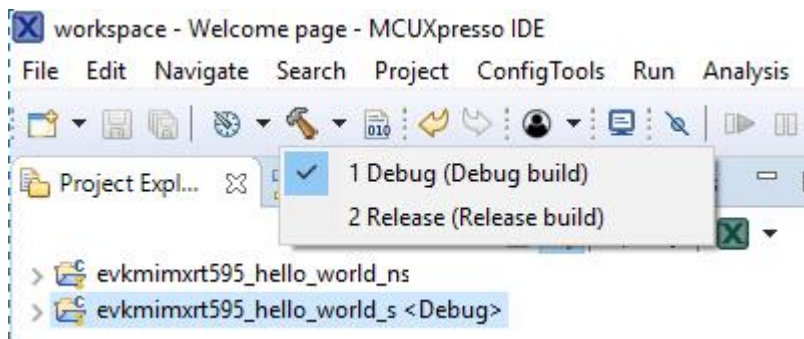


**Build a TrustZone example application** This section describes the steps required to configure MCUXpresso IDE to build, run, and debug TrustZone example applications. The TrustZone version of the `hello_world` example application targeted for the MIMXRT595-EVK hardware platform is used as an example,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

1. TrustZone examples are imported into the workspace in a similar way as single core applications. When the SDK zip package for MIMXRT595-EVK is installed and available in the **Installed SDKs** view, click **Import SDK example(s)...** on the Quickstart Panel. In the window that appears, expand the **MIMXRT500** folder and select **MIMXRT595S**. Then, select **evkmimxrt595** and click **Next**.
2. Expand the `trustzone_examples/` folder and select `hello_world_s`. Because TrustZone examples are linked together, the non-secure project is automatically imported with the secure project, and there is no need to select it explicitly. Then, click **Finish**.

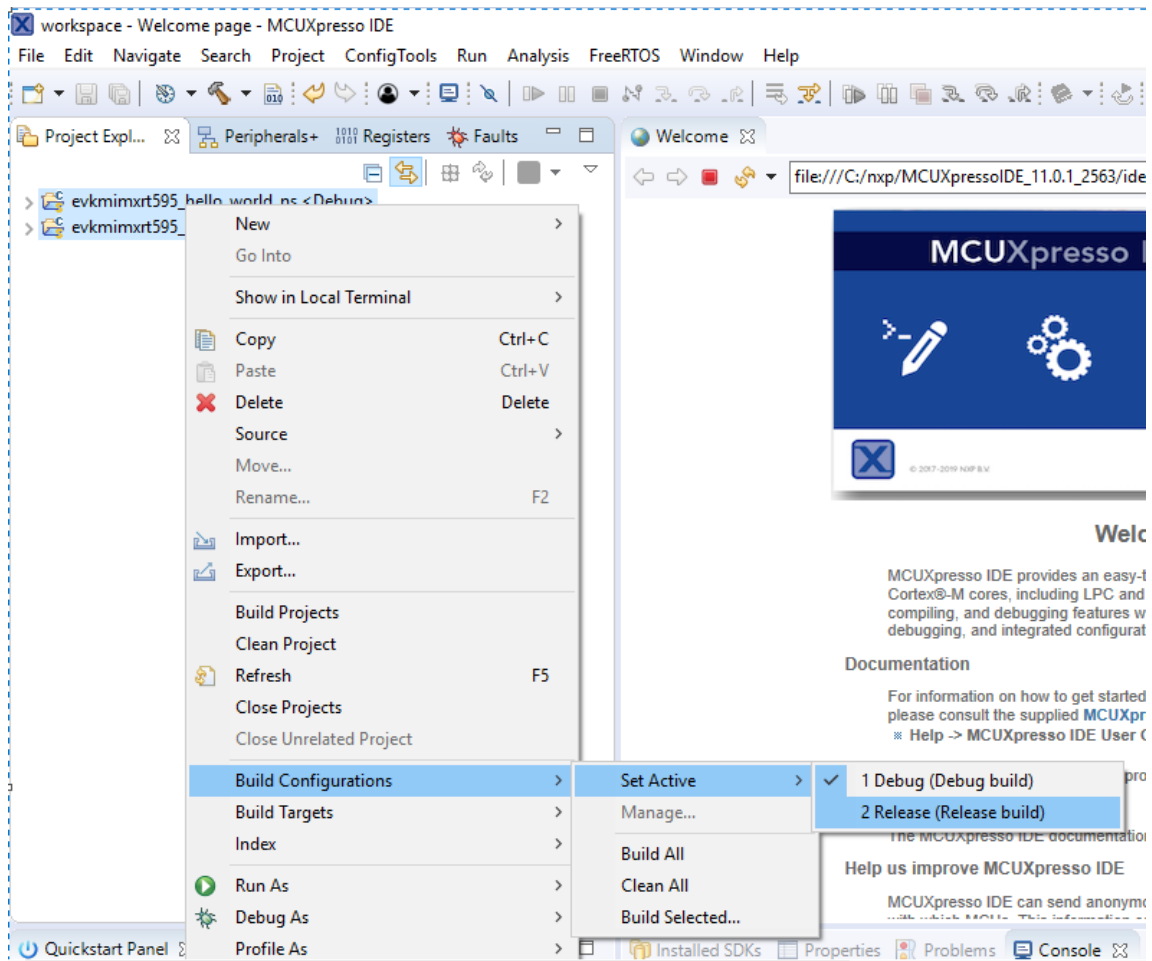


- Now, two projects should be imported into the workspace. To start building the TrustZone application, highlight the `evkmimxrt595_hello_world_s` project (TrustZone master project) in the Project Explorer. Then, choose the appropriate build target, **Debug** or **Release**, by clicking the downward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



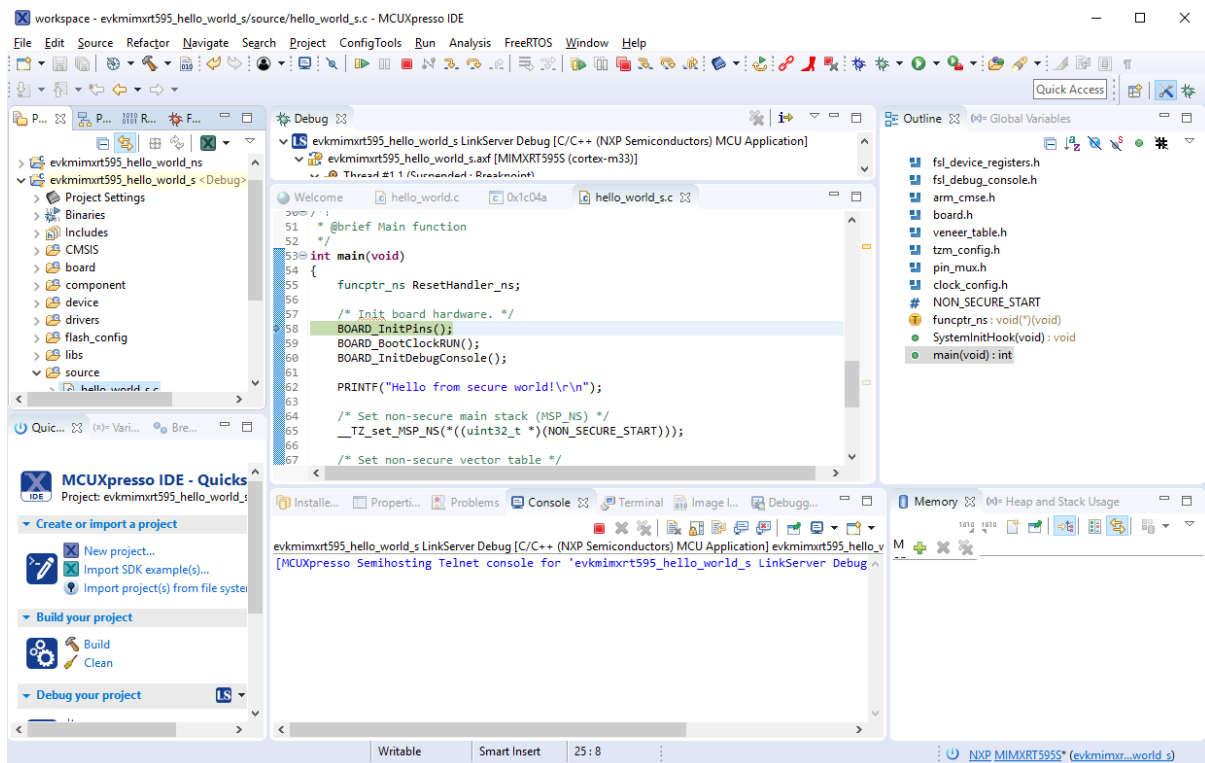
The project starts building after the build target is selected. It is requested to build the application for the secure project first, because the non-secure project must know the secure project since CMSE library when running the linker. It is not possible to finish the non-secure project linker when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

**Note:** When the **Release** build is requested, it is necessary to change the build configuration of both the secure and non-secure application projects first. To do this, select both projects in the Project Explorer view by clicking to select the first project, then using shift-click or control-click to select the second project. Right click in the Project Explorer view to display the context-sensitive menu and select **Build Configurations > Set Active > Release**. This is also possible by using the menu item of **Project > Build Configuration > Set Active > Release**. After switching to the **Release** build configuration. Build the application for the secure project first.



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

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



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

## Run a demo application using IAR

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

**Note:** IAR Embedded Workbench for Arm version 8.32.3 is used in the following example, and the IAR toolchain should correspond to the latest supported version, as described in the *MCUXpresso SDK Release Notes*.

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

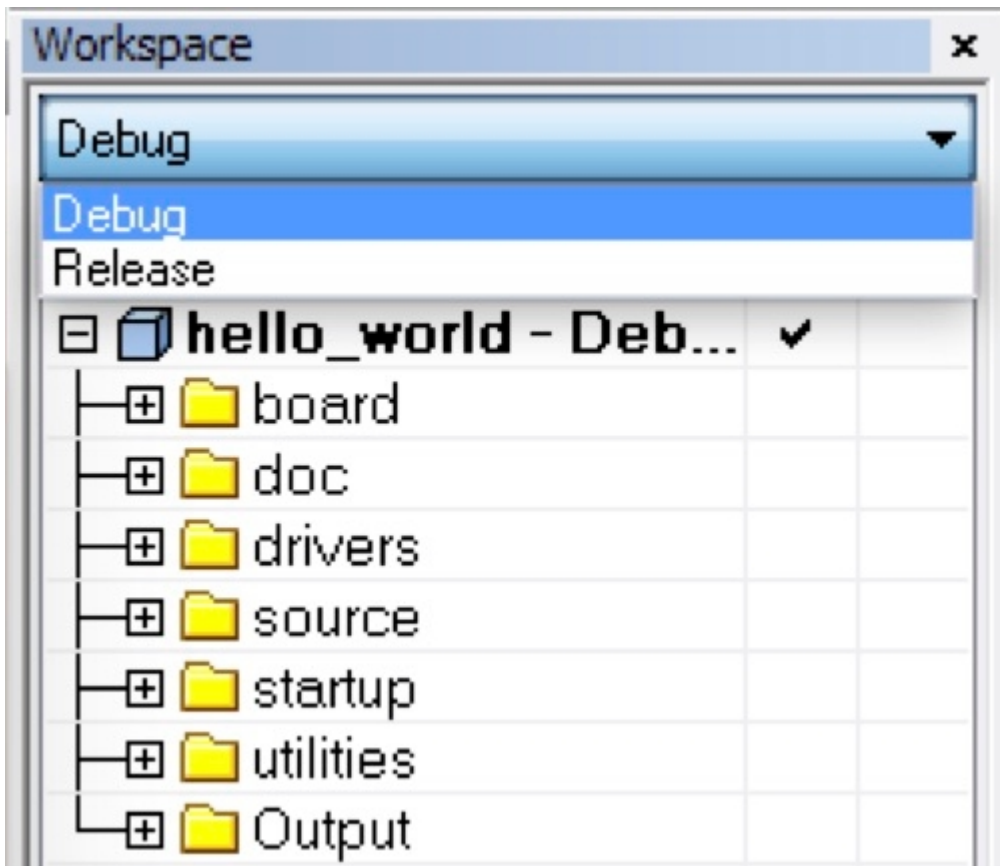
1. Open the desired demo application workspace. Most example application workspace files can be located using the following path:

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

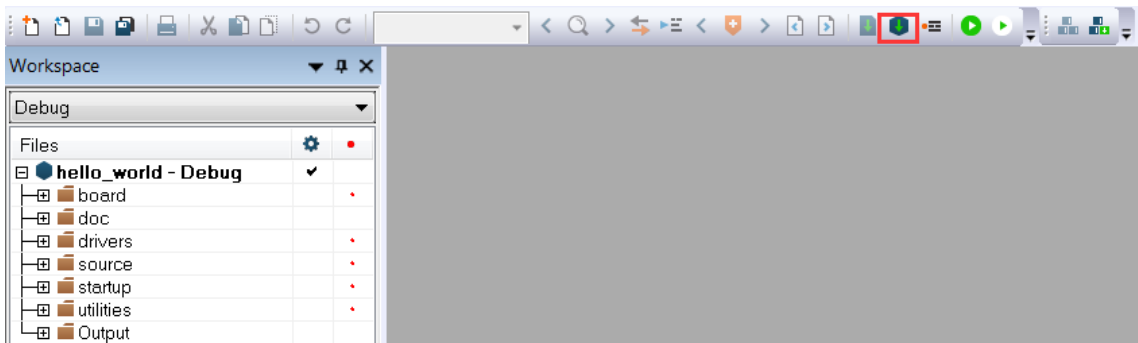
Other example applications may have additional folders in their path.

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

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



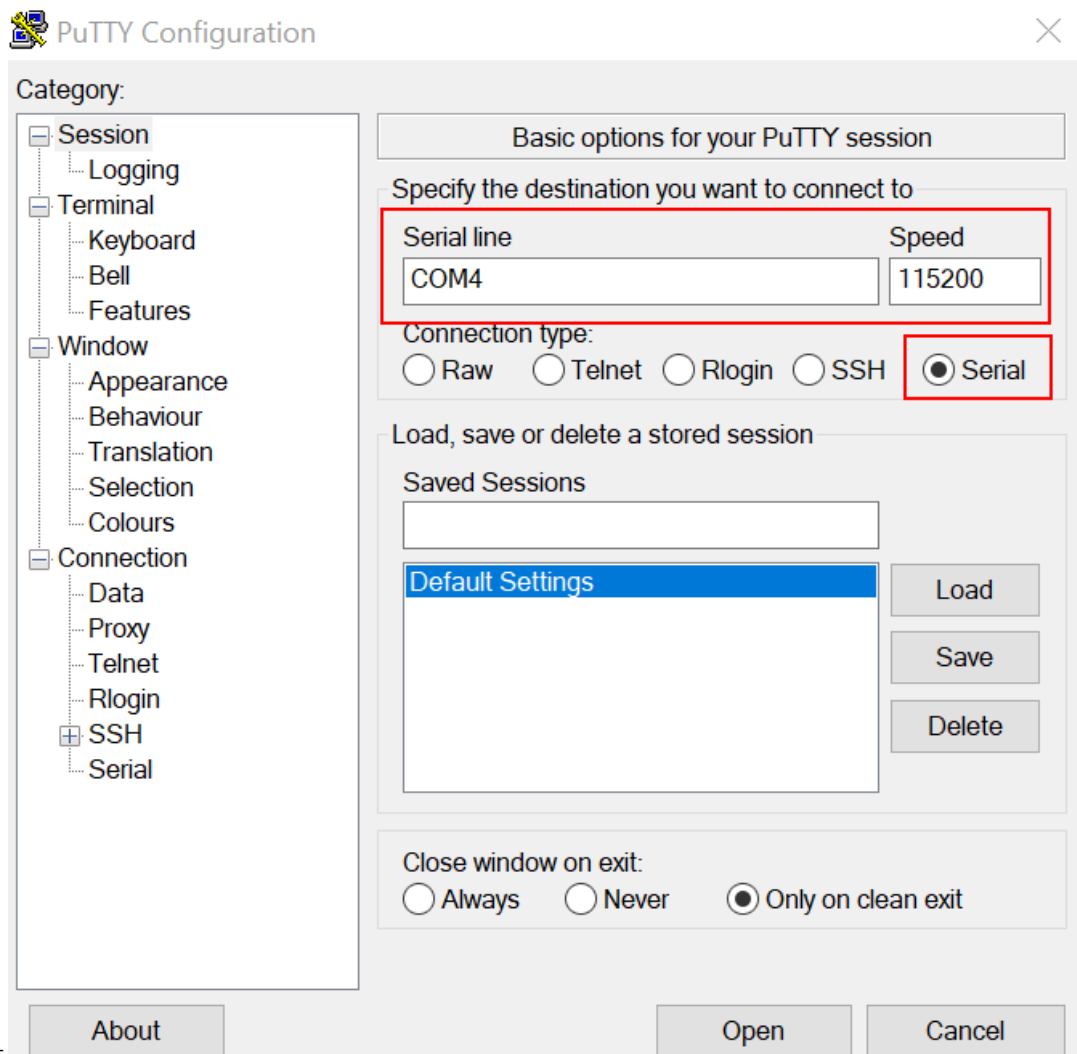
- To build the demo application, click **Make**, highlighted in red in following figure.



- 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.
- 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:
  - 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

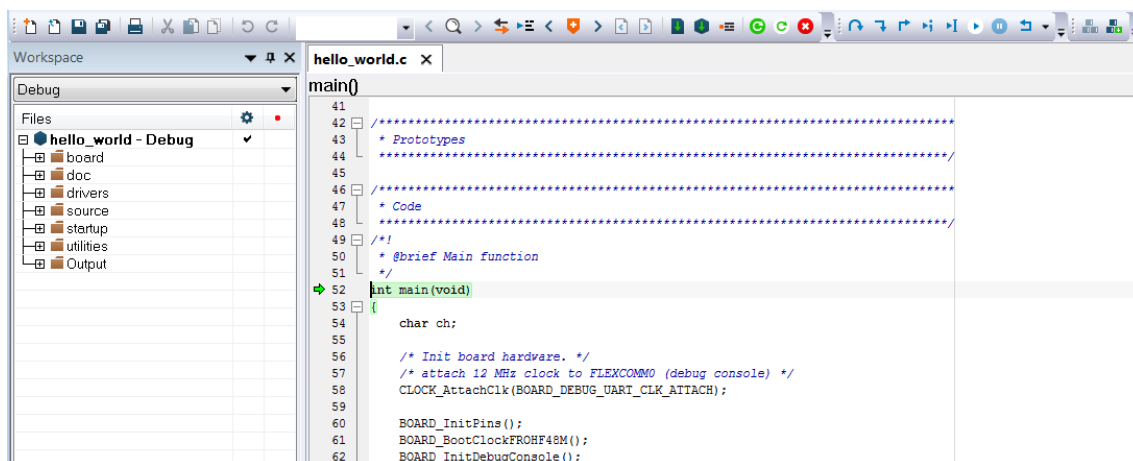


4. 1 stop bit

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
```

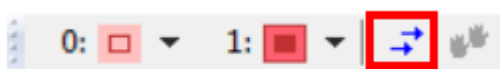
```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm4/iar/hello_world_cm4.eww
```

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.

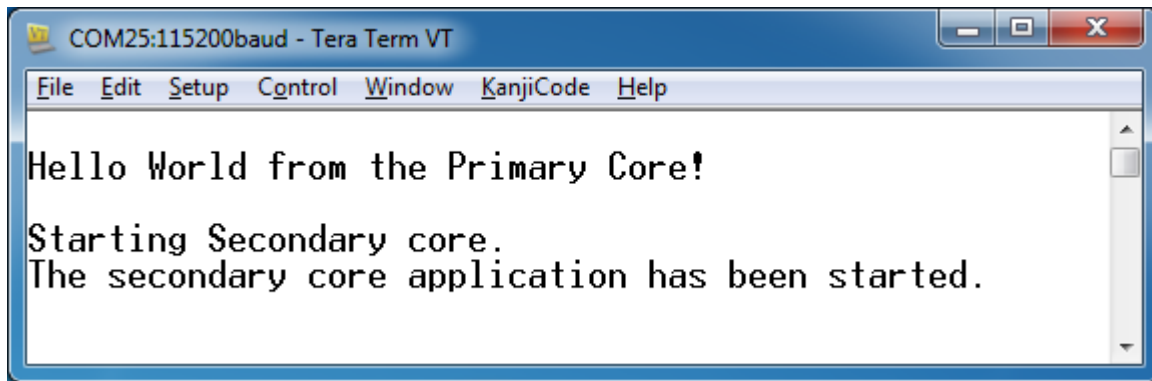
**Run a multicore example application** The primary core debugger handles flashing both primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 4 as described in **Run an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

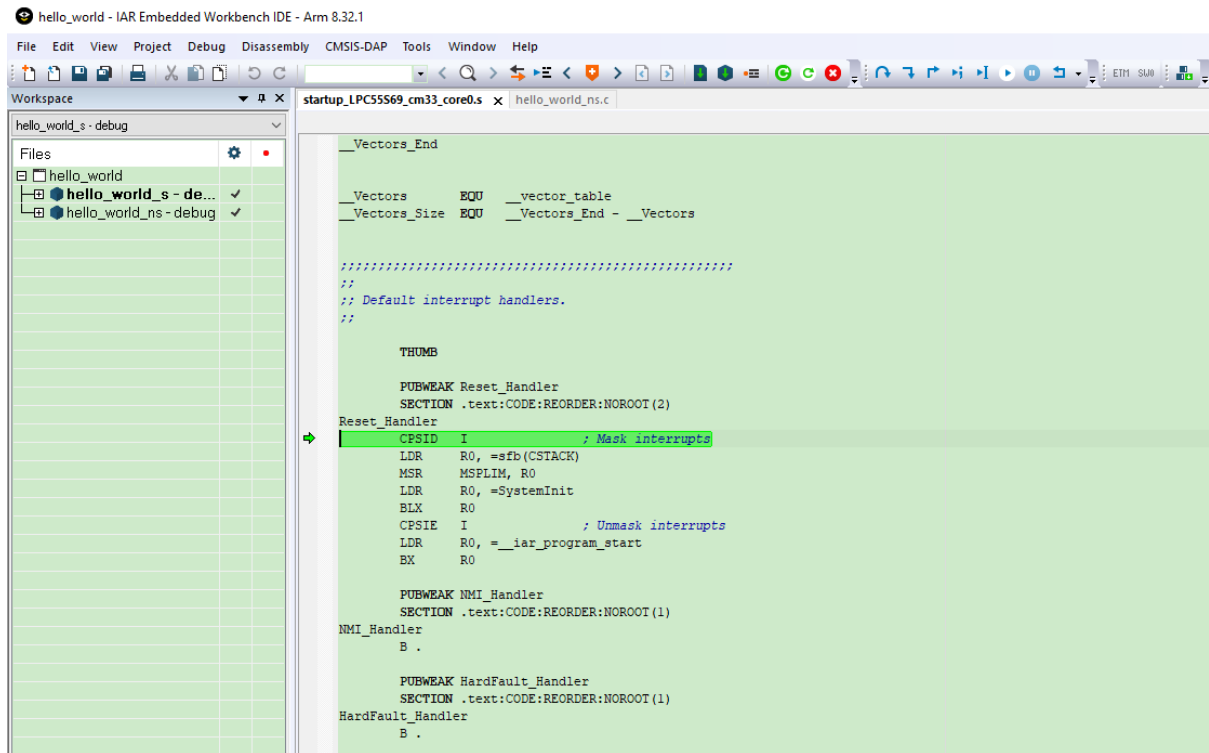
```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_ns/iar/hello_world_
↪ns.eww
```

```
<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
```

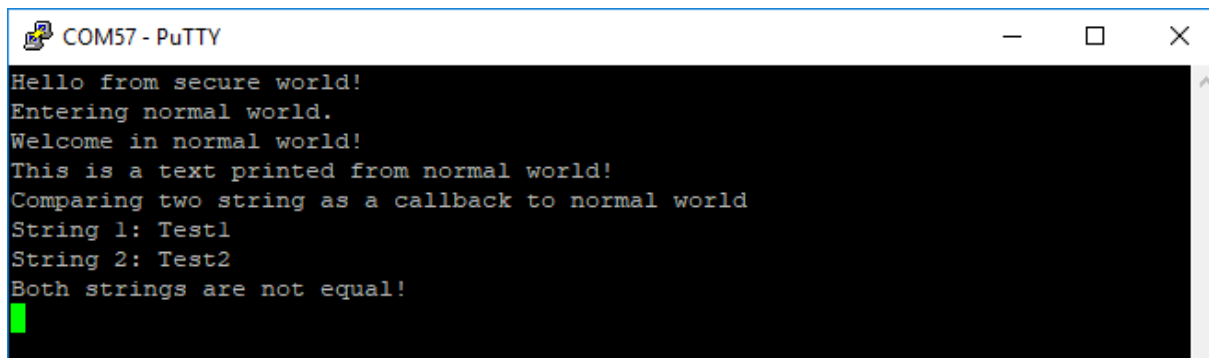
This project `hello_world.eww` contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another. Build both applications separately by clicking **Make**. It is requested to build the application for the secure project first, because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

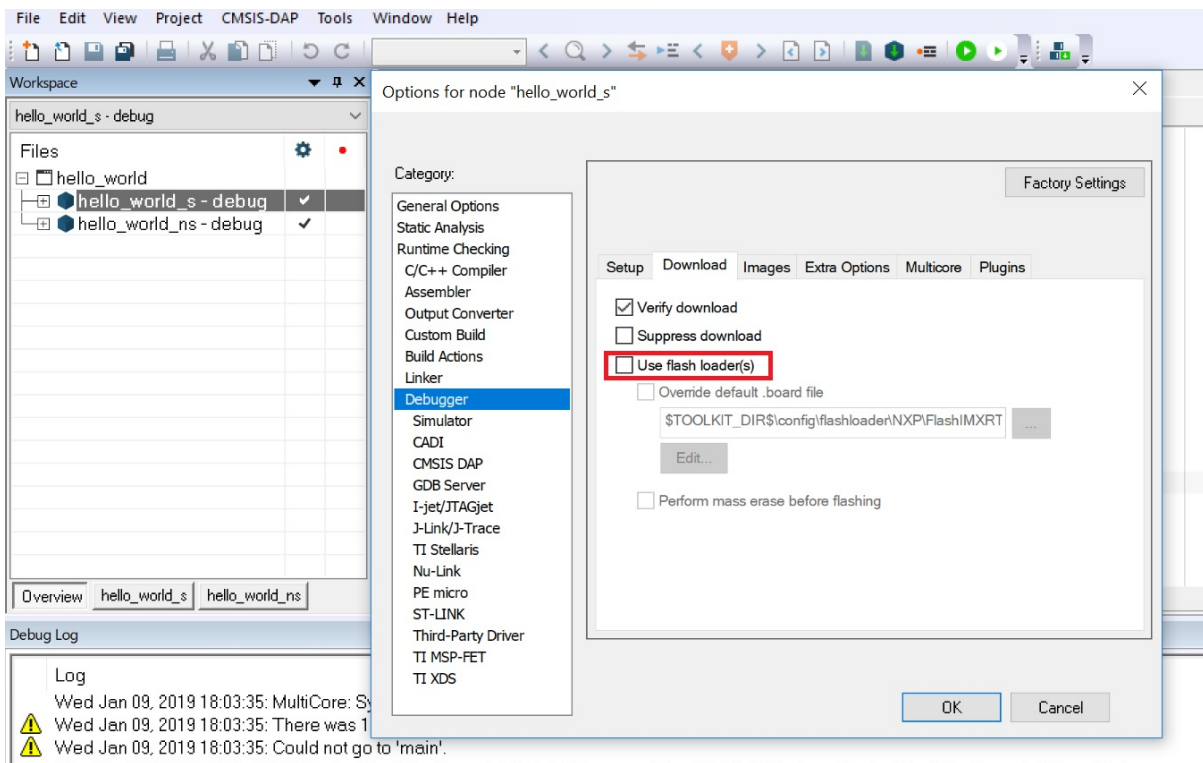


Run the code by clicking **Go** to start the application.

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.

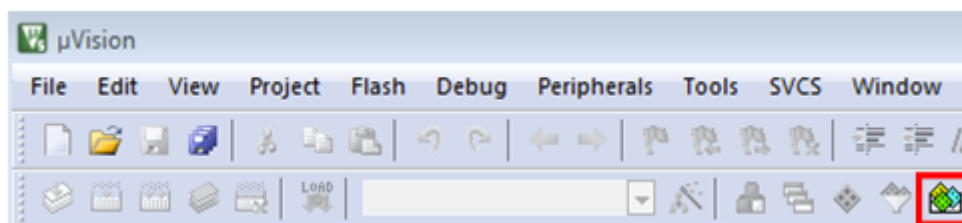


## Run a demo using Keil MDK/µVision

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:

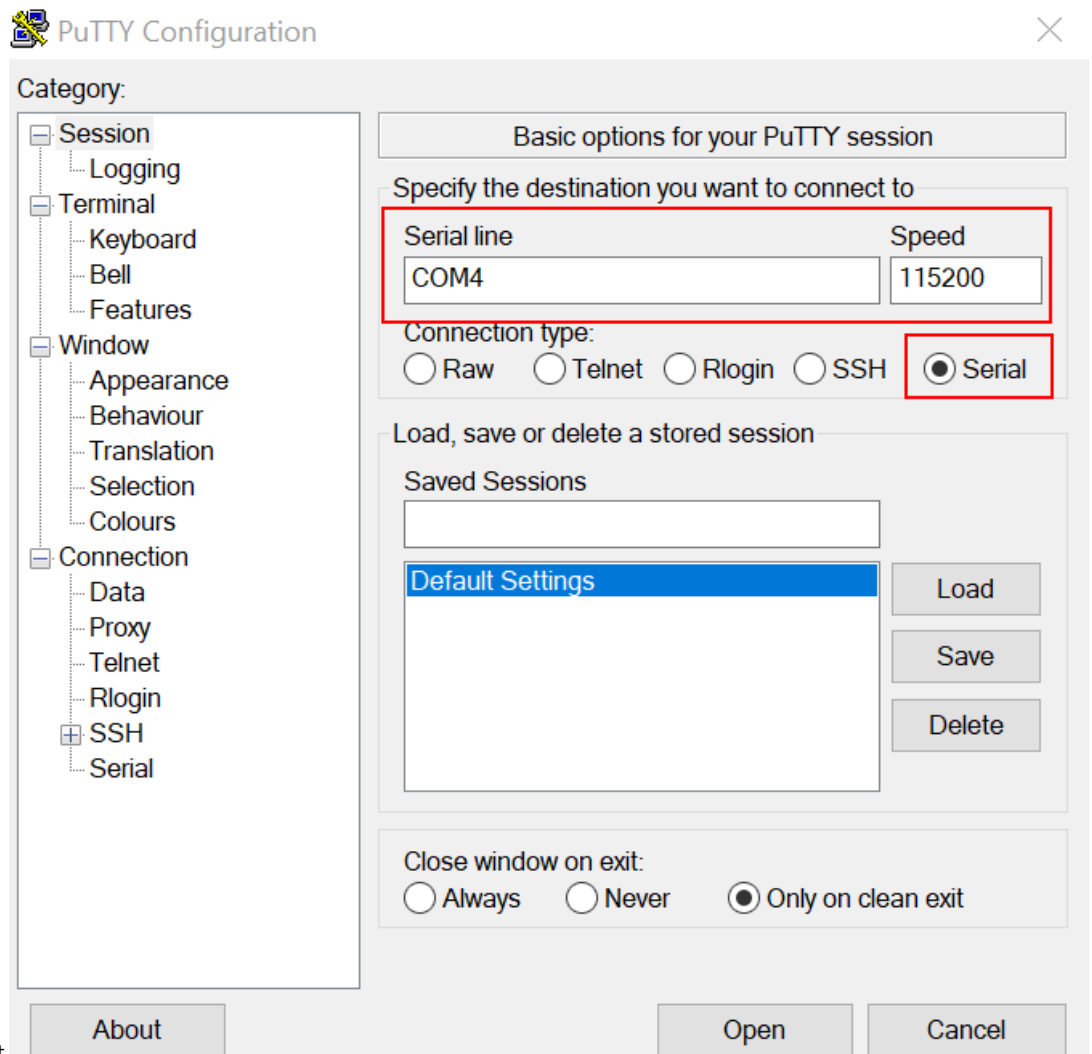
- To build the demo project, select **Rebuild**, highlighted in red.



- The build completes without errors.

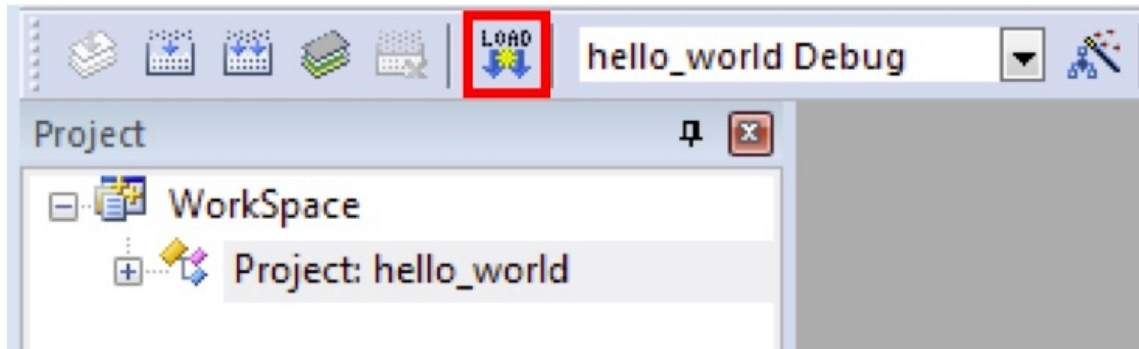
**Run an example application** To download and run the application, perform these steps:

- 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

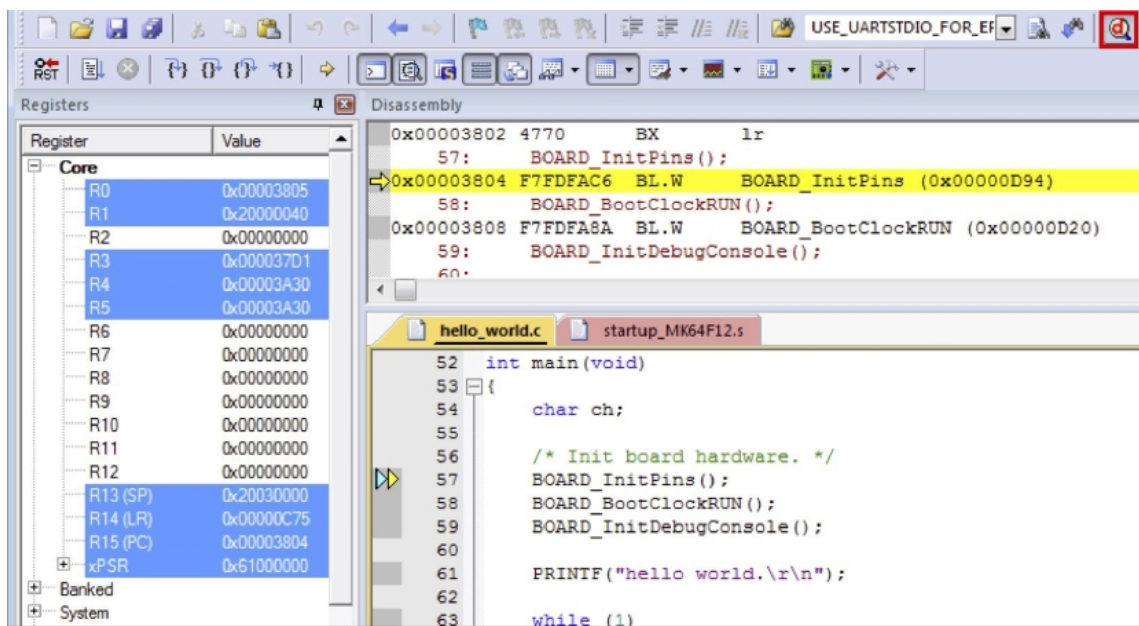


- 1 stop bit

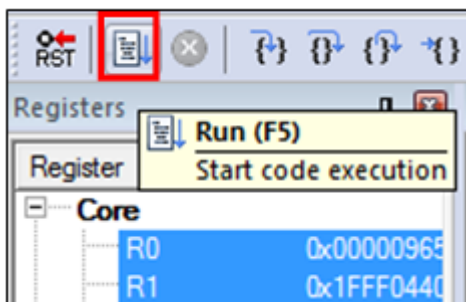
- In  $\mu$ Vision, after the application is built, click the **Download** button to download the application to the target.



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/ $\mu$ Vision workspaces are located in this folder:

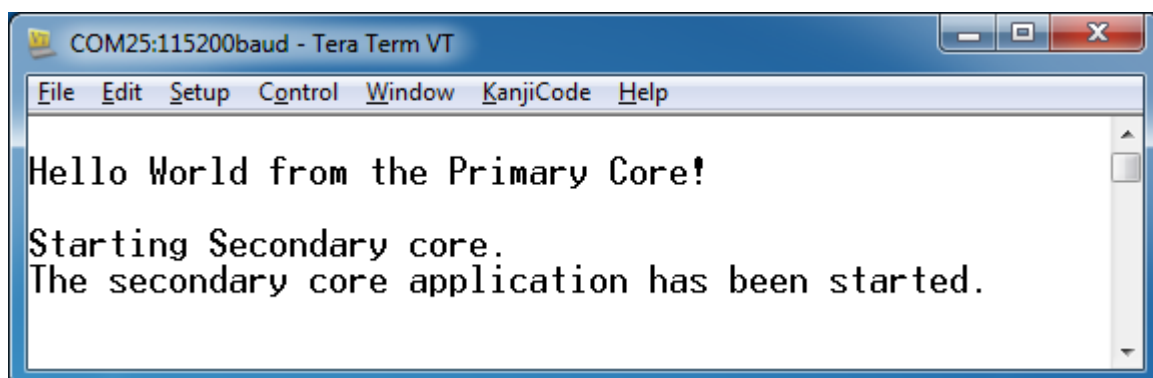
```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm0plus/mdk/hello_world_
↪cm0plus.uvmpw
```

```
<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.

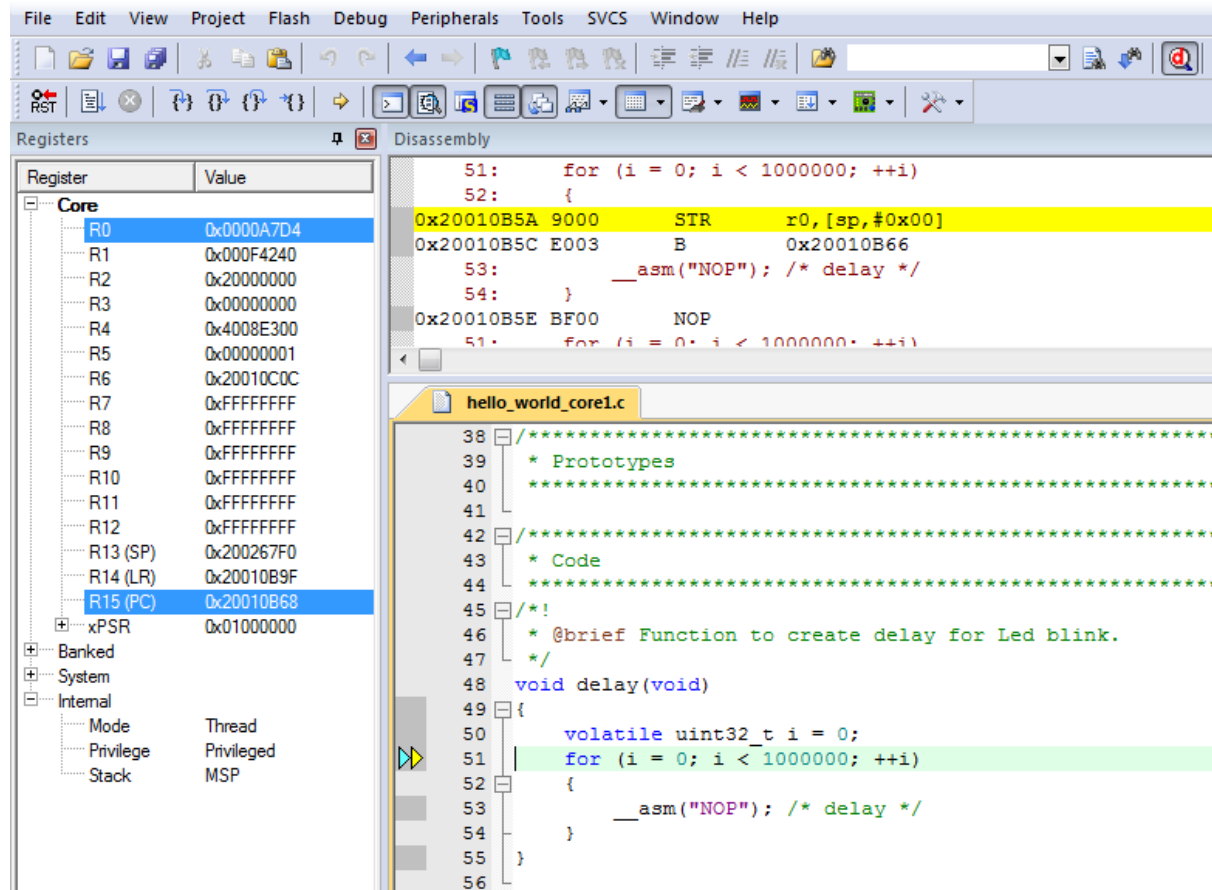
**Run a multicore example application** The primary core debugger flashes both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 5 as described in **Run an example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

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
```

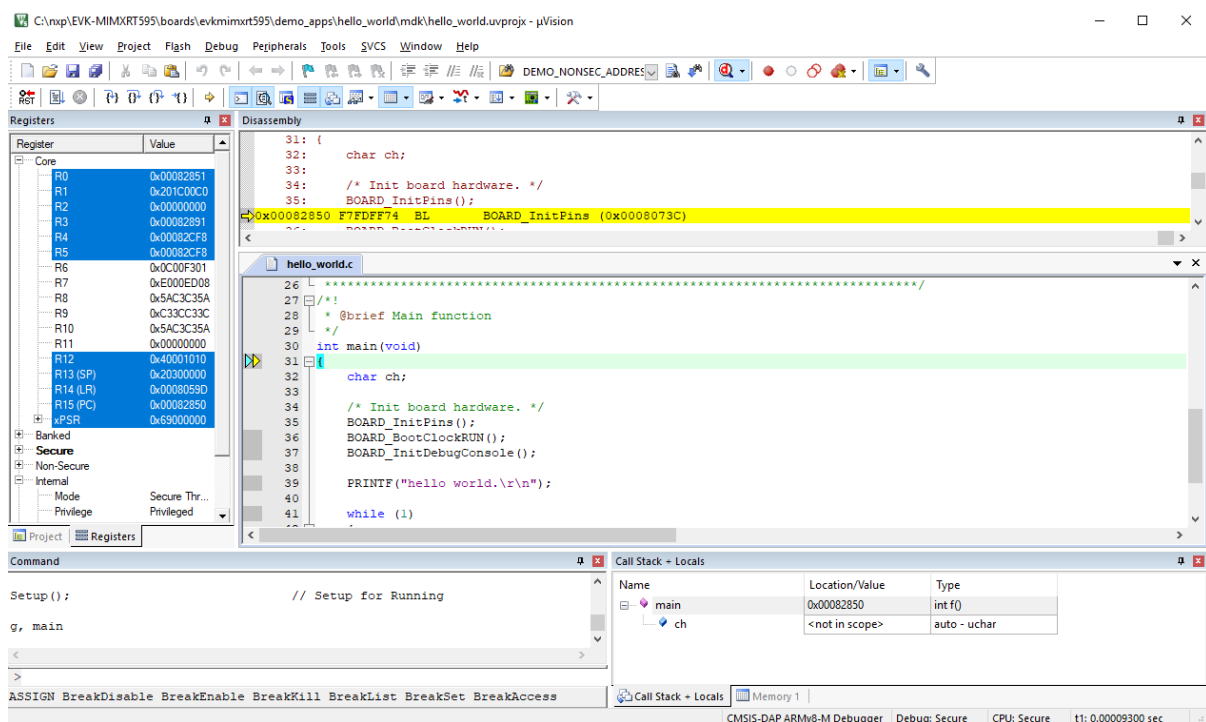
```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_s/mdk/hello_world.  
↪ uvmpw
```

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.

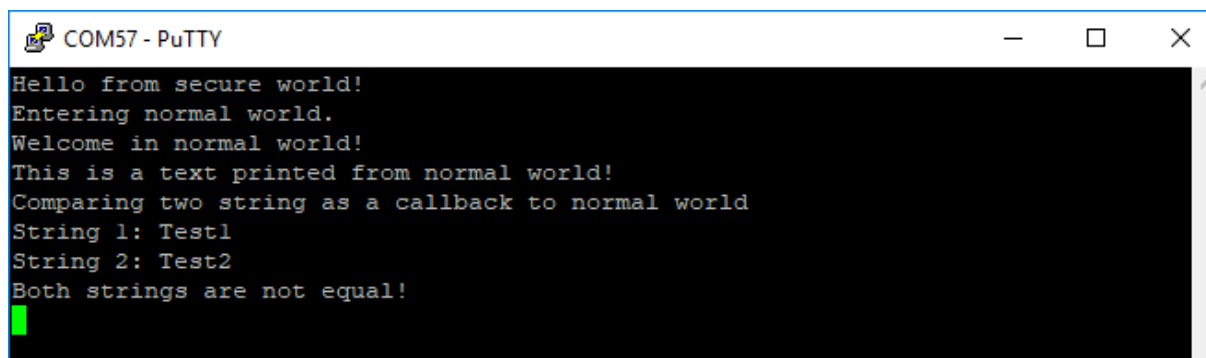
**Run a TrustZone example application** The secure project is configured to download both secure and non-secure output files so debugging can be fully managed from the secure project.

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.

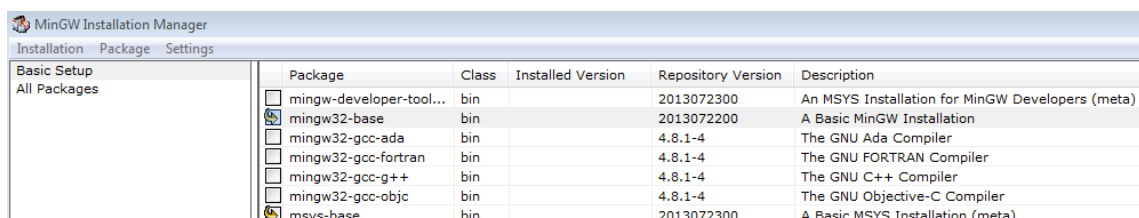
**Install GCC Arm Embedded tool chain** Download and run the installer from GNU Arm Embedded Toolchain. This is the actual toolset (in other words, compiler, linker, and so on). The GCC toolchain should correspond to the latest supported version, as described in **MCUXpresso SDK Release Notes**.

**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.

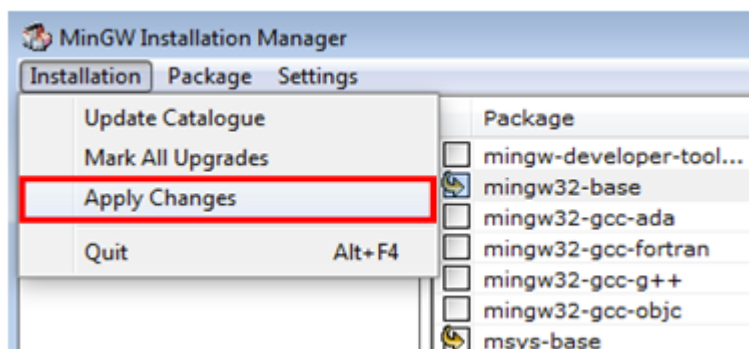
1. Download the latest MinGW mingw-get-setup installer from [MinGW](#).
2. Run the installer. The recommended installation path is `C:\MinGW`, however, you may install to any location.

**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.

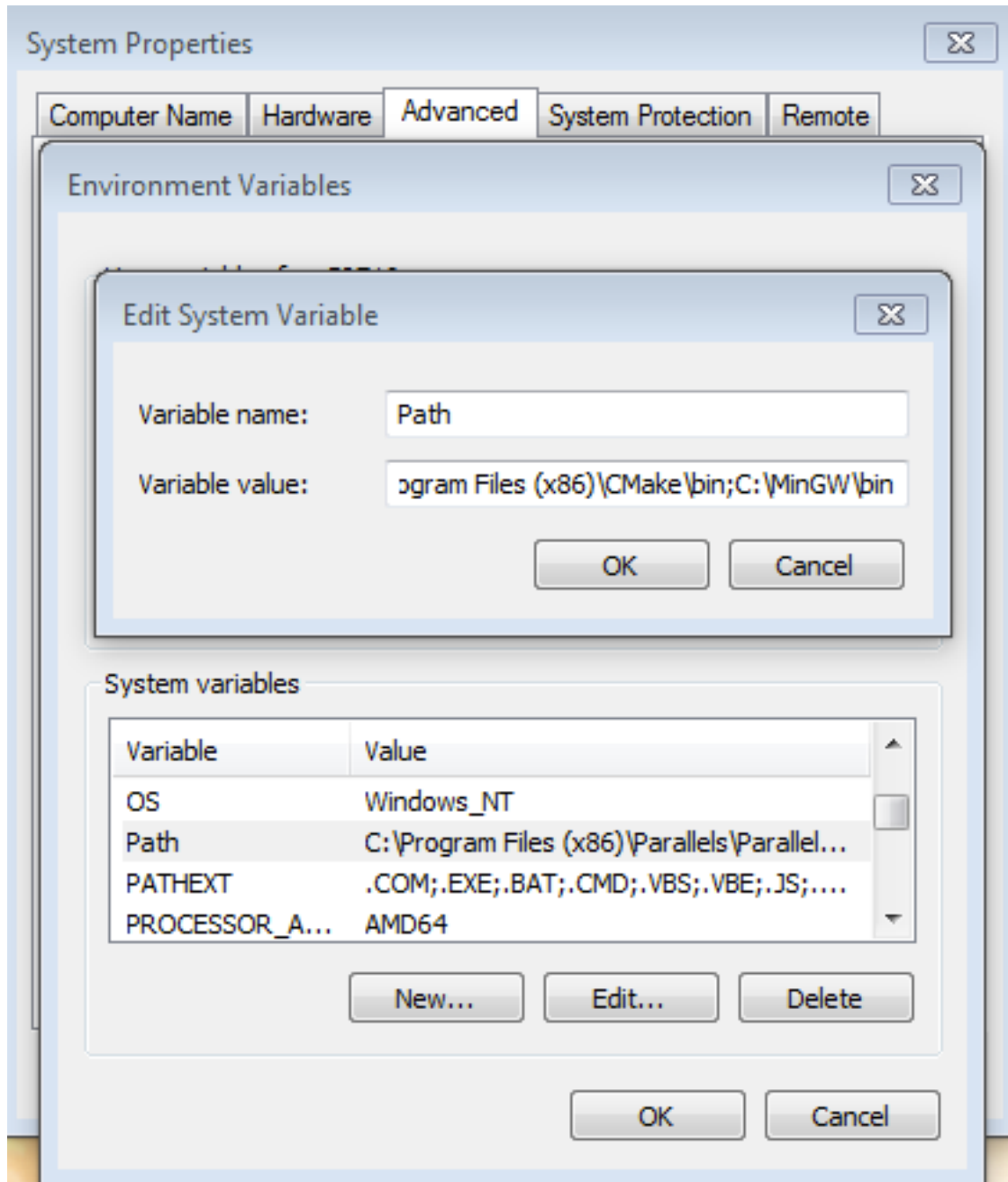


5. Add the appropriate item to the Windows operating system path environment variable. It can be found under **Control Panel->System and Security->System->Advanced System Settings** in the **Environment Variables...** section. The path is:

```
<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.



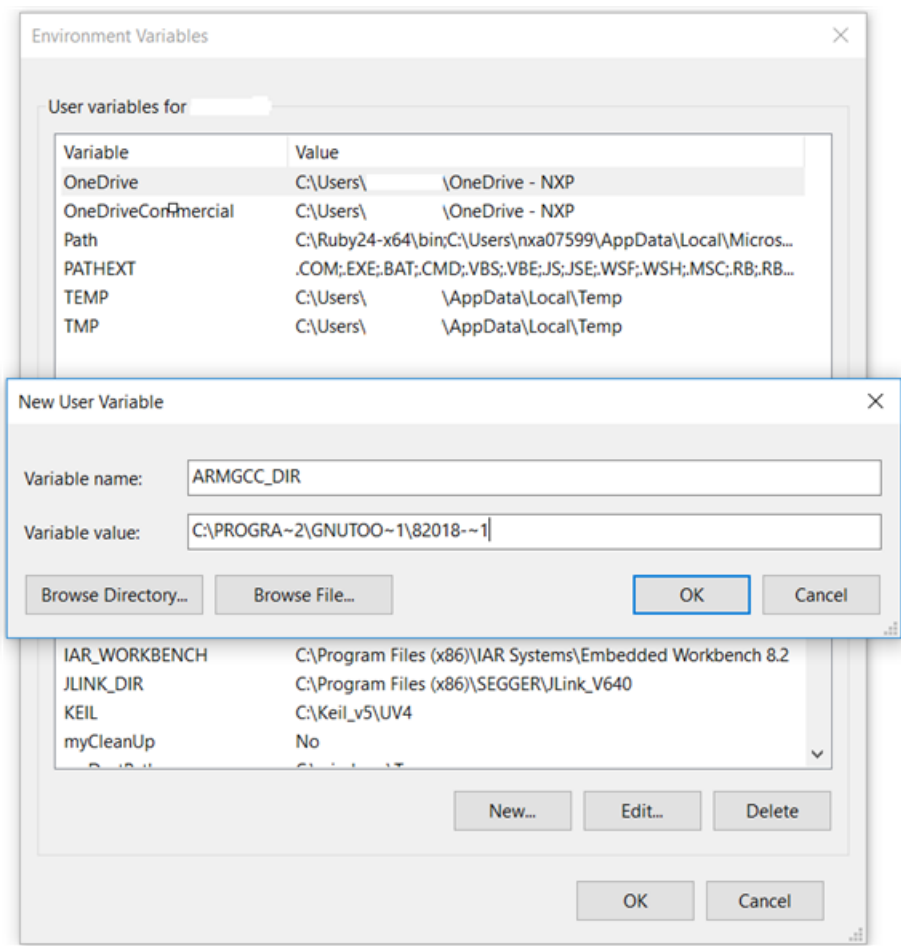
**Add a new system environment variable for ARMGCC\_DIR** Create a new *system* environment variable and name it as ARMGCC\_DIR. The value of this variable should point to the Arm GCC Embedded tool chain installation path. For this example, the path is:

```
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

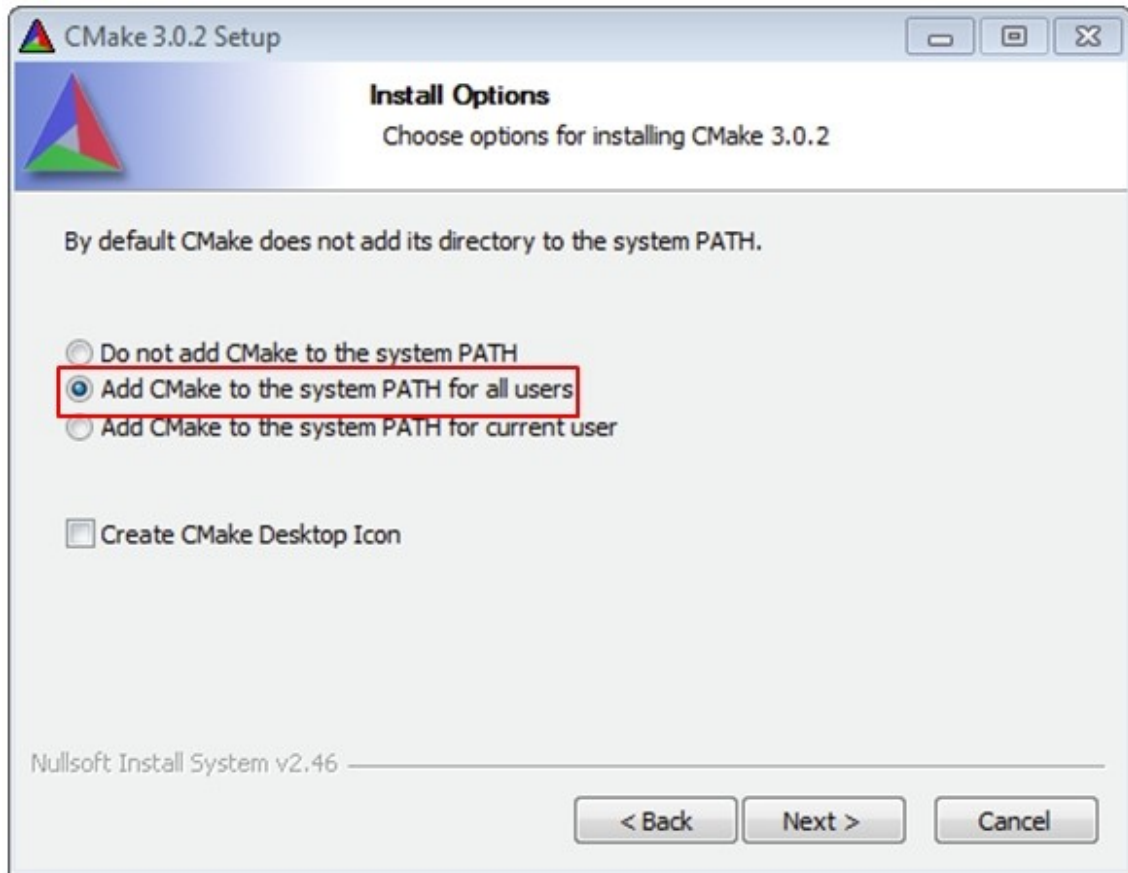
```
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>for %I in (.) do echo %~sI
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>echo C:\PROGRA~2\GNUTOO~1\82018~1
C:\PROGRA~2\GNUTOO~1\82018~1
```



## 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`.

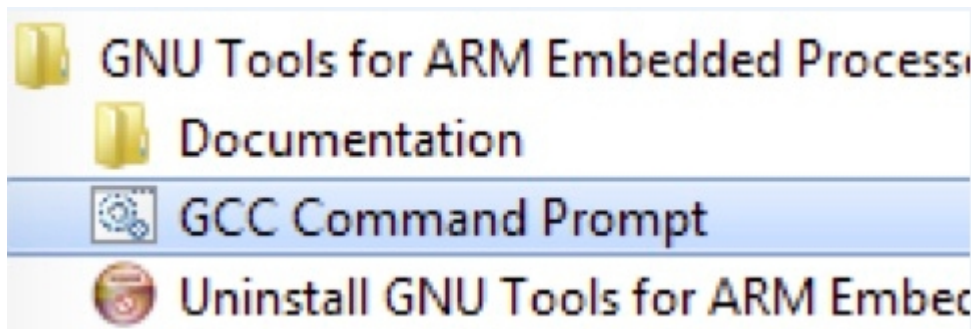
**Linux OS** It depends on the distributions of Linux Operation System. Here we use Ubuntu as an example.

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.

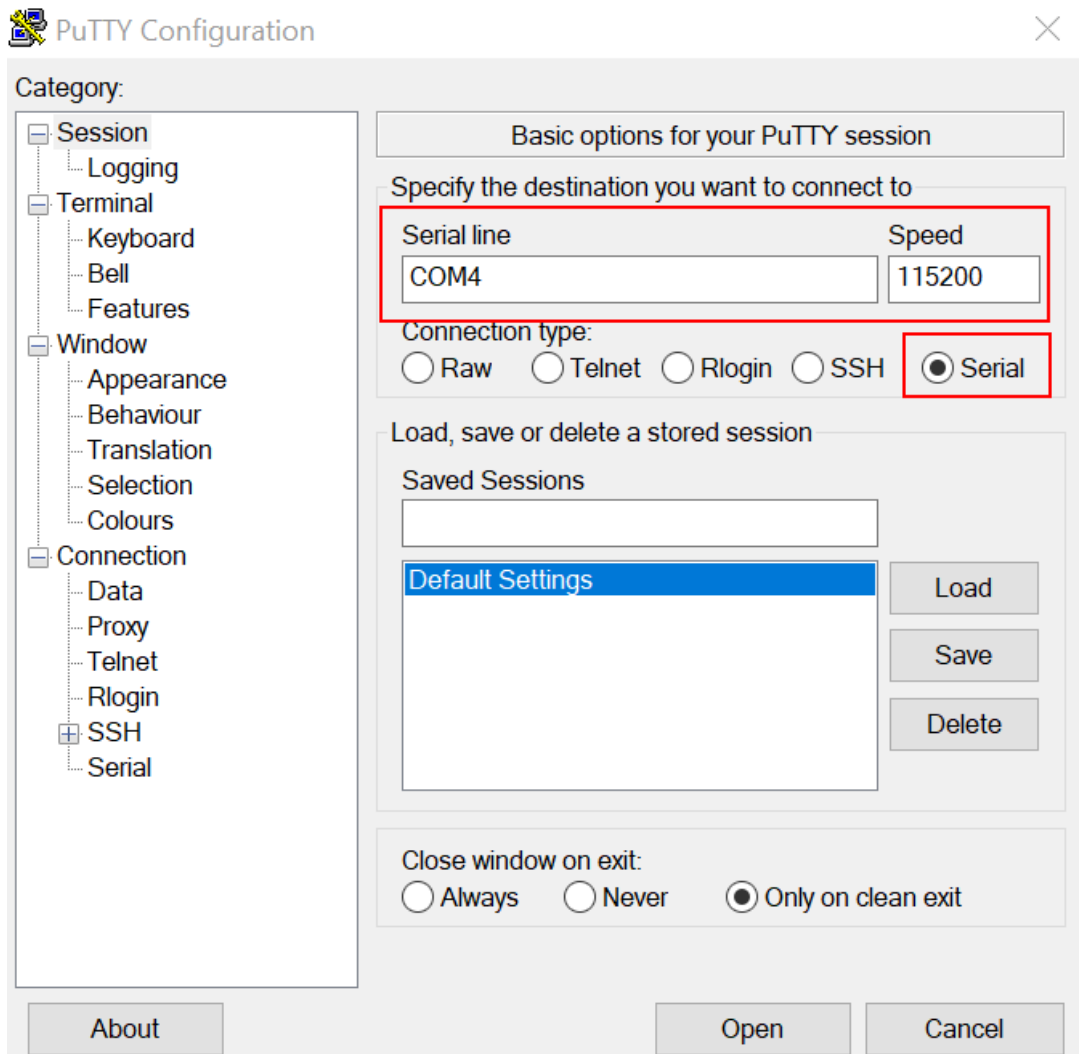
```
[ 84%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.0_FRDM-K64F
/devices/MK64F12/drivers/fsl_smc.c.obj
[ 92%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.0_FRDM-K64F
/devices/MK64F12/drivers/fsl_clock.c.obj
[100%] Linking C executable debug\hello_world.elf
[100%]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

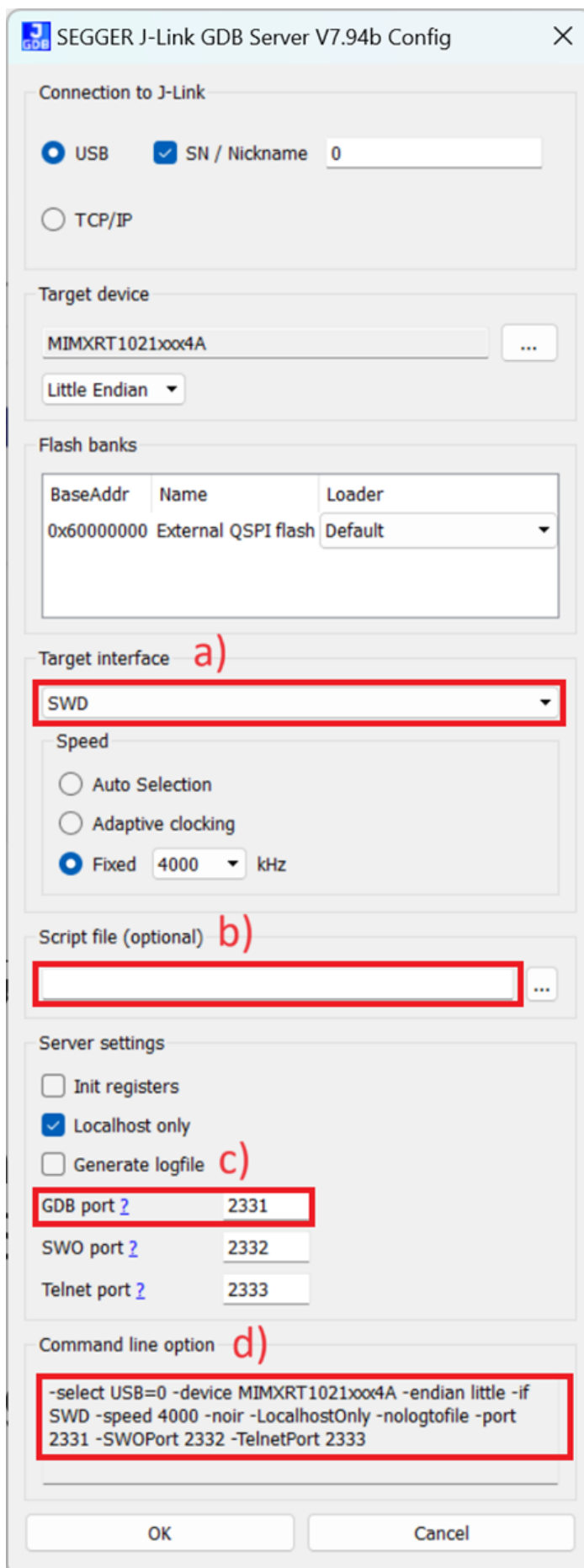


3. To launch the application, open the Windows **Start** menu and select **Programs > SEGGER > J-Link <version> J-Link GDB Server**.

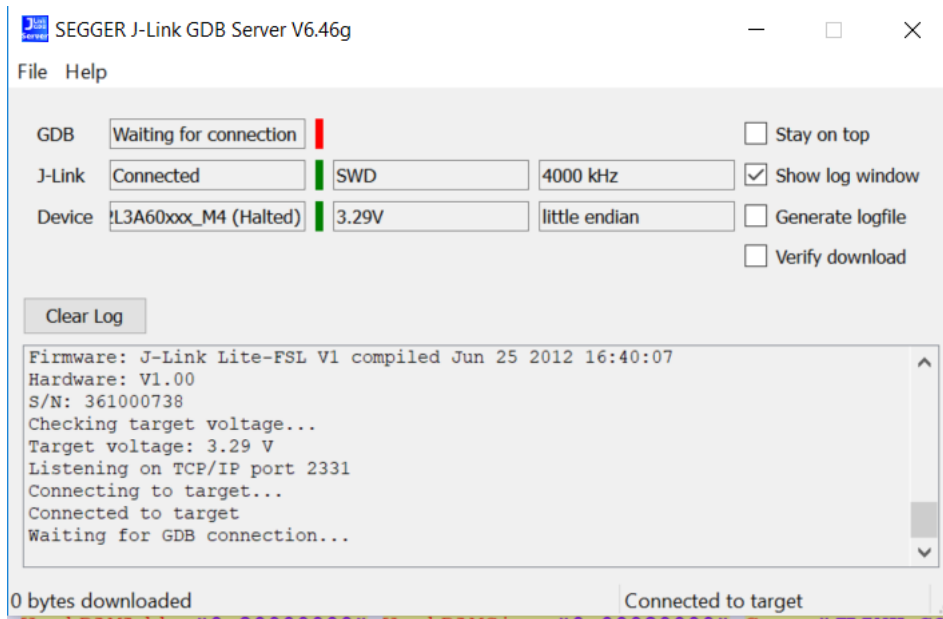
**Note:** It is assumed that the J-Link software is already installed.

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

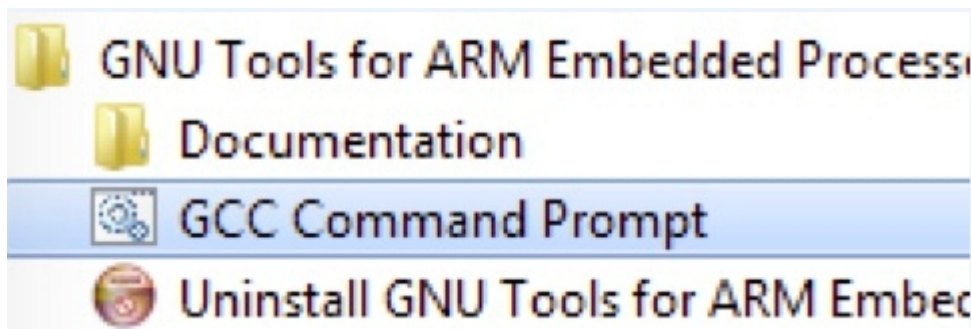
4. Make sure to check the following options.
  1. **Target interface:** The debug connection on board uses internal SWD signaling. In case of a wrong setting J-Link is unable to communicate with device under test.
  2. **Script file:** If required, a J-Link init script file can be used for board initialization.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

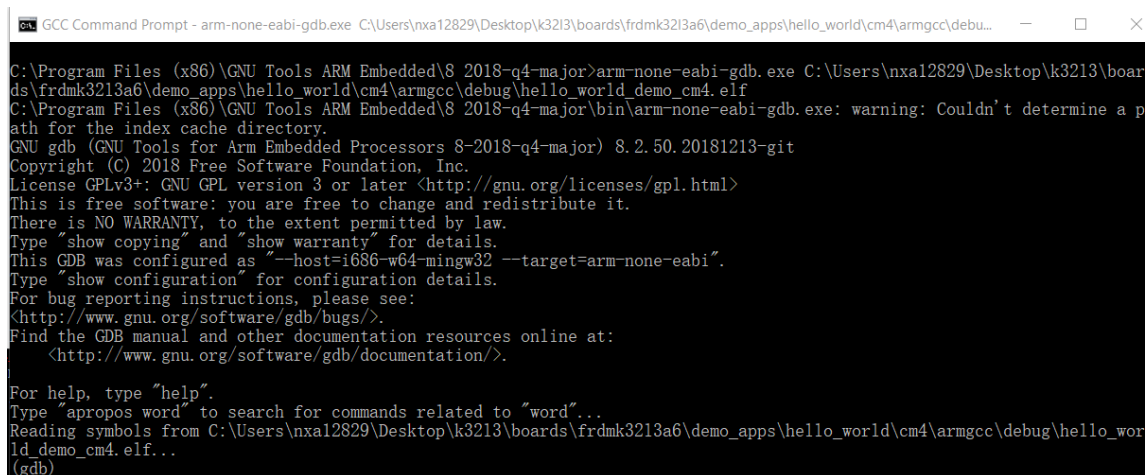


7. Change to the directory that contains the example application output. The output can be found in using one of these paths, depending on the build target selected:

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

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

```

### 9. Run these commands:

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

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

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



```

COM4 - PuTTY
hello world.

```

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

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

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

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

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

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

```

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

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

```

```

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

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

```

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

The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 to 10, as described in **Run an example application**. These steps are common for both single-core and dual-core applications in Arm GCC.

Both the primary and the auxiliary image is loaded into the SPI flash memory. After execution of the monitor go command, the primary core application is executed. During the primary core code execution, the auxiliary core code is reallocated from the flash memory to the RAM, and the auxiliary core is released from the reset. The hello\_world multicore application is now running

and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.

```

Administrator: GCC Command Prompt
c:\D\SDK_2.0_LPCXpresso54114\boards\lpcxpresso54114\multicore_examples\hello_world\cm4\armgcc>IF "" == "" <pause >
Press any key to continue . . .

c:\D\SDK_2.0_LPCXpresso54114\boards\lpcxpresso54114\multicore_examples\hello_world\cm4\armgcc>cd debug

c:\D\SDK_2.0_LPCXpresso54114\boards\lpcxpresso54114\multicore_examples\hello_world\cm4\armgcc\debug>arm-none-eabi-gdb.exe hello_world_cm4.elf
GNU gdb (GNU Tools for ARM Embedded Processors 6-2017-q2-update) 7.12.1.20170417-g
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_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/serial_manager.c.obj
[ 74%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/components/serial_manager/serial_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/startup_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_uart.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 "" == "" (pause)
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.

```

Command Prompt - arm-none-eabi-gdb
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.

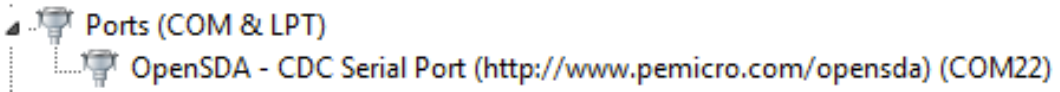
2. **Windows:** To determine the COM port open Device Manager in the Windows operating system. Click the **Start** menu and type **Device Manager** in the search bar.

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.

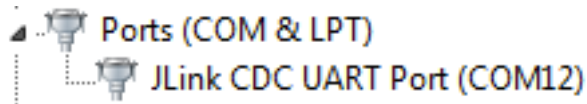
1. **CMSIS-DAP/mbed/DAPLink** interface:



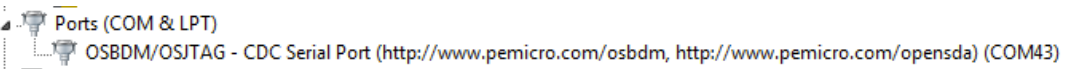
## 2. P&E Micro:



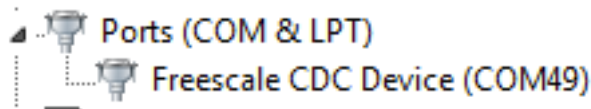
## 3. J-Link:



## 4. P&E Micro OSJTAG:



## 5. MRB-KW01:



## On-board Debugger

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

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

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

**The corresponding host driver must be installed before debugging.**

- For boards with CMSIS-DAP firmware, visit [developer.mbed.org/handbook/Windows-serial-configuration](http://developer.mbed.org/handbook/Windows-serial-configuration) and follow the instructions to install the Windows operating system serial driver. If running on Linux OS, this step is not required.
- If using J-Link with either a standalone debug pod or MCU-Link, install the J-Link software (drivers and utilities) from [www.segger.com/jlink-software.html](http://www.segger.com/jlink-software.html).

**Updating MCU-Link firmware** This firmware in this debug interface may be updated using the host computer utility called MCU-Link. This typically used when switching between the default debugger protocol (CMSIS-DAP) to SEGGER J-Link, or for updating this firmware with new releases of these. This section contains the steps to reprogram the debug probe firmware.

**Note:** If MCUXpresso IDE is used and the jumper making DFUlink is installed on the board (JP5 on some boards, but consult the board user manual or schematic for specific jumper number), MCU-Link debug probe boots to DFU mode, and MCUXpresso IDE automatically downloads the

CMSIS-DAP firmware to the probe before flash memory programming (after clicking **Debug**). Using DFU mode ensures that most up-to-date/compatible firmware is used with MCUXpresso IDE.

NXP provides the MCU-Link utility, which is the recommended tool for programming the latest versions of CMSIS-DAP and J-Link firmware onto MCU-Link or NXP boards. The utility can be downloaded from [MCU-Link](#).

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

1. Install the MCU-Link utility.
2. Unplug the board's USB cable.
3. Make the DFU link (install the jumper labeled DFUlink).
4. Connect the probe to the host via USB (use Link USB connector).
5. Open a command shell and call the appropriate script located in the MCU-Link installation directory (<MCU-Link install dir>).
  1. To program CMSIS-DAP debug firmware: <MCU-Link install dir>/scripts/program\_CMSIS
  2. To program J-Link debug firmware: <MCU-Link install dir>/scripts/program\_JLINK
6. Remove DFU link (remove the jumper installed in Step 3).
7. Repower the board by removing the USB cable and plugging it in again.

**On-board debugger LPC-Link** LPC-Link 2 is an extensible debug probe that can be used seamlessly with MCUXpresso IDE, and is also compatible with 3rd party IDEs that support CMSIS-DAP protocol. MCU-Link also includes a USB to UART bridge feature (VCOM) that can be used to provide a serial connection between the target MCU and a host computer. LPC-Link 2 is compatible with Windows, MacOS and Linux. A free utility from NXP provides an easy way to install firmware updates.

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

**The corresponding host driver must be installed before debugging.**

- For boards with CMSIS-DAP firmware, visit [developer.mbed.org/handbook/Windows-serial-configuration](https://developer.mbed.org/handbook/Windows-serial-configuration) and follow the instructions to install the Windows operating system serial driver. If running on Linux OS, this step is not required.
- If using J-Link with either a standalone debug pod or MCU-Link, install the J-Link software (drivers and utilities) from [www.segger.com/jlink-software.html](http://www.segger.com/jlink-software.html).

**Updating LPC-Link firmware** The LPCXpresso hardware platform comes with a CMSIS-DAP-compatible debug interface (known as LPC-Link2). This firmware in this debug interface may be updated using the host computer utility called LPCScript. This typically used when switching between the default debugger protocol (CMSIS-DAP) to SEGGER J-Link, or for updating this firmware with new releases of these. This section contains the steps to reprogram the debug probe firmware.

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

NXP provides the LPCScript utility, which is the recommended tool for programming the latest versions of CMSIS-DAP and J-Link firmware onto LPC-Link2 or LPCXpresso boards. The utility can be downloaded from [LPCScript](#).

These steps show how to update the debugger firmware on your board for Windows operating system. For Linux OS, follow the instructions described in LPCScript user guide ([LPCScript](#), select **LPCScript**, and then the documentation tab).

1. Install the LPCScript utility.
2. Unplug the board's USB cable.
3. Make the DFU link (install the jumper labeled DFUlink).
4. Connect the probe to the host via USB (use Link USB connector).
5. Open a command shell and call the appropriate script located in the LPCScript installation directory (<LPCScript install dir>).
  1. To program CMSIS-DAP debug firmware: <LPCScript install dir>/scripts/program\_CMSIS
  2. To program J-Link debug firmware: <LPCScript install dir>/scripts/program\_JLINK
6. Remove DFU link (remove the jumper installed in Step 3).
7. Repower the board by removing the USB cable and plugging it in again.

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

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

See the table in [Default debug interfaces](#) to determine the default debug interface that comes loaded on your specific hardware platform.

**The corresponding host driver must be installed before debugging.**

- For boards with CMSIS-DAP firmware, visit [developer.mbed.org/handbook/Windows-serial-configuration](https://developer.mbed.org/handbook/Windows-serial-configuration) and follow the instructions to install the Windows operating system serial driver. If running on Linux OS, this step is not required.
- For boards with a P&E Micro interface, see [PE micro](#) to download and install the P&E Micro Hardware Interface Drivers package.

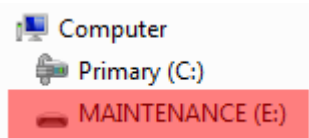
**Updating OpenSDA firmware** Any NXP hardware platform that comes with an OpenSDA-compatible debug interface has the ability to update the OpenSDA firmware. This typically means to switch from the default application (either CMSIS-DAP or P&E Micro) to a SEGGER J-Link. This section contains the steps to switch the OpenSDA firmware to a J-Link interface. However, the steps can be applied to restoring the original image also. For reference, OpenSDA firmware files can be found at the links below:

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

- P&E Micro: Downloading P&E Micro OpenSDA firmware images requires registration with P&E Micro ([www.pemicro.com](http://www.pemicro.com)).

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

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



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

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

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

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

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

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

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

**The host driver must be installed before debugging.**

- See [PE micro](#) to download and install the P&E Micro Hardware Interface Drivers package.

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

**The host driver must be installed before debugging.**

- See [PE micro](#) to download and install the P&E Micro Hardware Interface Drivers package.

## Default debug interfaces

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

| Hardware platform | Default debugger firmware | On-board debugger probe |
|-------------------|---------------------------|-------------------------|
| EVK-MCIMX7ULP     | N/A                       | N/A                     |
| EVK-MIMX8MM       | N/A                       | N/A                     |
| EVK-MIMX8MN       | N/A                       | N/A                     |
| EVK-MIMX8MND3L    | N/A                       | N/A                     |
| EVK-MIMX8MP       | N/A                       | N/A                     |
| EVK-MIMX8MQ       | N/A                       | N/A                     |
| EVK-MIMX8ULP      | N/A                       | N/A                     |
| EVK-MIMXRT1010    | CMSIS-DAP                 | LPC-Link2               |
| EVK-MIMXRT1015    | CMSIS-DAP                 | LPC-Link2               |
| EVK-MIMXRT1020    | CMSIS-DAP                 | LPC-Link2               |
| EVK-MIMXRT1064    | CMSIS-DAP                 | LPC-Link2               |
| EVK-MIMXRT595     | CMSIS-DAP                 | LPC-Link2               |
| EVK-MIMXRT685     | CMSIS-DAP                 | LPC-Link2               |
| EVK9-MIMX8ULP     | N/A                       | N/A                     |
| EVKB-IMXRT1050    | CMSIS-DAP                 | LPC-Link2               |
| FRDM-K22F         | CMSIS-DAP                 | OpenSDA v2              |
| FRDM-K32L2A4S     | CMSIS-DAP                 | OpenSDA v2              |
| FRDM-K32L2B       | CMSIS-DAP                 | OpenSDA v2              |
| FRDM-K32L3A6      | CMSIS-DAP                 | OpenSDA v2              |
| FRDM-KE02Z40M     | P&E Micro                 | OpenSDA v1              |
| FRDM-KE15Z        | CMSIS-DAP                 | OpenSDA v2              |
| FRDM-KE16Z        | CMSIS-DAP                 | OpenSDA v2              |
| FRDM-KE17Z        | CMSIS-DAP                 | OpenSDA v2              |
| FRDM-KE17Z512     | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXA153      | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXA156      | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXA266      | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXA344      | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXA346      | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXA366      | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXC041      | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXC242      | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXC444      | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXE247      | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXE31B      | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXN236      | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXN947      | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXW23       | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXW71       | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXW72       | CMSIS-DAP                 | MCU-Link                |
| FRDM-RW612        | CMSIS-DAP                 | MCU-Link                |
| IMX943-EVK        | N/A                       | N/A                     |
| IMX95LP4XEVK-15   | N/A                       | N/A                     |
| IMX95LPD5EVK-19   | N/A                       | N/A                     |
| IMX95VERDINEVK    | N/A                       | N/A                     |
| KW45B41Z-EVK      | CMSIS-DAP                 | MCU-Link                |
| KW45B41Z-LOC      | CMSIS-DAP                 | MCU-Link                |
| KW47-EVK          | CMSIS-DAP                 | MCU-Link                |

continues on next page

Table 1 – continued from previous page

| Hardware platform | Default debugger firmware | On-board debugger probe |
|-------------------|---------------------------|-------------------------|
| KW47-LOC          | CMSIS-DAP                 | MCU-Link                |
| LPC845BREAKOUT    | CMSIS-DAP                 | LPC-Link2               |
| LPCXpresso51U68   | CMSIS-DAP                 | LPC-Link2               |
| LPCXpresso54628   | CMSIS-DAP                 | LPC-Link2               |
| LPCXpresso54S018  | CMSIS-DAP                 | LPC-Link2               |
| LPCXpresso54S018M | CMSIS-DAP                 | LPC-Link2               |
| LPCXpresso55S06   | CMSIS-DAP                 | LPC-Link2               |
| LPCXpresso55S16   | CMSIS-DAP                 | LPC-Link2               |
| LPCXpresso55S28   | CMSIS-DAP                 | LPC-Link2               |
| LPCXpresso55S36   | CMSIS-DAP                 | MCU-Link                |
| LPCXpresso55S69   | CMSIS-DAP                 | LPC-Link2               |
| LPCXpresso802     | CMSIS-DAP                 | LPC-Link2               |
| LPCXpresso804     | CMSIS-DAP                 | LPC-Link2               |
| LPCXpresso824MAX  | CMSIS-DAP                 | LPC-Link2               |
| LPCXpresso845MAX  | CMSIS-DAP                 | LPC-Link2               |
| LPCXpresso860MAX  | CMSIS-DAP                 | LPC-Link2               |
| MC56F80000-EVK    | P&E Micro                 | Multilink               |
| MC56F81000-EVK    | P&E Micro                 | Multilink               |
| MC56F83000-EVK    | P&E Micro                 | OSJTAG                  |
| MCIMX93-EVK       | N/A                       | N/A                     |
| MCIMX93-QSB       | N/A                       | N/A                     |
| MCIMX93AUTO-EVK   | N/A                       | N/A                     |
| MCX-N5XX-EVK      | CMSIS-DAP                 | MCU-Link                |
| MCX-N9XX-EVK      | CMSIS-DAP                 | MCU-Link                |
| MCX-W71-EVK       | CMSIS-DAP                 | MCU-Link                |
| MCX-W72-EVK       | CMSIS-DAP                 | MCU-Link                |
| MIMXRT1024-EVK    | CMSIS-DAP                 | LPC-Link2               |
| MIMXRT1040-EVK    | CMSIS-DAP                 | LPC-Link2               |
| MIMXRT1060-EVKB   | CMSIS-DAP                 | LPC-Link2               |
| MIMXRT1060-EVKC   | CMSIS-DAP                 | MCU-Link                |
| MIMXRT1160-EVK    | CMSIS-DAP                 | LPC-Link2               |
| MIMXRT1170-EVKB   | CMSIS-DAP                 | MCU-Link                |
| MIMXRT1180-EVK    | CMSIS-DAP                 | MCU-Link                |
| MIMXRT685-AUD-EVK | CMSIS-DAP                 | LPC-Link2               |
| MIMXRT700-EVK     | CMSIS-DAP                 | MCU-Link                |
| RD-RW612-BGA      | CMSIS-DAP                 | MCU-Link                |
| TWR-KM34Z50MV3    | P&E Micro                 | OpenSDA v1              |
| TWR-KM34Z75M      | P&E Micro                 | OpenSDA v1              |
| TWR-KM35Z75M      | CMSIS-DAP                 | OpenSDA v2              |
| TWR-MC56F8200     | P&E Micro                 | OSJTAG                  |
| TWR-MC56F8400     | P&E Micro                 | OSJTAG                  |

### How to define IRQ handler in CPP files

With MCUXpresso SDK, users could define their own IRQ handler in application level to override the default IRQ handler. For example, to override the default PIT\_IRQHandler define in startup\_DEVICE.s, application code like app.c can be implement like:

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

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

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

## 1.3 Getting Started with MCUXpresso SDK GitHub

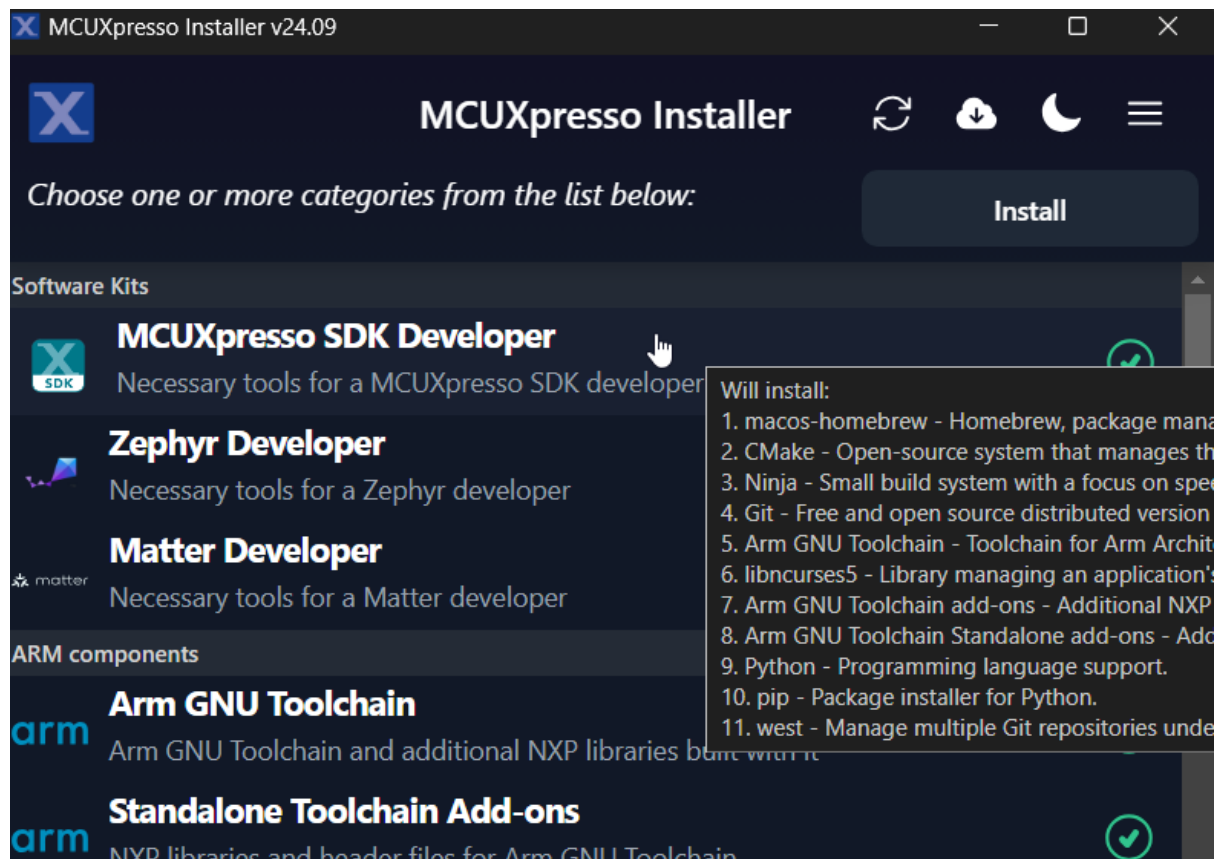
### 1.3.1 Getting Started with MCUXpresso SDK Repository

#### Installation

#### NOTE

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

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



## Alternative: Manual Installation

### Basic tools

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

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

Then configure your username and email using the commands:

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

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

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

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

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

## Build And Configuration System

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

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

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

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

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

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

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

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

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

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

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

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

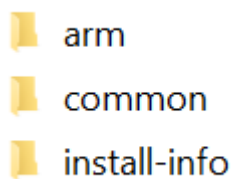
**Toolchain** MCUXpresso SDK supports all mainstream toolchains for embedded development. You can install your used or interested toolchains following the guides.

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

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

| Toolchain                    | Environment Variable | Example  | Cmd Line Argument        |
|------------------------------|----------------------|--|--------------------------|
| Armgcc                       | ARM-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_SE            | c:\NXP\zephyr-sdk-<version> for windows/opt/zephyr-sdk-<version> for Linux   | - toolchain zephyr       |
| CodeWarrior                  | CW_DIR               | C:\Freescale\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 | RISCVL-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 riscvl-llvm  |

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



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

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

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

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

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

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

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

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

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

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

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

## Get MCUXpresso SDK Repo

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

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

# Update the west projects
cd mcuxpresso-sdk
west update

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

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

1. Navigate to the core directory:

```
cd mcuxsdk
```

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

```
python -m venv .venv

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

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

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

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

3. Install the required Python packages:

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

## Explore Contents

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

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

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

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

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

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

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

- `demo_apps`: Basic demo set to start using SDK, including `hello_world` and `led_blinky`.
- `driver_examples`: Simple applications that show how to use the peripheral drivers for a single use case. These applications typically only use a single peripheral but there are cases where multiple peripherals are used (for example, SPI transfer using DMA).

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

### Run a demo using MCUXpresso for VS Code

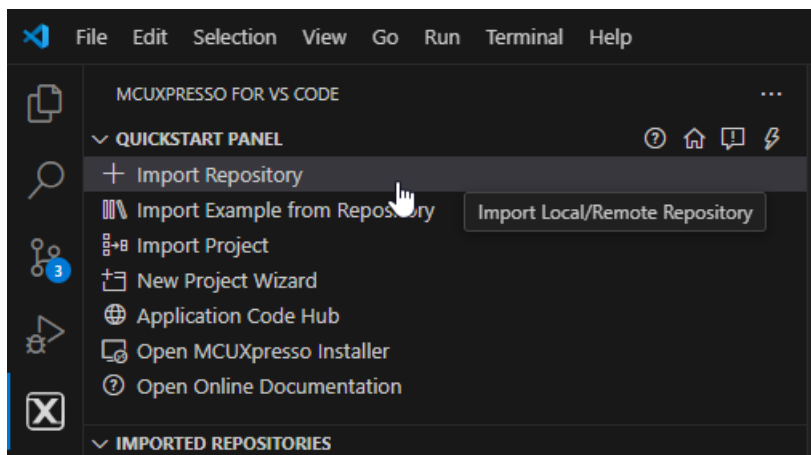
This section explains how to configure MCUXpresso for VS Code to build, run, and debug example applications. This guide uses the `hello_world` demo application as an example. However, these

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

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

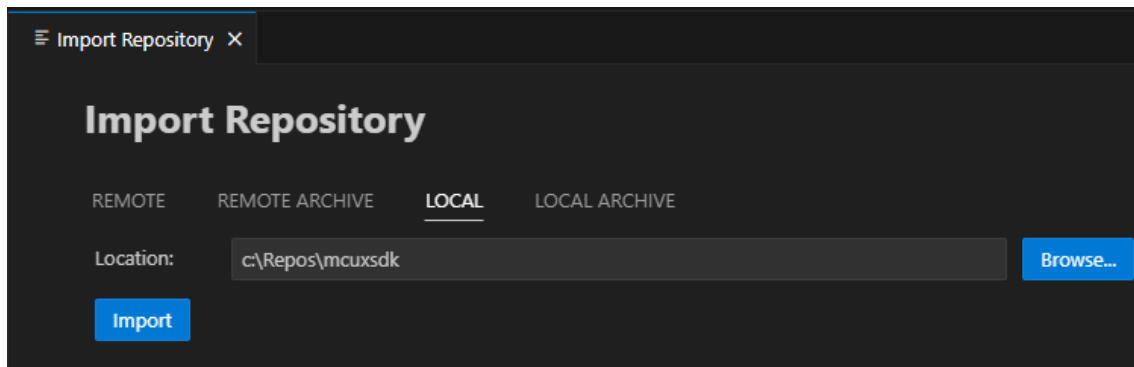
To build an example application:

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

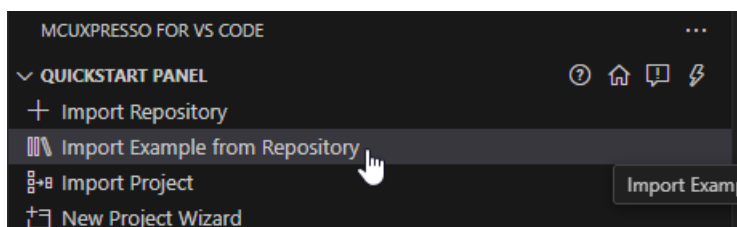


**Note:** You can import the SDK in several ways. Refer to [MCUXpresso for VS Code Wiki](#) for details.

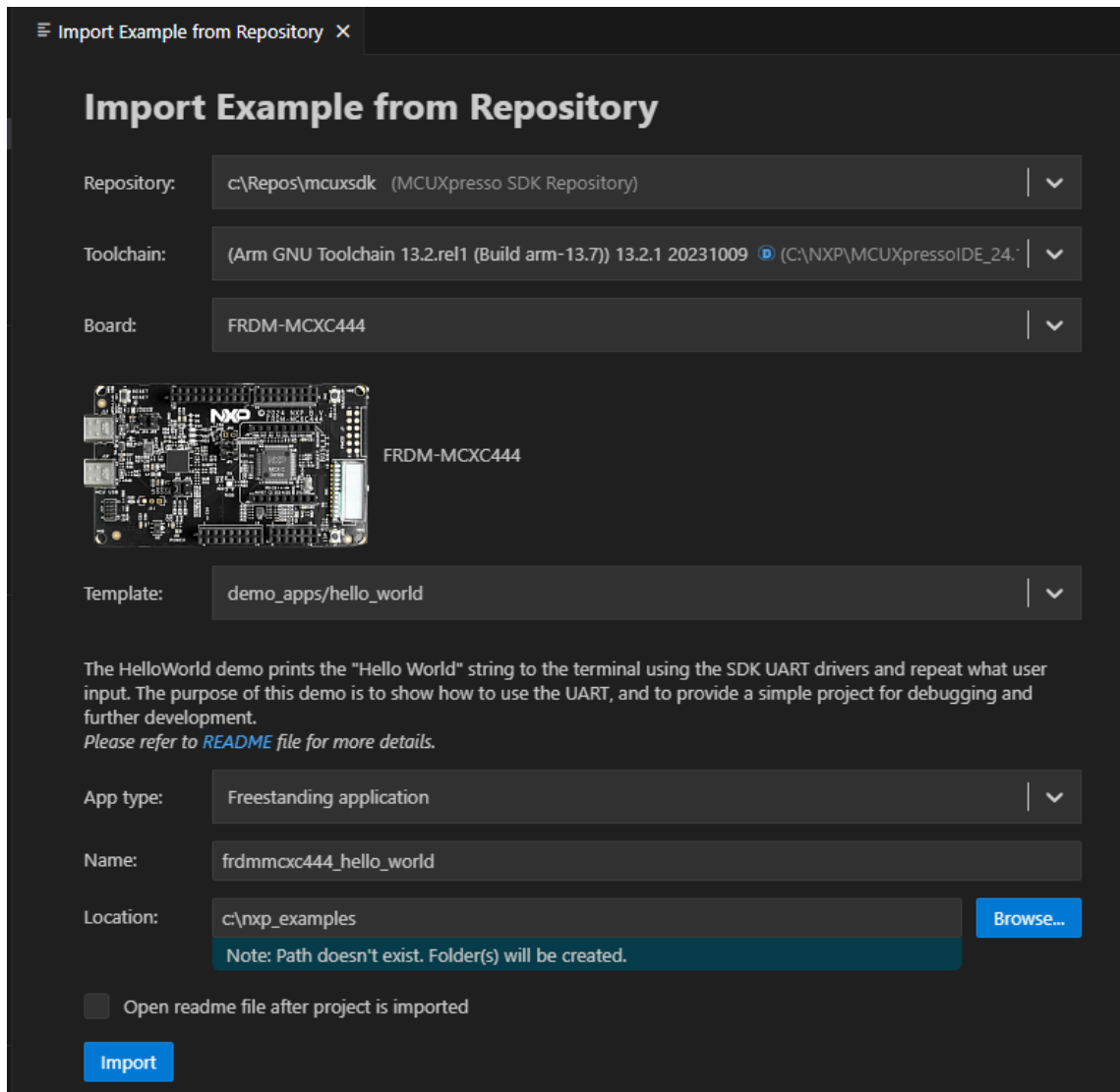
Select **Local** if you've already obtained the SDK as seen in [Get MCUXpresso SDK Repo](#). Select your location and click **Import**.



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

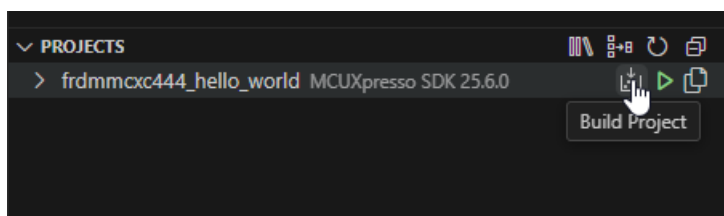


In the dropdown menu, select the MCUXpresso SDK, the Arm GNU Toolchain, your board, template, and application type. Click **Import**.



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

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



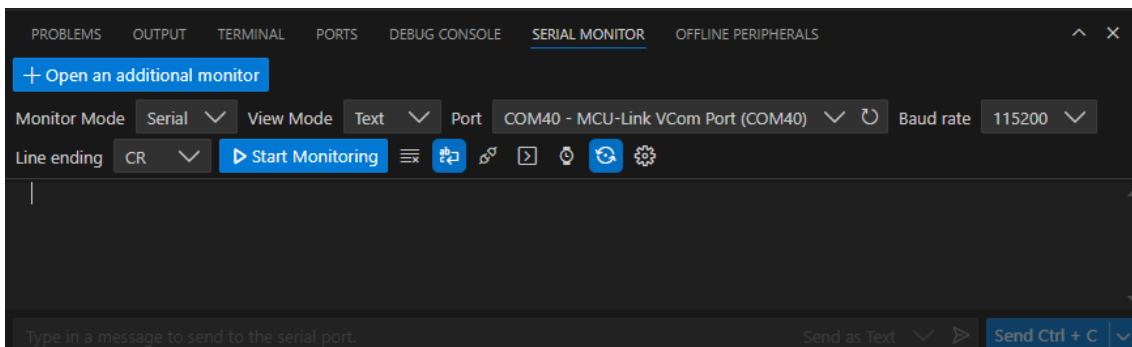
The integrated terminal will open at the bottom and will display the build output.

```

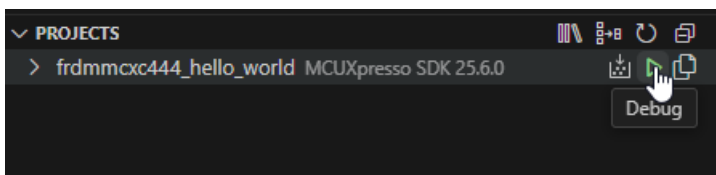
PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE SERIAL MONITOR OFFLINE PERIPHERALS
[17/21] Building C object C:\MakeFiles\hello_world.dir\C:/Repos/mcuxsdk/mcuxsdk/components/debug_console_lite/fs1_debug_console.c.obj
[18/21] Building C object C:\MakeFiles\hello_world.dir\C:/Repos/mcuxsdk/mcuxsdk/devices/MCX/MCX444/drivers/fs1_clock.c.obj
[19/21] Building C object C:\MakeFiles\hello_world.dir\C:/Repos/mcuxsdk/mcuxsdk/drivers/lpuart/fs1_lpuart.c.obj
[20/21] Building C object C:\MakeFiles\hello_world.dir\C:/Repos/mcuxsdk/mcuxsdk/drivers/uart/fs1_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
---- 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_
↪ evkbnimx7ulp]
...

```

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

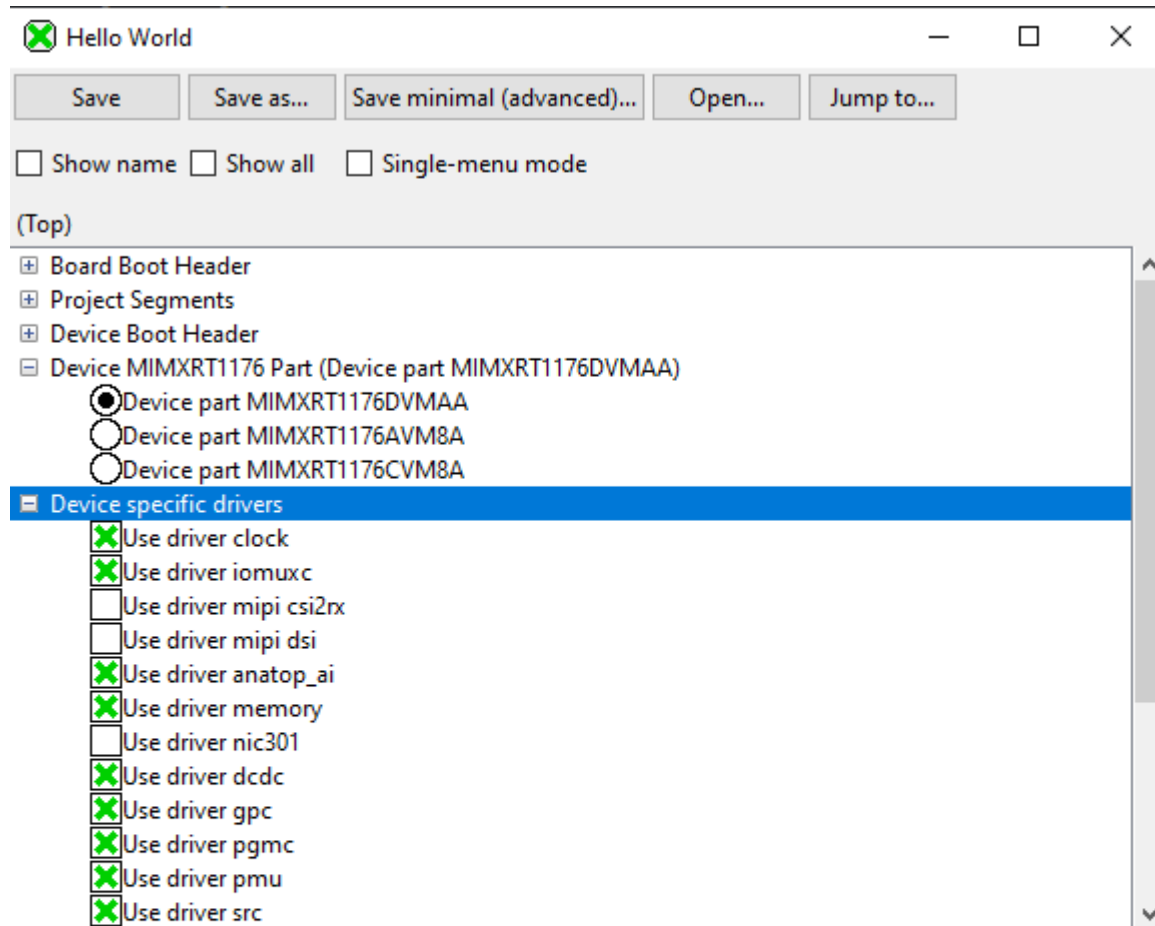
```
west build -b evkbnimxrt1170 examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

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



Kconfig definition, with parent deps. propagated to 'depends on'

```
=====  
At D:/sdk_next/mcuxsdk\devices\..\devices/RT/RT1170/MIMXRT1176\drivers/Kconfig: 5  
Included via D:/sdk_next/mcuxsdk/examples/demo_apps/hello_world/Kconfig: 6 ->  
D:/sdk_next/mcuxsdk/Kconfig.mcuxpresso: 9 -> D:/sdk_next/mcuxsdk\devices/Kconfig: 1  
-> D:/sdk_next/mcuxsdk\devices\..\devices/RT/RT1170/MIMXRT1176/Kconfig: 8  
Menu path: (Top)
```

```
menu "Device specific drivers"
```

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

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

**Flash** *Note:* Please refer Flash and Debug The Example to enable west flash/debug support.

Flash the hello\_world example:

```
west flash -r linkserver
```

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

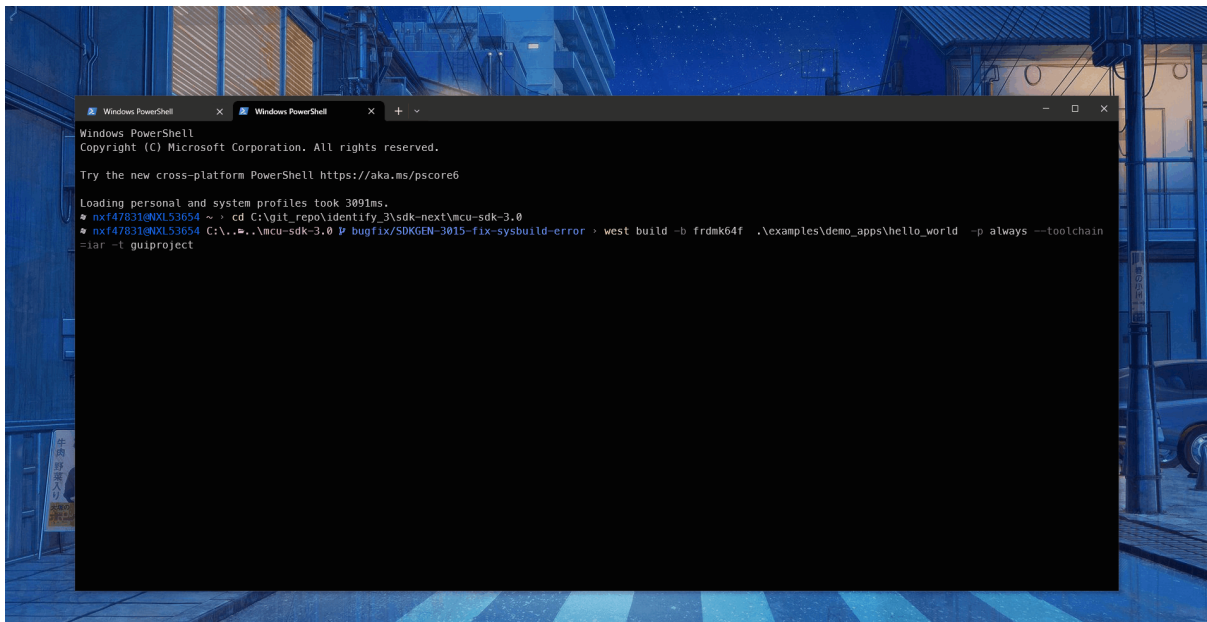
```
west debug -r linkserver
```

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

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

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

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



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

## 1.4 Release Notes

### 1.4.1 MCUXpresso SDK Release Notes

#### Overview

The MCUXpresso SDK is a comprehensive software enablement package designed to simplify and accelerate application development with Arm Cortex-M-based devices from NXP, including its general purpose, crossover and Bluetooth-enabled MCUs. MCUXpresso SW and Tools for DSC

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

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

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

## MCUXpresso SDK

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

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

## Development tools

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

- MCUXpresso IDE, Rev. 25.06.xx
- IAR Embedded Workbench for Arm, version is 9.60.4
- Keil MDK, version is 5.42
- MCUXpresso for VS Code v25.09
- GCC Arm Embedded Toolchain 14.2.x

## Supported development systems

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

| Development boards  | MCU devices                   |
|---------------------|-------------------------------|
| <b>FRDM-MCXC041</b> | MCXC041VFG, <b>MCXC041VFK</b> |

## 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.

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

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

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

**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.

### USBFS controller issue

Due to the USBFS controller design issues, the USB host suspend/resume demos (usb\_suspend\_resume\_host\_hid\_mouse) of the full speed controller do not support the low speed device directly.

### USB PID issue

Because the PID of all USB device examples is updated, uninstall the device drivers and then reinstall when the device (with new PID) is plugged in the first time

## 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
- 

**CLOCK**

**[2.0.0]**

- Initial version.
- 

**CMP**

**[2.0.3]**

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

**[2.0.2]**

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

**[2.0.1]**

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

**[2.0.0]**

- Initial version.
-

## COMMON

### [2.6.1]

- Improvements
  - Support Cortex M23.

### [2.6.0]

- Bug Fixes
  - Fix CERT-C violations.

### [2.5.0]

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

### [2.4.3]

- Improvements
  - Enable irq's that mount under `irqsteer` interrupt extender.

### [2.4.2]

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

### [2.4.1]

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

### [2.4.0]

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

### [2.3.3]

- New Features
  - Added `NETC` into status group.

### [2.3.2]

- Improvements
  - Make driver `aarch64` compatible

### [2.3.1]

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

### [2.3.0]

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

### [2.2.10]

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

### [2.2.9]

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

### [2.2.8]

- Improvements
  - Included 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.
- 

### COP

#### [2.0.2]

- Bug Fixes
  - Fixed CERT INT31-C violations.

#### [2.0.1]

- Bug Fixes
  - Fixed MISRA-2012 issues.
    - \* Rule 10.1 and rule 17.7.

#### [2.0.0]

- Initial version.
- 

### FLASH

#### [3.3.0]

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

### [3.2.0]

- New Feature
  - Basic support for FTFC

### [3.1.3]

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

### [3.1.2]

- Bug Fixes — Remove redundant comments.

### [3.1.1]

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

### [3.1.0]

- New Feature
  - Support erase flash asynchronously.

### [3.0.2]

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

### [3.0.1]

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

### [3.0.0]

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

### [2.3.1]

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

### [2.3.0]

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

### [2.2.0]

- New Features
  - Supported FTL 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
- 

## 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\_GetPinsDMAResetFlags 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”.
- 

## I2C

### [2.0.10]

- Bug Fixes
  - Fixed coverity issues.

### [2.0.9]

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

### [2.0.8]

- Bug Fixes
  - Fixed the bug that DFEN bit of I2C Status register 2 could not be set in `I2C_MasterInit`.
  - MISRA C-2012 issue fixed: rule 14.2, 15.7, and 16.4.
  - Eliminated IAR Pa082 warnings from `I2C_MasterTransferDMA` and `I2C_MasterTransferCallbackDMA` by assigning volatile variables to local variables and using local variables instead.
  - Fixed MISRA issues.
    - \* Fixed rules 10.1, 10.3, 10.4, 11.9, 14.4, 15.7, 17.7.
- Improvements
  - Improved timeout mechanism when waiting certain state in transfer API.

- Updated the I2C\_WAIT\_TIMEOUT macro to unified name I2C\_RETRY\_TIMES.
- Moved the master manually acknowledge byte operation into static function I2C\_MasterAckByte.
- Fixed control/status clean flow issue inside I2C\_MasterReadBlocking to avoid potential issue that pending status is cleaned before it's proceeded.

#### [2.0.7]

- Bug Fixes
  - Fixed the issue for MISRA-2012 check.
    - \* Fixed rule 11.9 ,15.7 ,14.4 ,10.4 ,10.8 ,10.3, 10.1, 10.6, 13.5, 11.3, 13.2, 17.7, 5.7, 8.3, 8.5, 11.1, 16.1.
  - Fixed Coverity issue of unchecked return value in I2C\_RTOS\_Transfer.
  - Fixed variable redefine issue by moving i2cBases from fsl\_i2c.h to fsl\_i2c.c.
- Improvements
  - Added I2C\_MASTER\_FACK\_CONTROL macro to enable FACK control for master transfer receive flow with IP supporting double buffer, then master could hold the SCL by manually setting TX AK/NAK during data transfer.

#### [2.0.6]

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

#### [2.0.5]

- Improvements
  - Added I2C\_WATI\_TIMEOUT macro to allow the user to specify the timeout times for waiting flags in functional API and blocking transfer API.

#### [2.0.4]

- Bug Fixes
  - Added a proper handle for transfer config flag kI2C\_TransferNoStartFlag to support transmit with kI2C\_TransferNoStartFlag flag. Support write only or write+read with no start flag; does not support read only with no start flag.

#### [2.0.3]

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

### [2.0.2]

- Bug Fixes
  - Fixed issue in master receive and slave transmit mode with no stop flag. The master could not succeed to start next transfer because the master could not send out re-start signal.
  - Fixed the out-of-order issue of data transfer due to memory barrier.
  - Added hold time configuration for slave. By leaving the SCL divider and MULT reset values when configured to slave mode, the setup and hold time of the slave is then reduced outside of spec for lower baudrates. This can cause intermittent arbitration loss on the master side.
- New Features
  - Added address nak event for master.
  - Added general call event for slave.

### [2.0.1]

- New Features
  - Added double buffer enable configuration for SoCs which have the DFEN bit in S2 register.
  - Added flexible transmit/receive buffer size support in I2C\_SlaveHandleIRQ.
  - Added start flag clear, address match, and release bus operation in I2C\_SlaveWrite/ReadBlocking API.
- Bug Fixes
  - Changed the kI2C\_SlaveRepeatedStartEvent to kI2C\_SlaveStartEvent.

### [2.0.0]

- Initial version.
- 

## LLWU

### [2.0.5]

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

### [2.0.4]

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

### [2.0.3]

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

### [2.0.2]

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

### [2.0.1]

- Other Changes
  - Updates for KL8x.

### [2.0.0]

- Initial version.
- 

## LPTMR

### [2.2.1]

- Bug Fixes
  - Fix CERT INT31-C issues.

### [2.2.0]

- Improvements
  - Updated lptmr\_prescaler\_clock\_select\_t, only define the valid options.

### [2.1.1]

- Improvements
  - Updated the characters from “PTMR” to “LPTMR” in “FSL\_FEATURE\_PTMR\_HAS\_NO\_PRESCALER\_CLOCK\_SOURCE\_1\_SUPPORT” feature definition.

### [2.1.0]

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

**[2.0.2]**

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

**[2.0.1]**

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

**[2.0.0]**

- Initial version.
- 

**LPUART**

**[2.10.0]**

- New Feature
  - Added support to configure RTS watermark.

**[2.9.4]**

- Improvements
  - Merged duplicate code.

**[2.9.3]**

- Improvements
  - Added timeout for while loops in LPUART\_Deinit().

**[2.9.2]**

- Bug Fixes
  - Fixed coverity issues.

**[2.9.1]**

- Bug Fixes
  - Fixed coverity issues.

**[2.9.0]**

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

### [2.8.3]

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

### [2.8.2]

- Bug Fix
  - Fixed the bug that LPUART\_TransferEnable16Bit controlled by wrong feature macro.

### [2.8.1]

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

### [2.8.0]

- Improvements
  - Added support of DATA register for 9bit or 10bit data transmit in write and read API. Such as: LPUART\_WriteBlocking16bit, LPUART\_ReadBlocking16bit, LPUART\_TransferEnable16Bit, LPUART\_WriteNonBlocking16bit, LPUART\_ReadNonBlocking16bit.

### [2.7.7]

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

### [2.7.6]

- Bug Fixes
  - Fixed LPUART\_EnableInterrupts and LPUART\_DisableInterrupts bug that blocks if the LPUART address doesn't support exclusive access.

### [2.7.5]

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

### [2.7.4]

- Improvements
  - Added support for atomic register accessing in LPUART\_EnableInterrupts and LPUART\_DisableInterrupts.

### [2.7.3]

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

### [2.7.2]

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

### [2.7.1]

- Improvements
  - Added support for LPUART\_BASE\_PTRS\_NS in security mode in file fsl\_lpuart.c.

### [2.7.0]

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

### [2.6.0]

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

### [2.5.3]

- Bug Fixes
  - Fixed comments by replacing unused status flags kLPUART\_NoiseErrorInRxDataRegFlag and kLPUART\_ParityErrorInRxDataRegFlag with kLPUART\_NoiseErrorFlag and kLPUART\_ParityErrorFlag.

### [2.5.2]

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

### [2.5.1]

- Improvements
  - Use separate data for TX and RX in lpuart\_transfer\_t.
- Bug Fixes

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

#### [2.5.0]

- Bug Fixes

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

- New Features

- Added APIs `LPUART_GetRxFifoCount/LPUART_GetTxFifoCount` to get rx/tx FIFO data count.
- Added APIs `LPUART_SetRxFifoWatermark/LPUART_SetTxFifoWatermark` to set rx/tx FIFO water mark.

#### [2.4.1]

- Bug Fixes

- Fixed MISRA advisory 17.7 issues.

#### [2.4.0]

- New Features

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

#### [2.3.1]

- Bug Fixes

- Fixed MISRA advisory 15.5 issues.

### [2.3.0]

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

### [2.2.8]

- Bug Fixes
  - Fixed issue for MISRA-2012 check.
    - \* Fixed rule-10.3, rule-14.4, rule-15.5.
  - Eliminated Pa082 warnings by assigning volatile variables to local variables and using local variables instead.
  - Fixed MISRA issues.
    - \* Fixed rules 10.1, 10.3, 10.4, 10.8, 14.4, 11.6, 17.7.
- Improvements
  - Added check for kLPUART\_TransmissionCompleteFlag in LPUART\_WriteBlocking, LPUART\_TransferHandleIRQ, LPUART\_TransferSendDMACallback and LPUART\_SendEDMACallback to ensure all the data would be sent out to bus.
  - Rounded up the calculated sbr value in LPUART\_SetBaudRate and LPUART\_Init to achieve more accurate baudrate setting. Changed osr from uint32\_t to uint8\_t since osr's biggest value is 31.
  - Modified LPUART\_ReadBlocking so that if more than one receiver errors occur, all status flags will be cleared and the most severe error status will be returned.

### [2.2.7]

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

### [2.2.6]

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

### [2.2.5]

- Bug Fixes
  - Do not set or clear the TIE/RIE bits when using LPUART\_EnableTxDMA and LPUART\_EnableRxDMA.

#### [2.2.4]

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

#### [2.2.3]

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

#### [2.2.2]

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

#### [2.2.1]

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

#### [2.2.0]

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

#### [2.1.1]

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

[2.1.0]

- Improvements
    - Update transactional APIs.
- 

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.
- 

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

### [2.4.0]

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

### [2.3.3]

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

### [2.3.2]

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

### [2.3.1]

- Bug Fixes
  - Fixed CERT INT31-C violations.

### [2.3.0]

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

### [2.2.2]

- Improvements
  - Refine `_rtc_interrupt_enable` order.

### [2.2.1]

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

### [2.2.0]

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

### [2.1.0]

- Improvements
  - Added feature macro check for many features.

### [2.0.0]

- Initial version.
-

## 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\_SetPowerModeVlpw implementation.

#### [2.0.1]

- Other Changes
  - Updated for KL8x.

#### [2.0.0]

- Initial version.
- 

## SPI

#### [2.1.4]

- Bug Fixes
  - Fixed coverity issues.

#### [2.1.3]

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

#### [2.1.2]

- Improvements
  - Changed SPI\_DUMMYDATA to 0x00.

### [2.1.1]

- Bug Fixes
  - Fixed MISRA 10.3 violation.

### [2.1.0]

- Improvements
  - Added timeout mechanism when waiting certain states in transfer driver.
- Bug Fixes
  - Fixed the bug that, when working as a slave, instance that does not have FIFO may miss some rx data.
  - Fixed master RX data overflow issue by synchronizing transmit and receive process.
  - Fixed issue that slave should not share the same non-blocking initialization API and IRQ handler with master to prevent dead lock issue.
  - Fixed issue that callback should be invoked after all data is sent out to bus.
  - Added code in SPI\_SlaveTransferNonBlocking to empty rx buffer before initializing transfer.

### [2.0.5]

- Bug Fixes
  - Eliminated Pa082 warnings from SPI\_WriteNonBlocking and SPI\_GetStatusFlags.
  - Fixed MISRA issues.
    - \* Fixed issues 10.1, 10.3, 10.4, 10.7, 10.8, 11.9, 14.4, 17.7.

### [2.0.4]

- New Features
  - Supported 3-wire mode for SPI driver. Added new API SPI\_SetPinMode() to control the transfer direction of the single wire. For master instance, MOSI is selected as I/O pin. For slave instance, MISO is selected as I/O pin.
  - Added dummy data setup API to allow users to configure the dummy data to be transferred.

### [2.0.3]

- Bug Fixes
  - Fixed the potential interrupt race condition at high baudrate when calling API SPI\_MasterTransferNonBlocking.

### [2.0.2]

- New Features
  - Allowed users to set the transfer size for SPI\_TransferNoBlocking non-integer times of watermark.
  - Allowed users to define the dummy data. Users only need to define the macro SPI\_DUMMYDATA in applications.

### [2.0.1]

- Bug Fixes
  - Fixed SPI\_Enable function parameter error.
  - Set the s\_dummy variable as static variable in fsl\_spi\_dma.c.
- Improvements
  - Optimized the code size while not using transactional API.
  - Improved performance in polling method.
  - Added #ifndef/#endif to allow users to change the default tx value at compile time.

### [2.0.0]

- Initial version.
- 

## TPM

### [2.4.1]

- Improvements
  - Add Coverage Justification for uncovered code.

### [2.4.0]

- New Feature
  - Added while loop timeout for MOD CnV CnSC and SC register write sequence.
  - Change the return type from void to status\_t for following API:
    - \* TPM\_DisableChannel
    - \* TPM\_EnableChannel
    - \* TPM\_SetupOutputCompare
    - \* TPM\_SetTimerPeriod
    - \* TPM\_StopTimer

### [2.3.6]

- Bug Fixes
  - Fixed CERT INT30-C INT31-C issue for TPM\_SetupDualEdgeCapture.

### [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\_UpdateChnIEdgeLevelSelect 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.
- 

## 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.
- 

## 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.

[MCXC041](#)

## 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

## MCXC041

### 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  $x < \text{value1} \ || \ x > \text{value2}$ ; else,  $\text{value1} > x > \text{value2}$ .

enumerator kADC16\_HardwareCompareMode3

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

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_FlashClk`

Flash clock

enumerator `kCLOCK_Er32kClk`

External reference 32K clock (ERCLK32K)

enumerator `kCLOCK_Osc0ErClk`

OSC0 external reference clock (OSC0ERCLK)

enumerator `kCLOCK_McgFixedFreqClk`

MCG fixed frequency clock (MCGFFCLK)

enumerator `kCLOCK_McgInternalRefClk`

MCG internal reference clock (MCGIRCLK)

enumerator `kCLOCK_McgFllClk`

MCGFLLCLK

enumerator `kCLOCK_McgPeriphClk`

MCG peripheral clock (MCGPCLK)

enumerator `kCLOCK_McgIrc48MClk`

MCG IRC48M clock

enumerator `kCLOCK_LpoClk`

LPO clock

enum `_clock_ip_name`

Clock gate name used for `CLOCK_EnableClock/CLOCK_DisableClock`.

*Values:*

enumerator `kCLOCK_IpInvalid`

enumerator `kCLOCK_I2c0`

enumerator `kCLOCK_Cmp0`

enumerator `kCLOCK_Vref0`

enumerator `kCLOCK_Spi0`

enumerator `kCLOCK_Lptmr0`

enumerator `kCLOCK_PortA`

enumerator `kCLOCK_PortB`

enumerator kCLOCK\_Lpuart0  
 enumerator kCLOCK\_Ftf0  
 enumerator kCLOCK\_Tpm0  
 enumerator kCLOCK\_Tpm1  
 enumerator kCLOCK\_Adc0  
 enumerator kCLOCK\_Rtc0

enum \_osc\_cap\_load  
 Oscillator capacitor load setting.

*Values:*

enumerator kOSC\_Cap2P  
 2 pF capacitor load  
 enumerator kOSC\_Cap4P  
 4 pF capacitor load  
 enumerator kOSC\_Cap8P  
 8 pF capacitor load  
 enumerator kOSC\_Cap16P  
 16 pF capacitor load

enum \_oscer\_enable\_mode  
 OSCERCLK enable mode.

*Values:*

enumerator kOSC\_ErClkEnable  
 Enable.  
 enumerator kOSC\_ErClkEnableInStop  
 Enable in stop mode.

enum \_osc\_mode  
 The OSC work mode.

*Values:*

enumerator kOSC\_ModeExt  
 Use external clock.  
 enumerator kOSC\_ModeOscLowPower  
 Oscillator low power.

enum \_mcglite\_clkout\_src  
 MCG\_Lite clock source selection.

*Values:*

enumerator kMCGLITE\_ClkSrcHirc  
 MCGOUTCLK source is HIRC  
 enumerator kMCGLITE\_ClkSrcLirc  
 MCGOUTCLK source is LIRC  
 enumerator kMCGLITE\_ClkSrcExt  
 MCGOUTCLK source is external clock source

enumerator kMCGLITE\_ClkSrcReserved

enum \_mcglite\_lirc\_mode  
MCG\_Lite LIRC select.

*Values:*

enumerator kMCGLITE\_Lirc2M  
Slow internal reference(LIRC) 2 MHz clock selected

enumerator kMCGLITE\_Lirc8M  
Slow internal reference(LIRC) 8 MHz clock selected

enum \_mcglite\_lirc\_div  
MCG\_Lite divider factor selection for clock source.

*Values:*

enumerator kMCGLITE\_LircDivBy1  
Divider is 1

enumerator kMCGLITE\_LircDivBy2  
Divider is 2

enumerator kMCGLITE\_LircDivBy4  
Divider is 4

enumerator kMCGLITE\_LircDivBy8  
Divider is 8

enumerator kMCGLITE\_LircDivBy16  
Divider is 16

enumerator kMCGLITE\_LircDivBy32  
Divider is 32

enumerator kMCGLITE\_LircDivBy64  
Divider is 64

enumerator kMCGLITE\_LircDivBy128  
Divider is 128

enum \_mcglite\_mode  
MCG\_Lite clock mode definitions.

*Values:*

enumerator kMCGLITE\_ModeHirc48M  
Clock mode is HIRC 48 M

enumerator kMCGLITE\_ModeLirc8M  
Clock mode is LIRC 8 M

enumerator kMCGLITE\_ModeLirc2M  
Clock mode is LIRC 2 M

enumerator kMCGLITE\_ModeExt  
Clock mode is EXT

enumerator kMCGLITE\_ModeError  
Unknown mode

enum `_mcglite_ircclk_enable_mode`

MCG internal reference clock (MCGIRCLK) enable mode definition.

*Values:*

enumerator `kMCGLITE_IrcclkEnable`

MCGIRCLK enable.

enumerator `kMCGLITE_IrcclkEnableInStop`

MCGIRCLK enable in stop mode.

typedef enum `_clock_name` `clock_name_t`

Clock name used to get clock frequency.

typedef enum `_clock_ip_name` `clock_ip_name_t`

Clock gate name used for `CLOCK_EnableClock/CLOCK_DisableClock`.

typedef struct `_sim_clock_config` `sim_clock_config_t`

SIM configuration structure for clock setting.

typedef struct `_oscer_config` `oscer_config_t`

The OSC configuration for OSCERCLK.

typedef enum `_osc_mode` `osc_mode_t`

The OSC work mode.

typedef struct `_osc_config` `osc_config_t`

OSC Initialization Configuration Structure.

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

- a. `freq`: The external frequency.
- b. `workMode`: The OSC module mode.

typedef enum `_mcglite_clkout_src` `mcglite_clkout_src_t`

MCG\_Lite clock source selection.

typedef enum `_mcglite_lirc_mode` `mcglite_lirc_mode_t`

MCG\_Lite LIRC select.

typedef enum `_mcglite_lirc_div` `mcglite_lirc_div_t`

MCG\_Lite divider factor selection for clock source.

typedef enum `_mcglite_mode` `mcglite_mode_t`

MCG\_Lite clock mode definitions.

typedef struct `_mcglite_config` `mcglite_config_t`

MCG\_Lite configure structure for mode change.

volatile uint32\_t `g_xtal0Freq`

External XTAL0 (OSC0) clock frequency.

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

```
CLOCK_InitOsc0(...); // Set up the OSC0
CLOCK_SetXtal0Freq(8000000); // Set the XTAL0 value to clock driver.
```

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

volatile uint32\_t g\_xtal32Freq

The external XTAL32/EXTAL32/RTC\_CLKIN clock frequency.

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

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

static inline void `CLOCK_EnableClock(clock_ip_name_t name)`

Enable the clock for specific IP.

#### Parameters

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

static inline void `CLOCK_DisableClock(clock_ip_name_t name)`

Disable the clock for specific IP.

#### Parameters

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

static inline void `CLOCK_SetEr32kClock(uint32_t src)`

Set ERCLK32K source.

#### Parameters

- `src` – The value to set ERCLK32K clock source.

static inline void `CLOCK_SetLpuart0Clock(uint32_t src)`

Set LPUART clock source.

#### Parameters

- `src` – The value to set LPUART clock source.

static inline void `CLOCK_SetTpmClock(uint32_t src)`

Set TPM clock source.

#### Parameters

- `src` – The value to set TPM clock source.

static inline void `CLOCK_SetClkOutClock(uint32_t src)`

Set CLKOUT source.

#### Parameters

- `src` – The value to set CLKOUT source.

static inline void `CLOCK_SetRtcClkOutClock(uint32_t src)`

Set RTC\_CLKOUT source.

#### Parameters

- `src` – The value to set RTC\_CLKOUT source.

static inline void `CLOCK_SetOutDiv(uint32_t outdiv1, uint32_t outdiv4)`

System clock divider.

Set the `SIM_CLKDIV1[OUTDIV1]`, `SIM_CLKDIV1[OUTDIV4]`.

#### Parameters

- `outdiv1` – Clock 1 output divider value.
- `outdiv4` – Clock 4 output divider value.

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

Gets the clock frequency for a specific clock name.

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

**Parameters**

- clockName – Clock names defined in *clock\_name\_t*

**Returns**

Clock frequency value in Hertz

uint32\_t CLOCK\_GetCoreSysClkFreq(void)

Get the core clock or system clock frequency.

**Returns**

Clock frequency in Hz.

uint32\_t CLOCK\_GetPlatClkFreq(void)

Get the platform clock frequency.

**Returns**

Clock frequency in Hz.

uint32\_t CLOCK\_GetBusClkFreq(void)

Get the bus clock frequency.

**Returns**

Clock frequency in Hz.

uint32\_t CLOCK\_GetFlashClkFreq(void)

Get the flash clock frequency.

**Returns**

Clock frequency in Hz.

uint32\_t CLOCK\_GetEr32kClkFreq(void)

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

**Returns**

Clock frequency in Hz.

uint32\_t CLOCK\_GetOsc0ErClkFreq(void)

Get the OSC0 external reference clock frequency (OSC0ERCLK).

**Returns**

Clock frequency in Hz.

void CLOCK\_SetSimConfig(*sim\_clock\_config\_t* const \*config)

Set the clock configure in SIM module.

This function sets system layer clock settings in SIM module.

**Parameters**

- config – Pointer to the configure structure.

static inline void CLOCK\_SetSimSafeDivs(void)

Set the system clock dividers in SIM to safe value.

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

**Parameters**

- config – Pointer to the configure structure.

FSL\_CLOCK\_DRIVER\_VERSION

CLOCK driver version 2.0.0.

SDK\_DEVICE\_MAXIMUM\_CPU\_CLOCK\_FREQUENCY

RTC\_CLOCKS

Clock ip name array for RTC.

LPUART\_CLOCKS

Clock ip name array for LPUART.

SPI\_CLOCKS

Clock ip name array for SPI.

LPTMR\_CLOCKS

Clock ip name array for LPTMR.

ADC16\_CLOCKS

Clock ip name array for ADC16.

TPM\_CLOCKS

Clock ip name array for TPM.

VREF\_CLOCKS

Clock ip name array for VREF.

I2C\_CLOCKS

Clock ip name array for I2C.

PORT\_CLOCKS

Clock ip name array for PORT.

FTF\_CLOCKS

Clock ip name array for FTF.

CMP\_CLOCKS

Clock ip name array for CMP.

LPO\_CLK\_FREQ

LPO clock frequency.

SYS\_CLK

Peripherals clock source definition.

BUS\_CLK

I2C0\_CLK\_SRC

SPI0\_CLK\_SRC

CLK\_GATE\_REG\_OFFSET\_SHIFT

CLK\_GATE\_REG\_OFFSET\_MASK

CLK\_GATE\_BIT\_SHIFT\_SHIFT

CLK\_GATE\_BIT\_SHIFT\_MASK

CLK\_GATE\_DEFINE(reg\_offset, bit\_shift)

CLK\_GATE\_ABSTRACT\_REG\_OFFSET(x)

CLK\_GATE\_ABSTRACT\_BITS\_SHIFT(x)

uint32\_t CLOCK\_GetOutClkFreq(void)

Gets the MCG\_Lite output clock (MCGOUTCLK) frequency.

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

**Returns**

The frequency of MCGOUTCLK.

uint32\_t CLOCK\_GetInternalRefClkFreq(void)

Gets the MCG internal reference clock (MCGIRCLK) frequency.

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

**Returns**

The frequency of MCGIRCLK.

uint32\_t CLOCK\_GetPeriphClkFreq(void)

Gets the current MCGPCLK frequency.

This function gets the MCGPCLK frequency in Hz based on the current MCG\_Lite register settings.

**Returns**

The frequency of MCGPCLK.

mcglite\_mode\_t CLOCK\_GetMode(void)

Gets the current MCG\_Lite mode.

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

**Returns**

The current MCG\_Lite mode or error code.

status\_t CLOCK\_SetMcgliteConfig(mcglite\_config\_t const \*targetConfig)

Sets the MCG\_Lite configuration.

This function configures the MCG\_Lite, includes the output clock source, MCGIRCLK settings, HIRC settings, and so on. See mcglite\_config\_t for details.

**Parameters**

- targetConfig – Pointer to the target MCG\_Lite mode configuration structure.

**Returns**

Error code.

static inline void OSC\_SetExtRefClkConfig(OSC\_Type \*base, oscr\_config\_t const \*config)

Configures the OSC external reference clock (OSCERCLK).

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

```
oscer_config_t config =
{
    .enableMode = kOSC_ErClkEnable | kOSC_ErClkEnableInStop,
    .erclkDiv = 1U,
};
OSC_SetExtRefClkConfig(OSC, &config);
```

**Parameters**

- base – OSC peripheral address.

- `config` – Pointer to the configuration structure.

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

Sets the capacitor load configuration for the oscillator.

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

Example:

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

### Parameters

- `base` – OSC peripheral address.
- `capLoad` – OR'ed value for the capacitor load option. See `_osc_cap_load`.

`void CLOCK_InitOsc0(osc_config_t const *config)`

Initializes the OSC0.

This function initializes the OSC0 according to the board configuration.

### Parameters

- `config` – Pointer to the OSC0 configuration structure.

`void CLOCK_DeinitOsc0(void)`

Deinitializes the OSC0.

This function deinitializes the OSC0.

`static inline void CLOCK_SetXtal0Freq(uint32_t freq)`

Sets the XTAL0 frequency based on board settings.

### Parameters

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

`static inline void CLOCK_SetXtal32Freq(uint32_t freq)`

Sets the XTAL32/RTC\_CLKIN frequency based on board settings.

### Parameters

- `freq` – The XTAL32/EXTAL32/RTC\_CLKIN input clock frequency in Hz.

`uint8_t er32kSrc`

ERCLK32K source selection.

`uint32_t clkdiv1`

SIM\_CLKDIV1.

`uint8_t enableMode`

OSCERCLK enable mode. OR'ed value of `_oscer_enable_mode`.

`uint32_t freq`

External clock frequency.

`uint8_t capLoad`

Capacitor load setting.

`osc_mode_t workMode`

OSC work mode setting.

*oscer\_config\_t* oscerConfig

Configuration for OSCERCLK.

*mcglite\_clkout\_src\_t* outSrc

MCGOUT clock select.

uint8\_t irclkEnableMode

MCGIRCLK enable mode, OR'ed value of `_mcglite_irclk_enable_mode`.

*mcglite\_lirc\_mode\_t* ircs

MCG\_C2[IRCS].

*mcglite\_lirc\_div\_t* fcrdiv

MCG\_SC[FCRDIV].

*mcglite\_lirc\_div\_t* lircDiv2

MCG\_MC[LIRC\_DIV2].

bool hircEnableInNotHircMode

HIRC enable when not in HIRC mode.

FSL\_SDK\_DISABLE\_DRIVER\_CLOCK\_CONTROL

Configure whether driver controls clock.

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

---

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

---

struct `_sim_clock_config`

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

struct `_oscer_config`

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

struct `_osc_config`

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

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

- a. `freq`: The external frequency.
- b. `workMode`: The OSC module mode.

struct `_mcglite_config`

`#include <fsl_clock.h>` MCG\_Lite configure structure for mode change.

## 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 COP: Watchdog Driver

void COP\_GetDefaultConfig(*cop\_config\_t* \*config)

Initializes the COP configuration structure.

This function initializes the COP configuration structure to default values. The default values are:

```
copConfig->enableWindowMode = false;
copConfig->timeoutMode = kCOP_LongTimeoutMode;
copConfig->enableStop = false;
copConfig->enableDebug = false;
copConfig->clockSource = kCOP_LpoClock;
copConfig->timeoutCycles = kCOP_2Power10CyclesOr2Power18Cycles;
```

### See also:

[cop\\_config\\_t](#)

### Parameters

- config – Pointer to the COP configuration structure.

void COP\_Init(SIM\_Type \*base, const *cop\_config\_t* \*config)

Initializes the COP module.

This function configures the COP. After it is called, the COP starts running according to the configuration. Because all COP control registers are write-once only, the COP\_Init function and the COP\_Disable function can be called only once. A second call has no effect.

Example:

```
cop_config_t config;
COP_GetDefaultConfig(&config);
config.timeoutCycles = kCOP_2Power8CyclesOr2Power16Cycles;
COP_Init(sim_base,&config);
```

### Parameters

- base – SIM peripheral base address.
- config – The configuration of COP.

```
static inline void COP_Disable(SIM_Type *base)
```

De-initializes the COP module. This dedicated function is not provided. Instead, the COP\_Disable function can be used to disable the COP.

Disables the COP module.

This function disables the COP Watchdog. Note: The COP configuration register is a write-once after reset. To disable the COP Watchdog, call this function first.

#### Parameters

- base – SIM peripheral base address.

```
void COP_Refresh(SIM_Type *base)
```

Refreshes the COP timer.

This function feeds the COP.

#### Parameters

- base – SIM peripheral base address.

```
FSL_COP_DRIVER_VERSION
```

COP driver version 2.0.2.

```
COP_FIRST_BYTE_OF_REFRESH
```

First byte of refresh sequence

```
COP_SECOND_BYTE_OF_REFRESH
```

Second byte of refresh sequence

```
enum _cop_clock_source
```

COP clock source selection.

*Values:*

```
enumerator kCOP_LpoClock
```

COP clock sourced from LPO

```
enumerator kCOP_McgIrClock
```

COP clock sourced from MCGIRCLK

```
enumerator kCOP_OscErClock
```

COP clock sourced from OSCERCLK

```
enumerator kCOP_BusClock
```

COP clock sourced from Bus clock

```
enum _cop_timeout_cycles
```

Define the COP timeout cycles.

*Values:*

```
enumerator kCOP_2Power5CyclesOr2Power13Cycles
```

2<sup>5</sup> or 2<sup>13</sup> clock cycles

```
enumerator kCOP_2Power8CyclesOr2Power16Cycles
```

2<sup>8</sup> or 2<sup>16</sup> clock cycles

```
enumerator kCOP_2Power10CyclesOr2Power18Cycles
```

2<sup>10</sup> or 2<sup>18</sup> clock cycles

```
enum _cop_timeout_mode
```

Define the COP timeout mode.

*Values:*

```
enumerator kCOP_ShortTimeoutMode
    COP selects long timeout
enumerator kCOP_LongTimeoutMode
    COP selects short timeout
typedef enum _cop_clock_source cop_clock_source_t
    COP clock source selection.
typedef enum _cop_timeout_cycles cop_timeout_cycles_t
    Define the COP timeout cycles.
typedef enum _cop_timeout_mode cop_timeout_mode_t
    Define the COP timeout mode.
typedef struct _cop_config cop_config_t
    Describes COP configuration structure.
struct _cop_config
    #include <fsl_cop.h> Describes COP configuration structure.
```

### Public Members

```
bool enableWindowMode
    COP run mode: window mode or normal mode
cop_timeout_mode_t timeoutMode
    COP timeout mode: long timeout or short timeout
bool enableStop
    Enable or disable COP in STOP mode
bool enableDebug
    Enable or disable COP in DEBUG mode
cop_clock_source_t clockSource
    Set COP clock source
cop_timeout_cycles_t timeoutCycles
    Set COP timeout value
```

## 2.5 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 =
```

(continues on next page)

(continued from previous page)

```
{
  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 – FGPIOpin 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

## 2.6 C90TFS Flash Driver

## 2.7 ftx adapter

## 2.8 Ftftx CACHE Driver

enum \_ftfx\_cache\_ram\_func\_constants

Constants for execute-in-RAM flash function.

*Values:*

enumerator kFTF<sub>x</sub>\_CACHE\_RamFuncMaxSizeInWords

The maximum size of execute-in-RAM function.

typedef struct \_flash\_prefetch\_speculation\_status ftx\_prefetch\_speculation\_status\_t

FTFx prefetch speculation status.

typedef struct \_ftfx\_cache\_config ftx\_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.

## 2.9 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 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.
- `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 kFTTFx\_FlexramFuncOptEepromQuickWriteRecovery

An option used to complete interrupted EEPROM quick write process

enumerator kFTTFx\_FlexramFuncOptEepromQuickWriteStatus

An option used to make EEPROM quick write status query

enumerator kFTTFx\_FlexramFuncOptAvailableForEepromQuickWrite

An option used to make FlexRAM available for EEPROM in Quick Write mode

enumerator kFTTFx\_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 kFTTFx\_SwapControlOptionInitializeSystem

An option used to initialize the Swap system

enumerator kFTTFx\_SwapControlOptionSetInUpdateState

An option used to set the Swap in an update state

enumerator kFTTFx\_SwapControlOptionSetInCompleteState

An option used to set the Swap in a complete state

enumerator kFTTFx\_SwapControlOptionReportStatus

An option used to report the Swap status

enumerator kFTTFx\_SwapControlOptionDisableSystem

An option used to disable the Swap status

enum \_ftfx\_swap\_state

Enumeration for the possible flash Swap status.

*Values:*

enumerator kFTTFx\_SwapStateUninitialized

Flash Swap system is in an uninitialized state.

enumerator kFTTFx\_SwapStateReady

Flash Swap system is in a ready state.

enumerator kFTTFx\_SwapStateUpdate

Flash Swap system is in an update state.

enumerator kFTTFx\_SwapStateUpdateErased

Flash Swap system is in an updateErased state.

enumerator kFTTFx\_SwapStateComplete

Flash Swap system is in a complete state.

enumerator kFTTFx\_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 blockBase
    A base address of the flash block
uint32_t aliasBlockBase
    A base address of the alias flash block
uint32_t totalSize
    The size of the flash block.
uint32_t sectorSize
    The size in bytes of a sector of flash.
uint32_t blockCount
    A number of flash blocks.
```

```
struct _ftfx_ops_config
    #include <fsl_ftfx_controller.h> Active FTFx information for the current operation.
```

### Public Members

```
uint32_t convertedAddress
    A converted address for the current flash type.
```

```
struct _ftfx_ifr_descriptor
    #include <fsl_ftfx_controller.h> Flash IFR memory descriptor.
union function_ptr_t
    #include <fsl_ftfx_controller.h>
```

### Public Members

```
uint32_t commadAddr
void (*callFlashCommand)(volatile uint8_t *FTMRx_fstat)
```

```
struct _ftfx_config
    #include <fsl_ftfx_controller.h> Flash driver state information.
    An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.
```

### Public Members

```
uint32_t flexramBlockBase
    The base address of the FlexRAM/acceleration RAM
uint32_t flexramTotalSize
    The size of the FlexRAM/acceleration RAM
uint16_t eepromTotalSize
    The size of EEPROM area which was partitioned from FlexRAM
function_ptr_t runCmdFuncAddr
    An buffer point to the flash execute-in-RAM function.
```

```
struct __unnamed3__
```

### 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
```

## 2.10 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
```

## 2.11 Ftfx 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-feild with given index and length.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `index` – The index indicating the area of program once field to be read.
- `src` – A pointer to the source buffer of data that is used to store data to be write.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; The index indicating the area of program once field was 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 feild with given index and length.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `index` – The index indicating the area of program once field to be read.
- `dst` – A pointer to the destination buffer of data that is used to store data to be read.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the data have been successfully read form Program flash0 IFR map and Program Once field based on index and length.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.

- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FLASH_VerifyErase(flash_config_t *config, uint32_t start, uint32_t lengthInBytes, ftfx_margin_value_t margin)`

Verifies an erasure of the desired flash area at a specified margin level.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- `margin` – Read margin choice.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the specified FLASH region has been erased.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – Parameter is not aligned with specified baseline.
- `kStatus_FTFx_AddressError` – Address is out of range.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t FLASH_VerifyEraseAll(flash_config_t *config, ftfx_margin_value_t margin)`

Verifies erasure of the entire flash at a specified margin level.

This function checks whether the flash is erased to the specified read margin level.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `margin` – Read margin choice.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; all program flash and flexnvm were in erased state.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.

- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

`status_t` FLASH\_VerifyProgram(*flash\_config\_t* \*config, uint32\_t start, uint32\_t lengthInBytes, const uint8\_t \*expectedData, *ftfx\_margin\_value\_t* margin, uint32\_t \*failedAddress, uint32\_t \*failedData)

Verifies programming of the desired flash area at a specified margin level.

This function verifies the data programmed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be verified. Must be word-aligned.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- `expectedData` – A pointer to the expected data that is to be verified against.
- `margin` – Read margin choice.
- `failedAddress` – A pointer to the returned failing address.
- `failedData` – A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the desired data have been successfully programmed 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 3.3.0.

`FSL_FLASH_DRIVER_VERSION_ROM`

Flash driver version for ROM.

Version 3.0.0.

`enum _flash_protection_state`

Enumeration for the three possible flash protection levels.

*Values:*

enumerator `kFLASH_ProtectionStateUnprotected`

Flash region is not protected.

enumerator kFLASH\_ProtectionStateProtected

Flash region is protected.

enumerator kFLASH\_ProtectionStateMixed

Flash is mixed with protected and unprotected region.

enum \_flash\_execute\_only\_access\_state

Enumeration for the three possible flash execute access levels.

*Values:*

enumerator kFLASH\_AccessStateUnLimited

Flash region is unlimited.

enumerator kFLASH\_AccessStateExecuteOnly

Flash region is execute only.

enumerator kFLASH\_AccessStateMixed

Flash is mixed with unlimited and execute only region.

enum \_flash\_property\_tag

Enumeration for various flash properties.

*Values:*

enumerator kFLASH\_PropertyPflash0SectorSize

Pflash sector size property.

enumerator kFLASH\_PropertyPflash0TotalSize

Pflash total size property.

enumerator kFLASH\_PropertyPflash0BlockSize

Pflash block size property.

enumerator kFLASH\_PropertyPflash0BlockCount

Pflash block count property.

enumerator kFLASH\_PropertyPflash0BlockBaseAddr

Pflash block base address property.

enumerator kFLASH\_PropertyPflash0FacSupport

Pflash fac support property.

enumerator kFLASH\_PropertyPflash0AccessSegmentSize

Pflash access segment size property.

enumerator kFLASH\_PropertyPflash0AccessSegmentCount

Pflash access segment count property.

enumerator kFLASH\_PropertyPflash1SectorSize

Pflash sector size property.

enumerator kFLASH\_PropertyPflash1TotalSize

Pflash total size property.

enumerator kFLASH\_PropertyPflash1BlockSize

Pflash block size property.

enumerator kFLASH\_PropertyPflash1BlockCount

Pflash block count property.

enumerator kFLASH\_PropertyPflash1BlockBaseAddr

Pflash block base address property.

enumerator kFLASH\_PropertyPflash1FacSupport

Pflash fac support property.

enumerator kFLASH\_PropertyPflash1AccessSegmentSize

Pflash access segment size property.

enumerator kFLASH\_PropertyPflash1AccessSegmentCount

Pflash access segment count property.

enumerator kFLASH\_PropertyFlexRamBlockBaseAddr

FlexRam block base address property.

enumerator kFLASH\_PropertyFlexRamTotalSize

FlexRam total size property.

typedef enum *\_flash\_protection\_state* flash\_prot\_state\_t

Enumeration for the three possible flash protection levels.

typedef union *\_pflash\_protection\_status* pflash\_prot\_status\_t

PFlash protection status.

typedef enum *\_flash\_execute\_only\_access\_state* flash\_xacc\_state\_t

Enumeration for the three possible flash execute access levels.

typedef enum *\_flash\_property\_tag* flash\_property\_tag\_t

Enumeration for various flash properties.

typedef struct *\_flash\_config* flash\_config\_t

Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

kStatus\_FLASH\_Success

kFLASH\_ApiEraseKey

union *\_pflash\_protection\_status*

*#include <fsl\_ftfx\_flash.h>* PFlash protection status.

### Public Members

uint32\_t protl

PROT[31:0].

uint32\_t proth

PROT[63:32].

uint8\_t protsl

PROTS[7:0].

uint8\_t protsh

PROTS[15:8].

uint8\_t reserved[2]

struct *\_flash\_config*

*#include <fsl\_ftfx\_flash.h>* Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

## 2.12 Ftftx FLEXNVM Driver

*status\_t* FLEXNVM\_Init(*flexnvm\_config\_t* \*config)

Initializes the global flash properties structure members.

This function checks and initializes the Flash module for the other Flash APIs.

### Parameters

- config – Pointer to the storage for the driver runtime state.

### Return values

- kStatus\_FTFx\_Success – API was executed successfully.
- kStatus\_FTFx\_InvalidArgument – An invalid argument is provided.
- kStatus\_FTFx\_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus\_FTFx\_PartitionStatusUpdateFailure – Failed to update the partition status.

*status\_t* FLEXNVM\_DflashErase(*flexnvm\_config\_t* \*config, uint32\_t start, uint32\_t lengthInBytes, uint32\_t key)

Erases the Dflash sectors encompassed by parameters passed into function.

This function erases the appropriate number of flash sectors based on the desired start address and length.

### Parameters

- config – The pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words) to be erased. Must be word-aligned.
- key – The value used to validate all flash erase APIs.

### Return values

- kStatus\_FTFx\_Success – API was executed successfully; the appropriate number of date flash sectors based on the desired start address and length were erased successfully.
- kStatus\_FTFx\_InvalidArgument – An invalid argument is provided.
- kStatus\_FTFx\_AlignmentError – The parameter is not aligned with the specified baseline.
- kStatus\_FTFx\_AddressError – The address is out of range.
- kStatus\_FTFx\_EraseKeyError – The API erase key is invalid.
- kStatus\_FTFx\_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus\_FTFx\_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus\_FTFx\_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus\_FTFx\_CommandFailure – Run-time error during the command execution.

*status\_t* FLEXNVM\_EraseAll(*flexnvm\_config\_t* \*config, uint32\_t key)

Erases entire flexnvm.

#### Parameters

- config – Pointer to the storage for the driver runtime state.
- key – A value used to validate all flash erase APIs.

#### Return values

- kStatus\_FTFx\_Success – API was executed successfully; the entire flexnvm has been erased successfully.
- kStatus\_FTFx\_InvalidArgument – An invalid argument is provided.
- kStatus\_FTFx\_EraseKeyError – API erase key is invalid.
- kStatus\_FTFx\_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus\_FTFx\_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus\_FTFx\_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus\_FTFx\_CommandFailure – Run-time error during command execution.
- kStatus\_FTFx\_PartitionStatusUpdateFailure – Failed to update the partition status.

*status\_t* FLEXNVM\_EraseAllUnsecure(*flexnvm\_config\_t* \*config, uint32\_t key)

Erases the entire flexnvm, including protected sectors.

#### Parameters

- config – Pointer to the storage for the driver runtime state.
- key – A value used to validate all flash erase APIs.

#### Return values

- kStatus\_FTFx\_Success – API was executed successfully; the flexnvm is not in 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 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 flash area.
- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_AlignmentError` – Parameter is not aligned with specified baseline.

- `kStatus_FTFx_AddressError` – Address is out of range.
- `kStatus_FTFx_SetFlexramAsRamError` – Failed to set flexram as RAM.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during command execution.
- `kStatus_FTFx_RecoverFlexramAsEepromError` – Failed to recover FlexRAM as EEPROM.

`status_t` FLEXNVM\_ProgramPartition(*flexnvm\_config\_t* \*config,  
*ftfx\_partition\_flexram\_load\_opt\_t* option, uint32\_t  
 eepromDataSizeCode, uint32\_t flexnvmPartitionCode)

Prepares the FlexNVM block for use as data flash, EEPROM backup, or a combination of both and initializes the FlexRAM.

#### Parameters

- `config` – Pointer to storage for the driver runtime state.
- `option` – The option used to set FlexRAM load behavior during reset.
- `eepromDataSizeCode` – Determines the amount of FlexRAM used in each of the available EEPROM subsystems.
- `flexnvmPartitionCode` – Specifies how to split the FlexNVM block between data flash memory and EEPROM backup memory supporting EEPROM functions.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the FlexNVM block for use as data flash, EEPROM backup, or a combination of both have been Prepared.
- `kStatus_FTFx_InvalidArgument` – Invalid argument is provided.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during command execution.

`status_t` FLEXNVM\_ProgramPartition\_CSE(*flexnvm\_config\_t* \*config,  
*ftfx\_partition\_flexram\_load\_opt\_t* option, uint32\_t  
 eepromDataSizeCode, uint32\_t  
 flexnvmPartitionCode, uint8\_t CSEcKeySize, uint8\_t  
 SFE)

Prepares the FlexNVM block for use as data flash, EEPROM backup, or a combination of both and initializes the FlexRAM. This is the CSE enabled version for IP's like FTFC.

#### Parameters

- `config` – Pointer to storage for the driver runtime state.
- `option` – The option used to set FlexRAM load behavior during reset.
- `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(*flexnvm\_config\_t* \*config, *ftfx\_margin\_value\_t* margin)

Verifies erasure of the entire flash at a specified margin level.

This function checks whether the flash is erased to the specified read margin level.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `margin` – Read margin choice.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the entire flexnvm region is in erased state.

- `kStatus_FTFx_InvalidArgument` – An invalid argument is provided.
- `kStatus_FTFx_ExecuteInRamFunctionNotReady` – Execute-in-RAM function is not available.
- `kStatus_FTFx_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FTFx_ProtectionViolation` – The program/erase operation is requested to execute on protected areas.
- `kStatus_FTFx_CommandFailure` – Run-time error during the command execution.

```
status_t FLEXNVM_DflashVerifyProgram(flexnvm_config_t *config, uint32_t start, uint32_t
lengthInBytes, const uint8_t *expectedData,
ftfx_margin_value_t margin, uint32_t *failedAddress,
uint32_t *failedData)
```

Verifies programming of the desired flash area at a specified margin level.

This function verifies the data programmed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

#### Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be verified. Must be word-aligned.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- `expectedData` – A pointer to the expected data that is to be verified against.
- `margin` – Read margin choice.
- `failedAddress` – A pointer to the returned failing address.
- `failedData` – A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

#### Return values

- `kStatus_FTFx_Success` – API was executed successfully; the desired data 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 desired data have been successfully programmed 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.

## 2.13 ftx 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)

## 2.14 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

## 2.15 GPIO Driver

void GPIO\_PortInit(GPIO\_Type \*base)

Initializes the GPIO peripheral.

This function ungates the GPIO clock.

### Parameters

- base – GPIO peripheral base pointer.

void GPIO\_PortDenit(GPIO\_Type \*base)

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

## 2.16 I2C: Inter-Integrated Circuit Driver

### 2.17 I2C Driver

```
void I2C_MasterInit(I2C_Type *base, const i2c_master_config_t *masterConfig, uint32_t srcClock_Hz)
```

Initializes the I2C peripheral. Call this API to ungate the I2C clock and configure the I2C with master configuration.

---

**Note:** This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can be custom filled or it can be set with default values by using the I2C\_MasterGetDefaultConfig(). After calling this API, the master is ready to transfer. This is an example.

```
i2c_master_config_t config = {
    .enableMaster = true,
    .enableStopHold = false,
    .highDrive = false,
    .baudRate_Bps = 100000,
    .glitchFilterWidth = 0
};
I2C_MasterInit(I2C0, &config, 12000000U);
```

---

#### Parameters

- base – I2C base pointer
- masterConfig – A pointer to the master configuration structure
- srcClock\_Hz – I2C peripheral clock frequency in Hz

```
void I2C_SlaveInit(I2C_Type *base, const i2c_slave_config_t *slaveConfig, uint32_t srcClock_Hz)
```

Initializes the I2C peripheral. Call this API to ungate the I2C clock and initialize the I2C with the slave configuration.

---

**Note:** This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can partly be set with default values by I2C\_SlaveGetDefaultConfig() or it can be custom filled by the user. This is an example.

```
i2c_slave_config_t config = {
    .enableSlave = true,
    .enableGeneralCall = false,
    .addressingMode = kI2C_Address7bit,
    .slaveAddress = 0x1DU,
    .enableWakeUp = false,
    .enablehighDrive = false,
    .enableBaudRateCtl = false,
    .sclStopHoldTime_ns = 4000
};
I2C_SlaveInit(I2C0, &config, 12000000U);
```

### Parameters

- base – I2C base pointer
- slaveConfig – A pointer to the slave configuration structure
- srcClock\_Hz – I2C peripheral clock frequency in Hz

```
void I2C_MasterDeinit(I2C_Type *base)
```

De-initializes the I2C master peripheral. Call this API to gate the I2C clock. The I2C master module can't work unless the I2C\_MasterInit is called.

### Parameters

- base – I2C base pointer

```
void I2C_SlaveDeinit(I2C_Type *base)
```

De-initializes the I2C slave peripheral. Calling this API gates the I2C clock. The I2C slave module can't work unless the I2C\_SlaveInit is called to enable the clock.

### Parameters

- base – I2C base pointer

```
uint32_t I2C_GetInstance(I2C_Type *base)
```

Get instance number for I2C module.

### Parameters

- base – I2C peripheral base address.

```
void I2C_MasterGetDefaultConfig(i2c_master_config_t *masterConfig)
```

Sets the I2C master configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in the I2C\_MasterConfigure(). Use the initialized structure unchanged in the I2C\_MasterConfigure() or modify the structure before calling the I2C\_MasterConfigure(). This is an example.

```
i2c_master_config_t config;
I2C_MasterGetDefaultConfig(&config);
```

### Parameters

- masterConfig – A pointer to the master configuration structure.

```
void I2C_SlaveGetDefaultConfig(i2c_slave_config_t *slaveConfig)
```

Sets the I2C slave configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in the I2C\_SlaveConfigure(). Modify fields of the structure before calling the I2C\_SlaveConfigure(). This is an example.

```
i2c_slave_config_t config;  
I2C_SlaveGetDefaultConfig(&config);
```

### Parameters

- slaveConfig – A pointer to the slave configuration structure.

```
static inline void I2C_Enable(I2C_Type *base, bool enable)
```

Enables or disables the I2C peripheral operation.

### Parameters

- base – I2C base pointer
- enable – Pass true to enable and false to disable the module.

```
uint32_t I2C_MasterGetStatusFlags(I2C_Type *base)
```

Gets the I2C status flags.

### Parameters

- base – I2C base pointer

### Returns

status flag, use status flag to AND `_i2c_flags` to get the related status.

```
static inline uint32_t I2C_SlaveGetStatusFlags(I2C_Type *base)
```

Gets the I2C status flags.

### Parameters

- base – I2C base pointer

### Returns

status flag, use status flag to AND `_i2c_flags` to get the related status.

```
static inline void I2C_MasterClearStatusFlags(I2C_Type *base, uint32_t statusMask)
```

Clears the I2C status flag state.

The following status register flags can be cleared `kI2C_ArbitrationLostFlag` and `kI2C_IntPendingFlag`.

### Parameters

- base – I2C base pointer
- statusMask – The status flag mask, defined in type `i2c_status_flag_t`. The parameter can be any combination of the following values:
  - `kI2C_StartDetectFlag` (if available)
  - `kI2C_StopDetectFlag` (if available)
  - `kI2C_ArbitrationLostFlag`
  - `kI2C_IntPendingFlag`

```
static inline void I2C_SlaveClearStatusFlags(I2C_Type *base, uint32_t statusMask)
```

Clears the I2C status flag state.

The following status register flags can be cleared `kI2C_ArbitrationLostFlag` and `kI2C_IntPendingFlag`.

### Parameters

- base – I2C base pointer
- statusMask – The status flag mask, defined in type `i2c_status_flag_t`. The parameter can be any combination of the following values:

- kI2C\_StartDetectFlag (if available)
- kI2C\_StopDetectFlag (if available)
- kI2C\_ArbitrationLostFlag
- kI2C\_IntPendingFlagFlag

`void I2C_EnableInterrupts(I2C_Type *base, uint32_t mask)`

Enables I2C interrupt requests.

#### Parameters

- base – I2C base pointer
- mask – interrupt source The parameter can be combination of the following source if defined:
  - kI2C\_GlobalInterruptEnable
  - kI2C\_StopDetectInterruptEnable/kI2C\_StartDetectInterruptEnable
  - kI2C\_SdaTimeoutInterruptEnable

`void I2C_DisableInterrupts(I2C_Type *base, uint32_t mask)`

Disables I2C interrupt requests.

#### Parameters

- base – I2C base pointer
- mask – interrupt source The parameter can be combination of the following source if defined:
  - kI2C\_GlobalInterruptEnable
  - kI2C\_StopDetectInterruptEnable/kI2C\_StartDetectInterruptEnable
  - kI2C\_SdaTimeoutInterruptEnable

`static inline void I2C_EnableDMA(I2C_Type *base, bool enable)`

Enables/disables the I2C DMA interrupt.

#### Parameters

- base – I2C base pointer
- enable – true to enable, false to disable

`static inline uint32_t I2C_GetDataRegAddr(I2C_Type *base)`

Gets the I2C tx/rx data register address. This API is used to provide a transfer address for I2C DMA transfer configuration.

#### Parameters

- base – I2C base pointer

#### Returns

data register address

`void I2C_MasterSetBaudRate(I2C_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)`

Sets the I2C master transfer baud rate.

#### Parameters

- base – I2C base pointer
- baudRate\_Bps – the baud rate value in bps
- srcClock\_Hz – Source clock

*status\_t* I2C\_MasterStart(I2C\_Type \*base, uint8\_t address, *i2c\_direction\_t* direction)

Sends a START on the I2C bus.

This function is used to initiate a new master mode transfer by sending the START signal. The slave address is sent following the I2C START signal.

#### Parameters

- base – I2C peripheral base pointer
- address – 7-bit slave device address.
- direction – Master transfer directions(transmit/receive).

#### Return values

- kStatus\_Success – Successfully send the start signal.
- kStatus\_I2C\_Busy – Current bus is busy.

*status\_t* I2C\_MasterStop(I2C\_Type \*base)

Sends a STOP signal on the I2C bus.

#### Return values

- kStatus\_Success – Successfully send the stop signal.
- kStatus\_I2C\_Timeout – Send stop signal failed, timeout.

*status\_t* I2C\_MasterRepeatedStart(I2C\_Type \*base, uint8\_t address, *i2c\_direction\_t* direction)

Sends a REPEATED START on the I2C bus.

#### Parameters

- base – I2C peripheral base pointer
- address – 7-bit slave device address.
- direction – Master transfer directions(transmit/receive).

#### Return values

- kStatus\_Success – Successfully send the start signal.
- kStatus\_I2C\_Busy – Current bus is busy but not occupied by current I2C master.

*status\_t* I2C\_MasterWriteBlocking(I2C\_Type \*base, const uint8\_t \*txBuff, size\_t txSize, uint32\_t flags)

Performs a polling send transaction on the I2C bus.

#### Parameters

- base – The I2C peripheral base pointer.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.
- flags – Transfer control flag to decide whether need to send a stop, use kI2C\_TransferDefaultFlag to issue a stop and kI2C\_TransferNoStop to not send a stop.

#### Return values

- kStatus\_Success – Successfully complete the data transmission.
- kStatus\_I2C\_ArbitrationLost – Transfer error, arbitration lost.
- kStatus\_I2C\_Nak – Transfer error, receive NAK during transfer.

*status\_t* I2C\_MasterReadBlocking(I2C\_Type \*base, uint8\_t \*rxBuff, size\_t rxSize, uint32\_t flags)

Performs a polling receive transaction on the I2C bus.

---

**Note:** The I2C\_MasterReadBlocking function stops the bus before reading the final byte. Without stopping the bus prior for the final read, the bus issues another read, resulting in garbage data being read into the data register.

---

#### Parameters

- base – I2C peripheral base pointer.
- rxBuff – The pointer to the data to store the received data.
- rxSize – The length in bytes of the data to be received.
- flags – Transfer control flag to decide whether need to send a stop, use kI2C\_TransferDefaultFlag to issue a stop and kI2C\_TransferNoStop to not send a stop.

#### Return values

- kStatus\_Success – Successfully complete the data transmission.
- kStatus\_I2C\_Timeout – Send stop signal failed, timeout.

*status\_t* I2C\_SlaveWriteBlocking(I2C\_Type \*base, const uint8\_t \*txBuff, size\_t txSize)

Performs a polling send transaction on the I2C bus.

#### Parameters

- base – The I2C peripheral base pointer.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

#### Return values

- kStatus\_Success – Successfully complete the data transmission.
- kStatus\_I2C\_ArbitrationLost – Transfer error, arbitration lost.
- kStatus\_I2C\_Nak – Transfer error, receive NAK during transfer.

*status\_t* I2C\_SlaveReadBlocking(I2C\_Type \*base, uint8\_t \*rxBuff, size\_t rxSize)

Performs a polling receive transaction on the I2C bus.

#### Parameters

- base – I2C peripheral base pointer.
- rxBuff – The pointer to the data to store the received data.
- rxSize – The length in bytes of the data to be received.

#### Return values

- kStatus\_Success – Successfully complete data receive.
- kStatus\_I2C\_Timeout – Wait status flag timeout.

*status\_t* I2C\_MasterTransferBlocking(I2C\_Type \*base, i2c\_master\_transfer\_t \*xfer)

Performs a master polling transfer on the I2C bus.

---

**Note:** The API does not return until the transfer succeeds or fails due to arbitration lost or receiving a NAK.

---

**Parameters**

- base – I2C peripheral base address.
- xfer – Pointer to the transfer structure.

**Return values**

- kStatus\_Success – Successfully complete the data transmission.
- kStatus\_I2C\_Busy – Previous transmission still not finished.
- kStatus\_I2C\_Timeout – Transfer error, wait signal timeout.
- kStatus\_I2C\_ArbitrationLost – Transfer error, arbitration lost.
- kStatus\_I2C\_Nak – Transfer error, receive NAK during transfer.

```
void I2C_MasterTransferCreateHandle(I2C_Type *base, i2c_master_handle_t *handle,  
                                   i2c_master_transfer_callback_t callback, void *userData)
```

Initializes the I2C handle which is used in transactional functions.

**Parameters**

- base – I2C base pointer.
- handle – pointer to i2c\_master\_handle\_t structure to store the transfer state.
- callback – pointer to user callback function.
- userData – user parameter passed to the callback function.

```
status_t I2C_MasterTransferNonBlocking(I2C_Type *base, i2c_master_handle_t *handle,  
                                       i2c_master_transfer_t *xfer)
```

Performs a master interrupt non-blocking transfer on the I2C bus.

---

**Note:** Calling the API returns immediately after transfer initiates. The user needs to call I2C\_MasterGetTransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus\_I2C\_Busy, the transfer is finished.

---

**Parameters**

- base – I2C base pointer.
- handle – pointer to i2c\_master\_handle\_t structure which stores the transfer state.
- xfer – pointer to i2c\_master\_transfer\_t structure.

**Return values**

- kStatus\_Success – Successfully start the data transmission.
- kStatus\_I2C\_Busy – Previous transmission still not finished.
- kStatus\_I2C\_Timeout – Transfer error, wait signal timeout.

```
status_t I2C_MasterTransferGetCount(I2C_Type *base, i2c_master_handle_t *handle, size_t  
                                   *count)
```

Gets the master transfer status during a interrupt non-blocking transfer.

**Parameters**

- base – I2C base pointer.
- handle – pointer to i2c\_master\_handle\_t structure which stores the transfer state.

- `count` – Number of bytes transferred so far by the non-blocking transaction.

**Return values**

- `kStatus_InvalidArgument` – `count` is Invalid.
- `kStatus_Success` – Successfully return the count.

`status_t I2C_MasterTransferAbort(I2C_Type *base, i2c_master_handle_t *handle)`

Aborts an interrupt non-blocking transfer early.

---

**Note:** This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

---

**Parameters**

- `base` – I2C base pointer.
- `handle` – pointer to `i2c_master_handle_t` structure which stores the transfer state

**Return values**

- `kStatus_I2C_Timeout` – Timeout during polling flag.
- `kStatus_Success` – Successfully abort the transfer.

`void I2C_MasterTransferHandleIRQ(I2C_Type *base, void *i2cHandle)`

Master interrupt handler.

**Parameters**

- `base` – I2C base pointer.
- `i2cHandle` – pointer to `i2c_master_handle_t` structure.

`void I2C_SlaveTransferCreateHandle(I2C_Type *base, i2c_slave_handle_t *handle, i2c_slave_transfer_callback_t callback, void *userData)`

Initializes the I2C handle which is used in transactional functions.

**Parameters**

- `base` – I2C base pointer.
- `handle` – pointer to `i2c_slave_handle_t` structure to store the transfer state.
- `callback` – pointer to user callback function.
- `userData` – user parameter passed to the callback function.

`status_t I2C_SlaveTransferNonBlocking(I2C_Type *base, i2c_slave_handle_t *handle, uint32_t eventMask)`

Starts accepting slave transfers.

Call this API after calling the `I2C_SlaveInit()` and `I2C_SlaveTransferCreateHandle()` to start processing transactions driven by an I2C master. The slave monitors the I2C bus and passes events to the callback that was passed into the call to `I2C_SlaveTransferCreateHandle()`. The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the `eventMask` parameter to the OR'd combination of `i2c_slave_transfer_event_t` enumerators for the events you wish to receive. The `kI2C_SlaveTransmitEvent` and `kLPI2C_SlaveReceiveEvent` events are always enabled and do not need to be included in the mask. Alternatively, pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the `kI2C_SlaveAllEvents` constant is provided as a convenient way to enable all events.

**Parameters**

- `base` – The I2C peripheral base address.
- `handle` – Pointer to `i2c_slave_handle_t` structure which stores the transfer state.
- `eventMask` – Bit mask formed by OR'ing together `i2c_slave_transfer_event_t` enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and `kI2C_SlaveAllEvents` to enable all events.

#### Return values

- `kStatus_Success` – Slave transfers were successfully started.
- `kStatus_I2C_Busy` – Slave transfers have already been started on this handle.

`void I2C_SlaveTransferAbort(I2C_Type *base, i2c_slave_handle_t *handle)`  
Aborts the slave transfer.

---

**Note:** This API can be called at any time to stop slave for handling the bus events.

---

#### Parameters

- `base` – I2C base pointer.
- `handle` – pointer to `i2c_slave_handle_t` structure which stores the transfer state.

`status_t I2C_SlaveTransferGetCount(I2C_Type *base, i2c_slave_handle_t *handle, size_t *count)`  
Gets the slave transfer remaining bytes during a interrupt non-blocking transfer.

#### Parameters

- `base` – I2C base pointer.
- `handle` – pointer to `i2c_slave_handle_t` structure.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

#### Return values

- `kStatus_InvalidArgument` – `count` is Invalid.
- `kStatus_Success` – Successfully return the count.

`void I2C_SlaveTransferHandleIRQ(I2C_Type *base, void *i2cHandle)`  
Slave interrupt handler.

#### Parameters

- `base` – I2C base pointer.
- `i2cHandle` – pointer to `i2c_slave_handle_t` structure which stores the transfer state

`FSL_I2C_DRIVER_VERSION`  
I2C driver version.

I2C status return codes.

*Values:*

enumerator `kStatus_I2C_Busy`  
I2C is busy with current transfer.

enumerator kStatus\_I2C\_Idle  
 Bus is Idle.

enumerator kStatus\_I2C\_Nak  
 NAK received during transfer.

enumerator kStatus\_I2C\_ArbitrationLost  
 Arbitration lost during transfer.

enumerator kStatus\_I2C\_Timeout  
 Timeout polling status flags.

enumerator kStatus\_I2C\_Addr\_Nak  
 NAK received during the address probe.

enum \_i2c\_flags  
 I2C peripheral flags.

---

**Note:** These enumerations are meant to be OR'd together to form a bit mask.

---

*Values:*

enumerator kI2C\_ReceiveNakFlag  
 I2C receive NAK flag.

enumerator kI2C\_IntPendingFlag  
 I2C interrupt pending flag. This flag can be cleared.

enumerator kI2C\_TransferDirectionFlag  
 I2C transfer direction flag.

enumerator kI2C\_RangeAddressMatchFlag  
 I2C range address match flag.

enumerator kI2C\_ArbitrationLostFlag  
 I2C arbitration lost flag. This flag can be cleared.

enumerator kI2C\_BusBusyFlag  
 I2C bus busy flag.

enumerator kI2C\_AddressMatchFlag  
 I2C address match flag.

enumerator kI2C\_TransferCompleteFlag  
 I2C transfer complete flag.

enumerator kI2C\_StopDetectFlag  
 I2C stop detect flag. This flag can be cleared.

enumerator kI2C\_StartDetectFlag  
 I2C start detect flag. This flag can be cleared.

enum \_i2c\_interrupt\_enable  
 I2C feature interrupt source.

*Values:*

enumerator kI2C\_GlobalInterruptEnable  
 I2C global interrupt.

enumerator kI2C\_StopDetectInterruptEnable  
 I2C stop detect interrupt.

enumerator kI2C\_StartStopDetectInterruptEnable  
I2C start&stop detect interrupt.

enum \_i2c\_direction  
The direction of master and slave transfers.

*Values:*

enumerator kI2C\_Write  
Master transmits to the slave.

enumerator kI2C\_Read  
Master receives from the slave.

enum \_i2c\_slave\_address\_mode  
Addressing mode.

*Values:*

enumerator kI2C\_Address7bit  
7-bit addressing mode.

enumerator kI2C\_RangeMatch  
Range address match addressing mode.

enum \_i2c\_master\_transfer\_flags  
I2C transfer control flag.

*Values:*

enumerator kI2C\_TransferDefaultFlag  
A transfer starts with a start signal, stops with a stop signal.

enumerator kI2C\_TransferNoStartFlag  
A transfer starts without a start signal, only support write only or write+read with no start flag, do not support read only with no start flag.

enumerator kI2C\_TransferRepeatedStartFlag  
A transfer starts with a repeated start signal.

enumerator kI2C\_TransferNoStopFlag  
A transfer ends without a stop signal.

enum \_i2c\_slave\_transfer\_event  
Set of events sent to the callback for nonblocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to I2C\_SlaveTransferNonBlocking() to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

---

**Note:** These enumerations are meant to be OR'd together to form a bit mask of events.

---

*Values:*

enumerator kI2C\_SlaveAddressMatchEvent  
Received the slave address after a start or repeated start.

enumerator kI2C\_SlaveTransmitEvent  
A callback is requested to provide data to transmit (slave-transmitter role).

enumerator `kI2C_SlaveReceiveEvent`

A callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator `kI2C_SlaveTransmitAckEvent`

A callback needs to either transmit an ACK or NACK.

enumerator `kI2C_SlaveStartEvent`

A start/repeated start was detected.

enumerator `kI2C_SlaveCompletionEvent`

A stop was detected or finished transfer, completing the transfer.

enumerator `kI2C_SlaveGeneralCallEvent`

Received the general call address after a start or repeated start.

enumerator `kI2C_SlaveAllEvents`

A bit mask of all available events.

Common sets of flags used by the driver.

*Values:*

enumerator `kClearFlags`

All flags which are cleared by the driver upon starting a transfer.

enumerator `kIrqFlags`

typedef enum `_i2c_direction` `i2c_direction_t`

The direction of master and slave transfers.

typedef enum `_i2c_slave_address_mode` `i2c_slave_address_mode_t`

Addressing mode.

typedef enum `_i2c_slave_transfer_event` `i2c_slave_transfer_event_t`

Set of events sent to the callback for nonblocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to `I2C_SlaveTransferNonBlocking()` to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

---

**Note:** These enumerations are meant to be OR'd together to form a bit mask of events.

---

typedef struct `_i2c_master_config` `i2c_master_config_t`

I2C master user configuration.

typedef struct `_i2c_slave_config` `i2c_slave_config_t`

I2C slave user configuration.

typedef struct `_i2c_master_handle` `i2c_master_handle_t`

I2C master handle typedef.

typedef void (`*i2c_master_transfer_callback_t`)(`I2C_Type *base`, `i2c_master_handle_t *handle`, `status_t status`, void `*userData`)

I2C master transfer callback typedef.

typedef struct `_i2c_slave_handle` `i2c_slave_handle_t`

I2C slave handle typedef.

`typedef struct _i2c_master_transfer i2c_master_transfer_t`  
I2C master transfer structure.

`typedef struct _i2c_slave_transfer i2c_slave_transfer_t`  
I2C slave transfer structure.

`typedef void (*i2c_slave_transfer_callback_t)(I2C_Type *base, i2c_slave_transfer_t *xfer, void *userData)`

I2C slave transfer callback typedef.

`I2C_RETRY_TIMES`

Retry times for waiting flag.

`I2C_MASTER_FACK_CONTROL`

Master Fast ack control, control if master needs to manually write ack, this is used to low the speed of transfer for SoCs with feature `FSL_FEATURE_I2C_HAS_DOUBLE_BUFFERING`.

`I2C_HAS_STOP_DETECT`

`struct i2c_master_config`

`#include <fsl_i2c.h>` I2C master user configuration.

### Public Members

`bool enableMaster`

Enables the I2C peripheral at initialization time.

`bool enableStopHold`

Controls the stop hold enable.

`bool enableDoubleBuffering`

Controls double buffer enable; notice that enabling the double buffer disables the clock stretch.

`uint32_t baudRate_Bps`

Baud rate configuration of I2C peripheral.

`uint8_t glitchFilterWidth`

Controls the width of the glitch.

`struct i2c_slave_config`

`#include <fsl_i2c.h>` I2C slave user configuration.

### Public Members

`bool enableSlave`

Enables the I2C peripheral at initialization time.

`bool enableGeneralCall`

Enables the general call addressing mode.

`bool enableWakeUp`

Enables/disables waking up MCU from low-power mode.

`bool enableDoubleBuffering`

Controls a double buffer enable; notice that enabling the double buffer disables the clock stretch.

`bool enableBaudRateCtl`

Enables/disables independent slave baud rate on SCL in very fast I2C modes.

uint16\_t slaveAddress

A slave address configuration.

uint16\_t upperAddress

A maximum boundary slave address used in a range matching mode.

*i2c\_slave\_address\_mode\_t* addressingMode

An addressing mode configuration of *i2c\_slave\_address\_mode\_config\_t*.

uint32\_t sclStopHoldTime\_ns

the delay from the rising edge of SCL (I2C clock) to the rising edge of SDA (I2C data) while SCL is high (stop condition), SDA hold time and SCL start hold time are also configured according to the SCL stop hold time.

struct *\_i2c\_master\_transfer*

*#include <fsl\_i2c.h>* I2C master transfer structure.

### Public Members

uint32\_t flags

A transfer flag which controls the transfer.

uint8\_t slaveAddress

7-bit slave address.

*i2c\_direction\_t* direction

A transfer direction, read or write.

uint32\_t subaddress

A sub address. Transferred MSB first.

uint8\_t subaddressSize

A size of the command buffer.

uint8\_t \*volatile data

A transfer buffer.

volatile size\_t dataSize

A transfer size.

struct *\_i2c\_master\_handle*

*#include <fsl\_i2c.h>* I2C master handle structure.

### Public Members

*i2c\_master\_transfer\_t* transfer

I2C master transfer copy.

size\_t transferSize

Total bytes to be transferred.

uint8\_t state

A transfer state maintained during transfer.

*i2c\_master\_transfer\_callback\_t* completionCallback

A callback function called when the transfer is finished.

void \*userData

A callback parameter passed to the callback function.

struct *\_i2c\_slave\_transfer*

*#include <fsl\_i2c.h>* I2C slave transfer structure.

### Public Members

*i2c\_slave\_transfer\_event\_t* event

A reason that the callback is invoked.

uint8\_t \*volatile data

A transfer buffer.

volatile size\_t dataSize

A transfer size.

*status\_t* completionStatus

Success or error code describing how the transfer completed. Only applies for I2C\_SlaveCompletionEvent.

size\_t transferredCount

A number of bytes actually transferred since the start or since the last repeated start.

struct *i2c\_slave\_handle*

*#include <fsl\_i2c.h>* I2C slave handle structure.

### Public Members

volatile bool isBusy

Indicates whether a transfer is busy.

*i2c\_slave\_transfer\_t* transfer

I2C slave transfer copy.

uint32\_t eventMask

A mask of enabled events.

*i2c\_slave\_transfer\_callback\_t* callback

A callback function called at the transfer event.

void \*userData

A callback parameter passed to the callback.

## 2.18 Common Driver

FSL\_COMMON\_DRIVER\_VERSION

common driver version.

DEBUG\_CONSOLE\_DEVICE\_TYPE\_NONE

No debug console.

DEBUG\_CONSOLE\_DEVICE\_TYPE\_UART

Debug console based on UART.

DEBUG\_CONSOLE\_DEVICE\_TYPE\_LPUART

Debug console based on LPUART.

DEBUG\_CONSOLE\_DEVICE\_TYPE\_LPSCI

Debug console based on LPSCI.

DEBUG\_CONSOLE\_DEVICE\_TYPE\_USBCDC

Debug console based on USBCDC.

DEBUG\_CONSOLE\_DEVICE\_TYPE\_FLEXCOMM

Debug console based on FLEXCOMM.

DEBUG\_CONSOLE\_DEVICE\_TYPE\_IUART

Debug console based on i.MX UART.

DEBUG\_CONSOLE\_DEVICE\_TYPE\_VUSART

Debug console based on LPC\_VUSART.

DEBUG\_CONSOLE\_DEVICE\_TYPE\_MINI\_USART

Debug console based on LPC\_USART.

DEBUG\_CONSOLE\_DEVICE\_TYPE\_SWO

Debug console based on SWO.

DEBUG\_CONSOLE\_DEVICE\_TYPE\_QSCI

Debug console based on QSCI.

MIN(a, b)

Computes the minimum of *a* and *b*.

MAX(a, b)

Computes the maximum of *a* and *b*.

UINT16\_MAX

Max value of uint16\_t type.

UINT32\_MAX

Max value of uint32\_t type.

SDK\_ATOMIC\_LOCAL\_ADD(addr, val)

Add value *val* from the variable at address *address*.

SDK\_ATOMIC\_LOCAL\_SUB(addr, val)

Subtract value *val* to the variable at address *address*.

SDK\_ATOMIC\_LOCAL\_SET(addr, bits)

Set the bits specified by *bits* to the variable at address *address*.

SDK\_ATOMIC\_LOCAL\_CLEAR(addr, bits)

Clear the bits specified by *bits* to the variable at address *address*.

SDK\_ATOMIC\_LOCAL\_TOGGLE(addr, bits)

Toggle the bits specified by *bits* to the variable at address *address*.

SDK\_ATOMIC\_LOCAL\_CLEAR\_AND\_SET(addr, clearBits, setBits)

For the variable at address *address*, clear the bits specified by *clearBits* and set the bits specified by *setBits*.

SDK\_ATOMIC\_LOCAL\_COMPARE\_AND\_SET(addr, expected, newValue)

For the variable at address *address*, check whether the value equal to *expected*. If value same as *expected* then update *newValue* to address and return **true** , else return **false** .

SDK\_ATOMIC\_LOCAL\_TEST\_AND\_SET(addr, newValue)

For the variable at address *address*, set as *newValue* value and return old value.

USEC\_TO\_COUNT(us, clockFreqInHz)

Macro to convert a microsecond period to raw count value

COUNT\_TO\_USEC(count, clockFreqInHz)

Macro to convert a raw count value to microsecond

MSEC\_TO\_COUNT(ms, clockFreqInHz)

Macro to convert a millisecond period to raw count value

COUNT\_TO\_MSEC(count, clockFreqInHz)

Macro to convert a raw count value to millisecond

SDK\_ISR\_EXIT\_BARRIER

SDK\_ALIGN(var, alignbytes)

Macro to define a variable with alignbytes alignment

SDK\_SIZEALIGN(var, alignbytes)

Macro to define a variable with L1 d-cache line size alignment

Macro to define a variable with L2 cache line size alignment

Macro to change a value to a given size aligned value

AT\_NONCACHEABLE\_SECTION(var)

Define a variable *var*, and place it in non-cacheable section.

AT\_NONCACHEABLE\_SECTION\_ALIGN(var, alignbytes)

Define a variable *var*, and place it in non-cacheable section, the start address of the variable is aligned to *alignbytes*.

AT\_NONCACHEABLE\_SECTION\_INIT(var)

Define a variable *var* with initial value, and place it in non-cacheable section.

AT\_NONCACHEABLE\_SECTION\_ALIGN\_INIT(var, alignbytes)

Define a variable *var* with initial value, and place it in non-cacheable section, the start address of the variable is aligned to *alignbytes*.

AT\_CACHE\_LINE\_SECTION(var)

Define a variable *var*, which is cache line size aligned and be placed in CacheLineData section.

AT\_CACHE\_LINE\_SECTION\_INIT(var)

Define a variable *var* with initial value, which is cache line size aligned and be placed in CacheLineData.init section.

AT\_QUICKACCESS\_SECTION\_CODE(func)

Place function in a section which can be accessed quickly by core.

AT\_QUICKACCESS\_SECTION\_DATA(var)

Place data in a section which can be accessed quickly by core.

AT\_QUICKACCESS\_SECTION\_DATA\_ALIGN(var, alignbytes)

Place data in a section which can be accessed quickly by core, and the variable address is set to align with *alignbytes*.

RAMFUNCTION\_SECTION\_CODE(func)

Place function in ram.

enum \_status\_groups

Status group numbers.

*Values:*

enumerator kStatusGroup\_Generic

Group number for generic status codes.

enumerator kStatusGroup\_FLASH

Group number for FLASH status codes.

enumerator kStatusGroup\_LPSPi  
Group number for LPSPi status codes.

enumerator kStatusGroup\_FLEXIO\_SPI  
Group number for FLEXIO SPI status codes.

enumerator kStatusGroup\_DSPI  
Group number for DSPI status codes.

enumerator kStatusGroup\_FLEXIO\_UART  
Group number for FLEXIO UART status codes.

enumerator kStatusGroup\_FLEXIO\_I2C  
Group number for FLEXIO I2C status codes.

enumerator kStatusGroup\_LPI2C  
Group number for LPI2C status codes.

enumerator kStatusGroup\_UART  
Group number for UART status codes.

enumerator kStatusGroup\_I2C  
Group number for I2C status codes.

enumerator kStatusGroup\_LPSCI  
Group number for LPSCI status codes.

enumerator kStatusGroup\_LPUART  
Group number for LPUART status codes.

enumerator kStatusGroup\_SPI  
Group number for SPI status code.

enumerator kStatusGroup\_XRDC  
Group number for XRDC status code.

enumerator kStatusGroup\_SEMA42  
Group number for SEMA42 status code.

enumerator kStatusGroup\_SDHC  
Group number for SDHC status code

enumerator kStatusGroup\_SDMMC  
Group number for SDMMC status code

enumerator kStatusGroup\_SAI  
Group number for SAI status code

enumerator kStatusGroup\_MCG  
Group number for MCG status codes.

enumerator kStatusGroup\_SCG  
Group number for SCG status codes.

enumerator kStatusGroup\_SDSPI  
Group number for SDSPI status codes.

enumerator kStatusGroup\_FLEXIO\_I2S  
Group number for FLEXIO I2S status codes

enumerator kStatusGroup\_FLEXIO\_MCULCD  
Group number for FLEXIO LCD status codes

enumerator kStatusGroup\_FLASHIAP  
Group number for FLASHIAP status codes

enumerator kStatusGroup\_FLEXCOMM\_I2C  
Group number for FLEXCOMM I2C status codes

enumerator kStatusGroup\_I2S  
Group number for I2S status codes

enumerator kStatusGroup\_IUART  
Group number for IUART status codes

enumerator kStatusGroup\_CSI  
Group number for CSI status codes

enumerator kStatusGroup\_MIPI\_DSI  
Group number for MIPI DSI status codes

enumerator kStatusGroup\_SDRAMC  
Group number for SDRAMC status codes.

enumerator kStatusGroup\_POWER  
Group number for POWER status codes.

enumerator kStatusGroup\_ENET  
Group number for ENET status codes.

enumerator kStatusGroup\_PHY  
Group number for PHY status codes.

enumerator kStatusGroup\_TRGMUX  
Group number for TRGMUX status codes.

enumerator kStatusGroup\_SMARTCARD  
Group number for SMARTCARD status codes.

enumerator kStatusGroup\_LMEM  
Group number for LMEM status codes.

enumerator kStatusGroup\_QSPI  
Group number for QSPI status codes.

enumerator kStatusGroup\_DMA  
Group number for DMA status codes.

enumerator kStatusGroup\_EDMA  
Group number for EDMA status codes.

enumerator kStatusGroup\_DMAMGR  
Group number for DMAMGR status codes.

enumerator kStatusGroup\_FLEXCAN  
Group number for FlexCAN status codes.

enumerator kStatusGroup\_LTC  
Group number for LTC status codes.

enumerator kStatusGroup\_FLEXIO\_CAMERA  
Group number for FLEXIO CAMERA status codes.

enumerator kStatusGroup\_LPC\_SPI  
Group number for LPC\_SPI status codes.

enumerator kStatusGroup\_LPC\_USART  
Group number for LPC\_USART status codes.

enumerator kStatusGroup\_DMIC  
Group number for DMIC status codes.

enumerator kStatusGroup\_SDIF  
Group number for SDIF status codes.

enumerator kStatusGroup\_SPIFI  
Group number for SPIFI status codes.

enumerator kStatusGroup\_OTP  
Group number for OTP status codes.

enumerator kStatusGroup\_MCAN  
Group number for MCAN status codes.

enumerator kStatusGroup\_CAAM  
Group number for CAAM status codes.

enumerator kStatusGroup\_ECSPi  
Group number for ECSPi status codes.

enumerator kStatusGroup\_USDHC  
Group number for USDHC status codes.

enumerator kStatusGroup\_LPC\_I2C  
Group number for LPC\_I2C status codes.

enumerator kStatusGroup\_DCP  
Group number for DCP status codes.

enumerator kStatusGroup\_MSCAN  
Group number for MSCAN status codes.

enumerator kStatusGroup\_ESAI  
Group number for ESAI status codes.

enumerator kStatusGroup\_FLEXSPI  
Group number for FLEXSPI status codes.

enumerator kStatusGroup\_MMDC  
Group number for MMDC status codes.

enumerator kStatusGroup\_PDM  
Group number for MIC status codes.

enumerator kStatusGroup\_SDMA  
Group number for SDMA status codes.

enumerator kStatusGroup\_ICS  
Group number for ICS status codes.

enumerator kStatusGroup\_SPDIF  
Group number for SPDIF status codes.

enumerator kStatusGroup\_LPC\_MINISPI  
Group number for LPC\_MINISPI status codes.

enumerator kStatusGroup\_HASHCRYPT  
Group number for Hashcrypt status codes

enumerator `kStatusGroup_LPC_SPI_SSP`  
Group number for LPC\_SPI\_SSP status codes.

enumerator `kStatusGroup_I3C`  
Group number for I3C status codes

enumerator `kStatusGroup_LPC_I2C_1`  
Group number for LPC\_I2C\_1 status codes.

enumerator `kStatusGroup_NOTIFIER`  
Group number for NOTIFIER status codes.

enumerator `kStatusGroup_DebugConsole`  
Group number for debug console status codes.

enumerator `kStatusGroup_SEMC`  
Group number for SEMC status codes.

enumerator `kStatusGroup_ApplicationRangeStart`  
Starting number for application groups.

enumerator `kStatusGroup_IAP`  
Group number for IAP status codes

enumerator `kStatusGroup_SFA`  
Group number for SFA status codes

enumerator `kStatusGroup_SPC`  
Group number for SPC status codes.

enumerator `kStatusGroup_PUF`  
Group number for PUF status codes.

enumerator `kStatusGroup_TOUCH_PANEL`  
Group number for touch panel status codes

enumerator `kStatusGroup_VBAT`  
Group number for VBAT status codes

enumerator `kStatusGroup_XSPI`  
Group number for XSPI status codes

enumerator `kStatusGroup_PNGDEC`  
Group number for PNGDEC status codes

enumerator `kStatusGroup_JPEGDEC`  
Group number for JPEGDEC status codes

enumerator `kStatusGroup_AUDMIX`  
Group number for AUDMIX status codes

enumerator `kStatusGroup_HAL_GPIO`  
Group number for HAL GPIO status codes.

enumerator `kStatusGroup_HAL_UART`  
Group number for HAL UART status codes.

enumerator `kStatusGroup_HAL_TIMER`  
Group number for HAL TIMER status codes.

enumerator `kStatusGroup_HAL_SPI`  
Group number for HAL SPI status codes.

- enumerator `kStatusGroup_HAL_I2C`  
Group number for HAL I2C status codes.
- enumerator `kStatusGroup_HAL_FLASH`  
Group number for HAL FLASH status codes.
- enumerator `kStatusGroup_HAL_PWM`  
Group number for HAL PWM status codes.
- enumerator `kStatusGroup_HAL_RNG`  
Group number for HAL RNG status codes.
- enumerator `kStatusGroup_HAL_I2S`  
Group number for HAL I2S status codes.
- enumerator `kStatusGroup_HAL_ADC_SENSOR`  
Group number for HAL ADC SENSOR status codes.
- enumerator `kStatusGroup_TIMERMANAGER`  
Group number for TiMER MANAGER status codes.
- enumerator `kStatusGroup_SERIALMANAGER`  
Group number for SERIAL MANAGER status codes.
- enumerator `kStatusGroup_LED`  
Group number for LED status codes.
- enumerator `kStatusGroup_BUTTON`  
Group number for BUTTON status codes.
- enumerator `kStatusGroup_EXTERN_EEPROM`  
Group number for EXTERN EEPROM status codes.
- enumerator `kStatusGroup_SHELL`  
Group number for SHELL status codes.
- enumerator `kStatusGroup_MEM_MANAGER`  
Group number for MEM MANAGER status codes.
- enumerator `kStatusGroup_LIST`  
Group number for List status codes.
- enumerator `kStatusGroup_OSA`  
Group number for OSA status codes.
- enumerator `kStatusGroup_COMMON_TASK`  
Group number for Common task status codes.
- enumerator `kStatusGroup_MSG`  
Group number for messaging status codes.
- enumerator `kStatusGroup_SDK_OCOTP`  
Group number for OCOTP status codes.
- enumerator `kStatusGroup_SDK_FLEXSPINOR`  
Group number for FLEXSPINOR status codes.
- enumerator `kStatusGroup_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_QUEUEDSPI`  
Group number for QSPI status codes.
- enumerator `kStatusGroup_POWER_MANAGER`  
Group number for POWER\_MANAGER status codes.
- enumerator `kStatusGroup_IPED`  
Group number for IPED status codes.
- enumerator `kStatusGroup_ELS_PKC`  
Group number for ELS PKC status codes.
- enumerator `kStatusGroup_CSS_PKC`  
Group number for CSS PKC status codes.
- enumerator `kStatusGroup_HOSTIF`  
Group number for HOSTIF status codes.
- enumerator `kStatusGroup_CLIF`  
Group number for CLIF status codes.
- enumerator `kStatusGroup_BMA`  
Group number for BMA status codes.
- enumerator `kStatusGroup_NETC`  
Group number for NETC status codes.
- enumerator `kStatusGroup_ELE`  
Group number for ELE status codes.
- enumerator `kStatusGroup_GLIKEY`  
Group number for GLIKEY status codes.
- enumerator `kStatusGroup_AON_POWER`  
Group number for AON\_POWER status codes.
- enumerator `kStatusGroup_AON_COMMON`  
Group number for AON\_COMMON status codes.

enumerator `kStatusGroup_ENDAT3`  
 Group number for ENDAT3 status codes.

enumerator `kStatusGroup_HIPERFACE`  
 Group number for HIPERFACE status codes.

enumerator `kStatusGroup_NPX`  
 Group number for NPX status codes.

enumerator `kStatusGroup_ELA_CSEC`  
 Group number for ELA\_CSEC status codes.

enumerator `kStatusGroup_FLEXIO_T_FORMAT`  
 Group number for T-format status codes.

enumerator `kStatusGroup_FLEXIO_A_FORMAT`  
 Group number for A-format status codes.

Generic status return codes.

*Values:*

enumerator `kStatus_Success`  
 Generic status for Success.

enumerator `kStatus_Fail`  
 Generic status for Fail.

enumerator `kStatus_ReadOnly`  
 Generic status for read only failure.

enumerator `kStatus_OutOfRange`  
 Generic status for out of range access.

enumerator `kStatus_InvalidArgument`  
 Generic status for invalid argument check.

enumerator `kStatus_Timeout`  
 Generic status for timeout.

enumerator `kStatus_NoTransferInProgress`  
 Generic status for no transfer in progress.

enumerator `kStatus_Busy`  
 Generic status for module is busy.

enumerator `kStatus_NoData`  
 Generic status for no data is found for the operation.

`typedef int32_t status_t`  
 Type used for all status and error return values.

`void *SDK_Malloc(size_t size, size_t alignbytes)`  
 Allocate memory with given alignment and aligned size.

This is provided to support the dynamically allocated memory used in cache-able region.

#### **Parameters**

- `size` – The length required to malloc.
- `alignbytes` – The alignment size.

#### **Return values**

The – allocated memory.

```
void SDK_Free(void *ptr)
```

Free memory.

#### Parameters

- ptr – The memory to be release.

```
void SDK_DelayAtLeastUs(uint32_t delayTime_us, uint32_t coreClock_Hz)
```

Delay at least for some time. Please note that, this API uses while loop for delay, different run-time environments make the time not precise, if precise delay count was needed, please implement a new delay function with hardware timer.

#### Parameters

- delayTime\_us – Delay time in unit of microsecond.
- coreClock\_Hz – Core clock frequency with Hz.

```
static inline status_t EnableIRQ(IRQn_Type interrupt)
```

Enable specific interrupt.

Enable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only enables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL\_FEATURE\_NUMBER\_OF\_LEVEL1\_INT\_VECTORS.

#### Parameters

- interrupt – The IRQ number.

#### Return values

- kStatus\_Success – Interrupt enabled successfully
- kStatus\_Fail – Failed to enable the interrupt

```
static inline status_t DisableIRQ(IRQn_Type interrupt)
```

Disable specific interrupt.

Disable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only disables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL\_FEATURE\_NUMBER\_OF\_LEVEL1\_INT\_VECTORS.

#### Parameters

- interrupt – The IRQ number.

#### Return values

- kStatus\_Success – Interrupt disabled successfully
- kStatus\_Fail – Failed to disable the interrupt

```
static inline status_t EnableIRQWithPriority(IRQn_Type interrupt, uint8_t priNum)
```

Enable the IRQ, and also set the interrupt priority.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL\_FEATURE\_NUMBER\_OF\_LEVEL1\_INT\_VECTORS.

**Parameters**

- interrupt – The IRQ to Enable.
- priNum – Priority number set to interrupt controller register.

**Return values**

- kStatus\_Success – Interrupt priority set successfully
- kStatus\_Fail – Failed to set the interrupt priority.

```
static inline status_t IRQ_SetPriority(IRQn_Type interrupt, uint8_t priNum)
```

Set the IRQ priority.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL\_FEATURE\_NUMBER\_OF\_LEVEL1\_INT\_VECTORS.

**Parameters**

- interrupt – The IRQ to set.
- priNum – Priority number set to interrupt controller register.

**Return values**

- kStatus\_Success – Interrupt priority set successfully
- kStatus\_Fail – Failed to set the interrupt priority.

```
static inline status_t IRQ_ClearPendingIRQ(IRQn_Type interrupt)
```

Clear the pending IRQ flag.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL\_FEATURE\_NUMBER\_OF\_LEVEL1\_INT\_VECTORS.

**Parameters**

- interrupt – The flag which IRQ to clear.

**Return values**

- kStatus\_Success – Interrupt priority set successfully
- kStatus\_Fail – Failed to set the interrupt priority.

```
static inline uint32_t DisableGlobalIRQ(void)
```

Disable the global IRQ.

Disable the global interrupt and return the current primask register. User is required to provided the primask register for the EnableGlobalIRQ().

**Returns**

Current primask value.

```
static inline void EnableGlobalIRQ(uint32_t primask)
```

Enable the global IRQ.

Set the primask register with the provided primask value but not just enable the primask. The idea is for the convenience of integration of RTOS. some RTOS get its own management

mechanism of primask. User is required to use the EnableGlobalIRQ() and DisableGlobalIRQ() in pair.

**Parameters**

- primask – value of primask register to be restored. The primask value is supposed to be provided by the DisableGlobalIRQ().

```
static inline bool _SDK_AtomicLocalCompareAndSet(uint32_t *addr, uint32_t expected, uint32_t
newValue)
```

```
static inline uint32_t _SDK_AtomicTestAndSet(uint32_t *addr, uint32_t newValue)
```

FSL\_DRIVER\_TRANSFER\_DOUBLE\_WEAK\_IRQ

Macro to use the default weak IRQ handler in drivers.

MAKE\_STATUS(group, code)

Construct a status code value from a group and code number.

MAKE\_VERSION(major, minor, bugfix)

Construct the version number for drivers.

The driver version is a 32-bit number, for both 32-bit platforms(such as Cortex M) and 16-bit platforms(such as DSC).

|          |               |               |         |
|----------|---------------|---------------|---------|
| Unused   | Major Version | Minor Version | Bug Fix |
| 31 25 24 | 17 16         | 9 8           | 0       |

ARRAY\_SIZE(x)

Computes the number of elements in an array.

UINT64\_H(X)

Macro to get upper 32 bits of a 64-bit value

UINT64\_L(X)

Macro to get lower 32 bits of a 64-bit value

SUPPRESS\_FALL\_THROUGH\_WARNING()

For switch case code block, if case section ends without “break;” statement, there will be fallthrough warning with compiler flag -Wextra or -Wimplicit-fallthrough=n when using armgcc. To suppress this warning, “SUPPRESS\_FALL\_THROUGH\_WARNING();” need to be added at the end of each case section which misses “break;”statement.

MSDK\_REG\_SECURE\_ADDR(x)

Convert the register address to the one used in secure mode.

MSDK\_REG\_NONSECURE\_ADDR(x)

Convert the register address to the one used in non-secure mode.

MSDK\_HAS\_DWT\_CYCCNT

The chip supports DWT CYCCNT or not.

MSDK\_INVALID\_IRQ\_HANDLER

Invalid IRQ handler address.

## 2.19 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

## 2.20 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` `dmass`  
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

## 2.21 LPTMR: Low-Power Timer

void LPTMR\_Init(LPTMR\_Type \*base, const *lptmr\_config\_t* \*config)

Ungates the LPTMR clock and configures the peripheral for a basic operation.

---

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

---

### Parameters

- base – LPTMR peripheral base address
- config – A pointer to the LPTMR configuration structure.

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

Gates the LPTMR clock.

### Parameters

- base – LPTMR peripheral base address

void LPTMR\_GetDefaultConfig(*lptmr\_config\_t* \*config)

Fills in the LPTMR configuration structure with default settings.

The default values are as follows.

```
config->timerMode = kLPTMR_TimerModeTimeCounter;
config->pinSelect = kLPTMR_PinSelectInput_0;
config->pinPolarity = kLPTMR_PinPolarityActiveHigh;
config->enableFreeRunning = false;
config->bypassPrescaler = true;
config->prescalerClockSource = kLPTMR_PrescalerClock_1;
config->value = kLPTMR_Prescale_Glitch_0;
```

### Parameters

- config – A pointer to the LPTMR configuration structure.

static inline void LPTMR\_EnableInterrupts(LPTMR\_Type \*base, uint32\_t mask)

Enables the selected LPTMR interrupts.

### Parameters

- base – LPTMR peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *lptmr\_interrupt\_enable\_t*

static inline void LPTMR\_DisableInterrupts(LPTMR\_Type \*base, uint32\_t mask)

Disables the selected LPTMR interrupts.

**Parameters**

- base – LPTMR peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration `lptmr_interrupt_enable_t`.

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

Gets the enabled LPTMR interrupts.

**Parameters**

- base – LPTMR peripheral base address

**Returns**

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

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

Gets the LPTMR status flags.

**Parameters**

- base – LPTMR peripheral base address

**Returns**

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

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

Clears the LPTMR status flags.

**Parameters**

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

static inline void LPTMR\_SetTimerPeriod(LPTMR\_Type \*base, uint32\_t ticks)

Sets the timer period in units of count.

Timers counts from 0 until it equals the count value set here. The count value is written to the CMR register.

---

**Note:**

- a. The TCF flag is set with the CNR equals the count provided here and then increments.
  - b. Call the utility macros provided in the `fsl_common.h` to convert to ticks.
- 

**Parameters**

- base – LPTMR peripheral base address
- ticks – A timer period in units of ticks

static inline uint32\_t LPTMR\_GetCurrentTimerCount(LPTMR\_Type \*base)

Reads the current timer counting value.

This function returns the real-time timer counting value in a range from 0 to a timer period.

---

**Note:** Call the utility macros provided in the `fsl_common.h` to convert ticks to usec or msec.

---

**Parameters**

- base – LPTMR peripheral base address

**Returns**

The current counter value in ticks

```
static inline void LPTMR_StartTimer(LPTMR_Type *base)
```

Starts the timer.

After calling this function, the timer counts up to the CMR register value. Each time the timer reaches the CMR value and then increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

**Parameters**

- base – LPTMR peripheral base address

```
static inline void LPTMR_StopTimer(LPTMR_Type *base)
```

Stops the timer.

This function stops the timer and resets the timer's counter register.

**Parameters**

- base – LPTMR peripheral base address

```
FSL_LPTMR_DRIVER_VERSION
```

Driver Version

```
enum _lptmr_pin_select
```

LPTMR pin selection used in pulse counter mode.

*Values:*

```
enumerator kLPTMR_PinSelectInput_0
```

Pulse counter input 0 is selected

```
enumerator kLPTMR_PinSelectInput_1
```

Pulse counter input 1 is selected

```
enumerator kLPTMR_PinSelectInput_2
```

Pulse counter input 2 is selected

```
enumerator kLPTMR_PinSelectInput_3
```

Pulse counter input 3 is selected

```
enum _lptmr_pin_polarity
```

LPTMR pin polarity used in pulse counter mode.

*Values:*

```
enumerator kLPTMR_PinPolarityActiveHigh
```

Pulse Counter input source is active-high

```
enumerator kLPTMR_PinPolarityActiveLow
```

Pulse Counter input source is active-low

```
enum _lptmr_timer_mode
```

LPTMR timer mode selection.

*Values:*

```
enumerator kLPTMR_TimerModeTimeCounter
```

Time Counter mode

enumerator kLPTMR\_TimerModePulseCounter  
Pulse Counter mode

enum \_lptmr\_prescaler\_glitch\_value  
LPTMR prescaler/glitch filter values.

*Values:*

enumerator kLPTMR\_Prescale\_Glitch\_0  
Prescaler divide 2, glitch filter does not support this setting

enumerator kLPTMR\_Prescale\_Glitch\_1  
Prescaler divide 4, glitch filter 2

enumerator kLPTMR\_Prescale\_Glitch\_2  
Prescaler divide 8, glitch filter 4

enumerator kLPTMR\_Prescale\_Glitch\_3  
Prescaler divide 16, glitch filter 8

enumerator kLPTMR\_Prescale\_Glitch\_4  
Prescaler divide 32, glitch filter 16

enumerator kLPTMR\_Prescale\_Glitch\_5  
Prescaler divide 64, glitch filter 32

enumerator kLPTMR\_Prescale\_Glitch\_6  
Prescaler divide 128, glitch filter 64

enumerator kLPTMR\_Prescale\_Glitch\_7  
Prescaler divide 256, glitch filter 128

enumerator kLPTMR\_Prescale\_Glitch\_8  
Prescaler divide 512, glitch filter 256

enumerator kLPTMR\_Prescale\_Glitch\_9  
Prescaler divide 1024, glitch filter 512

enumerator kLPTMR\_Prescale\_Glitch\_10  
Prescaler divide 2048 glitch filter 1024

enumerator kLPTMR\_Prescale\_Glitch\_11  
Prescaler divide 4096, glitch filter 2048

enumerator kLPTMR\_Prescale\_Glitch\_12  
Prescaler divide 8192, glitch filter 4096

enumerator kLPTMR\_Prescale\_Glitch\_13  
Prescaler divide 16384, glitch filter 8192

enumerator kLPTMR\_Prescale\_Glitch\_14  
Prescaler divide 32768, glitch filter 16384

enumerator kLPTMR\_Prescale\_Glitch\_15  
Prescaler divide 65536, glitch filter 32768

enum \_lptmr\_prescaler\_clock\_select  
LPTMR prescaler/glitch filter clock select.

---

**Note:** Clock connections are SoC-specific

---

*Values:*

enumerator kLPTMR\_PrescalerClock\_0  
 Prescaler/glitch filter clock 0 selected.

enumerator kLPTMR\_PrescalerClock\_1  
 Prescaler/glitch filter clock 1 selected.

enumerator kLPTMR\_PrescalerClock\_2  
 Prescaler/glitch filter clock 2 selected.

enumerator kLPTMR\_PrescalerClock\_3  
 Prescaler/glitch filter clock 3 selected.

enum \_lptmr\_interrupt\_enable  
 List of the LPTMR interrupts.

*Values:*

enumerator kLPTMR\_TimerInterruptEnable  
 Timer interrupt enable

enum \_lptmr\_status\_flags  
 List of the LPTMR status flags.

*Values:*

enumerator kLPTMR\_TimerCompareFlag  
 Timer compare flag

typedef enum \_lptmr\_pin\_select lptmr\_pin\_select\_t  
 LPTMR pin selection used in pulse counter mode.

typedef enum \_lptmr\_pin\_polarity lptmr\_pin\_polarity\_t  
 LPTMR pin polarity used in pulse counter mode.

typedef enum \_lptmr\_timer\_mode lptmr\_timer\_mode\_t  
 LPTMR timer mode selection.

typedef enum \_lptmr\_prescaler\_glitch\_value lptmr\_prescaler\_glitch\_value\_t  
 LPTMR prescaler/glitch filter values.

typedef enum \_lptmr\_prescaler\_clock\_select lptmr\_prescaler\_clock\_select\_t  
 LPTMR prescaler/glitch filter clock select.

---

**Note:** Clock connections are SoC-specific

---

typedef enum \_lptmr\_interrupt\_enable lptmr\_interrupt\_enable\_t  
 List of the LPTMR interrupts.

typedef enum \_lptmr\_status\_flags lptmr\_status\_flags\_t  
 List of the LPTMR status flags.

typedef struct \_lptmr\_config lptmr\_config\_t  
 LPTMR config structure.

This structure holds the configuration settings for the LPTMR peripheral. To initialize this structure to reasonable defaults, call the LPTMR\_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration struct can be made constant so it resides in flash.

```
static inline void LPTMR_EnableTimerDMA(LPTMR_Type *base, bool enable)
```

Enable or disable timer DMA request.

#### 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

## 2.22 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

### 2.23 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, 2000000U);
```

### Parameters

- base – LPUART peripheral base address.
- config – Pointer to a user-defined configuration structure.
- srcClock\_Hz – LPUART clock source frequency in HZ.

### Return values

- kStatus\_LPUART\_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus\_Success – LPUART initialize succeed

*status\_t* LPUART\_Deinit(LPUART\_Type \*base)

Deinitializes a LPUART instance.

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

### Parameters

- base – LPUART peripheral base address.

### Return values

- kStatus\_Success – Deinit is success.
- kStatus\_LPUART\_Timeout – Timeout during deinit.

void LPUART\_GetDefaultConfig(*lpuart\_config\_t* \*config)

Gets the default configuration structure.

This function initializes the LPUART configuration structure to a default value. The default values are: `lpuartConfig->baudRate_Bps = 115200U`; `lpuartConfig->parityMode = kLPUART_ParityDisabled`; `lpuartConfig->dataBitsCount = kLPUART_EightDataBits`; `lpuartConfig->isMsb = false`; `lpuartConfig->stopBitCount = kLPUART_OneStopBit`; `lpuartConfig->txFifoWatermark = 0`; `lpuartConfig->rxFifoWatermark = 1`; `lpuartConfig->rxIdleType = kLPUART_IdleTypeStartBit`; `lpuartConfig->rxIdleConfig = kLPUART_IdleCharacter1`; `lpuartConfig->enableTx = false`; `lpuartConfig->enableRx = false`;

### Parameters

- config – Pointer to a configuration structure.

```
status_t LPUART_SetBaudRate(LPUART_Type *base, uint32_t baudRate_Bps, uint32_t
                           srcClock_Hz)
```

Sets the LPUART instance baudrate.

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART\_Init.

```
LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
```

#### Parameters

- base – LPUART peripheral base address.
- baudRate\_Bps – LPUART baudrate to be set.
- srcClock\_Hz – LPUART clock source frequency in HZ.

#### Return values

- kStatus\_LPUART\_BaudrateNotSupport – Baudrate is not supported in the current clock source.
- kStatus\_Success – Set baudrate succeeded.

```
void LPUART_Enable9bitMode(LPUART_Type *base, bool enable)
```

Enable 9-bit data mode for LPUART.

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

#### Parameters

- base – LPUART peripheral base address.
- enable – true to enable, false to disable.

```
static inline void LPUART_SetMatchAddress(LPUART_Type *base, uint16_t address1, uint16_t
                                         address2)
```

Set the LPUART address.

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receives with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer; otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

---

**Note:** Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

---

#### Parameters

- base – LPUART peripheral base address.
- address1 – LPUART slave address1.
- address2 – LPUART slave address2.

```
static inline void LPUART_EnableMatchAddress(LPUART_Type *base, bool match1, bool
                                             match2)
```

Enable the LPUART match address feature.

#### Parameters

- base – LPUART peripheral base address.
- match1 – true to enable match address1, false to disable.
- match2 – true to enable match address2, false to disable.

static inline void LPUART\_SetRxFifoWatermark(LPUART\_Type \*base, uint8\_t water)

Sets the rx FIFO watermark.

#### Parameters

- base – LPUART peripheral base address.
- water – Rx FIFO watermark.

static inline void LPUART\_SetTxFifoWatermark(LPUART\_Type \*base, uint8\_t water)

Sets the tx FIFO watermark.

#### Parameters

- base – LPUART peripheral base address.
- water – Tx FIFO watermark.

static inline void LPUART\_TransferEnable16Bit(*lpuart\_handle\_t* \*handle, bool enable)

Sets the LPUART using 16bit transmit, only for 9bit or 10bit mode.

This function Enable 16bit Data transmit in *lpuart\_handle\_t*.

#### Parameters

- handle – LPUART handle pointer.
- enable – true to enable, false to disable.

uint32\_t LPUART\_GetStatusFlags(LPUART\_Type \*base)

Gets LPUART status flags.

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators *\_lpuart\_flags*. To check for a specific status, compare the return value with enumerators in the *\_lpuart\_flags*. For example, to check whether the TX is empty:

```
if (kLPUART_TxDataRegEmptyFlag & LPUART_GetStatusFlags(LPUART1))
{
    ...
}
```

#### Parameters

- base – LPUART peripheral base address.

#### Returns

LPUART status flags which are ORed by the enumerators in the *\_lpuart\_flags*.

*status\_t* LPUART\_ClearStatusFlags(LPUART\_Type \*base, uint32\_t mask)

Clears status flags with a provided mask.

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only be cleared or set by hardware are: *kLPUART\_TxDataRegEmptyFlag*, *kLPUART\_TransmissionCompleteFlag*, *kLPUART\_RxDataRegFullFlag*, *kLPUART\_RxActiveFlag*, *kLPUART\_NoiseErrorFlag*, *kLPUART\_ParityErrorFlag*, *kLPUART\_TxFifoEmptyFlag*, *kLPUART\_RxFifoEmptyFlag* Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

#### Parameters

- base – LPUART peripheral base address.

- mask – the status flags to be cleared. The user can use the enumerators in the `_lpuart_status_flag_t` to do the OR operation and get the mask.

### Return values

- `kStatus_LPUART_FlagCannotClearManually` – The flag can't be cleared by this function but it is cleared automatically by hardware.
- `kStatus_Success` – Status in the mask are cleared.

### Returns

0 succeed, others failed.

```
void LPUART_EnableInterrupts(LPUART_Type *base, uint32_t mask)
```

Enables LPUART interrupts according to a provided mask.

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the `_lpuart_interrupt_enable`. This 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 \_\_unnamed13\_\_

### 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 \_\_unnamed15\_\_

### 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 \_\_unnamed17\_\_

**Public Members**

uint8\_t \*volatile rxData  
Address of remaining data to receive.

uint16\_t \*volatile rxData16  
Address of remaining data to receive.

union \_\_unnamed19\_\_

**Public Members**

uint8\_t \*rxRingBuffer  
Start address of the receiver ring buffer.

uint16\_t \*rxRingBuffer16  
Start address of the receiver ring buffer.

## 2.24 MCM: Miscellaneous Control Module

FSL\_MCM\_DRIVER\_VERSION

MCM driver version.

Enum `_mcm_interrupt_flag`. Interrupt status flag mask. .

*Values:*

enumerator `kMCM_CacheWriteBuffer`  
Cache Write Buffer Error Enable.

enumerator `kMCM_ParityError`  
Cache Parity Error Enable.

enumerator `kMCM_FPUInvalidOperation`  
FPU Invalid Operation Interrupt Enable.

enumerator `kMCM_FPUDivideByZero`  
FPU Divide-by-zero Interrupt Enable.

enumerator `kMCM_FPUOverflow`  
FPU Overflow Interrupt Enable.

enumerator `kMCM_FPUUnderflow`  
FPU Underflow Interrupt Enable.

enumerator `kMCM_FPUInexact`  
FPU Inexact Interrupt Enable.

enumerator `kMCM_FPUInputDenormalInterrupt`  
FPU Input Denormal Interrupt Enable.

typedef union `_mcm_buffer_fault_attribute` `mcm_buffer_fault_attribute_t`  
The union of buffer fault attribute.

typedef union `_mcm_lmem_fault_attribute` `mcm_lmem_fault_attribute_t`  
The union of LMEM fault attribute.

static inline void MCM\_EnableCrossbarRoundRobin(MCM\_Type \*base, bool enable)  
Enables/Disables crossbar round robin.

**Parameters**

- base – MCM peripheral base address.
- enable – Used to enable/disable crossbar round robin.
  - **true** Enable crossbar round robin.
  - **false** disable crossbar round robin.

static inline void MCM\_EnableInterruptStatus(MCM\_Type \*base, uint32\_t mask)  
Enables the interrupt.

**Parameters**

- base – MCM peripheral base address.
- mask – Interrupt status flags mask(`mcm_interrupt_flag`).

static inline void MCM\_DisableInterruptStatus(MCM\_Type \*base, uint32\_t mask)  
Disables the interrupt.

**Parameters**

- base – MCM peripheral base address.
- mask – Interrupt status flags mask(`mcm_interrupt_flag`).

static inline uint16\_t MCM\_GetInterruptStatus(MCM\_Type \*base)  
Gets the Interrupt status .

**Parameters**

- base – MCM peripheral base address.

static inline void MCM\_ClearCacheWriteBufferErroStatus(MCM\_Type \*base)  
Clears the Interrupt status .

**Parameters**

- base – MCM peripheral base address.

static inline uint32\_t MCM\_GetBufferFaultAddress(MCM\_Type \*base)  
Gets buffer fault address.

**Parameters**

- base – MCM peripheral base address.

static inline void MCM\_GetBufferFaultAttribute(MCM\_Type \*base, *mcm\_buffer\_fault\_attribute\_t* \*bufferfault)

Gets buffer fault attributes.

**Parameters**

- base – MCM peripheral base address.

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.

## 2.25 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.

## 2.26 PORT: Port Control and Interrupts

```
static inline void PORT_SetPinConfig(PORT_Type *base, uint32_t pin, const port_pin_config_t
                                     *config)
```

Sets the port PCR register.

This is an example to define an input pin or output pin PCR configuration.

```
// Define a digital input pin PCR configuration
port_pin_config_t config = {
    kPORT_PullUp,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainDisable,
    kPORT_LowDriveStrength,
    kPORT_MuxAsGpio,
    kPORT_UnLockRegister,
};
```

### Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- config – PORT PCR register configuration structure.

```
static inline void PORT_SetMultiplePinsConfig(PORT_Type *base, uint32_t mask, const
                                              port_pin_config_t *config)
```

Sets the port PCR register for multiple pins.

This is an example to define input pins or output pins PCR configuration.

```
Define a digital input pin PCR configuration
port_pin_config_t config = {
    kPORT_PullUp ,
    kPORT_PullEnable,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainDisable,
    kPORT_LowDriveStrength,
    kPORT_MuxAsGpio,
    kPORT_UnlockRegister,
};
```

### Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- config – PORT PCR register configuration structure.

```
static inline void PORT_SetMultipleInterruptPinsConfig(PORT_Type *base, uint32_t mask,
                                                       port_interrupt_t config)
```

Sets the port interrupt configuration in PCR register for multiple pins.

### Parameters

- `base` – PORT peripheral base pointer.
- `mask` – PORT pin number macro.
- `config` – PORT pin interrupt configuration.
  - `kPORT_InterruptOrDMADisabled`: Interrupt/DMA request disabled.
  - `kPORT_DMARisingEdge` : DMA request on rising edge(if the DMA requests exit).
  - `kPORT_DMAFallingEdge`: DMA request on falling edge(if the DMA requests exit).
  - `kPORT_DMAEitherEdge` : DMA request on either edge(if the DMA requests exit).
  - `kPORT_FlagRisingEdge` : Flag sets on rising edge(if the Flag states exit).
  - `kPORT_FlagFallingEdge` : Flag sets on falling edge(if the Flag states exit).
  - `kPORT_FlagEitherEdge` : Flag sets on either edge(if the Flag states exit).
  - `kPORT_InterruptLogicZero` : Interrupt when logic zero.
  - `kPORT_InterruptRisingEdge` : Interrupt on rising edge.
  - `kPORT_InterruptFallingEdge`: Interrupt on falling edge.
  - `kPORT_InterruptEitherEdge` : Interrupt on either edge.
  - `kPORT_InterruptLogicOne` : Interrupt when logic one.
  - `kPORT_ActiveHighTriggerOutputEnable` : Enable active high-trigger output (if the trigger states exit).
  - `kPORT_ActiveLowTriggerOutputEnable` : Enable active low-trigger output (if the trigger states exit)..

```
static inline void PORT_SetPinMux(PORT_Type *base, uint32_t pin, port_mux_t mux)
```

Configures the pin muxing.

---

**Note:** : This function is NOT recommended to use together with the `PORT_SetPinsConfig`, because the `PORT_SetPinsConfig` need to configure the pin mux anyway (Otherwise the pin mux is reset to zero : `kPORT_PinDisabledOrAnalog`). This function is recommended to use to reset the pin mux

---

### Parameters

- `base` – PORT peripheral base pointer.
- `pin` – PORT pin number.
- `mux` – pin muxing slot selection.
  - `kPORT_PinDisabledOrAnalog`: Pin disabled or work in analog function.
  - `kPORT_MuxAsGpio` : Set as GPIO.
  - `kPORT_MuxAlt2` : chip-specific.
  - `kPORT_MuxAlt3` : chip-specific.
  - `kPORT_MuxAlt4` : chip-specific.
  - `kPORT_MuxAlt5` : chip-specific.
  - `kPORT_MuxAlt6` : chip-specific.
  - `kPORT_MuxAlt7` : chip-specific.

```
static inline void PORT_EnablePinsDigitalFilter(PORT_Type *base, uint32_t mask, bool enable)
```

Enables the digital filter in one port, each bit of the 32-bit register represents one pin.

#### Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- enable – PORT digital filter configuration.

```
static inline void PORT_SetDigitalFilterConfig(PORT_Type *base, const
                                             port_digital_filter_config_t *config)
```

Sets the digital filter in one port, each bit of the 32-bit register represents one pin.

#### Parameters

- base – PORT peripheral base pointer.
- config – PORT digital filter configuration structure.

```
static inline void PORT_SetPinInterruptConfig(PORT_Type *base, uint32_t pin, port_interrupt_t
                                             config)
```

Configures the port pin interrupt/DMA request.

#### Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- config – PORT pin interrupt configuration.
  - kPORT\_InterruptOrDMADisabled: Interrupt/DMA request disabled.
  - kPORT\_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).
  - kPORT\_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit).
  - kPORT\_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).
  - kPORT\_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).
  - kPORT\_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).
  - kPORT\_FlagEitherEdge : Flag sets on either edge(if the Flag states exit).
  - kPORT\_InterruptLogicZero : Interrupt when logic zero.
  - kPORT\_InterruptRisingEdge : Interrupt on rising edge.
  - kPORT\_InterruptFallingEdge: Interrupt on falling edge.
  - kPORT\_InterruptEitherEdge : Interrupt on either edge.
  - kPORT\_InterruptLogicOne : Interrupt when logic one.
  - kPORT\_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit).
  - kPORT\_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit).

```
static inline void PORT_SetPinDriveStrength(PORT_Type *base, uint32_t pin, uint8_t strength)
```

Configures the port pin drive strength.

#### Parameters

- base – PORT peripheral base pointer.

- pin – PORT pin number.
- strength – PORT pin drive strength
  - kPORT\_LowDriveStrength = 0U - Low-drive strength is configured.
  - kPORT\_HighDriveStrength = 1U - High-drive strength is configured.

static inline uint32\_t PORT\_GetPinsInterruptFlags(PORT\_Type \*base)

Reads the whole port status flag.

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

#### Parameters

- base – PORT peripheral base pointer.

#### Returns

Current port interrupt status flags, for example, 0x00010001 means the pin 0 and 16 have the interrupt.

static inline void PORT\_ClearPinsInterruptFlags(PORT\_Type \*base, uint32\_t mask)

Clears the multiple pin interrupt status flag.

#### Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.

FSL\_PORT\_DRIVER\_VERSION

PORT driver version.

enum \_port\_pull

Internal resistor pull feature selection.

*Values:*

enumerator kPORT\_PullDisable

Internal pull-up/down resistor is disabled.

enumerator kPORT\_PullDown

Internal pull-down resistor is enabled.

enumerator kPORT\_PullUp

Internal pull-up resistor is enabled.

enum \_port\_slew\_rate

Slew rate selection.

*Values:*

enumerator kPORT\_FastSlewRate

Fast slew rate is configured.

enumerator kPORT\_SlowSlewRate

Slow slew rate is configured.

enum \_port\_open\_drain\_enable

Open Drain feature enable/disable.

*Values:*

enumerator kPORT\_OpenDrainDisable  
Open drain output is disabled.

enumerator kPORT\_OpenDrainEnable  
Open drain output is enabled.

enum \_\_port\_passive\_filter\_enable  
Passive filter feature enable/disable.

*Values:*

enumerator kPORT\_PassiveFilterDisable  
Passive input filter is disabled.

enumerator kPORT\_PassiveFilterEnable  
Passive input filter is enabled.

enum \_\_port\_drive\_strength  
Configures the drive strength.

*Values:*

enumerator kPORT\_LowDriveStrength  
Low-drive strength is configured.

enumerator kPORT\_HighDriveStrength  
High-drive strength is configured.

enum \_\_port\_lock\_register  
Unlock/lock the pin control register field[15:0].

*Values:*

enumerator kPORT\_UnlockRegister  
Pin Control Register fields [15:0] are not locked.

enumerator kPORT\_LockRegister  
Pin Control Register fields [15:0] are locked.

enum \_\_port\_mux  
Pin mux selection.

*Values:*

enumerator kPORT\_PinDisabledOrAnalog  
Corresponding pin is disabled, but is used as an analog pin.

enumerator kPORT\_MuxAsGpio  
Corresponding pin is configured as GPIO.

enumerator kPORT\_MuxAlt0  
Chip-specific

enumerator kPORT\_MuxAlt1  
Chip-specific

enumerator kPORT\_MuxAlt2  
Chip-specific

enumerator kPORT\_MuxAlt3  
Chip-specific

enumerator kPORT\_MuxAlt4  
Chip-specific

enumerator kPORT\_MuxAlt5  
Chip-specific

enumerator kPORT\_MuxAlt6  
Chip-specific

enumerator kPORT\_MuxAlt7  
Chip-specific

enumerator kPORT\_MuxAlt8  
Chip-specific

enumerator kPORT\_MuxAlt9  
Chip-specific

enumerator kPORT\_MuxAlt10  
Chip-specific

enumerator kPORT\_MuxAlt11  
Chip-specific

enumerator kPORT\_MuxAlt12  
Chip-specific

enumerator kPORT\_MuxAlt13  
Chip-specific

enumerator kPORT\_MuxAlt14  
Chip-specific

enumerator kPORT\_MuxAlt15  
Chip-specific

enum \_port\_interrupt  
Configures the interrupt generation condition.

*Values:*

enumerator kPORT\_InterruptOrDMADisabled  
Interrupt/DMA request is disabled.

enumerator kPORT\_DMARisingEdge  
DMA request on rising edge.

enumerator kPORT\_DMAFallingEdge  
DMA request on falling edge.

enumerator kPORT\_DMAEitherEdge  
DMA request on either edge.

enumerator kPORT\_FlagRisingEdge  
Flag sets on rising edge.

enumerator kPORT\_FlagFallingEdge  
Flag sets on falling edge.

enumerator kPORT\_FlagEitherEdge  
Flag sets on either edge.

enumerator kPORT\_InterruptLogicZero  
Interrupt when logic zero.

enumerator kPORT\_InterruptRisingEdge

Interrupt on rising edge.

enumerator kPORT\_InterruptFallingEdge

Interrupt on falling edge.

enumerator kPORT\_InterruptEitherEdge

Interrupt on either edge.

enumerator kPORT\_InterruptLogicOne

Interrupt when logic one.

enumerator kPORT\_ActiveHighTriggerOutputEnable

Enable active high-trigger output.

enumerator kPORT\_ActiveLowTriggerOutputEnable

Enable active low-trigger output.

enum \_port\_digital\_filter\_clock\_source

Digital filter clock source selection.

*Values:*

enumerator kPORT\_BusClock

Digital filters are clocked by the bus clock.

enumerator kPORT\_LpoClock

Digital filters are clocked by the 1 kHz LPO clock.

typedef enum \_port\_mux port\_mux\_t

Pin mux selection.

typedef enum \_port\_interrupt port\_interrupt\_t

Configures the interrupt generation condition.

typedef enum \_port\_digital\_filter\_clock\_source port\_digital\_filter\_clock\_source\_t

Digital filter clock source selection.

typedef struct \_port\_digital\_filter\_config port\_digital\_filter\_config\_t

PORT digital filter feature configuration definition.

typedef struct \_port\_pin\_config port\_pin\_config\_t

PORT pin configuration structure.

FSL\_COMPONENT\_ID

struct \_port\_digital\_filter\_config

*#include <fsl\_port.h>* PORT digital filter feature configuration definition.

### Public Members

uint32\_t digitalFilterWidth

Set digital filter width

port\_digital\_filter\_clock\_source\_t clockSource

Set digital filter clockSource

struct \_port\_pin\_config

*#include <fsl\_port.h>* PORT pin configuration structure.

### Public Members

`uint16_t pullSelect`  
No-pull/pull-down/pull-up select

`uint16_t slewRate`  
Fast/slow slew rate Configure

`uint16_t passiveFilterEnable`  
Passive filter enable/disable

`uint16_t openDrainEnable`  
Open drain enable/disable

`uint16_t driveStrength`  
Fast/slow drive strength configure

`uint16_t lockRegister`  
Lock/unlock the PCR field[15:0]

## 2.27 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\_SourceMdma

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.

```

## 2.28 RTC: Real Time Clock

`void RTC_Init(RTC_Type *base, const rtc_config_t *config)`

Ungates the RTC clock and configures the peripheral for basic operation.

This function issues a software reset if the timer invalid flag is set.

---

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

---

### Parameters

- `base` – RTC peripheral base address
- `config` – Pointer to the user's RTC configuration structure.

`static inline void RTC_Deinit(RTC_Type *base)`

Stops the timer and gate the RTC clock.

### Parameters

- `base` – RTC peripheral base address

`void RTC_GetDefaultConfig(rtc_config_t *config)`

Fills in the RTC config struct with the default settings.

The default values are as follows.

```
config->clockOutput = false;
config->wakeupSelect = false;
config->updateMode = false;
config->supervisorAccess = false;
config->compensationInterval = 0;
config->compensationTime = 0;
```

### Parameters

- `config` – Pointer to the user's RTC configuration structure.

`status_t RTC_SetDatetime(RTC_Type *base, const rtc_datetime_t *datetime)`

Sets the RTC date and time according to the given time structure.

The RTC counter must be stopped prior to calling this function because writes to the RTC seconds register fail if the RTC counter is running.

### Parameters

- `base` – RTC peripheral base address
- `datetime` – Pointer to the structure where the date and time details are stored.

### Returns

`kStatus_Success`: Success in setting the time and starting the RTC  
`kStatus_InvalidArgument`: Error because the datetime format is incorrect

`void RTC_GetDatetime(RTC_Type *base, rtc_datetime_t *datetime)`

Gets the RTC time and stores it in the given time structure.

### Parameters

- `base` – RTC peripheral base address
- `datetime` – Pointer to the structure where the date and time details are stored.

*status\_t* RTC\_SetAlarm(RTC\_Type \*base, const *rtc\_datetime\_t* \*alarmTime)

Sets the RTC alarm time.

The function checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

#### Parameters

- base – RTC peripheral base address
- alarmTime – Pointer to the structure where the alarm time is stored.

#### Returns

kStatus\_Success: success in setting the RTC alarm  
 kStatus\_InvalidArgument: Error because the alarm datetime format is incorrect  
 kStatus\_Fail: Error because the alarm time has already passed

void RTC\_GetAlarm(RTC\_Type \*base, *rtc\_datetime\_t* \*datetime)

Returns the RTC alarm time.

#### Parameters

- base – RTC peripheral base address
- datetime – Pointer to the structure where the alarm date and time details are stored.

void RTC\_EnableInterrupts(RTC\_Type \*base, uint32\_t mask)

Enables the selected RTC interrupts.

#### Parameters

- base – RTC peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *rtc\_interrupt\_enable\_t*

void RTC\_DisableInterrupts(RTC\_Type \*base, uint32\_t mask)

Disables the selected RTC interrupts.

#### Parameters

- base – RTC peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *rtc\_interrupt\_enable\_t*

uint32\_t RTC\_GetEnabledInterrupts(RTC\_Type \*base)

Gets the enabled RTC interrupts.

#### Parameters

- base – RTC peripheral base address

#### Returns

The enabled interrupts. This is the logical OR of members of the enumeration *rtc\_interrupt\_enable\_t*

uint32\_t RTC\_GetStatusFlags(RTC\_Type \*base)

Gets the RTC status flags.

#### Parameters

- base – RTC peripheral base address

#### Returns

The status flags. This is the logical OR of members of the enumeration *rtc\_status\_flags\_t*

void RTC\_ClearStatusFlags(RTC\_Type \*base, uint32\_t mask)

Clears the RTC status flags.

**Parameters**

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

static inline void RTC\_EnableOscillatorClock(RTC\_Type \*base, bool enable)

Enable/Disable RTC 32kHz Oscillator clock.

---

**Note:** After setting this bit, wait the oscillator startup time before enabling the time counter to allow the 32.768 kHz clock time to stabilize.

---

**Parameters**

- base – RTC peripheral base address
- enable – Enable/Disable RTC 32.768 kHz clock

static inline void RTC\_SetClockSource(RTC\_Type \*base)

Set RTC clock source.

*Deprecated:*

Do not use this function. It has been superseded by `RTC_EnableOscillatorClock`

---

**Note:** After setting this bit, wait the oscillator startup time before enabling the time counter to allow the 32.768 kHz clock time to stabilize.

---

**Parameters**

- base – RTC peripheral base address

static inline void RTC\_EnableLPOClock(RTC\_Type \*base, bool enable)

Enable/Disable RTC 1kHz LPO clock.

---

**Note:** After setting this bit, RTC prescaler increments using the LPO 1kHz clock and not the RTC 32kHz crystal clock.

---

**Parameters**

- base – RTC peripheral base address
- enable – Enable/Disable RTC 1kHz LPO clock

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

Starts the RTC time counter.

After calling this function, the timer counter increments once a second provided `SR[TOF]` or `SR[TIF]` are not set.

**Parameters**

- base – RTC peripheral base address

```
static inline void RTC_StopTimer(RTC_Type *base)
```

Stops the RTC time counter.

RTC's seconds register can be written to only when the timer is stopped.

#### Parameters

- base – RTC peripheral base address

```
void RTC_GetMonotonicCounter(RTC_Type *base, uint64_t *counter)
```

Reads the values of the Monotonic Counter High and Monotonic Counter Low and returns them as a single value.

#### Parameters

- base – RTC peripheral base address
- counter – Pointer to variable where the value is stored.

```
void RTC_SetMonotonicCounter(RTC_Type *base, uint64_t counter)
```

Writes values Monotonic Counter High and Monotonic Counter Low by decomposing the given single value. The Monotonic Overflow Flag in RTC\_SR is cleared due to the API.

#### Parameters

- base – RTC peripheral base address
- counter – Counter value

```
status_t RTC_IncrementMonotonicCounter(RTC_Type *base)
```

Increments the Monotonic Counter by one.

Increments the Monotonic Counter (registers RTC\_MCLR and RTC\_MCHR accordingly) by setting the monotonic counter enable (MER[MCE]) and then writing to the RTC\_MCLR register. A write to the monotonic counter low that causes it to overflow also increments the monotonic counter high.

#### Parameters

- base – RTC peripheral base address

#### Returns

kStatus\_Success: success  
kStatus\_Fail: error occurred, either time invalid or monotonic overflow flag was found

```
FSL_RTC_DRIVER_VERSION
```

Version 2.4.0

```
enum _rtc_interrupt_enable
```

List of RTC interrupts.

*Values:*

```
enumerator kRTC_TimeInvalidInterruptEnable
```

Time invalid interrupt.

```
enumerator kRTC_TimeOverflowInterruptEnable
```

Time overflow interrupt.

```
enumerator kRTC_AlarmInterruptEnable
```

Alarm interrupt.

```
enumerator kRTC_MonotonicOverflowInterruptEnable
```

Monotonic Overflow Interrupt Enable

```
enumerator kRTC_SecondsInterruptEnable
```

Seconds interrupt.

enumerator kRTC\_TestModeInterruptEnable  
enumerator kRTC\_FlashSecurityInterruptEnable  
enumerator kRTC\_TamperPinInterruptEnable  
enumerator kRTC\_SecurityModuleInterruptEnable  
enumerator kRTC\_LossOfClockInterruptEnable

enum \_rtc\_status\_flags

List of RTC flags.

*Values:*

enumerator kRTC\_TimeInvalidFlag  
    Time invalid flag  
enumerator kRTC\_TimeOverflowFlag  
    Time overflow flag  
enumerator kRTC\_AlarmFlag  
    Alarm flag  
enumerator kRTC\_MonotonicOverflowFlag  
    Monotonic Overflow Flag  
enumerator kRTC\_TamperInterruptDetectFlag  
    Tamper interrupt detect flag  
enumerator kRTC\_TestModeFlag  
enumerator kRTC\_FlashSecurityFlag  
enumerator kRTC\_TamperPinFlag  
enumerator kRTC\_SecurityTamperFlag  
enumerator kRTC\_LossOfClockTamperFlag

enum \_rtc\_osc\_cap\_load

List of RTC Oscillator capacitor load settings.

*Values:*

enumerator kRTC\_Capacitor\_2p  
    2 pF capacitor load  
enumerator kRTC\_Capacitor\_4p  
    4 pF capacitor load  
enumerator kRTC\_Capacitor\_8p  
    8 pF capacitor load  
enumerator kRTC\_Capacitor\_16p  
    16 pF capacitor load

enum \_rtc\_timer\_seconds\_interrupt\_frequency

List of RTC Timer Seconds Interrupt Frequencies.

*Values:*

enumerator kRTC\_TimerSecondsFrequency1Hz  
    Timer seconds frequency is 1Hz

enumerator `kRTC_TimerSecondsFrequency2Hz`

Timer seconds frequency is 2Hz

enumerator `kRTC_TimerSecondsFrequency4Hz`

Timer seconds frequency is 4Hz

enumerator `kRTC_TimerSecondsFrequency8Hz`

Timer seconds frequency is 8Hz

enumerator `kRTC_TimerSecondsFrequency16Hz`

Timer seconds frequency is 16Hz

enumerator `kRTC_TimerSecondsFrequency32Hz`

Timer seconds frequency is 32Hz

enumerator `kRTC_TimerSecondsFrequency64Hz`

Timer seconds frequency is 64Hz

enumerator `kRTC_TimerSecondsFrequency128Hz`

Timer seconds frequency is 128Hz

typedef enum `_rtc_interrupt_enable` `rtc_interrupt_enable_t`

List of RTC interrupts.

typedef enum `_rtc_status_flags` `rtc_status_flags_t`

List of RTC flags.

typedef enum `_rtc_osc_cap_load` `rtc_osc_cap_load_t`

List of RTC Oscillator capacitor load settings.

typedef enum `_rtc_timer_seconds_interrupt_frequency` `rtc_timer_seconds_interrupt_frequency_t`

List of RTC Timer Seconds Interrupt Frequencies.

typedef struct `_rtc_datetime` `rtc_datetime_t`

Structure is used to hold the date and time.

typedef struct `_rtc_pin_config` `rtc_pin_config_t`

RTC pin config structure.

typedef struct `_rtc_config` `rtc_config_t`

RTC config structure.

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

The config struct can be made const so it resides in flash

static inline uint32\_t `RTC_GetTamperTimeSeconds(RTC_Type *base)`

Get the RTC tamper time seconds.

#### Parameters

- `base` – RTC peripheral base address

static inline void `RTC_SetOscCapLoad(RTC_Type *base, uint32_t capLoad)`

This function sets the specified capacitor configuration for the RTC oscillator.

#### Parameters

- `base` – RTC peripheral base address
- `capLoad` – Oscillator loads to enable. This is a logical OR of members of the enumeration `rtc_osc_cap_load_t`

```
static inline void RTC_Reset(RTC_Type *base)
```

Performs a software reset on the RTC module.

This resets all RTC registers except for the SWR bit and the RTC\_WAR and RTC\_RAR registers. The SWR bit is cleared by software explicitly clearing it.

#### Parameters

- base – RTC peripheral base address

```
static inline void RTC_EnableWakeUpPin(RTC_Type *base, bool enable)
```

Enables or disables the RTC Wakeup Pin Operation.

This function enable or disable RTC Wakeup Pin. The wakeup pin is optional and not available on all devices.

#### Parameters

- base – RTC\_Type base pointer.
- enable – true to enable, false to disable.

```
static inline void RTC_EnableClockOutput(RTC_Type *base, bool enable)
```

Enables or disables the RTC 32 kHz clock output.

This function enables or disables the RTC 32 kHz clock output.

#### Parameters

- base – RTC\_Type base pointer.
- enable – true to enable, false to disable.

```
void RTC_SetTimerSecondsInterruptFrequency(RTC_Type *base,  
                                           rtc_timer_seconds_interrupt_frequency_t freq)
```

Sets the RTC timer seconds interrupt frequency.

This function sets the RTC timer seconds interrupt frequency.

#### Parameters

- base – RTC peripheral base address
- freq – The timer seconds interrupt frequency. This is a member of the enumeration `rtc_timer_seconds_interrupt_frequency_t`

```
struct _rtc_datetime
```

*#include <fsl\_rtc.h>* Structure is used to hold the date and time.

#### Public Members

```
uint16_t year
```

Range from 1970 to 2099.

```
uint8_t month
```

Range from 1 to 12.

```
uint8_t day
```

Range from 1 to 31 (depending on month).

```
uint8_t hour
```

Range from 0 to 23.

```
uint8_t minute
```

Range from 0 to 59.

uint8\_t second

Range from 0 to 59.

struct \_rtc\_pin\_config

*#include <fsl\_rtc.h>* RTC pin config structure.

### Public Members

bool inputLogic

true: Tamper pin input data is logic one. false: Tamper pin input data is logic zero.

bool pinActiveLow

true: Tamper pin is active low. false: Tamper pin is active high.

bool filterEnable

true: Input filter is enabled on the tamper pin. false: Input filter is disabled on the tamper pin.

bool pullSelectNegate

true: Tamper pin pull resistor direction will negate the tamper pin. false: Tamper pin pull resistor direction will assert the tamper pin.

bool pullEnable

true: Pull resistor is enabled on tamper pin. false: Pull resistor is disabled on tamper pin.

struct \_rtc\_config

*#include <fsl\_rtc.h>* RTC config structure.

This structure holds the configuration settings for the RTC peripheral. To initialize this structure to reasonable defaults, call the RTC\_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

### Public Members

bool clockOutput

true: The 32 kHz clock is not output to other peripherals; false: The 32 kHz clock is output to other peripherals

bool wakeupSelect

true: Wakeup pin outputs the 32 KHz clock; false: Wakeup pin used to wakeup the chip

bool updateMode

true: Registers can be written even when locked under certain conditions, false: No writes allowed when registers are locked

bool supervisorAccess

true: Non-supervisor accesses are allowed; false: Non-supervisor accesses are not supported

uint32\_t compensationInterval

Compensation interval that is written to the CIR field in RTC TCR Register

uint32\_t compensationTime

Compensation time that is written to the TCR field in RTC TCR Register

## 2.29 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\_SetUsbVoltageRegulatorEnableMode(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_SetUsbVoltageRegulatorEnableMode(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.

## 2.30 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\_SetPowerModeVlpw(SMC\_Type \*base)

Configures the system to VLPW power mode.

**Parameters**

- base – SMC peripheral base address.

**Returns**

SMC configuration error code.

*status\_t* SMC\_SetPowerModeVlps(SMC\_Type \*base)

Configures the system to VLPS power mode.

**Parameters**

- base – SMC peripheral base address.

**Returns**

SMC configuration error code.

*status\_t* SMC\_SetPowerModeLls(SMC\_Type \*base, const *smc\_power\_mode\_lls\_config\_t* \*config)

Configures the system to LLS power mode.

**Parameters**

- base – SMC peripheral base address.
- config – The LLS power mode configuration structure

**Returns**

SMC configuration error code.

*status\_t* SMC\_SetPowerModeVlls(SMC\_Type \*base, const *smc\_power\_mode\_vlls\_config\_t* \*config)

Configures the system to VLLS power mode.

**Parameters**

- base – SMC peripheral base address.
- config – The VLLS power mode configuration structure.

**Returns**

SMC configuration error code.

FSL\_SMC\_DRIVER\_VERSION

SMC driver version.

enum *\_smc\_power\_mode\_protection*

Power Modes Protection.

*Values:*

enumerator kSMC\_AllowPowerModeVlls

Allow Very-low-leakage Stop Mode.

enumerator kSMC\_AllowPowerModeLls

Allow Low-leakage Stop Mode.

enumerator kSMC\_AllowPowerModeVlp  
Allow Very-Low-power Mode.

enumerator kSMC\_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

## 2.31 SPI: Serial Peripheral Interface Driver

### 2.32 SPI Driver

`void` SPI\_MasterGetDefaultConfig(*spi\_master\_config\_t* \*config)

Sets the SPI master configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in SPI\_MasterInit(). User may use the initialized structure unchanged in SPI\_MasterInit(), or modify some fields of the structure before calling SPI\_MasterInit(). After calling this API, the master is ready to transfer. Example:

```
spi_master_config_t config;
SPI_MasterGetDefaultConfig(&config);
```

#### Parameters

- config – pointer to master config structure

`void` SPI\_MasterInit(SPI\_Type \*base, const *spi\_master\_config\_t* \*config, uint32\_t srcClock\_Hz)

Initializes the SPI with master configuration.

The configuration structure can be filled by user from scratch, or be set with default values by SPI\_MasterGetDefaultConfig(). After calling this API, the slave is ready to transfer. Example

```
spi_master_config_t config = {
    .baudRate_Bps = 400000,
    ...
};
SPI_MasterInit(SPI0, &config);
```

#### Parameters

- base – SPI base pointer
- config – pointer to master configuration structure
- srcClock\_Hz – Source clock frequency.

`void` SPI\_SlaveGetDefaultConfig(*spi\_slave\_config\_t* \*config)

Sets the SPI slave configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in SPI\_SlaveInit(). Modify some fields of the structure before calling SPI\_SlaveInit(). Example:

```
spi_slave_config_t config;  
SPI_SlaveGetDefaultConfig(&config);
```

### Parameters

- config – pointer to slave configuration structure

```
void SPI_SlaveInit(SPI_Type *base, const spi_slave_config_t *config)
```

Initializes the SPI with slave configuration.

The configuration structure can be filled by user from scratch or be set with default values by SPI\_SlaveGetDefaultConfig(). After calling this API, the slave is ready to transfer.

Example

```
spi_slave_config_t config = {  
.polarity = kSPIClockPolarity_ActiveHigh;  
.phase = kSPIClockPhase_FirstEdge;  
.direction = kSPIMsbFirst;  
...  
};  
SPI_MasterInit(SPI0, &config);
```

### Parameters

- base – SPI base pointer
- config – pointer to master configuration structure

```
void SPI_Deinit(SPI_Type *base)
```

De-initializes the SPI.

Calling this API resets the SPI module, gates the SPI clock. The SPI module can't work unless calling the SPI\_MasterInit/SPI\_SlaveInit to initialize module.

### Parameters

- base – SPI base pointer

```
static inline void SPI_Enable(SPI_Type *base, bool enable)
```

Enables or disables the SPI.

### Parameters

- base – SPI base pointer
- enable – pass true to enable module, false to disable module

```
uint32_t SPI_GetStatusFlags(SPI_Type *base)
```

Gets the status flag.

### Parameters

- base – SPI base pointer

### Returns

SPI Status, use status flag to AND `_spi_flags` could get the related status.

```
static inline void SPI_ClearInterrupt(SPI_Type *base, uint8_t mask)
```

Clear the interrupt if enable INCTLR.

### Parameters

- base – SPI base pointer
- mask – Interrupt need to be cleared The parameter could be any combination of the following values:

- kSPI\_RxFullAndModfInterruptEnable
- kSPI\_TxEmptyInterruptEnable
- kSPI\_MatchInterruptEnable
- kSPI\_RxFifoNearFullInterruptEnable
- kSPI\_TxFifoNearEmptyInterruptEnable

`void SPI_EnableInterrupts(SPI_Type *base, uint32_t mask)`

Enables the interrupt for the SPI.

#### Parameters

- `base` – SPI base pointer
- `mask` – SPI interrupt source. The parameter can be any combination of the following values:
  - kSPI\_RxFullAndModfInterruptEnable
  - kSPI\_TxEmptyInterruptEnable
  - kSPI\_MatchInterruptEnable
  - kSPI\_RxFifoNearFullInterruptEnable
  - kSPI\_TxFifoNearEmptyInterruptEnable

`void SPI_DisableInterrupts(SPI_Type *base, uint32_t mask)`

Disables the interrupt for the SPI.

#### Parameters

- `base` – SPI base pointer
- `mask` – SPI interrupt source. The parameter can be any combination of the following values:
  - kSPI\_RxFullAndModfInterruptEnable
  - kSPI\_TxEmptyInterruptEnable
  - kSPI\_MatchInterruptEnable
  - kSPI\_RxFifoNearFullInterruptEnable
  - kSPI\_TxFifoNearEmptyInterruptEnable

`static inline void SPI_EnableDMA(SPI_Type *base, uint8_t mask, bool enable)`

Enables the DMA source for SPI.

#### Parameters

- `base` – SPI base pointer
- `mask` – SPI DMA source.
- `enable` – True means enable DMA, false means disable DMA

`static inline uint32_t SPI_GetDataRegisterAddress(SPI_Type *base)`

Gets the SPI tx/rx data register address.

This API is used to provide a transfer address for the SPI DMA transfer configuration.

#### Parameters

- `base` – SPI base pointer

#### Returns

data register address

uint32\_t SPI\_GetInstance(SPI\_Type \*base)

Get the instance for SPI module.

**Parameters**

- base – SPI base address

static inline void SPI\_SetPinMode(SPI\_Type \*base, spi\_pin\_mode\_t pinMode)

Sets the pin mode for transfer.

**Parameters**

- base – SPI base pointer
- pinMode – pin mode for transfer AND `_spi_pin_mode` could get the related configuration.

void SPI\_MasterSetBaudRate(SPI\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)

Sets the baud rate for SPI transfer. This is only used in master.

**Parameters**

- base – SPI base pointer
- baudRate\_Bps – baud rate needed in Hz.
- srcClock\_Hz – SPI source clock frequency in Hz.

static inline void SPI\_SetMatchData(SPI\_Type \*base, uint32\_t matchData)

Sets the match data for SPI.

The match data is a hardware comparison value. When the value received in the SPI receive data buffer equals the hardware comparison value, the SPI Match Flag in the S register (S[SPMF]) sets. This can also generate an interrupt if the enable bit sets.

**Parameters**

- base – SPI base pointer
- matchData – Match data.

void SPI\_EnableFIFO(SPI\_Type \*base, bool enable)

Enables or disables the FIFO if there is a FIFO.

**Parameters**

- base – SPI base pointer
- enable – True means enable FIFO, false means disable FIFO.

status\_t SPI\_WriteBlocking(SPI\_Type \*base, uint8\_t \*buffer, size\_t size)

Sends a buffer of data bytes using a blocking method.

---

**Note:** This function blocks via polling until all bytes have been sent.

---

**Parameters**

- base – SPI base pointer
- buffer – The data bytes to send
- size – The number of data bytes to send

**Returns**

kStatus\_SPI\_Timeout The transfer timed out and was aborted.

void SPI\_WriteData(SPI\_Type \*base, uint16\_t data)

Writes a data into the SPI data register.

**Parameters**

- base – SPI base pointer
- data – needs to be write.

uint16\_t SPI\_ReadData(SPI\_Type \*base)

Gets a data from the SPI data register.

**Parameters**

- base – SPI base pointer

**Returns**

Data in the register.

void SPI\_SetDummyData(SPI\_Type \*base, uint8\_t dummyData)

Set up the dummy data.

**Parameters**

- base – SPI peripheral address.
- dummyData – Data to be transferred when tx buffer is NULL.

void SPI\_MasterTransferCreateHandle(SPI\_Type \*base, spi\_master\_handle\_t \*handle, spi\_master\_callback\_t callback, void \*userData)

Initializes the SPI master handle.

This function initializes the SPI master handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, call this API once to get the initialized handle.

**Parameters**

- base – SPI peripheral base address.
- handle – SPI handle pointer.
- callback – Callback function.
- userData – User data.

status\_t SPI\_MasterTransferBlocking(SPI\_Type \*base, spi\_transfer\_t \*xfer)

Transfers a block of data using a polling method.

**Parameters**

- base – SPI base pointer
- xfer – pointer to spi\_xfer\_config\_t structure

**Return values**

- kStatus\_Success – Successfully start a transfer.
- kStatus\_InvalidArgument – Input argument is invalid.

status\_t SPI\_MasterTransferNonBlocking(SPI\_Type \*base, spi\_master\_handle\_t \*handle, spi\_transfer\_t \*xfer)

Performs a non-blocking SPI interrupt transfer.

---

**Note:** The API immediately returns after transfer initialization is finished. Call SPI\_GetStatusIRQ() to get the transfer status.

---

---

**Note:** If SPI transfer data frame size is 16 bits, the transfer size cannot be an odd number.

---

### Parameters

- `base` – SPI peripheral base address.
- `handle` – pointer to `spi_master_handle_t` structure which stores the transfer state
- `xfer` – pointer to `spi_xfer_config_t` structure

### Return values

- `kStatus_Success` – Successfully start a transfer.
- `kStatus_InvalidArgument` – Input argument is invalid.
- `kStatus_SPI_Busy` – SPI is not idle, is running another transfer.

`status_t SPI_MasterTransferGetCount(SPI_Type *base, spi_master_handle_t *handle, size_t *count)`

Gets the bytes of the SPI interrupt transferred.

### Parameters

- `base` – SPI peripheral base address.
- `handle` – Pointer to SPI transfer handle, this should be a static variable.
- `count` – Transferred bytes of SPI master.

### Return values

- `kStatus_SPI_Success` – Succeed get the transfer count.
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

`void SPI_MasterTransferAbort(SPI_Type *base, spi_master_handle_t *handle)`

Aborts an SPI transfer using interrupt.

### Parameters

- `base` – SPI peripheral base address.
- `handle` – Pointer to SPI transfer handle, this should be a static variable.

`void SPI_MasterTransferHandleIRQ(SPI_Type *base, spi_master_handle_t *handle)`

Interrupts the handler for the SPI.

### Parameters

- `base` – SPI peripheral base address.
- `handle` – pointer to `spi_master_handle_t` structure which stores the transfer state.

`void SPI_SlaveTransferCreateHandle(SPI_Type *base, spi_slave_handle_t *handle, spi_slave_callback_t callback, void *userData)`

Initializes the SPI slave handle.

This function initializes the SPI slave handle which can be used for other SPI slave transactional APIs. Usually, for a specified SPI instance, call this API once to get the initialized handle.

### Parameters

- `base` – SPI peripheral base address.

- handle – SPI handle pointer.
- callback – Callback function.
- userData – User data.

```
status_t SPI_SlaveTransferNonBlocking(SPI_Type *base, spi_slave_handle_t *handle,  
                                     spi_transfer_t *xfer)
```

Performs a non-blocking SPI slave interrupt transfer.

---

**Note:** The API returns immediately after the transfer initialization is finished. Call SPI\_GetStatusIRQ() to get the transfer status.

---

---

**Note:** If SPI transfer data frame size is 16 bits, the transfer size cannot be an odd number.

---

### Parameters

- base – SPI peripheral base address.
- handle – pointer to spi\_slave\_handle\_t structure which stores the transfer state
- xfer – pointer to spi\_xfer\_config\_t structure

### Return values

- kStatus\_Success – Successfully start a transfer.
- kStatus\_InvalidArgument – Input argument is invalid.
- kStatus\_SPI\_Busy – SPI is not idle, is running another transfer.

```
static inline status_t SPI_SlaveTransferGetCount(SPI_Type *base, spi_slave_handle_t *handle,  
                                               size_t *count)
```

Gets the bytes of the SPI interrupt transferred.

### Parameters

- base – SPI peripheral base address.
- handle – Pointer to SPI transfer handle, this should be a static variable.
- count – Transferred bytes of SPI slave.

### Return values

- kStatus\_SPI\_Success – Succeed get the transfer count.
- kStatus\_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

```
static inline void SPI_SlaveTransferAbort(SPI_Type *base, spi_slave_handle_t *handle)
```

Aborts an SPI slave transfer using interrupt.

### Parameters

- base – SPI peripheral base address.
- handle – Pointer to SPI transfer handle, this should be a static variable.

```
void SPI_SlaveTransferHandleIRQ(SPI_Type *base, spi_slave_handle_t *handle)
```

Interrupts a handler for the SPI slave.

### Parameters

- base – SPI peripheral base address.

- `handle` – pointer to `spi_slave_handle_t` structure which stores the transfer state

`FSL_SPI_DRIVER_VERSION`

SPI driver version.

Return status for the SPI driver.

*Values:*

enumerator `kStatus_SPI_Busy`  
SPI bus is busy

enumerator `kStatus_SPI_Idle`  
SPI is idle

enumerator `kStatus_SPI_Error`  
SPI error

enumerator `kStatus_SPI_Timeout`  
SPI timeout polling status flags.

enum `_spi_clock_polarity`

SPI clock polarity configuration.

*Values:*

enumerator `kSPI_ClockPolarityActiveHigh`  
Active-high SPI clock (idles low).

enumerator `kSPI_ClockPolarityActiveLow`  
Active-low SPI clock (idles high).

enum `_spi_clock_phase`

SPI clock phase configuration.

*Values:*

enumerator `kSPI_ClockPhaseFirstEdge`  
First edge on SPCK occurs at the middle of the first cycle of a data transfer.

enumerator `kSPI_ClockPhaseSecondEdge`  
First edge on SPCK occurs at the start of the first cycle of a data transfer.

enum `_spi_shift_direction`

SPI data shifter direction options.

*Values:*

enumerator `kSPI_MsbFirst`  
Data transfers start with most significant bit.

enumerator `kSPI_LsbFirst`  
Data transfers start with least significant bit.

enum `_spi_ss_output_mode`

SPI slave select output mode options.

*Values:*

enumerator `kSPI_SlaveSelectAsGpio`  
Slave select pin configured as GPIO.

enumerator kSPI\_SlaveSelectFaultInput  
Slave select pin configured for fault detection.

enumerator kSPI\_SlaveSelectAutomaticOutput  
Slave select pin configured for automatic SPI output.

enum \_spi\_pin\_mode  
SPI pin mode options.

*Values:*

enumerator kSPI\_PinModeNormal  
Pins operate in normal, single-direction mode.

enumerator kSPI\_PinModeInput  
Bidirectional mode. Master: MOSI pin is input; Slave: MISO pin is input.

enumerator kSPI\_PinModeOutput  
Bidirectional mode. Master: MOSI pin is output; Slave: MISO pin is output.

enum \_spi\_data\_bitcount\_mode  
SPI data length mode options.

*Values:*

enumerator kSPI\_8BitMode  
8-bit data transmission mode

enumerator kSPI\_16BitMode  
16-bit data transmission mode

enum \_spi\_interrupt\_enable  
SPI interrupt sources.

*Values:*

enumerator kSPI\_RxFullAndModfInterruptEnable  
Receive buffer full (SPRF) and mode fault (MODF) interrupt

enumerator kSPI\_TxEmptyInterruptEnable  
Transmit buffer empty interrupt

enumerator kSPI\_MatchInterruptEnable  
Match interrupt

enumerator kSPI\_RxFifoNearFullInterruptEnable  
Receive FIFO nearly full interrupt

enumerator kSPI\_TxFifoNearEmptyInterruptEnable  
Transmit FIFO nearly empty interrupt

enum \_spi\_flags  
SPI status flags.

*Values:*

enumerator kSPI\_RxBufferFullFlag  
Read buffer full flag

enumerator kSPI\_MatchFlag  
Match flag

enumerator kSPI\_TxBufferEmptyFlag  
Transmit buffer empty flag

enumerator kSPI\_ModeFaultFlag

Mode fault flag

enumerator kSPI\_RxFifoNearFullFlag

Rx FIFO near full

enumerator kSPI\_TxFifoNearEmptyFlag

Tx FIFO near empty

enumerator kSPI\_TxFifoFullFlag

Tx FIFO full

enumerator kSPI\_RxFifoEmptyFlag

Rx FIFO empty

enumerator kSPI\_TxFifoError

Tx FIFO error

enumerator kSPI\_RxFifoError

Rx FIFO error

enumerator kSPI\_TxOverflow

Tx FIFO Overflow

enumerator kSPI\_RxOverflow

Rx FIFO Overflow

enum \_spi\_w1c\_interrupt

SPI FIFO write-1-to-clear interrupt flags.

*Values:*

enumerator kSPI\_RxFifoFullClearInterrupt

Receive FIFO full interrupt

enumerator kSPI\_TxFifoEmptyClearInterrupt

Transmit FIFO empty interrupt

enumerator kSPI\_RxNearFullClearInterrupt

Receive FIFO nearly full interrupt

enumerator kSPI\_TxNearEmptyClearInterrupt

Transmit FIFO nearly empty interrupt

enum \_spi\_txfifo\_watermark

SPI TX FIFO watermark settings.

*Values:*

enumerator kSPI\_TxFifoOneFourthEmpty

SPI tx watermark at 1/4 FIFO size

enumerator kSPI\_TxFifoOneHalfEmpty

SPI tx watermark at 1/2 FIFO size

enum \_spi\_rxfifo\_watermark

SPI RX FIFO watermark settings.

*Values:*

enumerator kSPI\_RxFifoThreeFourthsFull

SPI rx watermark at 3/4 FIFO size

enumerator `kSPI_RxFifoOneHalfFull`  
 SPI rx watermark at 1/2 FIFO size

enum `_spi_dma_enable_t`  
 SPI DMA source.

*Values:*

enumerator `kSPI_TxDmaEnable`  
 Tx DMA request source

enumerator `kSPI_RxDmaEnable`  
 Rx DMA request source

enumerator `kSPI_DmaAllEnable`  
 All DMA request source

typedef enum `_spi_clock_polarity` `spi_clock_polarity_t`  
 SPI clock polarity configuration.

typedef enum `_spi_clock_phase` `spi_clock_phase_t`  
 SPI clock phase configuration.

typedef enum `_spi_shift_direction` `spi_shift_direction_t`  
 SPI data shifter direction options.

typedef enum `_spi_ss_output_mode` `spi_ss_output_mode_t`  
 SPI slave select output mode options.

typedef enum `_spi_pin_mode` `spi_pin_mode_t`  
 SPI pin mode options.

typedef enum `_spi_data_bitcount_mode` `spi_data_bitcount_mode_t`  
 SPI data length mode options.

typedef enum `_spi_w1c_interrupt` `spi_w1c_interrupt_t`  
 SPI FIFO write-1-to-clear interrupt flags.

typedef enum `_spi_txfifo_watermark` `spi_txfifo_watermark_t`  
 SPI TX FIFO watermark settings.

typedef enum `_spi_rxfifo_watermark` `spi_rxfifo_watermark_t`  
 SPI RX FIFO watermark settings.

typedef struct `_spi_master_config` `spi_master_config_t`  
 SPI master user configure structure.

typedef struct `_spi_slave_config` `spi_slave_config_t`  
 SPI slave user configure structure.

typedef struct `_spi_transfer` `spi_transfer_t`  
 SPI transfer structure.

typedef struct `_spi_master_handle` `spi_master_handle_t`

typedef `spi_master_handle_t` `spi_slave_handle_t`  
 Slave handle is the same with master handle

typedef void (`*spi_master_callback_t`)(`SPI_Type *base`, `spi_master_handle_t *handle`, `status_t status`, `void *userData`)  
 SPI master callback for finished transmit.

```
typedef void (*spi_slave_callback_t)(SPI_Type *base, spi_slave_handle_t *handle, status_t status, void *userData)
```

SPI master callback for finished transmit.

```
volatile uint8_t g_spiDummyData[]
```

Global variable for dummy data value setting.

```
SPI_DUMMYDATA
```

SPI dummy transfer data, the data is sent while txBuff is NULL.

```
SPI_RETRY_TIMES
```

Retry times for waiting flag.

```
struct _spi_master_config
```

*#include <fsl\_spi.h>* SPI master user configure structure.

### Public Members

```
bool enableMaster
```

Enable SPI at initialization time

```
bool enableStopInWaitMode
```

SPI stop in wait mode

```
spi_clock_polarity_t polarity
```

Clock polarity

```
spi_clock_phase_t phase
```

Clock phase

```
spi_shift_direction_t direction
```

MSB or LSB

```
spi_data_bitcount_mode_t dataMode
```

8bit or 16bit mode

```
spi_txfifo_watermark_t txWatermark
```

Tx watermark settings

```
spi_rxfifo_watermark_t rxWatermark
```

Rx watermark settings

```
spi_ss_output_mode_t outputMode
```

SS pin setting

```
spi_pin_mode_t pinMode
```

SPI pin mode select

```
uint32_t baudRate_Bps
```

Baud Rate for SPI in Hz

```
struct _spi_slave_config
```

*#include <fsl\_spi.h>* SPI slave user configure structure.

### Public Members

```
bool enableSlave
```

Enable SPI at initialization time

bool enableStopInWaitMode  
 SPI stop in wait mode

*spi\_clock\_polarity\_t* polarity  
 Clock polarity

*spi\_clock\_phase\_t* phase  
 Clock phase

*spi\_shift\_direction\_t* direction  
 MSB or LSB

*spi\_data\_bitcount\_mode\_t* dataMode  
 8bit or 16bit mode

*spi\_txfifo\_watermark\_t* txWatermark  
 Tx watermark settings

*spi\_rxfifo\_watermark\_t* rxWatermark  
 Rx watermark settings

*spi\_pin\_mode\_t* pinMode  
 SPI pin mode select

struct *\_spi\_transfer*  
*#include <fsl\_spi.h>* SPI transfer structure.

### Public Members

const uint8\_t \*txData  
 Send buffer

uint8\_t \*rxData  
 Receive buffer

size\_t dataSize  
 Transfer bytes

uint32\_t flags  
 SPI control flag, useless to SPI.

struct *\_spi\_master\_handle*  
*#include <fsl\_spi.h>* SPI transfer handle structure.

### Public Members

const uint8\_t \*volatile txData  
 Transfer buffer

uint8\_t \*volatile rxData  
 Receive buffer

volatile size\_t txRemainingBytes  
 Send data remaining in bytes

volatile size\_t rxRemainingBytes  
 Receive data remaining in bytes

volatile uint32\_t state  
 SPI internal state

size\_t transferSize  
Bytes to be transferred

uint8\_t bytePerFrame  
SPI mode, 2bytes or 1byte in a frame

uint8\_t watermark  
Watermark value for SPI transfer

spi\_master\_callback\_t callback  
SPI callback

void \*userData  
Callback parameter

## 2.33 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;
```

(continues on next page)

(continued from previous page)

```

#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
#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 PWM setup successful  
kStatus\_Error PWM setup failed  
kStatus\_Timeout PWM setup timeout when write register CnV or MOD

*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 *status\_t* 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

**Returns**

kStatus\_Success PWM setup successful kStatus\_Timeout PWM setup timeout when write register CnSC

```
static inline status_t 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`.

#### Returns

`kStatus_Success` PWM setup successful  
`kStatus_Timeout` PWM setup timeout when write register CnSC

```
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

```
status_t 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.

#### Returns

`kStatus_Success` PWM setup successful  
`kStatus_Timeout` PWM setup timeout when write register CnV

```
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 `status_t` 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:**

- 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.
  - 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.

**Returns**

`kStatus_Success` PWM setup successful  
`kStatus_Timeout` PWM setup timeout when write register CnSC

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 `status_t` TPM\_StopTimer(TPM\_Type \*base)

Stops the TPM counter.

**Parameters**

- `base` – TPM peripheral base address

**Returns**

`kStatus_Success` PWM setup successful  
`kStatus_Timeout` PWM setup timeout when write register CnSC

FSL\_TPM\_DRIVER\_VERSION

TPM driver version 2.4.0.

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\_NoPwmSignal  
No PWM output on pin

enumerator kTPM\_LowTrue

Low true pulses

enumerator kTPM\_HighTrue

High true pulses

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 EL SB 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_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\_TIMEOUT

Max loops to wait for writing register.

When writing MOD CnV CnSC and SC register, driver will wait until register is updated. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

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\_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;

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

## 2.34 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)



# 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\_QUEUE\_SIZE

```
#define FMSTR_COMM_QUEUE_SIZE [number]
```

**Value Type** Value in range 0...255

**Description** Specify the size of the FIFO receiver queue used to quickly receive and store characters in the FMSTR\_SHORT\_INTR interrupt mode. The default value is 32 B.

### FMSTR\_SERIAL\_SINGLEWIRE

```
#define FMSTR_SERIAL_SINGLEWIRE [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Set to non-zero to enable the “True” single-wire mode which uses a single MCU pin to communicate. The low-level driver enables the pin direction switching when the MCU peripheral supports it.

**CAN Bus transport** This section describes configuration parameters used when CAN transport is used:

```
#define FMSTR_TRANSPORT FMSTR_CAN
```

**FMSTR\_CAN\_DRV** Select what low-level driver interface will be used when implementing the CAN communication.

```
#define FMSTR_CAN_DRV [identifier]
```

**Value Type** Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing CAN driver instances.

**Description** When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/can implementation*):

- FMSTR\_CAN\_MCUX\_FLEXCAN - FlexCAN driver
- FMSTR\_CAN\_MCUX\_MCAN - MCAN driver
- FMSTR\_CAN\_MCUX\_MSCAN - msCAN driver
- FMSTR\_CAN\_MCUX\_DSCFLEXCAN - DSC FlexCAN driver
- FMSTR\_CAN\_MCUX\_DSCMSCAN - DSC msCAN driver

Other SDKs or BSPs may define the custom low-level driver interface structure which may be used as FMSTR\_CAN\_DRV.

### FMSTR\_CAN\_BASE

```
#define FMSTR_CAN_BASE [address|symbol]
```

**Value Type** Optional address value (numeric or symbolic)

**Description** Specify the base address of the FlexCAN, msCAN, or other CAN peripheral module to be used for the communication. This value is not defined by default. User application should call `FMSTR_SetCanBaseAddress()` to select the peripheral module.

#### FMSTR\_CAN\_CMDID

```
#define FMSTR_CAN_CMDID [number]
```

**Value Type** CAN identifier (11-bit or 29-bit number)

**Description** CAN message identifier used for FreeMASTER commands (direction from PC Host tool to target application). When declaring 29-bit identifier, combine the numeric value with `FMSTR_CAN_EXTID` bit. Default value is 0x7AA.

#### FMSTR\_CAN\_RSPID

```
#define FMSTR_CAN_RSPID [number]
```

**Value Type** CAN identifier (11-bit or 29-bit number)

**Description** CAN message identifier used for responding messages (direction from target application to PC Host tool). When declaring 29-bit identifier, combine the numeric value with `FMSTR_CAN_EXTID` bit. Note that both *CMDID* and *RSPID* values may be the same. Default value is 0x7AA.

#### FMSTR\_FLEXCAN\_TXMB

```
#define FMSTR_FLEXCAN_TXMB [number]
```

**Value Type** Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

**Description** Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame transmission. Default value is 0.

#### FMSTR\_FLEXCAN\_RXMB

```
#define FMSTR_FLEXCAN_RXMB [number]
```

**Value Type** Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

**Description** Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame reception. Note that the FreeMASTER driver may also operate with a common message buffer used by both TX and RX directions. Default value is 1.

**Network transport** This section describes configuration parameters used when Network transport is used:

```
#define FMSTR_TRANSPORT FMSTR_NET
```

**FMSTR\_NET\_DRV** Select network interface implementation.

```
#define FMSTR_NET_DRV [identifier]
```

**Value Type** Identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing NET driver instances.

**Description** When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/network implementation*):

- **FMSTR\_NET\_LWIP\_TCP** - TCP communication using lwIP stack
- **FMSTR\_NET\_LWIP\_UDP** - UDP communication using lwIP stack
- **FMSTR\_NET\_SEGGER\_RTT** - Communication using SEGGER J-Link RTT interface

Other SDKs or BSPs may define the custom networking interface which may be used as FMSTR\_CAN\_DRV.

Add another row below:

#### FMSTR\_NET\_PORT

```
#define FMSTR_NET_PORT [number]
```

**Value Type** TCP or UDP port number (short integer)

**Description** Specifies the server port number used by TCP or UDP protocols.

#### FMSTR\_NET\_BLOCKING\_TIMEOUT

```
#define FMSTR_NET_BLOCKING_TIMEOUT [number]
```

**Value Type** Timeout as number of milliseconds

**Description** This value specifies a timeout in milliseconds for which the network socket operations may block the execution inside *FMSTR\_Poll*. This may be set high (e.g. 250) when a dedicated RTOS task is used to handle FreeMASTER protocol polling. Set to a lower value when the polling task is also responsible for other operations. Set to 0 to attempt to use non-blocking socket operations.

### FMSTR\_NET\_AUTODISCOVERY

```
#define FMSTR_NET_AUTODISCOVERY [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** This option enables the FreeMASTER driver to use a separate UDP socket to broadcast auto-discovery messages to network. This helps the FreeMASTER tool to discover the target device address, port and protocol options.

### Debugging options

#### FMSTR\_DISABLE

```
#define FMSTR_DISABLE [0|1]
```

**Value Type** boolean (0 or 1)

**Description** Define as non-zero to disable all FreeMASTER features, exclude the driver code from build, and compile all its API functions empty. This may be useful to remove FreeMASTER without modifying any application source code. Default value is 0 (false).

#### FMSTR\_DEBUG\_TX

```
#define FMSTR_DEBUG_TX [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to enable the driver to periodically transmit test frames out on the selected communication interface (SCI or CAN). With the debug transmission enabled, it is simpler to detect problems in the baudrate or other communication configuration settings.

The test frames are transmitted until the first valid command frame is received from the PC Host tool. The test frame is a valid error status frame, as defined by the protocol format. On the serial line, the test frame consists of three printable characters (+©W) which are easy to capture using the serial terminal tools.

This feature requires the FMSTR\_Poll() function to be called periodically. Default value is 0 (false).

#### FMSTR\_APPLICATION\_STR

```
#define FMSTR_APPLICATION_STR
```

**Value Type** String.

**Description** Name of the application visible in FreeMASTER host application.

### Memory access

### FMSTR\_USE\_READMEM

```
#define FMSTR_USE_READMEM [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the Memory Read command and enable FreeMASTER to have read access to memory and variables. The access can be further restricted by using a TSA feature.  
Default value is 1 (true).

### FMSTR\_USE\_WRITEMEM

```
#define FMSTR_USE_WRITEMEM [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the Memory Write command.  
The default value is 1 (true).

### Oscilloscope options

#### FMSTR\_USE\_SCOPE

```
#define FMSTR_USE_SCOPE [number]
```

**Value Type** Integer number.

**Description** Number of Oscilloscope instances to be supported. Set to 0 to disable the Oscilloscope feature.  
Default value is 0.

#### FMSTR\_MAX\_SCOPE\_VARS

```
#define FMSTR_MAX_SCOPE_VARS [number]
```

**Value Type** Integer number larger than 2.

**Description** Number of variables to be supported by each Oscilloscope instance.  
Default value is 8.

### Recorder options

#### FMSTR\_USE\_RECORDER

```
#define FMSTR_USE_RECORDER [number]
```

**Value Type** Integer number.

**Description** Number of Recorder instances to be supported. Set to 0 to disable the Recorder feature.

Default value is 0.

#### FMSTR\_REC\_BUFF\_SIZE

```
#define FMSTR_REC_BUFF_SIZE [number]
```

**Value Type** Integer number larger than 2.

**Description** Defines the size of the memory buffer used by the Recorder instance #0. Default: not defined, user shall call 'FMSTR\_RecorderCreate()' API function to specify this parameter in run time.

#### FMSTR\_REC\_TIMEBASE

```
#define FMSTR_REC_TIMEBASE [time specification]
```

**Value Type** Number (nanoseconds time).

**Description** Defines the base sampling rate in nanoseconds (sampling speed) Recorder instance #0.

Use one of the following macros:

- FMSTR\_REC\_BASE\_SECONDS(x)
- FMSTR\_REC\_BASE\_MILLISEC(x)
- FMSTR\_REC\_BASE\_MICROSEC(x)
- FMSTR\_REC\_BASE\_NANOSEC(x)

Default: not defined, user shall call 'FMSTR\_RecorderCreate()' API function to specify this parameter in run time.

#### FMSTR\_REC\_FLOAT\_TRIG

```
#define FMSTR_REC_FLOAT_TRIG [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the floating-point triggering. Be aware that floating-point triggering may grow the code size by linking the floating-point standard library.

Default value is 0 (false).

### Application Commands options

### FMSTR\_USE\_APPCMD

```
#define FMSTR_USE_APPCMD [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the Application Commands feature. Default value is 0 (false).

### FMSTR\_APPCMD\_BUFF\_SIZE

```
#define FMSTR_APPCMD_BUFF_SIZE [size]
```

**Value Type** Numeric buffer size in range 1..255

**Description** The size of the Application Command data buffer allocated by the driver. The buffer stores the (optional) parameters of the Application Command which waits to be processed.

### FMSTR\_MAX\_APPCMD\_CALLS

```
#define FMSTR_MAX_APPCMD_CALLS [number]
```

**Value Type** Number in range 0..255

**Description** The number of different Application Commands that can be assigned a callback handler function using FMSTR\_RegisterAppCmdCall(). Default value is 0.

## TSA options

### FMSTR\_USE\_TSA

```
#define FMSTR_USE_TSA [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable the FreeMASTER TSA feature to be used. With this option enabled, the TSA tables defined in the applications are made available to the FreeMASTER host tool. Default value is 0 (false).

### FMSTR\_USE\_TSA\_SAFETY

```
#define FMSTR_USE_TSA_SAFETY [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable the memory access validation in the FreeMASTER driver. With this option, the host tool is not able to access the memory which is not described by at least one TSA descriptor. Also a write access is denied for objects defined as read-only in TSA tables. Default value is 0 (false).

#### FMSTR\_USE\_TSA\_INROM

```
#define FMSTR_USE_TSA_INROM [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Declare all TSA descriptors as *const*, which enables the linker to put the data into the flash memory. The actual result depends on linker settings or the linker commands used in the project. Default value is 0 (false).

#### FMSTR\_USE\_TSA\_DYNAMIC

```
#define FMSTR_USE_TSA_DYNAMIC [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable runtime-defined TSA entries to be added to the TSA table by the FMSTR\_SetUpTsaBuff() and FMSTR\_TsaAddVar() functions. Default value is 0 (false).

### Pipes options

#### FMSTR\_USE\_PIPES

```
#define FMSTR_USE_PIPES [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable the FreeMASTER Pipes feature to be used. Default value is 0 (false).

#### FMSTR\_MAX\_PIPES\_COUNT

```
#define FMSTR_MAX_PIPES_COUNT [number]
```

**Value Type** Number in range 1..63.

**Description** The number of simultaneous pipe connections to support. The default value is 1.

**Driver interrupt modes** To implement the communication, the FreeMASTER driver handles the Serial or CAN module's receive and transmit requests. Use the *freemaster\_cfg.h* configuration file to select whether the driver processes the communication automatically in the interrupt service routine handler or if it only polls the status of the module (typically during the application idle time).

This section describes each of the interrupt mode in more details.

**Completely Interrupt-Driven operation** Activated using:

```
#define FMSTR_LONG_INTR 1
```

In this mode, both the communication and the FreeMASTER protocol decoding is done in the *FMSTR\_SerialIsr*, *FMSTR\_CanIsr*, or other interrupt service routine. Because the protocol execution may be a lengthy task (especially with the TSA-Safety enabled) it is recommended to use this mode only if the interrupt prioritization scheme is possible in the application and the FreeMASTER interrupt is assigned to a lower (the lowest) priority.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR\_SerialIsr* or *FMSTR\_CanIsr* functions from that handler.

**Mixed Interrupt and Polling Modes** Activated using:

```
#define FMSTR_SHORT_INTR 1
```

In this mode, the communication processing time is split between the interrupt routine and the main application loop or task. The raw communication is handled by the *FMSTR\_SerialIsr*, *FMSTR\_CanIsr*, or other interrupt service routine, while the protocol decoding and execution is handled by the *FMSTR\_Poll* routine. Call *FMSTR\_Poll* during the idle time in the application main loop.

The interrupt processing in this mode is relatively fast and deterministic. Upon a serial-receive event, the received character is only placed into a FIFO-like queue and it is not further processed. Upon a CAN receive event, the received frame is stored into a receive buffer. When transmitting, the characters are fetched from the prepared transmit buffer.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR\_SerialIsr* or *FMSTR\_CanIsr* functions from that handler.

When the serial interface is used as the serial communication interface, ensure that the *FMSTR\_Poll* function is called at least once per *N* character time periods. *N* is the length of the FreeMASTER FIFO queue (*FMSTR\_COMM\_QUEUE\_SIZE*) and the character time is the time needed to transmit or receive a single byte over the SCI line.

**Completely Poll-driven**

```
#define FMSTR_POLL_DRIVEN 1
```

In this mode, both the communication and the FreeMASTER protocol decoding are done in the *FMSTR\_Poll* routine. No interrupts are needed and the *FMSTR\_SerialIsr*, *FMSTR\_CanIsr*, and similar handlers compile to an empty code.

When using this mode, ensure that the *FMSTR\_Poll* function is called by the application at least once per the serial "character time" which is the time needed to transmit or receive a single character.

In the latter two modes (*FMSTR\_SHORT\_INTR* and *FMSTR\_POLL\_DRIVEN*), the protocol handling takes place in the *FMSTR\_Poll* routine. An application interrupt can occur in the middle of the

Read Memory or Write Memory commands' execution and corrupt the variable being accessed by the FreeMASTER driver. In these two modes, some issues or glitches may occur when using FreeMASTER to visualize or monitor volatile variables modified in interrupt servicing code.

The same issue may appear even in the full interrupt mode (FMSTR\_LONG\_INTR), if volatile variables are modified in the interrupt code with a priority higher than the priority of the communication interrupt.

**Data types** Simple portability was one of the main requirements when writing the FreeMASTER driver. This is why the driver code uses the privately-declared data types and the vast majority of the platform-dependent code is separated in the platform-dependent source files. The data types used in the driver API are all defined in the platform-specific header file.

To prevent name conflicts with the symbols used in the application, all data types, macros, and functions have the FMSTR\_ prefix. The only global variables used in the driver are the transport and low-level API structures exported from the driver-implementation layer to upper layers. Other than that, all private variables are declared as static and named using the fmstr\_ prefix.

**Communication interface initialization** The FreeMASTER driver does not perform neither the initialization nor the configuration of the peripheral module that it uses to communicate. It is the application startup code responsibility to configure the communication module before the FreeMASTER driver is initialized by the FMSTR\_Init call.

When the Serial communication module is used as the FreeMASTER communication interface, configure the UART receive and transmit pins, the serial communication baud rate, parity (no-parity), the character length (eight bits), and the number of stop bits (one) before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see [Driver interrupt modes](#)), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected serial peripheral module. Call the FMSTR\_SerialIsr function from the application handler.

When a CAN module is used as the FreeMASTER communication interface, configure the CAN receive and transmit pins and the CAN module bit rate before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see [Driver interrupt modes](#)), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected CAN peripheral module. Call the FMSTR\_CanIsr function from the application handler.

**Note:** It is not necessary to enable or unmask the serial nor the CAN interrupts before initializing the FreeMASTER driver. The driver enables or disables the interrupts and communication lines, as required during runtime.

**FreeMASTER Recorder calls** When using the FreeMASTER Recorder in the application (FMSTR\_USE\_RECORDER > 0), call the FMSTR\_RecorderCreate function early after FMSTR\_Init to set up each recorder instance to be used in the application. Then call the FMSTR\_Recorder function periodically in the code where the data recording should occur. A typical place to call the Recorder routine is at the timer or PWM interrupts, but it can be anywhere else. The example applications provided together with the driver code call the FMSTR\_Recorder in the main application loop.

In applications where FMSTR\_Recorder is called periodically with a constant period, specify the period in the Recorder configuration structure before calling FMSTR\_RecorderCreate. This setting enables the PC Host FreeMASTER tool to display the X-axis of the Recorder graph properly scaled for the time domain.

**Driver usage** Start using or evaluating FreeMASTER by opening some of the example applications available in the driver setup package.

Follow these steps to enable the basic FreeMASTER connectivity in the application:

- Make sure that all \*.c files of the FreeMASTER driver from the `src/common/platforms/[your_platform]` folder are a part of the project. See [Driver files](#) for more details.
- Configure the FreeMASTER driver by creating or editing the `freemaster_cfg.h` file and by saving it into the application project directory. See [Driver configuration](#) for more details.
- Include the `freemaster.h` file into any application source file that makes the FreeMASTER API calls.
- Initialize the Serial or CAN modules. Set the baud rate, parity, and other parameters of the communication. Do not enable the communication interrupts in the interrupt mask registers.
- For the FMSTR\_LONG\_INTR and FMSTR\_SHORT\_INTR modes, install the application-specific interrupt routine and call the FMSTR\_SerialIsr or FMSTR\_CanIsr functions from this handler.
- Call the FMSTR\_Init function early on in the application initialization code.
- Call the FMSTR\_RecorderCreate functions for each Recorder instance to enable the Recorder feature.
- In the main application loop, call the FMSTR\_Poll API function periodically when the application is idle.
- For the FMSTR\_SHORT\_INTR and FMSTR\_LONG\_INTR modes, enable the interrupts globally so that the interrupts can be handled by the CPU.

**Communication troubleshooting** The most common problem that causes communication issues is a wrong baud rate setting or a wrong pin multiplexer setting of the target MCU. When a communication between the PC Host running FreeMASTER and the target MCU cannot be established, try enabling the FMSTR\_DEBUG\_TX option in the `freemaster_cfg.h` file and call the FMSTR\_Poll function periodically in the main application task loop.

With this feature enabled, the FreeMASTER driver periodically transmits a test frame through the Serial or CAN lines. Use a logic analyzer or an oscilloscope to monitor the signals at the communication pins of the CPU device to examine whether the bit rate and signal polarity are configured properly.

## Driver API

This section describes the driver Application Programmers' Interface (API) needed to initialize and use the FreeMASTER serial communication driver.

**Control API** There are three key functions to initialize and use the driver.

### FMSTR\_Init

#### Prototype

```
FMSTR_BOOL FMSTR_Init(void);
```

- Declaration: `freemaster.h`
- Implementation: `freemaster_protocol.c`

**Description** This function initializes the internal variables of the FreeMASTER driver and enables the communication interface. This function does not change the configuration of the selected communication module. The hardware module must be initialized before the *FMSTR\_Init* function is called.

A call to this function must occur before calling any other FreeMASTER driver API functions.

## FMSTR\_Poll

### Prototype

```
void FMSTR_Poll(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_protocol.c*

**Description** In the poll-driven or short interrupt modes, this function handles the protocol decoding and execution (see *Driver interrupt modes*). In the poll-driven mode, this function also handles the communication interface with the PC. Typically, the *FMSTR\_Poll* function is called during the “idle” time in the main application task loop.

To prevent the receive data overflow (loss) on a serial interface, make sure that the *FMSTR\_Poll* function is called at least once per the time calculated as:

$$N * Tchar$$

where:

- *N* is equal to the length of the receive FIFO queue (configured by the *FMSTR\_COMM\_QUEUE\_SIZE* macro). *N* is 1 for the poll-driven mode.
- *Tchar* is the character time, which is the time needed to transmit or receive a single byte over the SCI line.

**Note:** In the long interrupt mode, this function typically compiles as an empty function and can still be called. It is worthwhile to call this function regardless of the interrupt mode used in the application. This approach enables a convenient switching between the different interrupt modes only by changing the configuration macros in the *freemaster\_cfg.h* file.

## FMSTR\_SerialIsr / FMSTR\_CanIsr

### Prototype

```
void FMSTR_SerialIsr(void);
void FMSTR_CanIsr(void);
```

- Declaration: *freemaster.h*
- Implementation: *hw-specific low-level driver C file*

**Description** This function contains the interrupt-processing code of the FreeMASTER driver. In long or short interrupt modes (see *Driver interrupt modes*), this function must be called from the application interrupt service routine registered for the communication interrupt vector. On platforms where the communication module uses multiple interrupt vectors, the application should register a handler for all vectors and call this function at each interrupt.

**Note:** In a poll-driven mode, this function is compiled as an empty function and does not have to be used.

## Recorder API

### FMSTR\_RecorderCreate

#### Prototype

```
FMSTR_BOOL FMSTR_RecorderCreate(FMSTR_INDEX recIndex, FMSTR_REC_BUFF* buffCfg);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_rec.c*

**Description** This function registers a recorder instance and enables it to be used by the PC Host tool. Call this function for all recorder instances from 0 to the maximum number defined by the FMSTR\_USE\_RECORDER configuration option (minus one). An exception to this requirement is the recorder of instance 0 which may be automatically configured by FMSTR\_Init when the *freemaster\_cfg.h* configuration file defines the *FMSTR\_REC\_BUFF\_SIZE* and *FMSTR\_REC\_TIMEBASE* options.

For more information, see [Configurable items](#).

### FMSTR\_Recorder

#### Prototype

```
void FMSTR_Recorder(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_rec.c*

**Description** This function takes a sample of the variables being recorded using the FreeMASTER Recorder instance *recIndex*. If the selected Recorder is not active when the *FMSTR\_Recorder* function is being called, the function returns immediately. When the Recorder is active, the values of the variables being recorded are copied into the recorder buffer and the trigger conditions are evaluated.

If a trigger condition is satisfied, the Recorder enters the post-trigger mode, where it counts down the follow-up samples (number of *FMSTR\_Recorder* function calls) and de-activates the Recorder when the required post-trigger samples are finished.

The *FMSTR\_Recorder* function is typically called in the timer or PWM interrupt service routines. This function can also be called in the application main loop (for testing purposes).

### FMSTR\_RecorderTrigger

#### Prototype

```
void FMSTR_RecorderTrigger(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_rec.c*

**Description** This function forces the Recorder trigger condition to happen, which causes the Recorder to be automatically deactivated after the post-trigger samples are sampled. Use this function in the application code for programmatic control over the Recorder triggering. This can be useful when a more complex triggering conditions need to be used.

**Fast Recorder API** The Fast Recorder feature is not available in the FreeMASTER driver version 3. This feature was heavily dependent on the target platform and it was only available for the 56F8xxxx DSCs.

**TSA Tables** When the TSA is enabled in the FreeMASTER driver configuration file (by setting the FMSTR\_USE\_TSA macro to a non-zero value), it defines the so-called TSA tables in the application. This section describes the macros that must to be used to define the TSA tables.

There can be any number of TSA tables spread across the application source files. There must be always exactly one TSA Table List defined, which informs the FreeMASTER driver about the active TSA tables.

When there is at least one TSA table and one TSA Table List defined in the application, the TSA information automatically appears in the FreeMASTER symbols list. The symbols can then be used to create FreeMASTER variables for visualization or control.

**TSA table definition** The TSA table describes the static or global variables together with their address, size, type, and access-protection information. If the TSA-described variables are of a structure type, the TSA table may also describe this type and provide an access to the individual structure members of the variable.

The TSA table definition begins with the FMSTR\_TSA\_TABLE\_BEGIN macro with a *table\_id* identifying the table. The *table\_id* shall be a valid C-language symbol.

```
FMSTR_TSA_TABLE_BEGIN(table_id)
```

After this opening macro, the TSA descriptors are placed using these macros:

```
/* Adding variable descriptors */
FMSTR_TSA_RW_VAR(name, type) /* read/write variable entry */
FMSTR_TSA_RO_VAR(name, type) /* read-only variable entry */

/* Description of complex data types */
FMSTR_TSA_STRUCT(struct_name) /* structure or union type entry */
FMSTR_TSA_MEMBER(struct_name, member_name, type) /* structure member entry */

/* Memory blocks */
FMSTR_TSA_RW_MEM(name, type, address, size) /* read/write memory block */
FMSTR_TSA_RO_MEM(name, type, address, size) /* read-only memory block */
```

The table is closed using the FMSTR\_TSA\_TABLE\_END macro:

```
FMSTR_TSA_TABLE_END()
```

**TSA descriptor parameters** The TSA descriptor macros accept these parameters:

- *name* — variable name. The variable must be defined before the TSA descriptor references it.
- *type* — variable or member type. Only one of the pre-defined type constants may be used (see below).
- *struct\_name* — structure type name. The type must be defined (typedef) before the TSA descriptor references it.

- *member\_name* — structure member name.

**Note:** The structure member descriptors (FMSTR\_TSA\_MEMBER) must immediately follow the parent structure descriptor (FMSTR\_TSA\_STRUCT) in the table.

**Note:** To write-protect the variables in the FreeMASTER driver (FMSTR\_TSA\_RO\_VAR), enable the TSA-Safety feature in the configuration file.

**TSA variable types** The table lists *type* identifiers which can be used in TSA descriptors:

| Constant                           | Description  |
|------------------------------------|--|
| FMSTR_TSA_UINTn                    | Unsigned integer type of size <i>n</i> bits (n=8,16,32,64)     |
| FMSTR_TSA_SINTn                    | Signed integer type of size <i>n</i> bits (n=8,16,32,64)       |
| FMSTR_TSA_FRACn                    | Fractional number of size <i>n</i> bits (n=16,32,64).          |
| FMSTR_TSA_FRAC_Q( <i>m,n</i> )     | Signed fractional number in general Q form (m+n+1 total bits)  |
| FMSTR_TSA_FRAC_UQ( <i>m,n</i> )    | Unsigned fractional number in general UQ form (m+n total bits) |
| FMSTR_TSA_FLOAT                    | 4-byte standard IEEE floating-point type                       |
| FMSTR_TSA_DOUBLE                   | 8-byte standard IEEE floating-point type                       |
| FMSTR_TSA_POINTER                  | Generic pointer type defined (platform-specific 16 or 32 bit)  |
| FM-STR_TSA_USERTYPE( <i>name</i> ) | Structure or union type declared with FMSTR_TSA_STRUCT record  |

**TSA table list** There shall be exactly one TSA Table List in the application. The list contains one entry for each TSA table defined anywhere in the application.

The TSA Table List begins with the FMSTR\_TSA\_TABLE\_LIST\_BEGIN macro and continues with the TSA table entries for each table.

```
FMSTR_TSA_TABLE_LIST_BEGIN()

FMSTR_TSA_TABLE(table_id)
FMSTR_TSA_TABLE(table_id2)
FMSTR_TSA_TABLE(table_id3)
...
```

The list is closed with the FMSTR\_TSA\_TABLE\_LIST\_END macro:

```
FMSTR_TSA_TABLE_LIST_END()
```

**TSA Active Content entries** FreeMASTER v2.0 and higher supports TSA Active Content, enabling the TSA tables to describe the memory-mapped files, virtual directories, and URL hyperlinks. FreeMASTER can access such objects similarly to accessing the files and folders on the local hard drive.

With this set of TSA entries, the FreeMASTER pages can be embedded directly into the target MCU flash and accessed by FreeMASTER directly over the communication line. The HTML-coded pages rendered inside the FreeMASTER window can access the TSA Active Content resources using a special URL referencing the *fmstr:* protocol.

This example provides an overview of the supported TSA Active Content entries:

```
FMSTR_TSA_TABLE_BEGIN(files_and_links)

/* Directory entry applies to all subsequent MEMFILE entries */
FMSTR_TSA_DIRECTORY("/text_files") /* entering a new virtual directory */
```

(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 responseDataAddr, FMSTR_SIZE responseDataLen);
```

- 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. |
|-----------------------|--|

By default, this is defined as *FM-STR\_SIZE*.

---

|                      |   |
|----------------------|---|
| <i>FM-STR_TSA_TS</i> | Data type used to hold a memory block size, as used in the TSA descriptors. |
|----------------------|---|

By default, this is defined as *FM-STR\_SIZE*.

---

**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. |
|---------------------|---|

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. |
|-----------------------|--|

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. |
|-----------------------|---|

This is used to store the data buffer sizes.

---

|                      |                                       |
|----------------------|---------------------------------------|
| <i>FM-STR_PPIPEF</i> | Pointer to the pipe handler function. |
|----------------------|---------------------------------------|

See [FM-STR\\_PipeOpen](#) for more details.

---

**Internal types** The table describes the data types used internally by the FreeMASTER driver. The data types are declared in the platform-specific header file and they are not available in the application code.

|  |   |
|--|---|
| <i>FMSTR_U8</i>  | The smallest memory entity.   |
| On the vast majority of platforms, this is an unsigned 8-bit integer.                                    |   |
| On the 56F8xx DSP platform, this is defined as an unsigned 16-bit integer.                               |   |
| <i>FM-STR_U16</i>  | Unsigned 16-bit integer.  |
| <i>FM-STR_U32</i>  | Unsigned 32-bit integer.  |
| <i>FMSTR_S8</i>  | Signed 8-bit integer.   |
| <i>FM-STR_S16</i>  | Signed 16-bit integer.  |
| <i>FM-STR_S32</i>  | Signed 32-bit integer.  |
| <i>FM-STR_FLOAT</i>  | 4-byte standard IEEE floating-point type.                               |
| <i>FM-STR_FLAGS</i>  | Data type forming a union with a structure of flag bit-fields.          |
| <i>FM-STR_SIZE8</i>  | Data type holding a general size value, at least 8 bits wide.           |
| <i>FM-STR_INDEX</i>  | General for-loop index. Must be signed, at least 16 bits wide.          |
| <i>FM-STR_BCHR</i>   | A single character in the communication buffer.                         |
| Typically, this is an 8-bit unsigned integer, except for the DSP platforms where it is a 16-bit integer. |   |
| <i>FM-STR_BPTR</i>   | A pointer to the communication buffer (an array of <i>FMSTR_BCHR</i> ). |

## Document references

### Links

- This document online: <https://mcuxpresso.nxp.com/mcuxsdk/latest/html/middleware/freemaster/doc/index.html>

- FreeMASTER tool home: [www.nxp.com/freemaster](http://www.nxp.com/freemaster)
- FreeMASTER community area: [community.nxp.com/community/freemaster](http://community.nxp.com/community/freemaster)
- FreeMASTER GitHub code repo: <https://github.com/nxp-mcuxpresso/mcux-freemaster>
- MCUXpresso SDK home: [www.nxp.com/mcuxpresso](http://www.nxp.com/mcuxpresso)
- MCUXpresso SDK builder: [mcuxpresso.nxp.com/en](http://mcuxpresso.nxp.com/en)

## Documents

- *FreeMASTER Usage Serial Driver Implementation* (document [AN4752](#))
- *Integrating FreeMASTER Time Debugging Tool With CodeWarrior For Microcontrollers v10.X Project* (document [AN4771](#))
- *Flash Driver Library For MC56F847xx And MC56F827xx DSC Family* (document [AN4860](#))

**Revision history** This Table summarizes the changes done to this document since the initial release.

| Revision | Date    | Description  |
|----------|---------|--|
| 1.0      | 03/2006 | Limited initial release  |
| 2.0      | 09/2007 | Updated for FreeMASTER version. New Freescale document template used.  |
| 2.1      | 12/2007 | Added description of the new Fast Recorder feature and its API.  |
| 2.2      | 04/2010 | Added support for MPC56xx platform, Added new API for use CAN interface.   |
| 2.3      | 04/2011 | Added support for Kxx Kinetis platform and MQX operating system.   |
| 2.4      | 06/2011 | Serial driver update, adds support for USB CDC interface.  |
| 2.5      | 08/2011 | Added Packet Driven BDM interface.   |
| 2.7      | 12/2013 | Added FLEXCAN32 interface, byte access and isr callback configuration option.  |
| 2.8      | 06/2014 | Removed obsolete license text, see the software package content for up-to-date license.  |
| 2.9      | 03/2015 | Update for driver version 1.8.2 and 1.9: FreeMASTER Pipes, TSA Active Content, LIN Transport Layer support, DEBUG-TX communication troubleshooting, Kinetis SDK support.   |
| 3.0      | 08/2016 | Update for driver version 2.0: Added support for MPC56xx, MPC57xx, KEAxx and S32Kxx platforms. New NXP document template as well as new license agreement used. added MCAN interface. Folders structure at the installation destination was rearranged.  |
| 4.0      | 04/2019 | Update for driver released as part of FreeMASTER v3.0 and MCUXpresso SDK 2.6. Updated to match new V4 serial communication protocol and new configuration options. This version of the document removes substantial portion of outdated information related to S08, S12, ColdFire, Power and other legacy platforms. |
| 4.1      | 04/2020 | Minor update for FreeMASTER driver included in MCUXpresso SDK 2.8.   |
| 4.2      | 09/2020 | Added example applications description and information about the MCUXpresso Config Tools. Fixed the pipe-related API description.  |
| 4.3      | 10/2024 | Added description of Network and Segger J-Link RTT interface configuration. Accompanying the MCUXpresso SDK version 24.12.00.  |
| 4.4      | 04/2025 | Added Zephyr-specific information. Accompanying the MCUXpresso SDK version 25.06.00.   |

# Chapter 4

## RTOS

### 4.1 FreeRTOS

#### 4.1.1 FreeRTOS kernel

Open source RTOS kernel for small devices.

[FreeRTOS kernel for MCUXpresso SDK Readme](#)

[FreeRTOS kernel for MCUXpresso SDK ChangeLog](#)

[FreeRTOS kernel Readme](#)

#### 4.1.2 FreeRTOS drivers

This is set of NXP provided FreeRTOS reentrant bus drivers.

#### 4.1.3 backoffalgorithm

Algorithm for calculating exponential backoff with jitter for network retry attempts.

[Readme](#)

#### 4.1.4 corehttp

C language HTTP client library designed for embedded platforms.

#### 4.1.5 corejson

JSON parser.

**Readme**

#### **4.1.6 coremqtt**

MQTT publish/subscribe messaging library.

#### **4.1.7 corepkcs11**

PKCS #11 key management library.

**Readme**

#### **4.1.8 freertos-plus-tcp**

Open source RTOS FreeRTOS Plus TCP.

**Readme**