



# MCUXpresso SDK Documentation

Release 25.12.00-pvw2



NXP  
Nov 14, 2025



# Table of contents

|          |                                                    |            |
|----------|----------------------------------------------------|------------|
| <b>1</b> | <b>LPCXpresso54S018</b>                            | <b>3</b>   |
| 1.1      | Overview                                           | 3          |
| 1.2      | Getting Started with MCUXpresso SDK Package        | 4          |
| 1.2.1    | Getting Started with MCUXpresso SDK Package        | 4          |
| 1.3      | Getting Started with MCUXpresso SDK GitHub         | 56         |
| 1.3.1    | Getting Started with MCUXpresso SDK Repository     | 56         |
| 1.4      | Release Notes                                      | 69         |
| 1.4.1    | MCUXpresso SDK Release Notes                       | 69         |
| 1.5      | ChangeLog                                          | 74         |
| 1.5.1    | MCUXpresso SDK Changelog                           | 74         |
| 1.6      | Driver API Reference Manual                        | 128        |
| 1.7      | Middleware Documentation                           | 128        |
| 1.7.1    | FreeMASTER                                         | 128        |
| 1.7.2    | AWS IoT                                            | 128        |
| 1.7.3    | FreeRTOS                                           | 128        |
| 1.7.4    | lwIP                                               | 128        |
| 1.7.5    | File systemFatfs                                   | 128        |
| <b>2</b> | <b>LPC54S018</b>                                   | <b>129</b> |
| 2.1      | AES: AES encryption decryption driver              | 129        |
| 2.2      | Clock Driver                                       | 133        |
| 2.3      | CRC: Cyclic Redundancy Check Driver                | 163        |
| 2.4      | CTIMER: Standard counter/timers                    | 166        |
| 2.5      | DMA: Direct Memory Access Controller Driver        | 176        |
| 2.6      | DMIC: Digital Microphone                           | 193        |
| 2.7      | DMIC DMA Driver                                    | 193        |
| 2.8      | DMIC Driver                                        | 195        |
| 2.9      | EMC: External Memory Controller Driver             | 204        |
| 2.10     | FLEXCOMM: FLEXCOMM Driver                          | 211        |
| 2.11     | FLEXCOMM Driver                                    | 211        |
| 2.12     | FMEAS: Frequency Measure Driver                    | 211        |
| 2.13     | GINT: Group GPIO Input Interrupt Driver            | 212        |
| 2.14     | I2C: Inter-Integrated Circuit Driver               | 215        |
| 2.15     | I2C DMA Driver                                     | 215        |
| 2.16     | I2C Driver                                         | 217        |
| 2.17     | I2C Master Driver                                  | 221        |
| 2.18     | I2C Slave Driver                                   | 230        |
| 2.19     | I2S: I2S Driver                                    | 239        |
| 2.20     | I2S DMA Driver                                     | 239        |
| 2.21     | I2S Driver                                         | 243        |
| 2.22     | IAP: In Application Programming Driver             | 251        |
| 2.23     | INPUTMUX: Input Multiplexing Driver                | 254        |
| 2.24     | Common Driver                                      | 260        |
| 2.25     | ADC: 12-bit SAR Analog-to-Digital Converter Driver | 273        |
| 2.26     | ENET: Ethernet Driver                              | 284        |
| 2.27     | GPIO: General Purpose I/O                          | 311        |

|          |                                                                       |            |
|----------|-----------------------------------------------------------------------|------------|
| 2.28     | IOCON: I/O pin configuration                                          | 314        |
| 2.29     | LCDC: LCD Controller Driver                                           | 315        |
| 2.30     | MCAN: Controller Area Network Driver                                  | 325        |
| 2.31     | MRT: Multi-Rate Timer                                                 | 348        |
| 2.32     | OTP: One-Time Programmable memory and API                             | 352        |
| 2.33     | PINT: Pin Interrupt and Pattern Match Driver                          | 356        |
| 2.34     | Power Driver                                                          | 364        |
| 2.35     | PUF: Physical Unclonable Function                                     | 369        |
| 2.36     | Reset Driver                                                          | 371        |
| 2.37     | RIT: Repetitive Interrupt Timer                                       | 376        |
| 2.38     | RNG: Random Number Generator                                          | 379        |
| 2.39     | RTC: Real Time Clock                                                  | 380        |
| 2.40     | SCTimer: SCTimer/PWM (SCT)                                            | 385        |
| 2.41     | SDIF: SD/MMC/SDIO card interface                                      | 402        |
| 2.42     | SHA: SHA encryption decryption driver                                 | 418        |
| 2.43     | Sha_algorithm_level_api                                               | 420        |
| 2.44     | SPI: Serial Peripheral Interface Driver                               | 421        |
| 2.45     | SPI DMA Driver                                                        | 421        |
| 2.46     | SPI Driver                                                            | 425        |
| 2.47     | SPIFI: SPIFI flash interface driver                                   | 434        |
| 2.48     | SPIFI DMA Driver                                                      | 443        |
| 2.49     | SPIFI Driver                                                          | 443        |
| 2.50     | USART: Universal Synchronous/Asynchronous Receiver/Transmitter Driver | 443        |
| 2.51     | USART DMA Driver                                                      | 443        |
| 2.52     | USART Driver                                                          | 445        |
| 2.53     | UTICK: MicroTick Timer Driver                                         | 461        |
| 2.54     | WWDT: Windowed Watchdog Timer Driver                                  | 462        |
| <b>3</b> | <b>Middleware</b>                                                     | <b>467</b> |
| 3.1      | Motor Control                                                         | 467        |
| 3.1.1    | FreeMASTER                                                            | 467        |
| <b>4</b> | <b>RTOS</b>                                                           | <b>505</b> |
| 4.1      | FreeRTOS                                                              | 505        |
| 4.1.1    | FreeRTOS kernel                                                       | 505        |
| 4.1.2    | FreeRTOS drivers                                                      | 505        |
| 4.1.3    | backoffalgorithm                                                      | 505        |
| 4.1.4    | corehttp                                                              | 505        |
| 4.1.5    | corejson                                                              | 505        |
| 4.1.6    | coremqtt                                                              | 506        |
| 4.1.7    | corepkcs11                                                            | 506        |
| 4.1.8    | freertos-plus-tcp                                                     | 506        |

This documentation contains information specific to the lpcxpresso54s018 board.



# Chapter 1

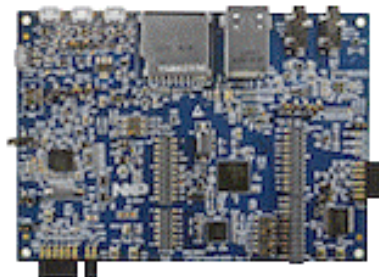
## LPCXpresso54S018

### 1.1 Overview

The LPCXpresso54S018 board provides a powerful and flexible platform for easy evaluation of the LPC54S00 family of microcontrollers. This board is an LPCXpresso V3 style, the latest generation of the original and highly successful LPCXpresso board family. These boards provide Arduino UNO compatible shield connectors with additional expansion ports around the Arduino footprint and also a PMod/host interface port. These boards feature an on-board LPC-Link2 debug probe based on the LPC4322 MCU for a performance debug experience over high speed USB, with easy firmware update options to support CMSIS-DSP or a special version of J-link OBD from SEGGER. The board can also be used with an external debug probe such as those from SEGGER and P&E.

The LPC54S00 series is fully supported by NXP's MCUXpresso suite of free software and tools, which include an Eclipse-based IDE, configuration tools and extensive SDK drivers/examples available at <https://mcuxpresso.nxp.com>. MCUXpresso SDK includes project files for use with IDEs from lead partners Keil and IAR, and these IDEs are also fully supported by the MCUXpresso pin, clock and peripheral configuration tools.

The LPCXpresso54S018 board includes an extensive set of connectors and on-board peripherals to enable full evaluation of the target MCU and development of highly functional prototypes. The on-board peripherals are complemented by a set of drivers and examples in the MCUXpresso SDK.



MCU device and part on board is shown below:

- Device: LPC54S018
- PartNumber: LPC54S018JET180

## 1.2 Getting Started with MCUXpresso SDK Package

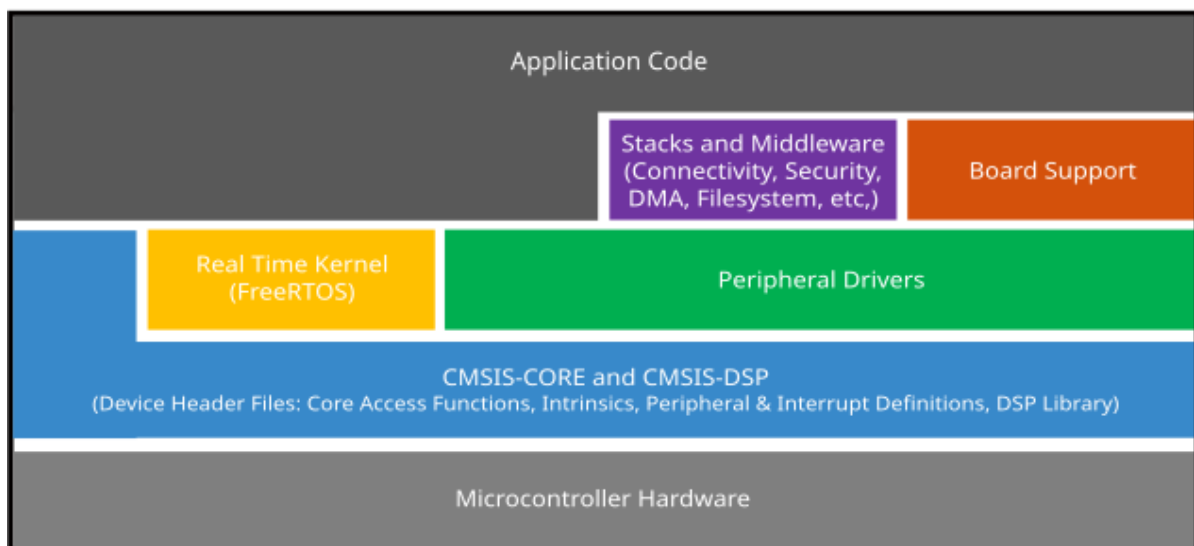
### 1.2.1 Getting Started with MCUXpresso SDK Package

#### 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.

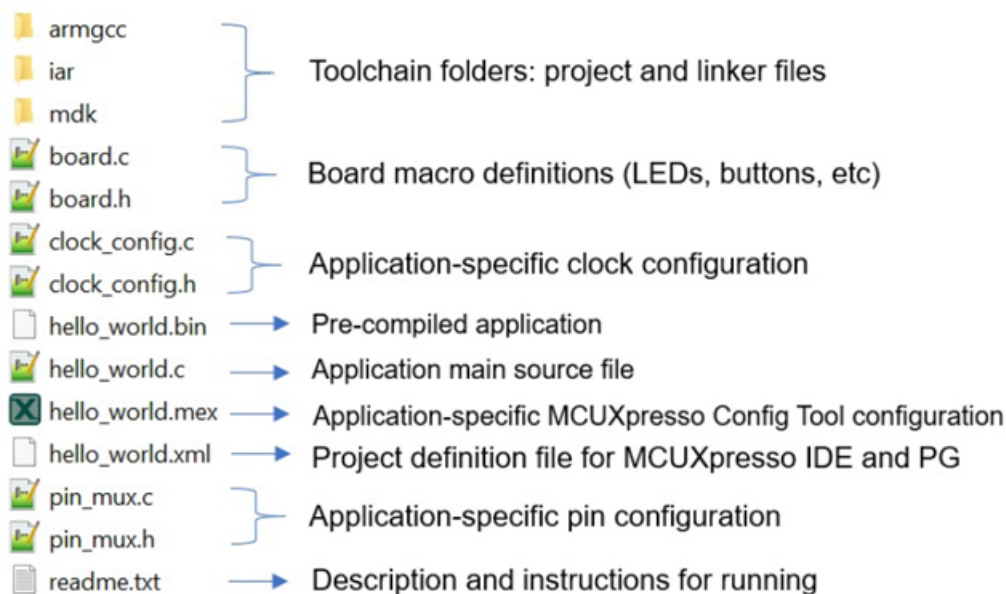


- `rtos_examples`: Basic FreeRTOS OS examples that show the use of various RTOS objects (semaphores, queues, and so on) and interfaces with the MCUXpresso SDK's RTOS drivers
- `usb_examples`: Applications that use the USB host/device/OTG stack.

**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/<devices_name>/project`: Project template used in CMSIS PACK new project creation

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

## Run a demo using MCUXpresso IDE

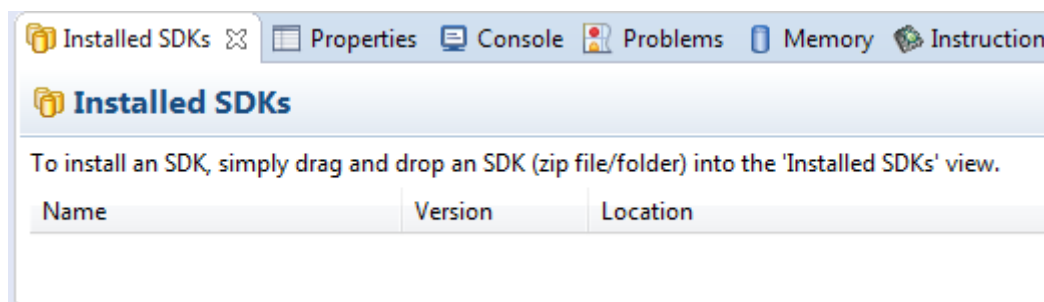
**Note:** Ensure that the MCUXpresso IDE toolchain is included when generating the MCUXpresso SDK package.

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

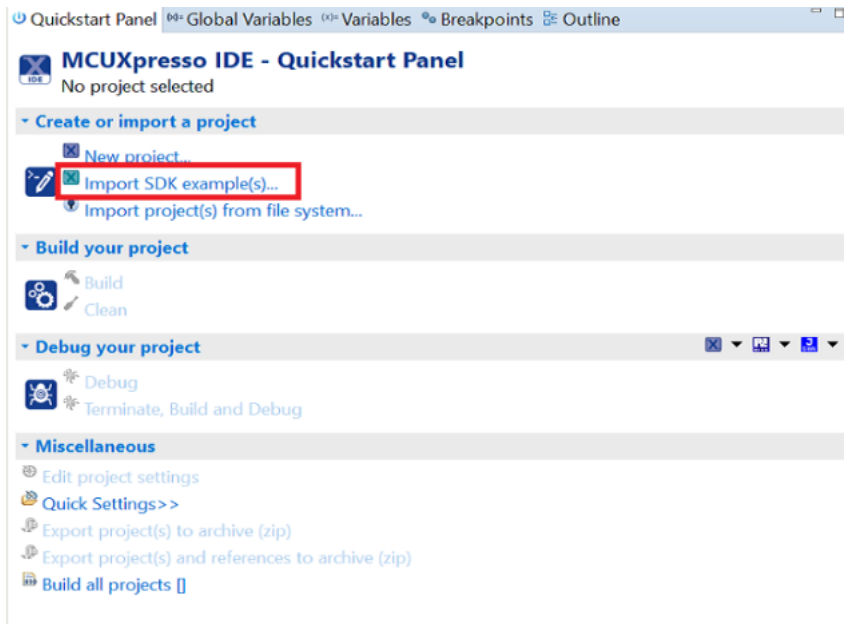
**Select the workspace location** Every time MCUXpresso IDE launches, it prompts the user to select a workspace location. MCUXpresso IDE is built on top of Eclipse which uses workspace to store information about its current configuration, and in some use cases, source files for the projects are in the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

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

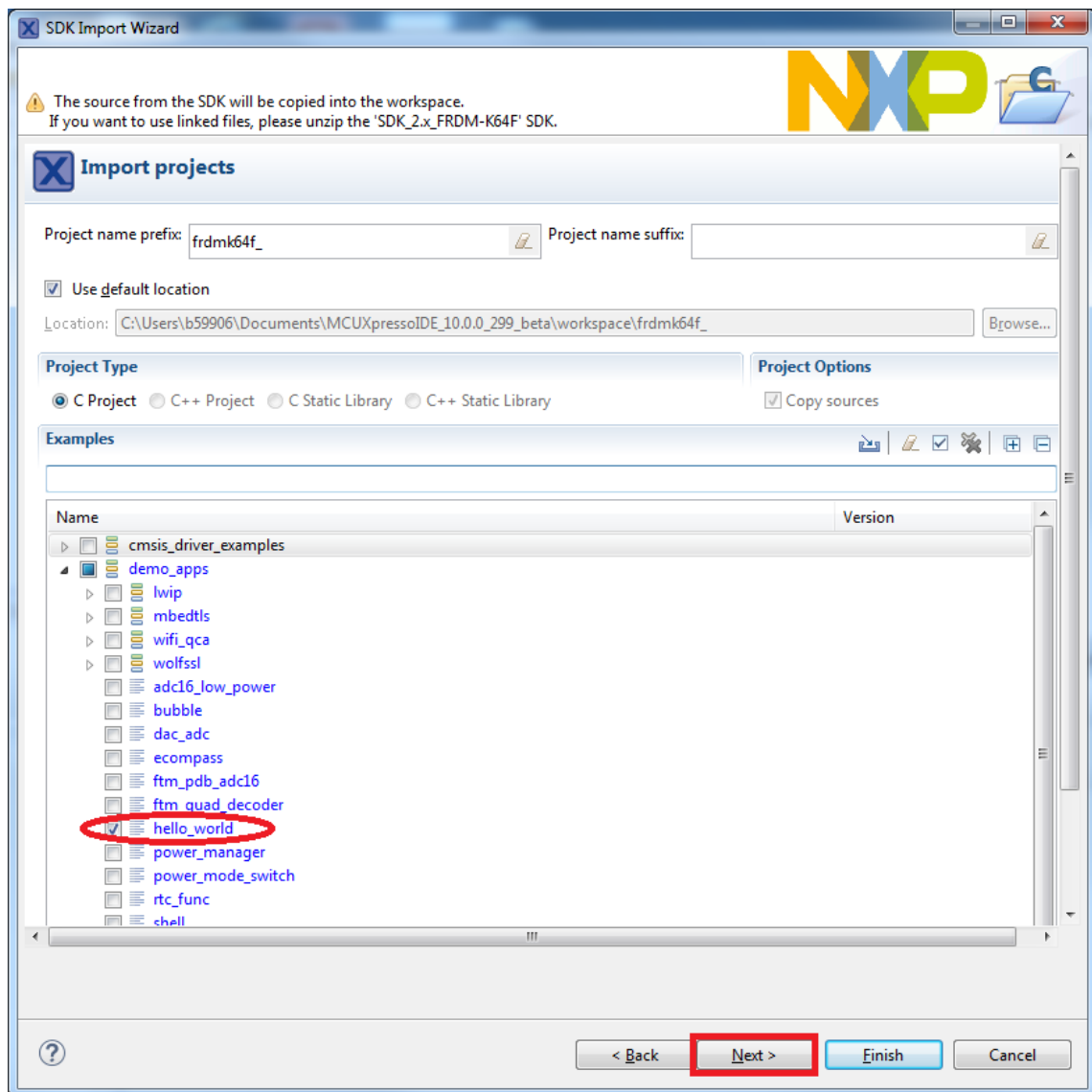
1. Drag and drop the SDK zip file into the **Installed SDKs** view to install an SDK. In the window that appears, click **OK** and wait until the import has finished.



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



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



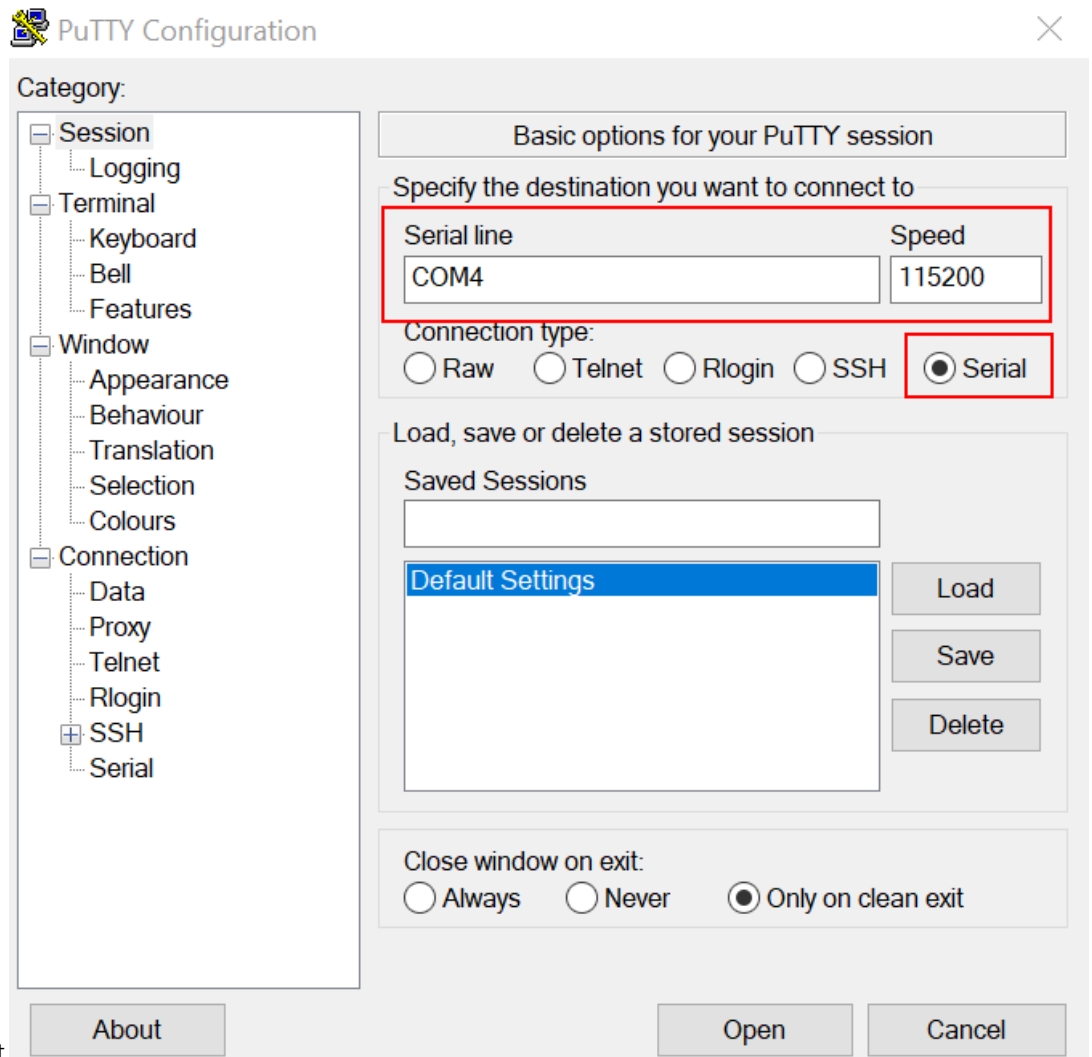
5. Ensure **Redlib: Use floating-point version of printf** is selected if the example prints floating-point numbers on the terminal for demo applications such as `adc_basic`, `adc_burst`, `adc_dma`, and `adc_interrupt`. Otherwise, it is not necessary to select this option. Then, click **Finish**.

**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:

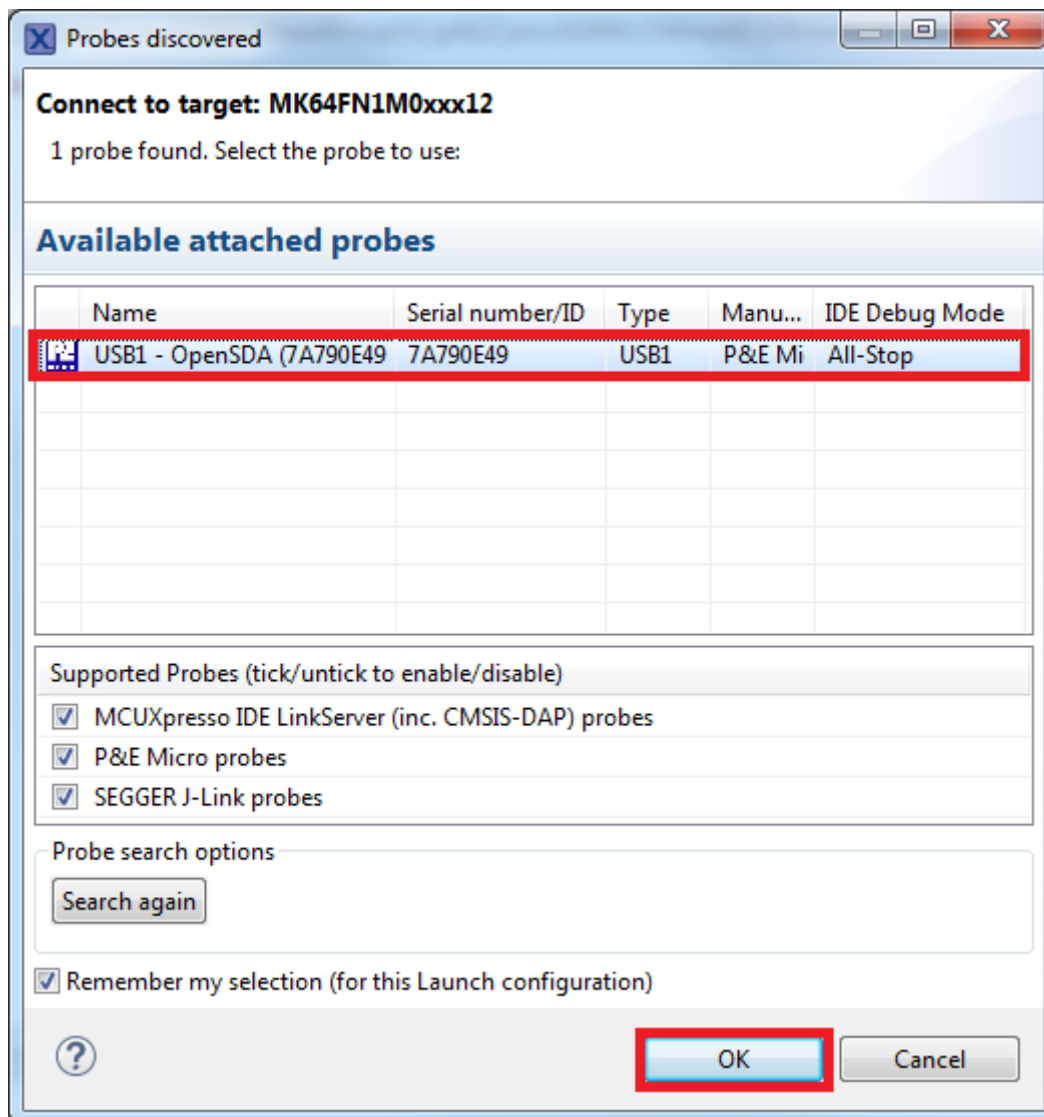
1. Ensure the host driver for the debugger firmware has been installed. See [On-board debugger](#).
2. Connect the development platform to your PC via a USB cable.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM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



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

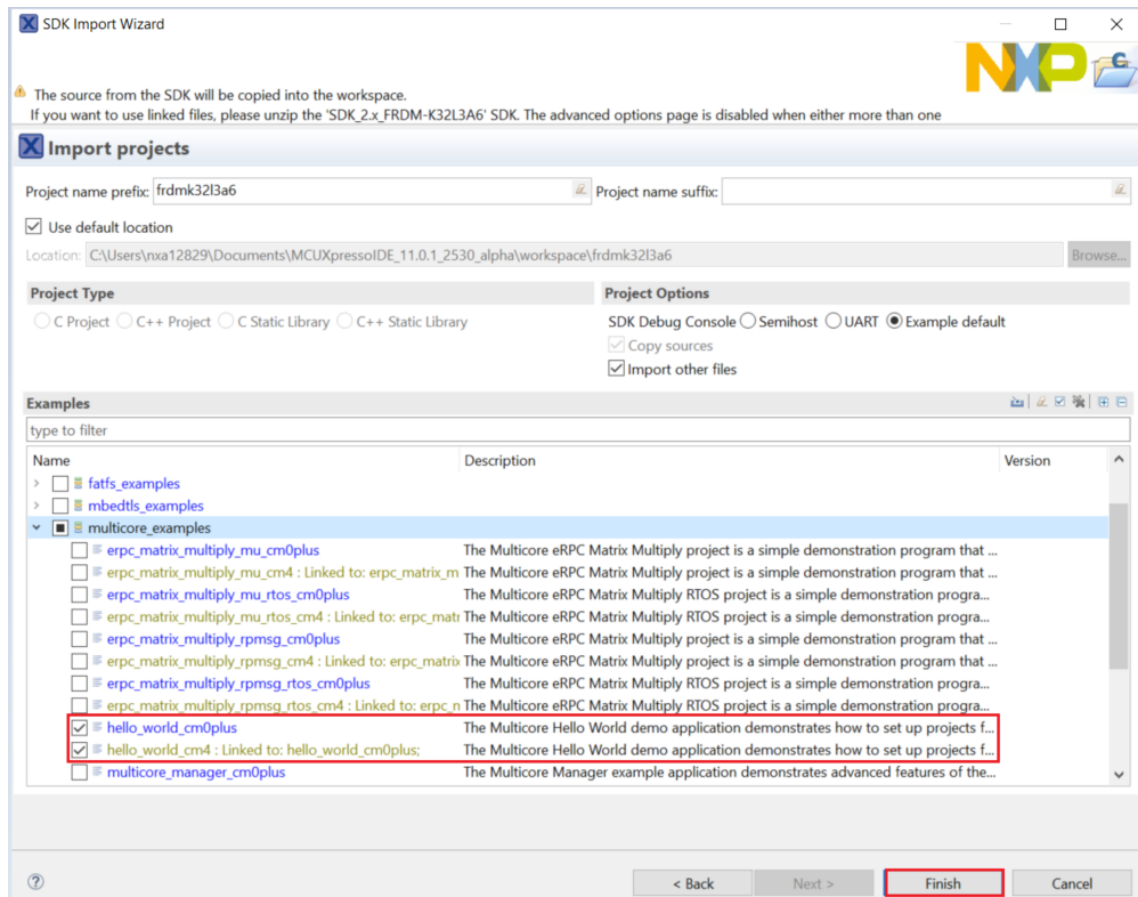


The `hello_world` application is now running and a banner is displayed on the terminal. If not, check your terminal settings and connections.

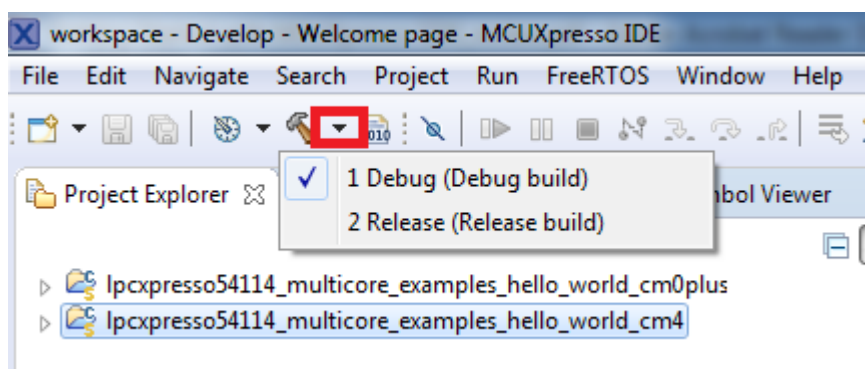


**Build a multicore example application** This section describes the steps required to configure MCUXpresso IDE to build, run, and debug multicore example applications. The following steps can be applied to any multicore example application in the MCUXpresso SDK. Here, the dual-core version of hello\_world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

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



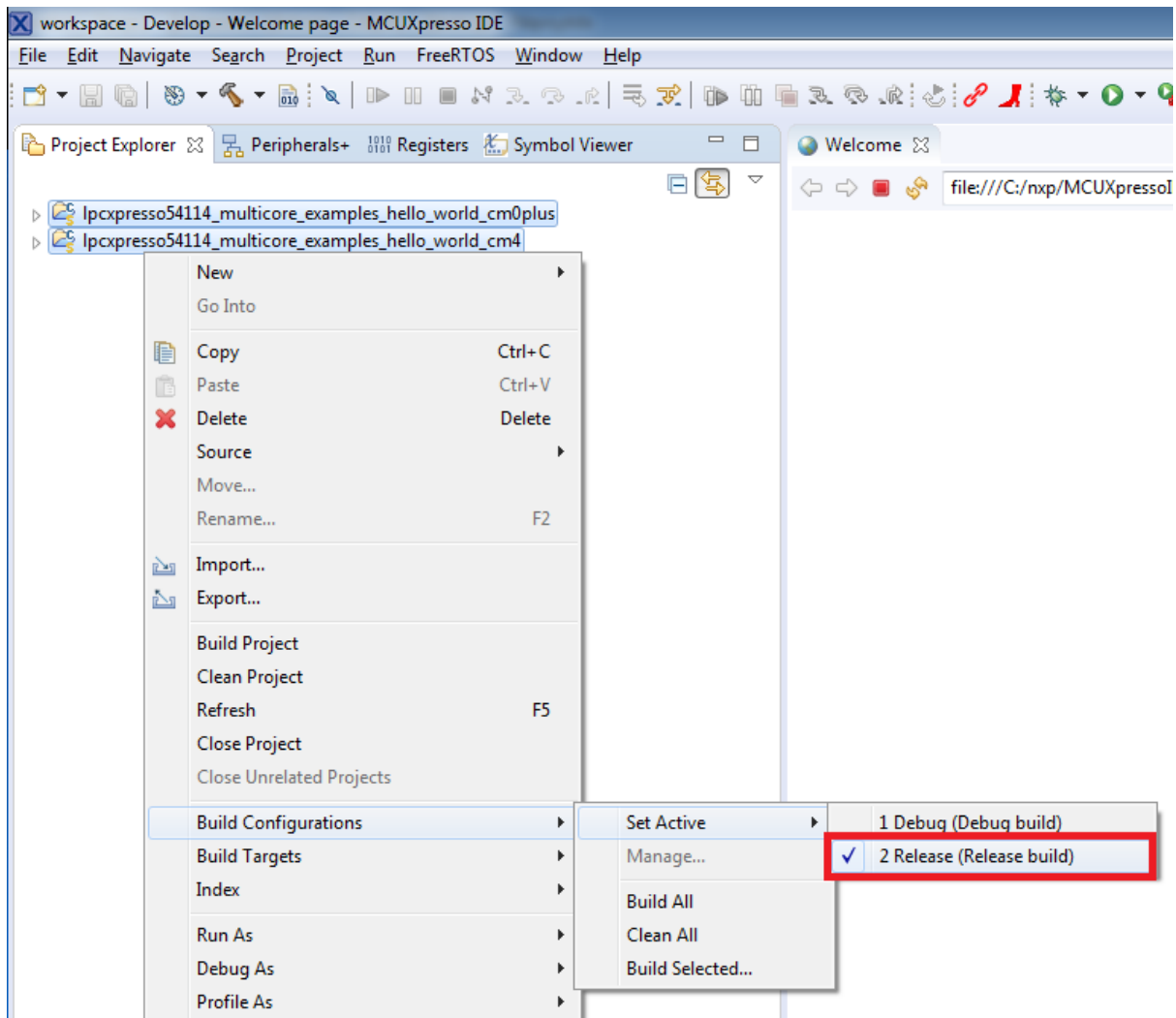
3. Now, two projects should be imported into the workspace. To start building the multicore application, highlight the `lpcxpresso54114_multicore_examples_hello_world_cm4` project (multicore master project) in the Project Explorer. Then choose the appropriate build target, **Debug** or **Release**, by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



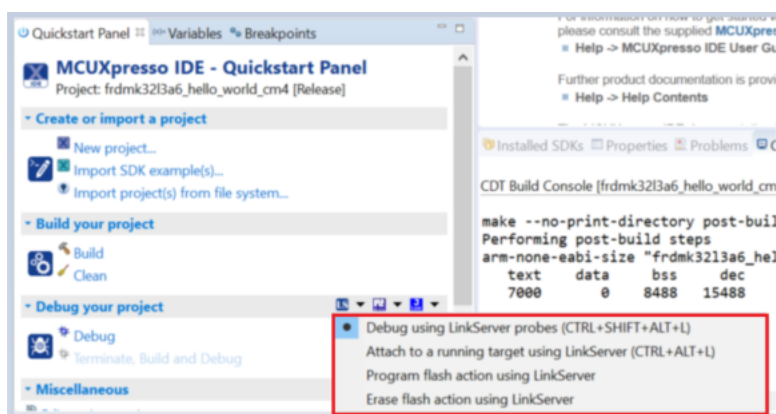
The project starts building after the build target is selected. Because of the project reference settings in multicore projects, triggering the build of the primary core application (cm4) also causes the referenced auxiliary core application (cm0plus) to build.

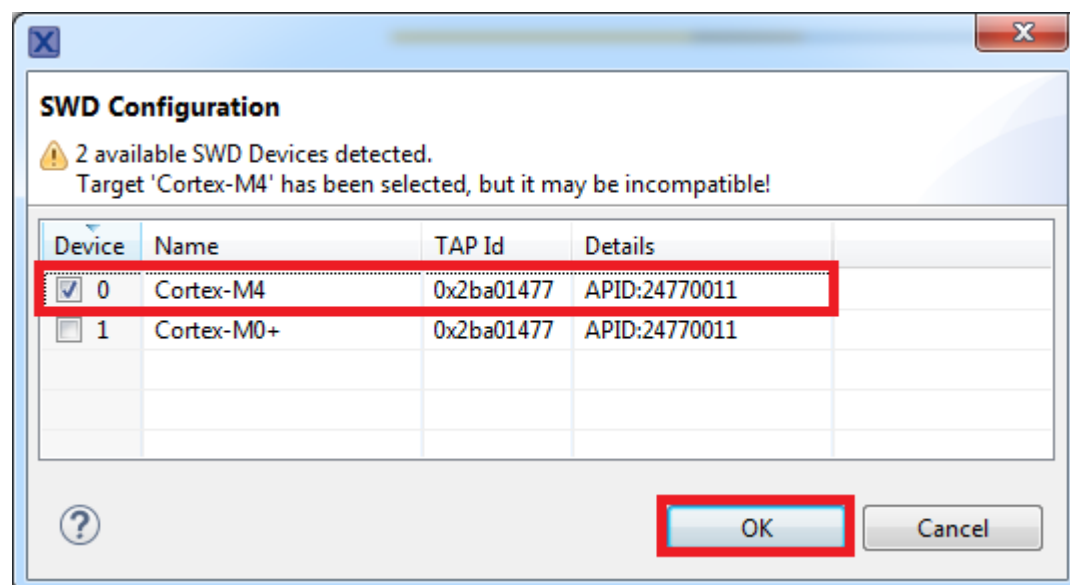
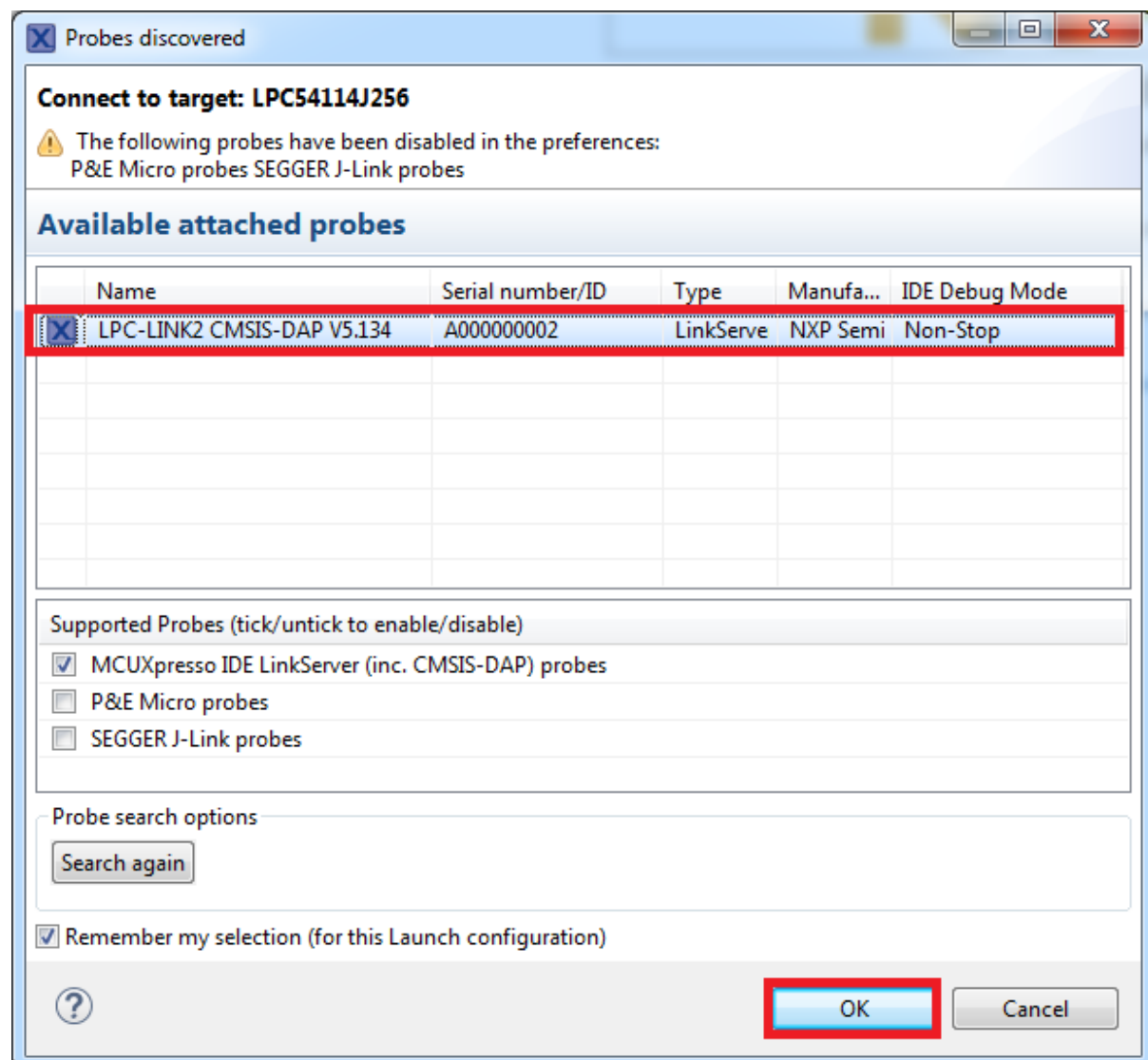
**Note:** When the **Release** build is requested, it is necessary to change the build configuration of both the primary and auxiliary core application projects first. To do this, select both projects in the Project Explorer view and then right click which displays the context-sensitive menu. Select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

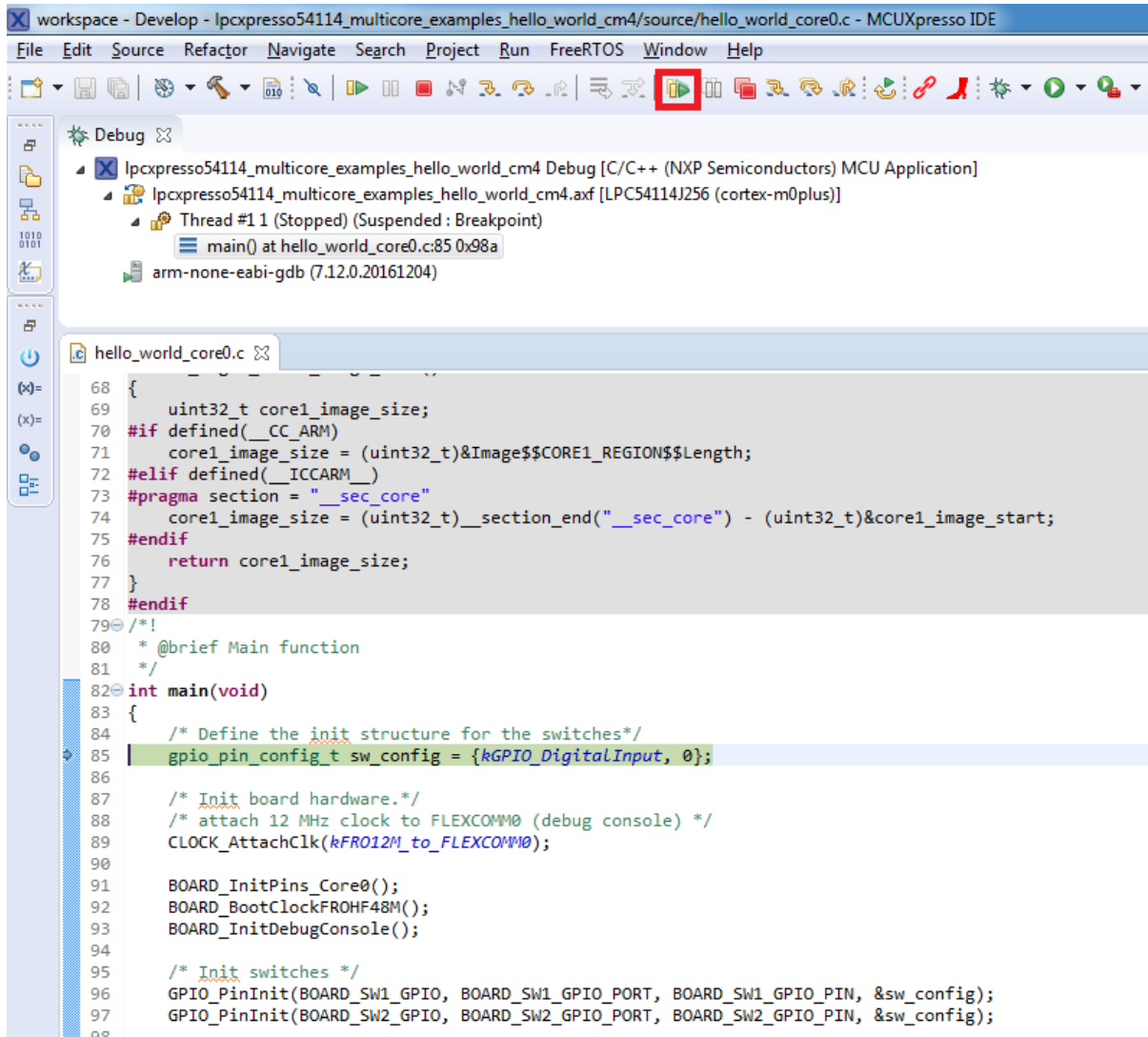




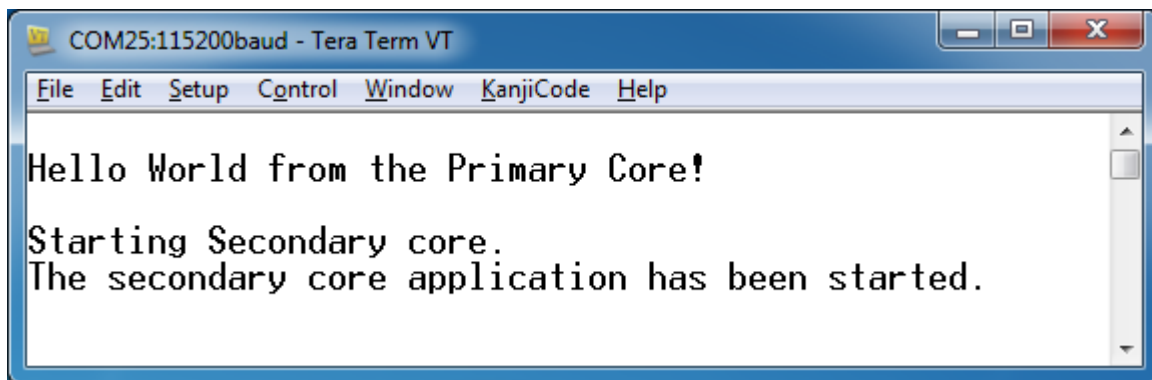
**Run a multicore example application** The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform all steps as described in **Run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





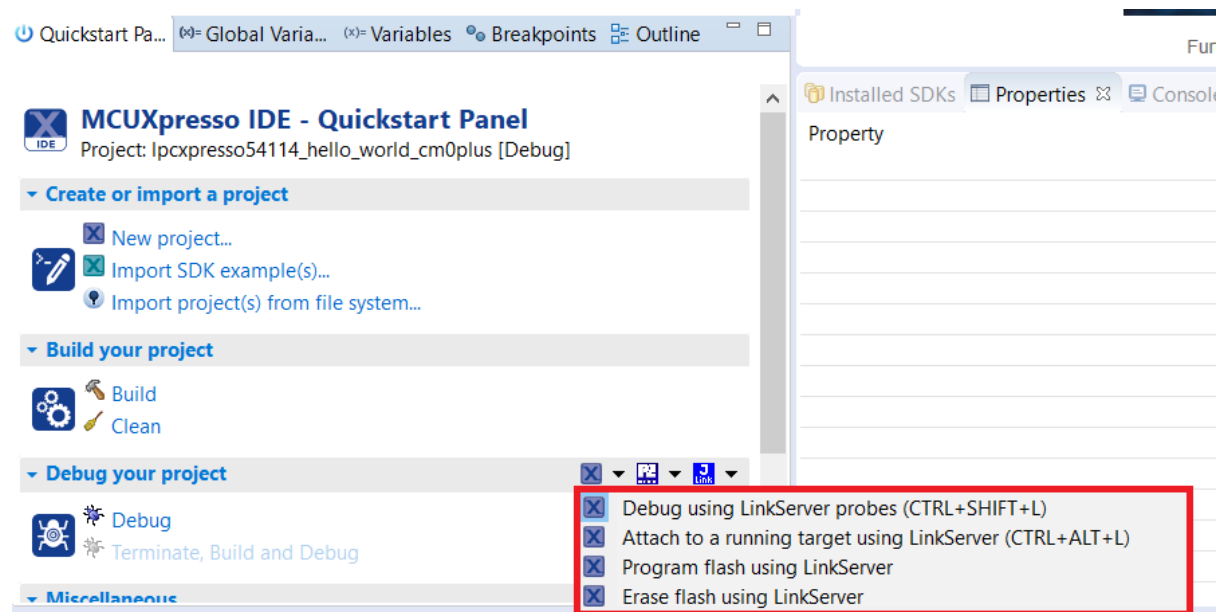


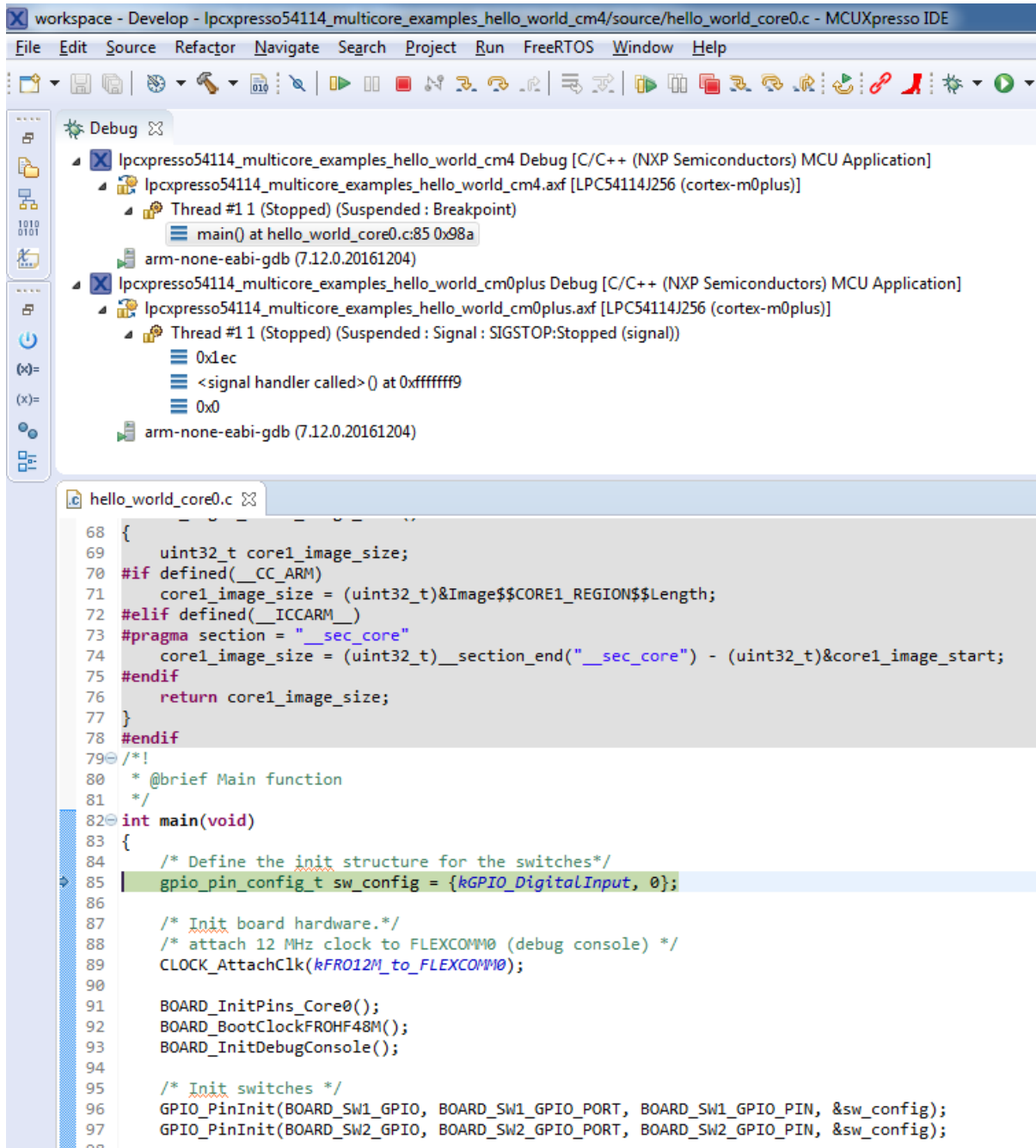
After clicking the “Resume All Debug sessions” button, the hello\_world multicore application runs and a banner is displayed on the terminal. If this is not the case, check your terminal settings and connections.



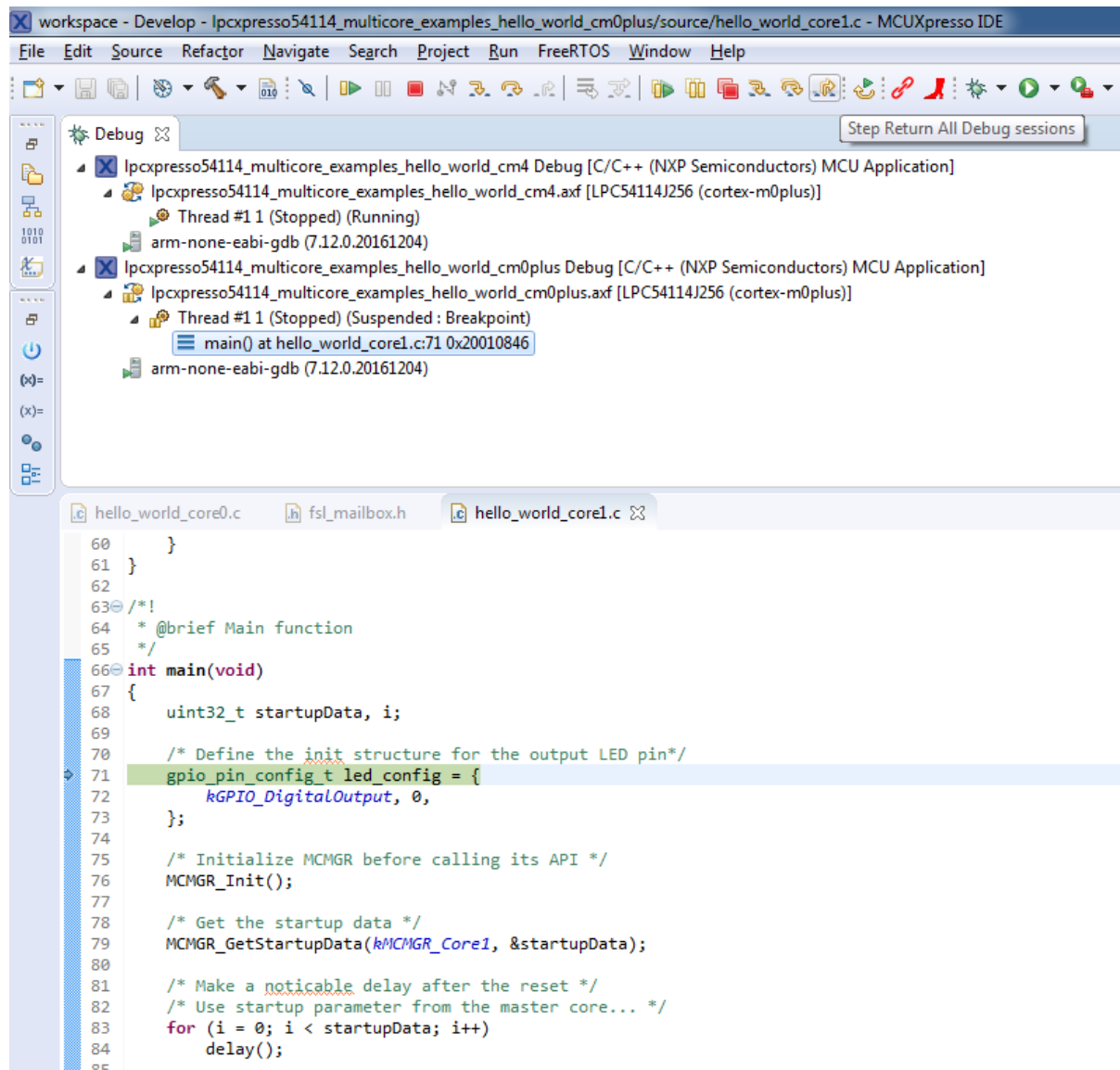
An LED controlled by the auxiliary core starts flashing, indicating that the auxiliary core has been released from the reset and running correctly. It is also possible to debug both sides of the multicore application in parallel. After creating the debug session for the primary core, perform same steps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

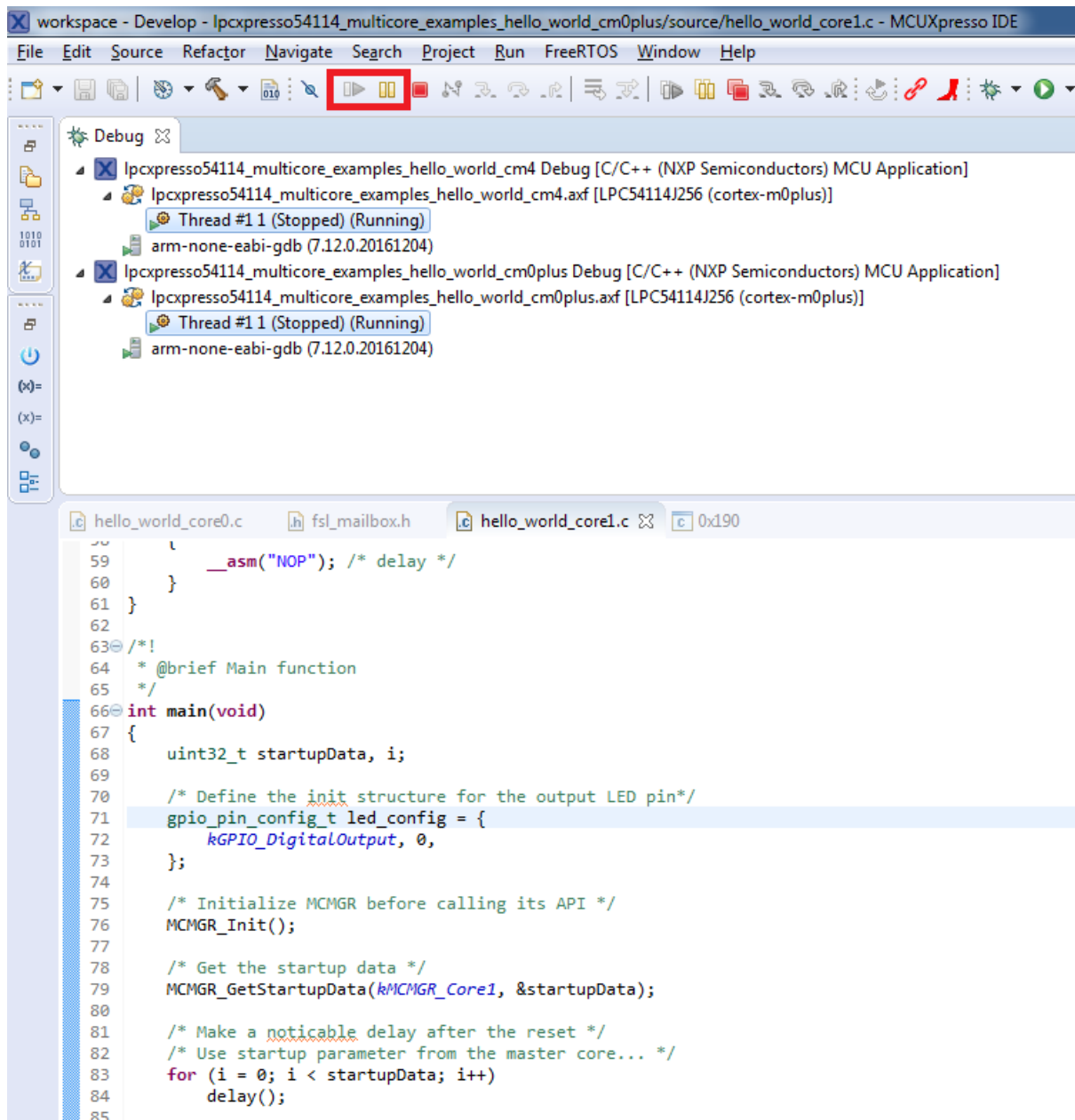




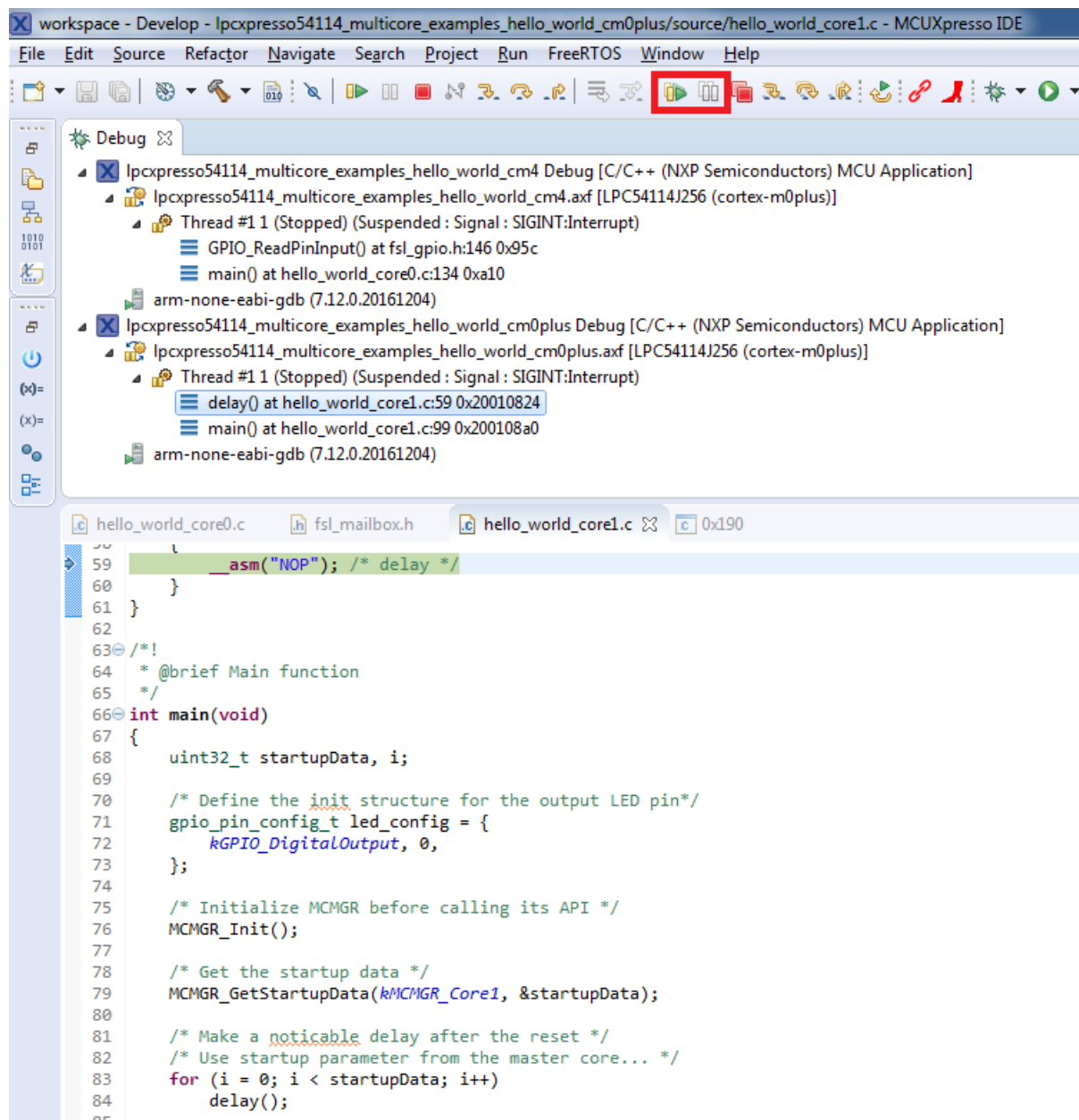
Now, the two debug sessions should be opened, and the debug controls can be used for both debug sessions depending on the debug session selection. Keep the primary core debug session selected by clicking the “Resume” button. The hello\_world multicore application then starts running. The primary core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



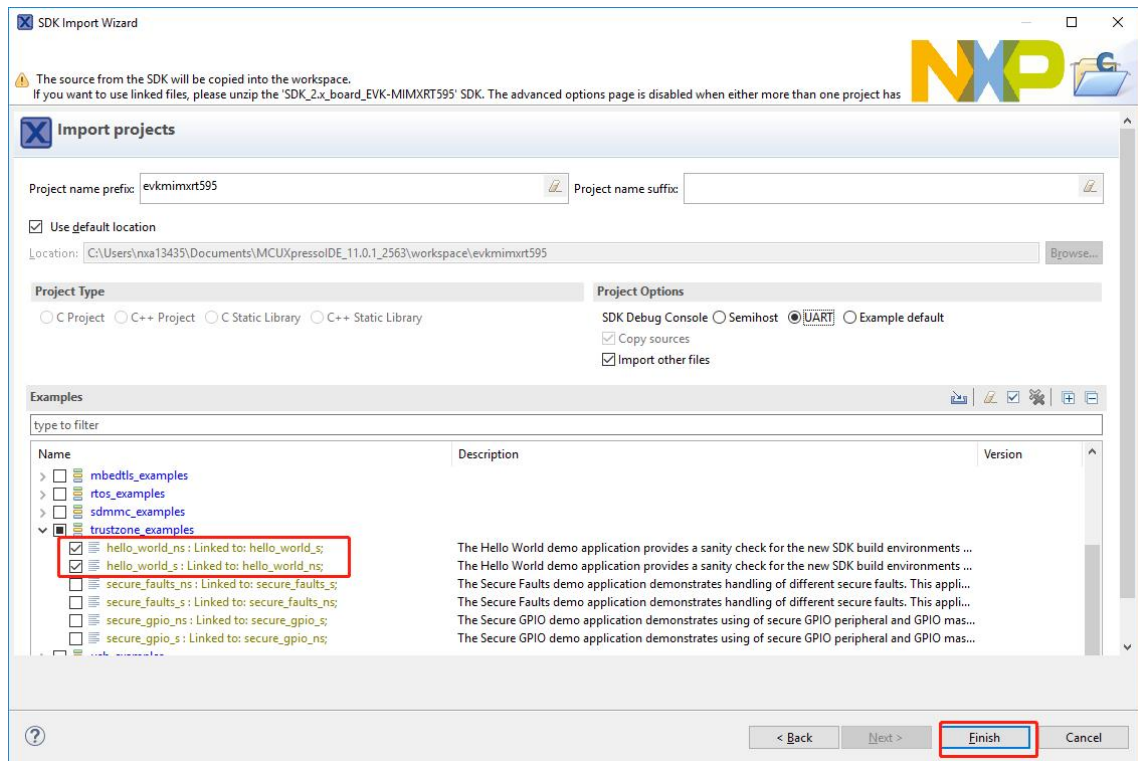




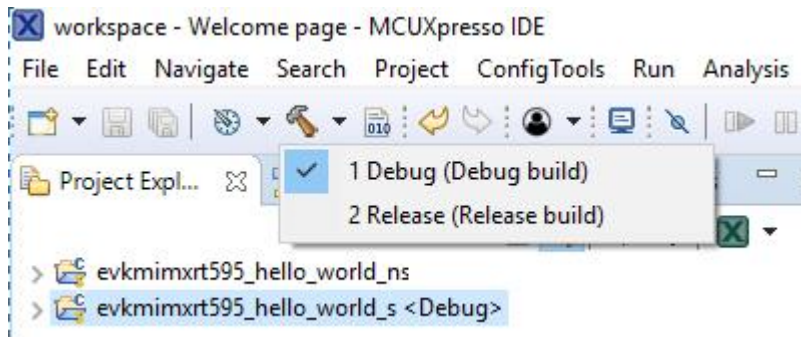
**Build a TrustZone example application** This section describes the steps required to configure MCUXpresso IDE to build, run, and debug TrustZone example applications. The TrustZone version of the `hello_world` example application targeted for the MIMXRT595-EVK hardware platform is used as an example,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



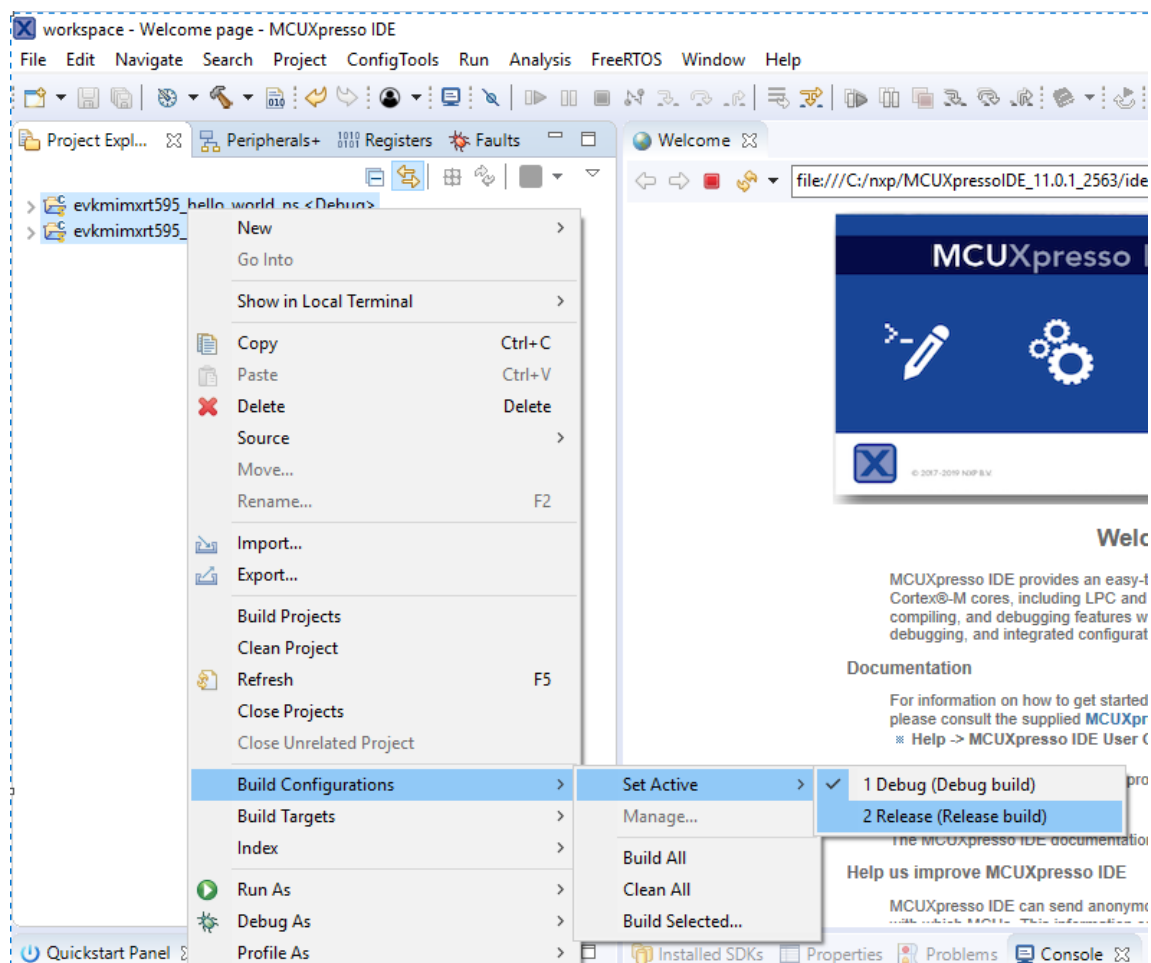


3. Now, two projects should be imported into the workspace. To start building the TrustZone application, highlight the `evkmimxrt595_hello_world_s` project (TrustZone master project) in the Project Explorer. Then, choose the appropriate build target, **Debug** or **Release**, by clicking the downward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



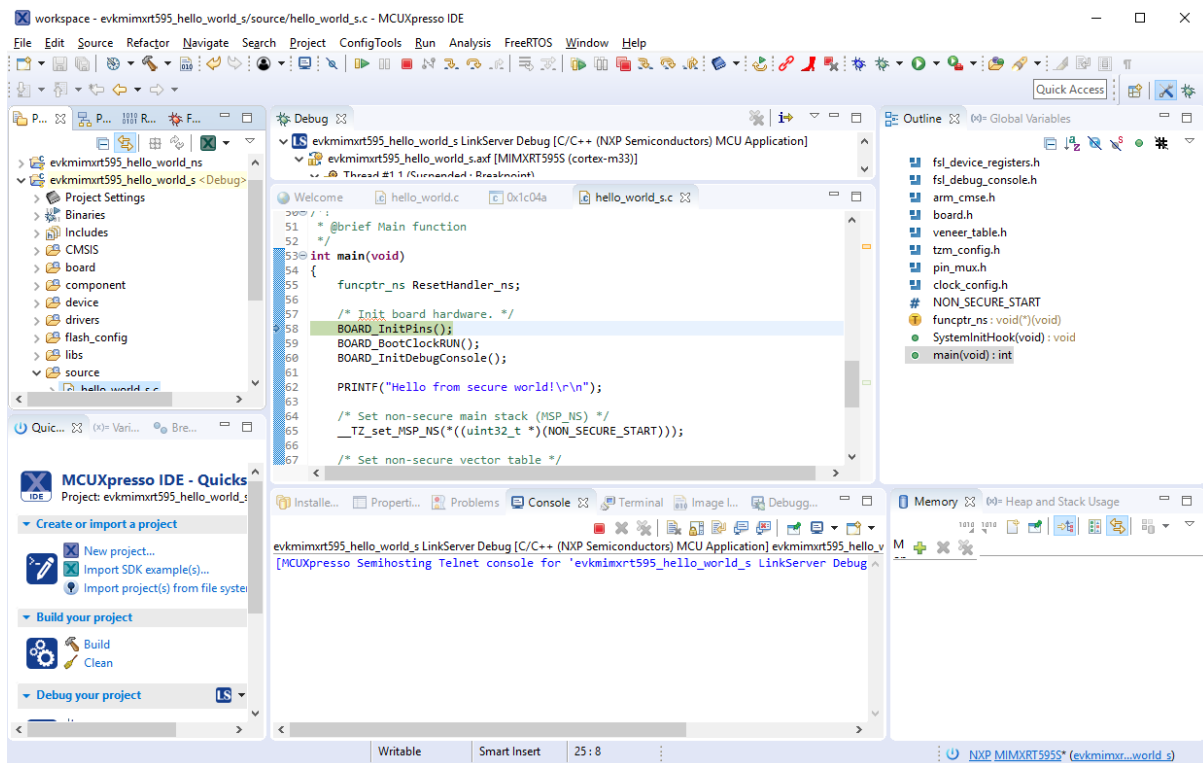
The project starts building after the build target is selected. It is requested to build the application for the secure project first, because the non-secure project must know the secure project since CMSE library when running the linker. It is not possible to finish the non-secure project linker when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

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



**Run a TrustZone example application** To download and run the application, perform all steps as described in **Run an example application**. These steps are common for single core, and TrustZone applications, ensuring <board\_name>\_hello\_world\_s is selected for debugging.

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



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

## Run a demo application using IAR

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

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

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

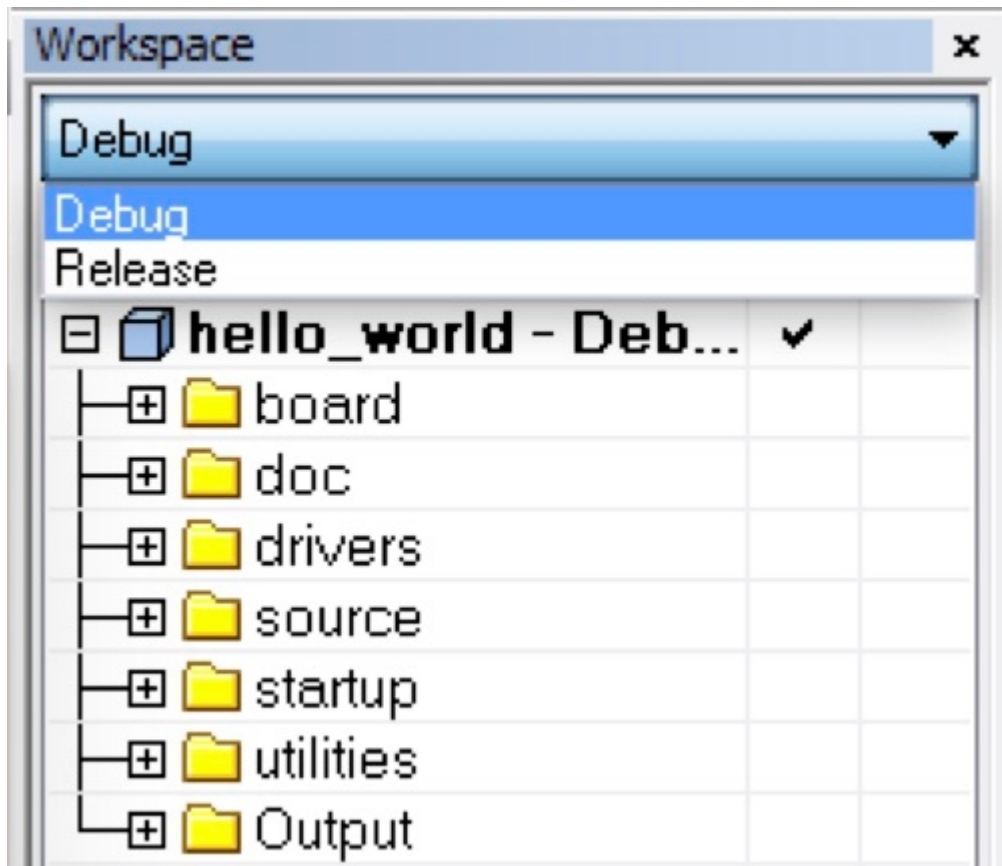
1. Open the desired demo application workspace. Most example application workspace files can be located using the following path:

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

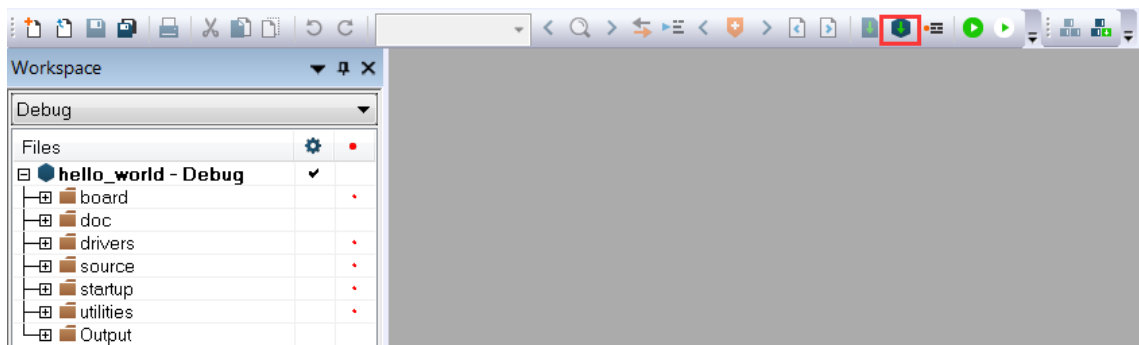
Other example applications may have additional folders in their path.

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

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



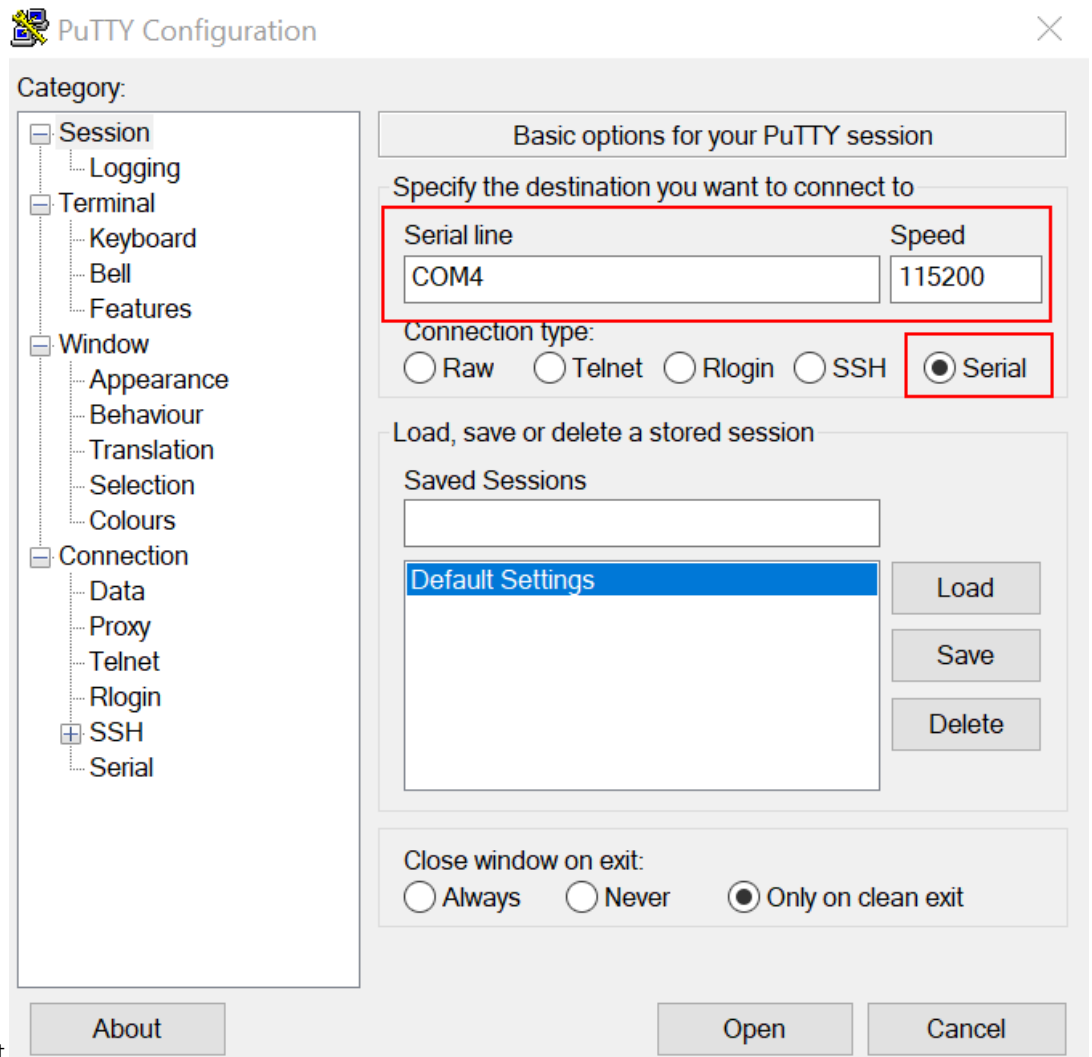
3. To build the demo application, click **Make**, highlighted in red in following figure.



4. The build completes without errors.

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

1. Ensure the host driver for the debugger firmware has been installed. See [On-board debugger](#).
2. Connect the development platform to your PC via USB cable.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (to determine the COM port number, see [How to determine COM port](#)). Configure the terminal with these settings:
  1. 115200 or 9600 baud rate, depending on your board (reference BOARD\_DEBUG\_UART\_BAUDRATE variable in the board.h file)
  2. No parity
  3. 8 data bits

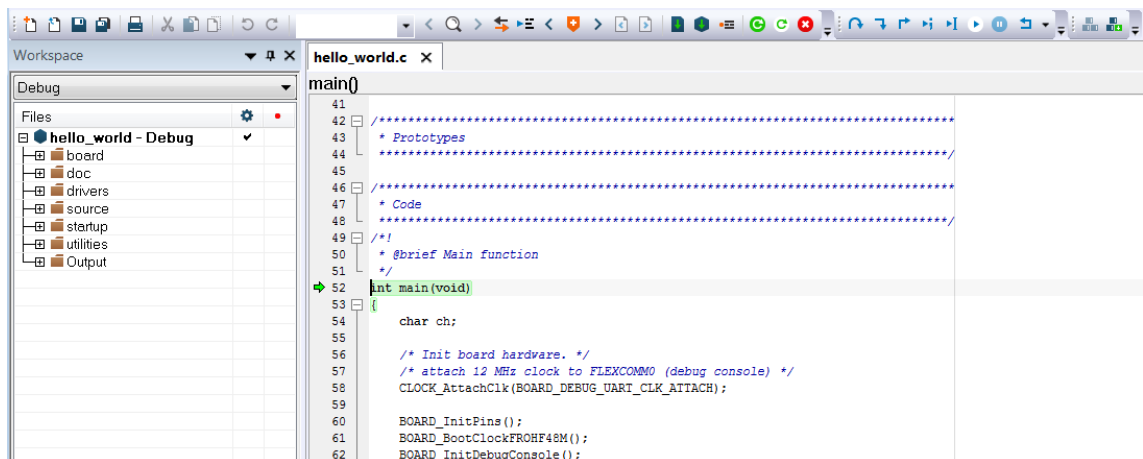


4. 1 stop bit

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



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



6. Run the code by clicking the **Go** button.

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



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

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

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

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

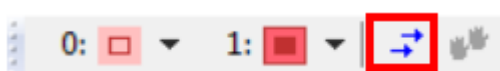
```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm4/iar/hello_world_cm4.eww
```

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

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

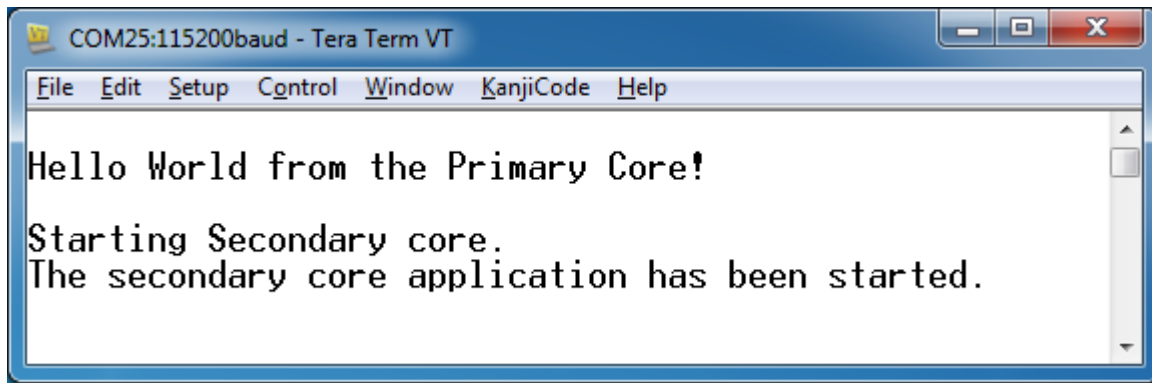
After clicking the “Download and Debug” button, the auxiliary core project is opened in the separate EWARM instance. Both the primary and auxiliary images are loaded into the device flash memory and the primary core application is executed. It stops at the default C language entry point in the `*main()*function`.

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



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





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



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

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

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

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

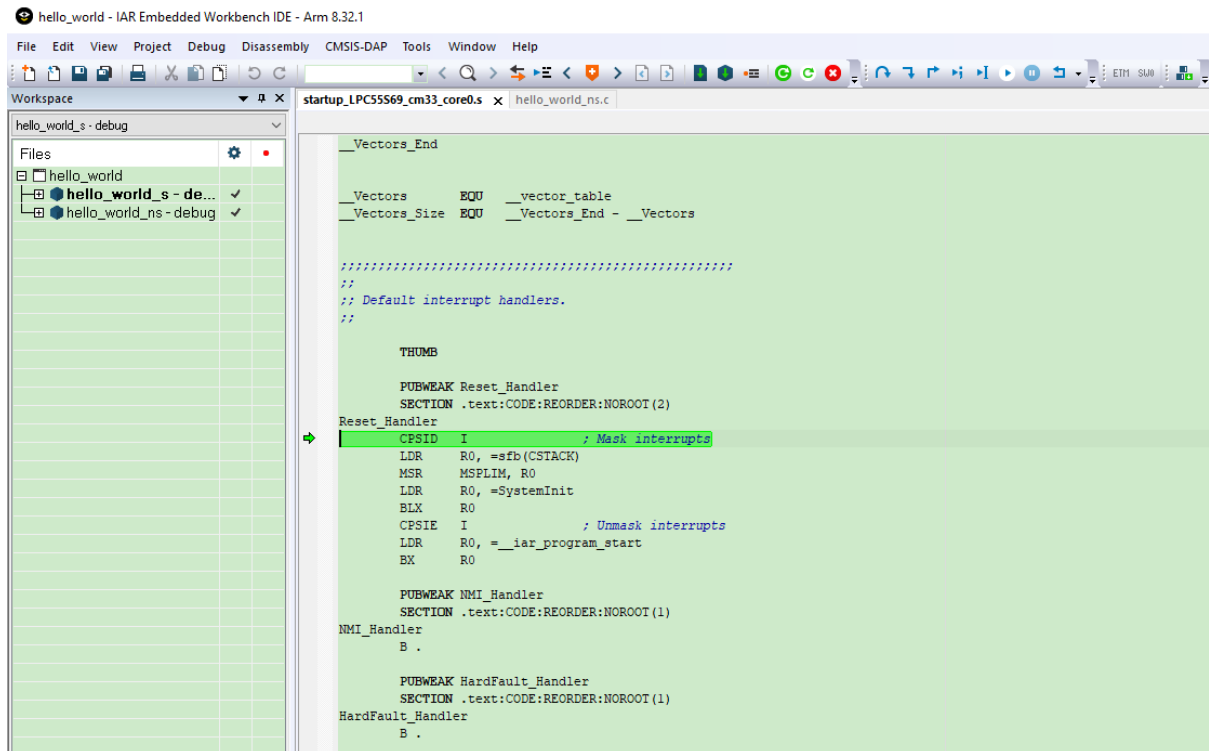
```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_ns/iar/hello_world_  
↪ns.eww
```

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

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

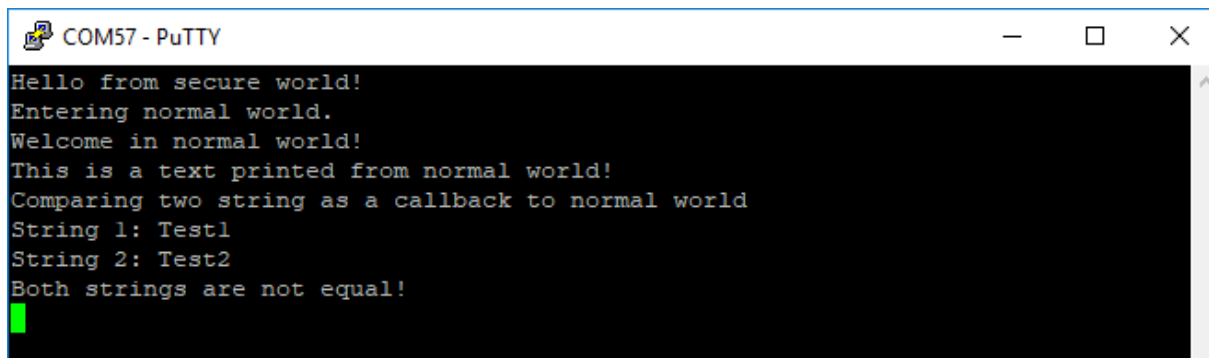
This project `hello_world.eww` contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another. Build both applications separately by clicking **Make**. It is requested to build the application for the secure project first, because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

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



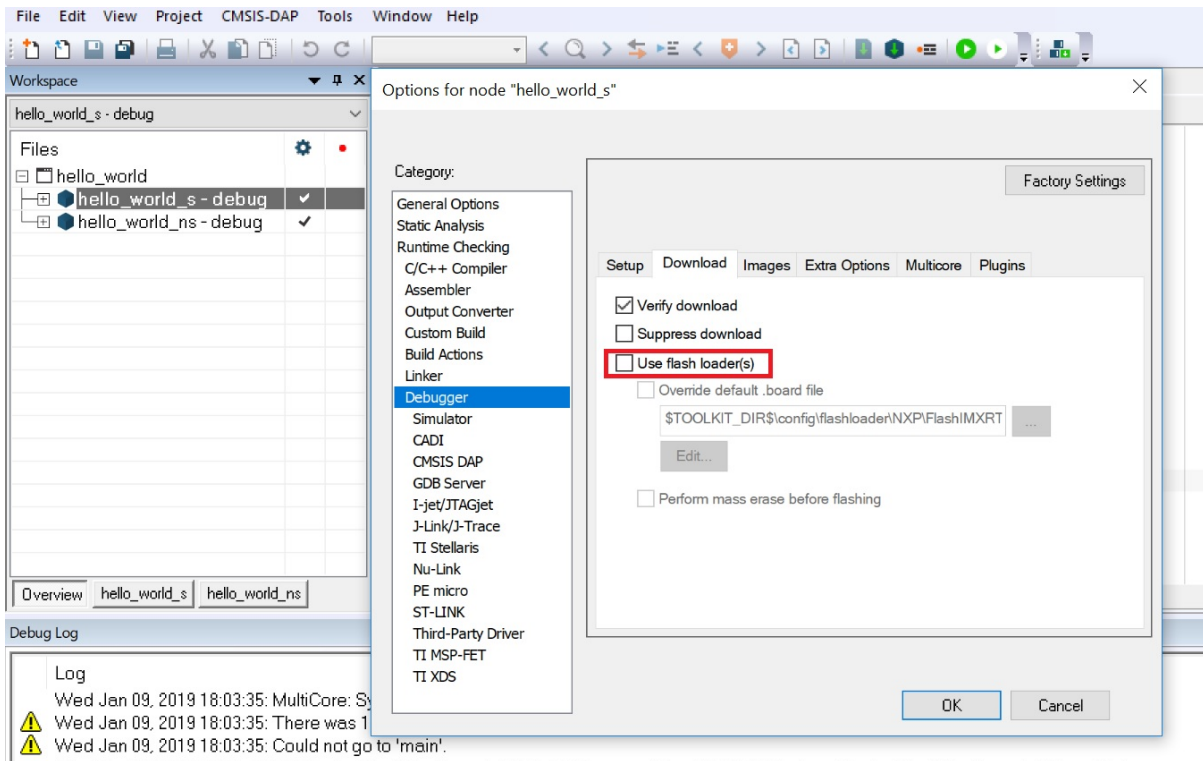
Run the code by clicking **Go** to start the application.

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



**Note:** If the application is running in RAM (debug/release build target), in **Options\*\*>\*\*Debugger > Download** tab, disable **Use flash loader(s)**. This can avoid the `__ns` download issue on i.MXRT500.



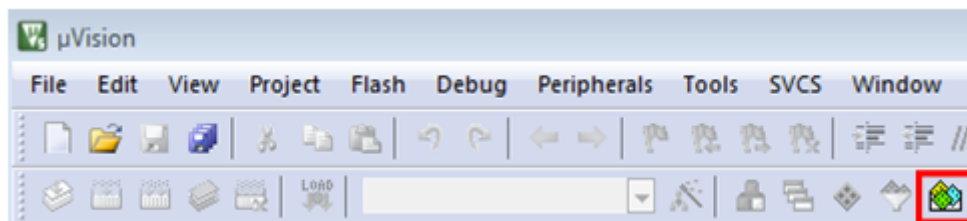


## Run a demo using Keil MDK/μVision

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

**Install CMSIS device pack** After the MDK tools are installed, Cortex Microcontroller Software Interface Standard (CMSIS) device packs must be installed to fully support the device from a debug perspective. These packs include things such as memory map information, register definitions, and flash programming algorithms. Follow these steps to install the appropriate CMSIS pack.

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



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

## Build an example application

1. Open the desired example application workspace in:

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

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

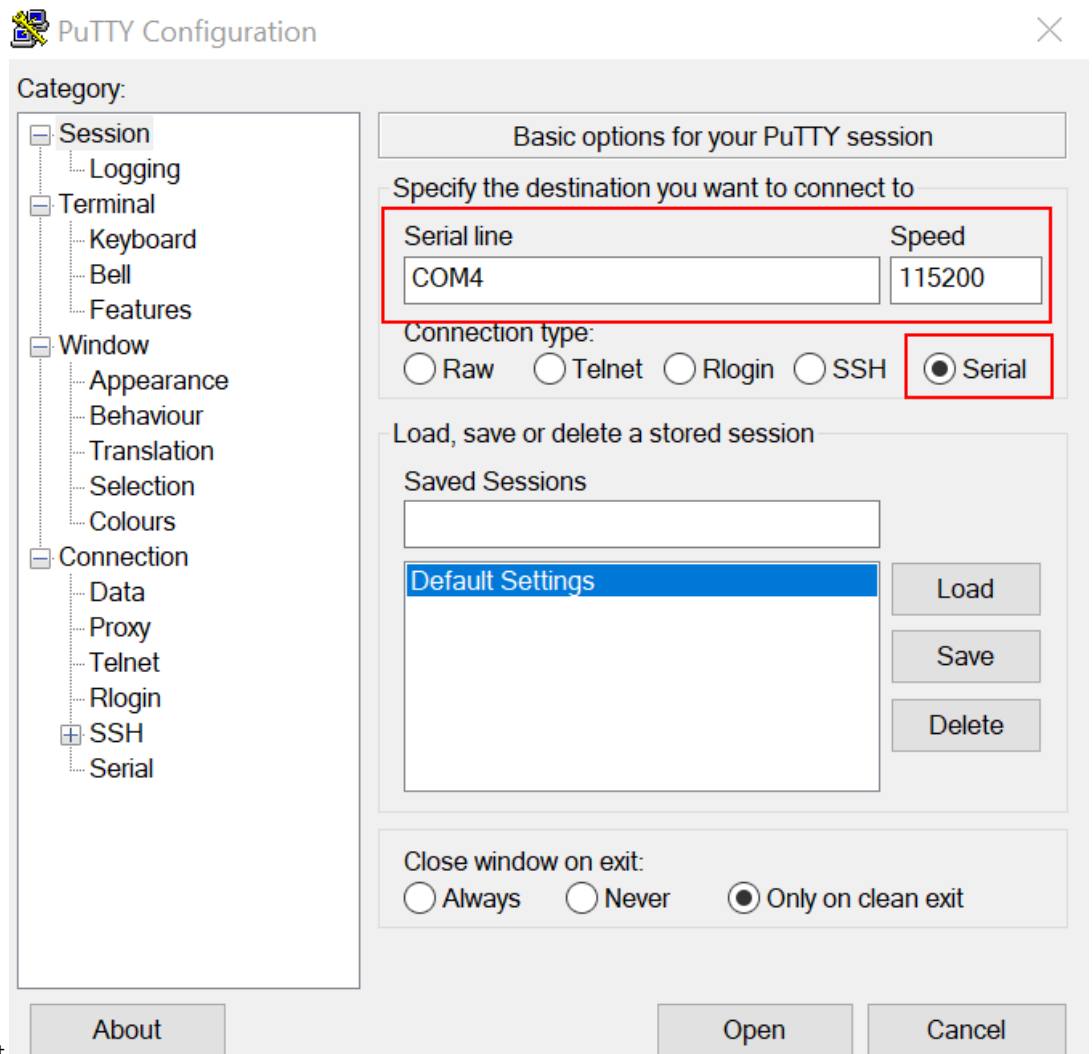
2. To build the demo project, select **Rebuild**, highlighted in red.



3. The build completes without errors.

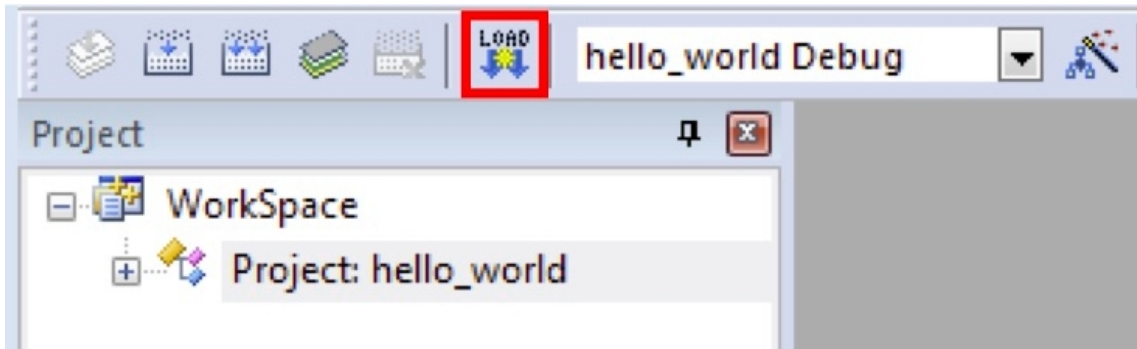
**Run an example application** To download and run the application, perform these steps:

1. Ensure the host driver for the debugger firmware has been installed. See [On-board debugger](#).
2. Connect the development platform to your PC via USB cable using USB connector.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm and connect to the debug serial port number (to determine the COM port number, see [How to determine COM port](#)). Configure the terminal with these settings:
  1. 115200 or 9600 baud rate, depending on your board (reference BOARD\_DEBUG\_UART\_BAUDRATE variable in the board.h file)
  2. No parity
  3. 8 data bits

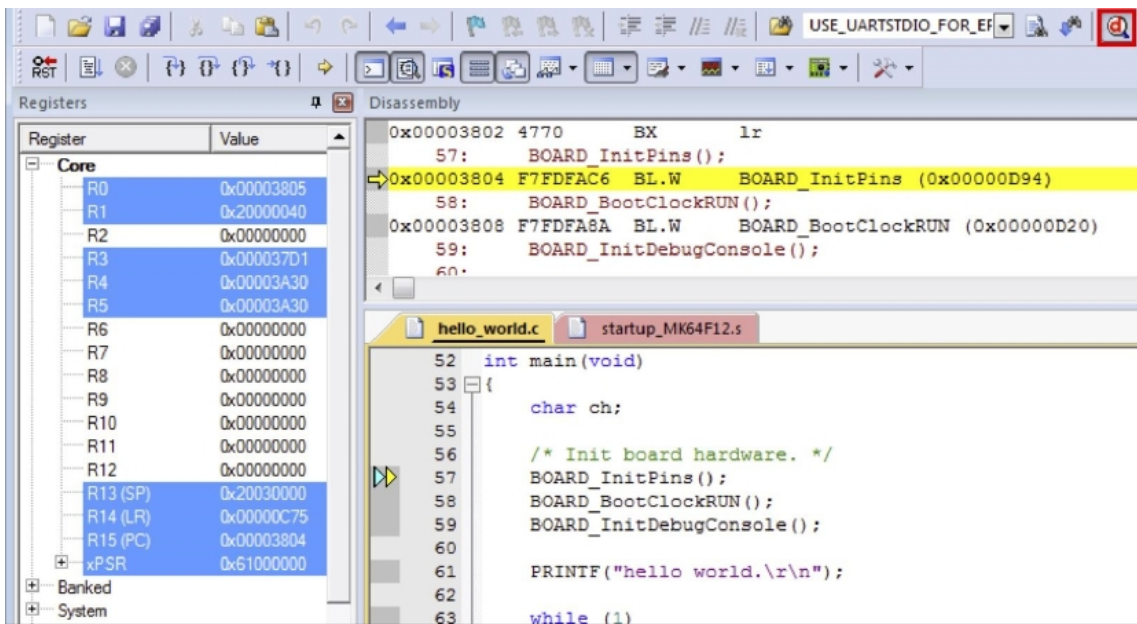


4. 1 stop bit

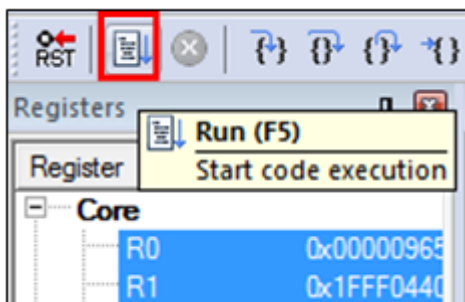
4. In  $\mu$ Vision, after the application is built, click the **Download** button to download the application to the target.



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



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



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



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

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

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

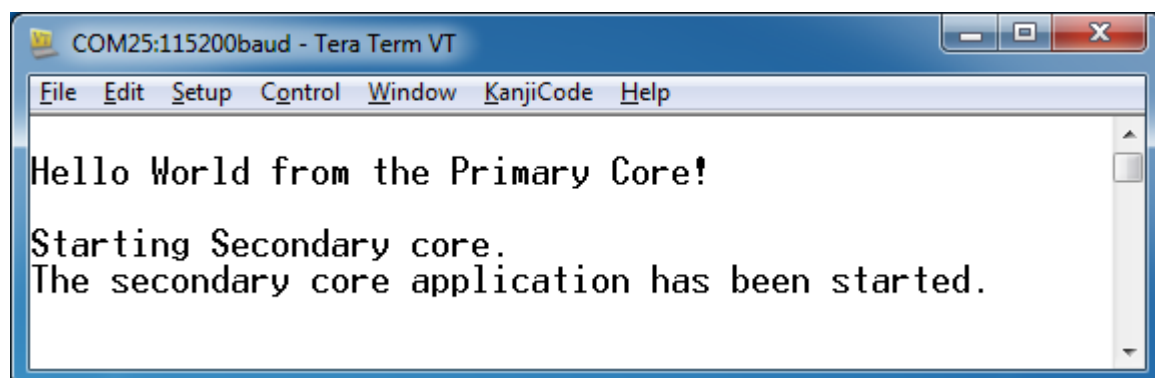
```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm0plus/mdk/hello_world_
→cm0plus.uvmpw
```

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

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

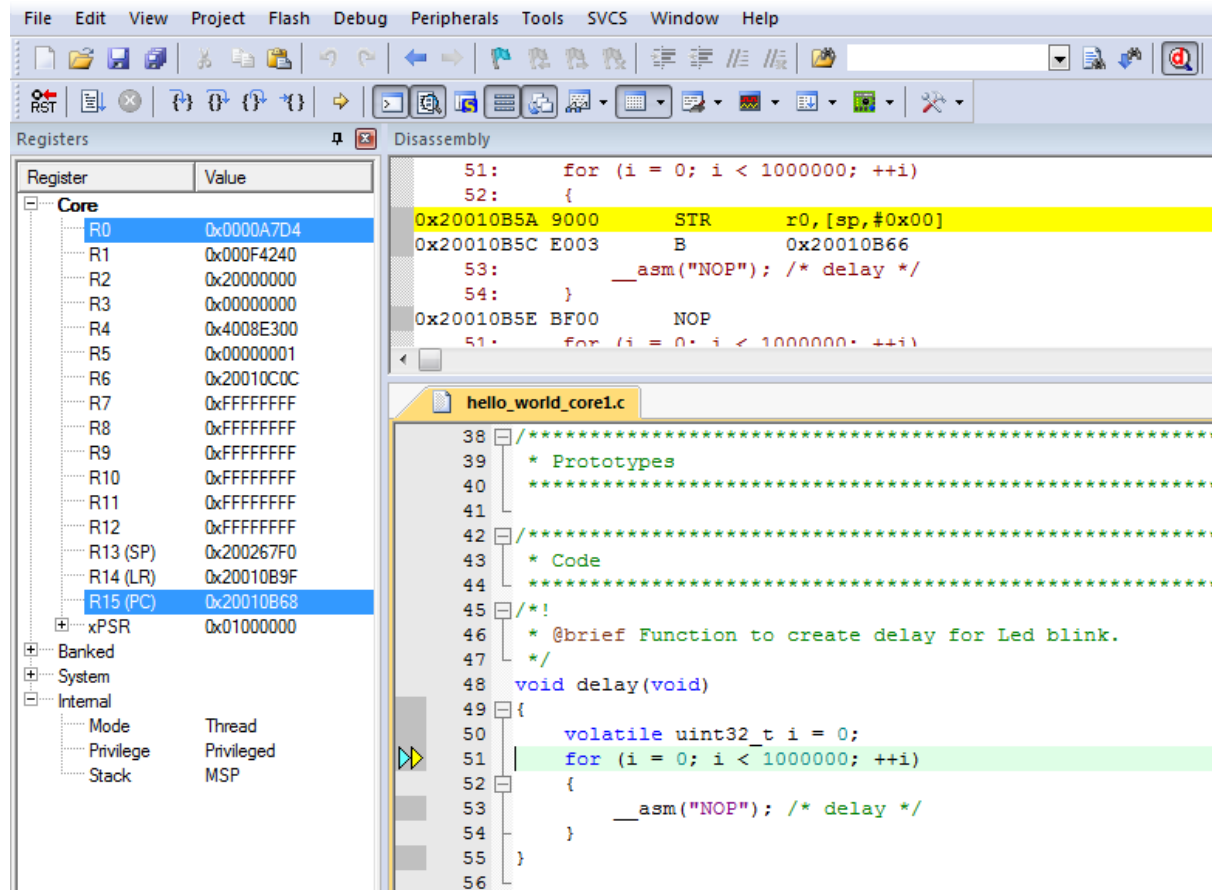
**Run a multicore example application** The primary core debugger flashes both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 5 as described in **Run an example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

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



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

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



Arm describes multicore debugging using the NXP LPC54114 Cortex-M4/M0+ dual-core processor and Keil uVision IDE in Application Note 318 at [www.keil.com/appnotes/docs/apnt\\_318.asp](http://www.keil.com/appnotes/docs/apnt_318.asp). The associated video can be found [here](#).

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

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

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

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

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

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

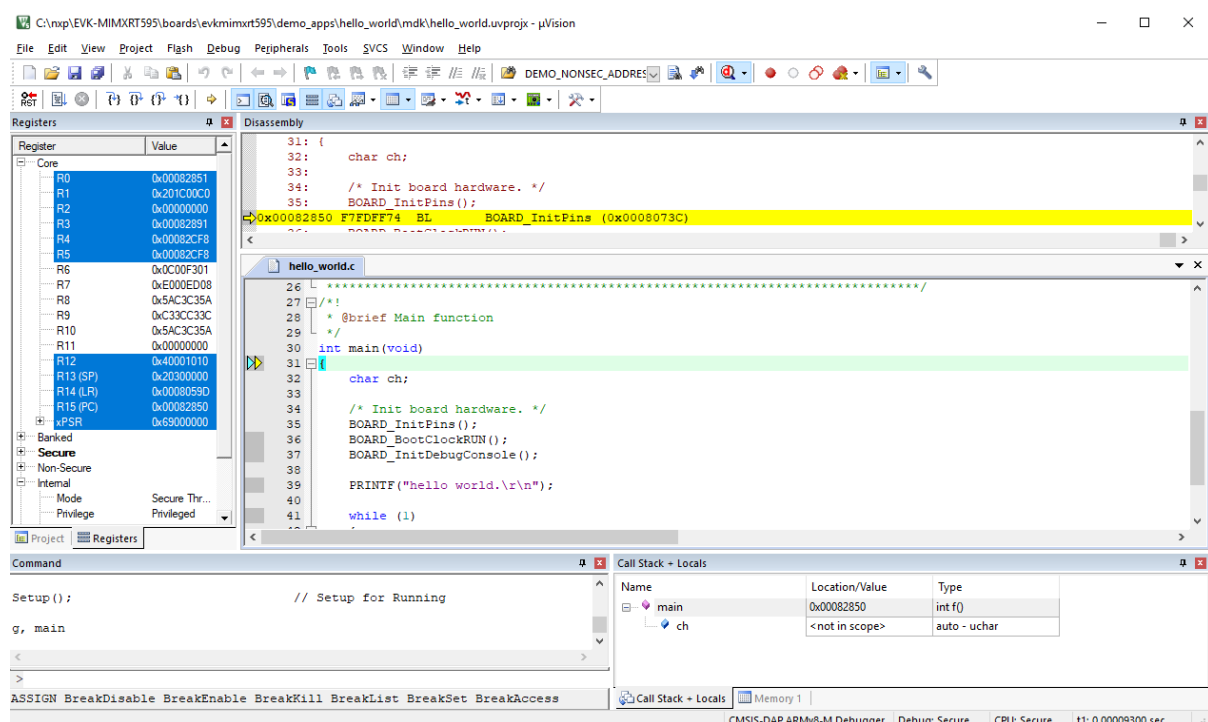
```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_s/mdk/hello_world.  
→ uvmpw
```

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

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

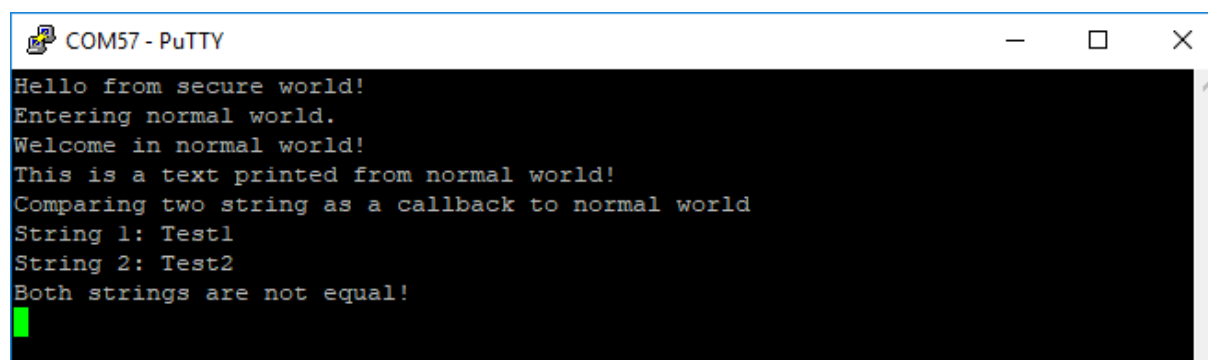
**Run a TrustZone example application** The secure project is configured to download both secure and non-secure output files so debugging can be fully managed from the secure project.

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



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

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





## Run a demo using Arm GCC

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

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

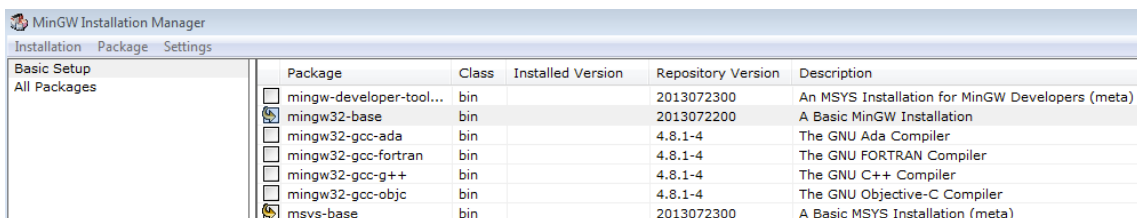
**Install GCC Arm Embedded tool chain** Download and run the installer from GNU Arm Embedded Toolchain. This is the actual toolset (in other words, compiler, linker, and so on). The GCC toolchain should correspond to the latest supported version, as described in **MCUXpresso SDK Release Notes**.

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

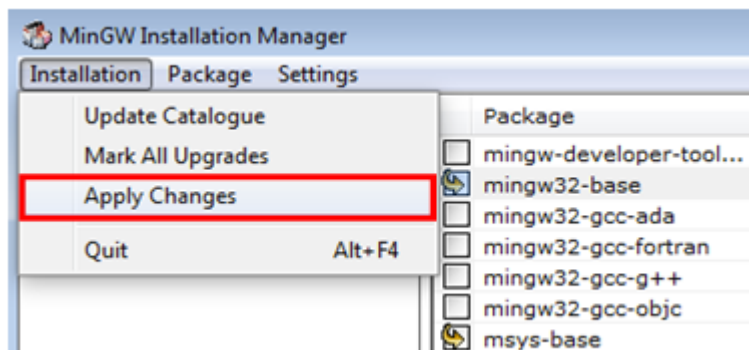
1. Download the latest MinGW mingw-get-setup installer from [MinGW](#).
2. Run the installer. The recommended installation path is `C:\MinGW`, however, you may install to any location.

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

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



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

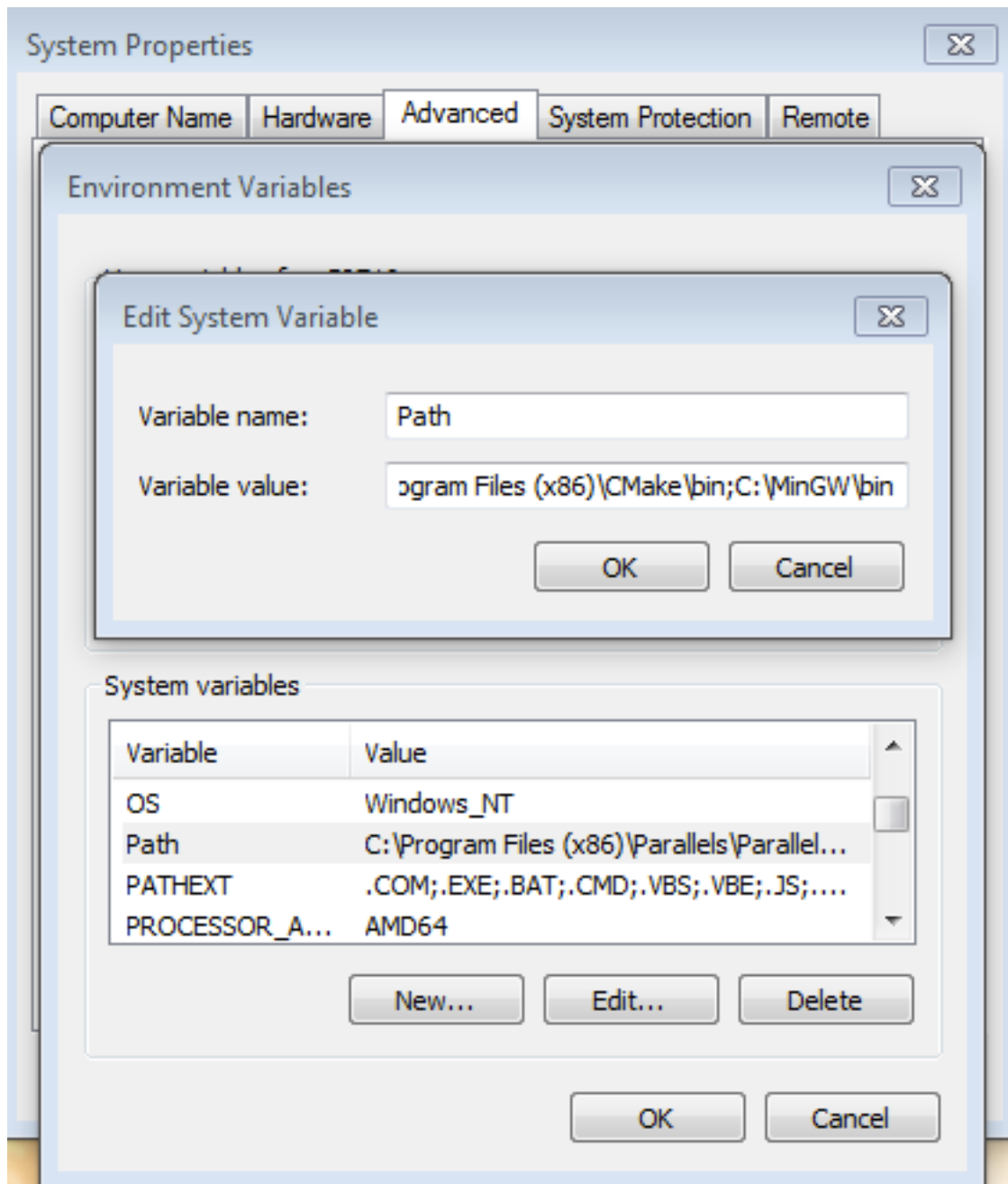


5. Add the appropriate item to the Windows operating system path environment variable. It can be found under **Control Panel->System and Security->System->Advanced System Settings** in the **Environment Variables...** section. The path is:

```
<mingw_install_dir>\bin
```

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

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



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

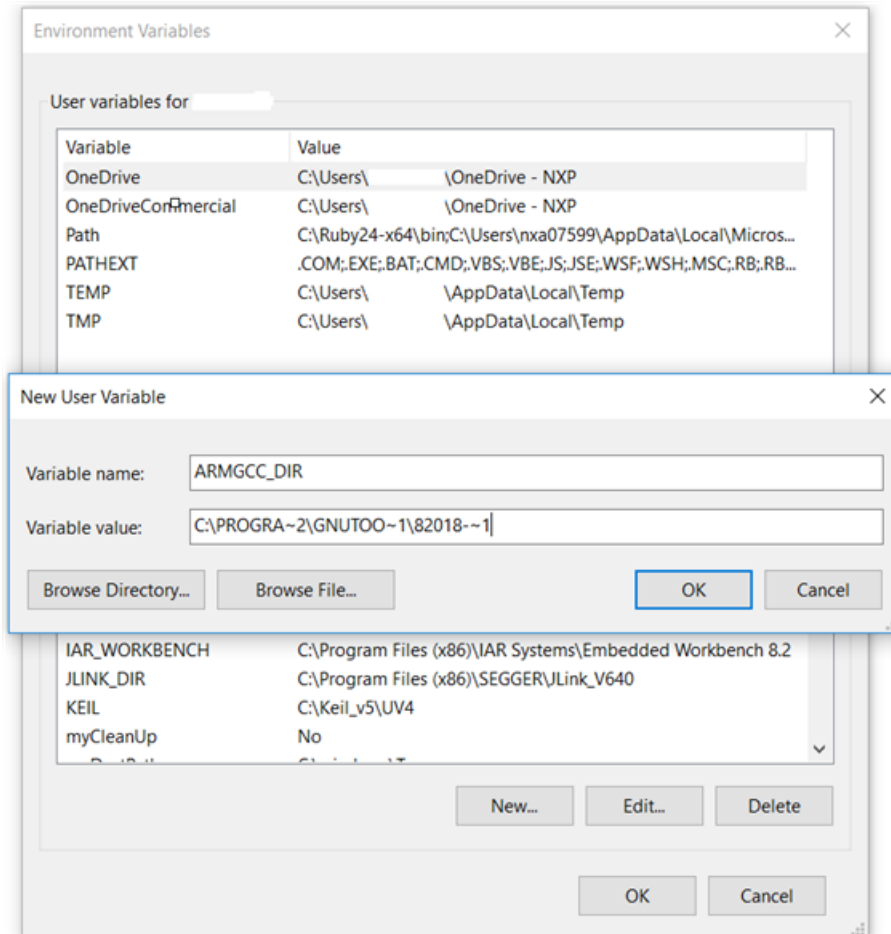
```
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major
```

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



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

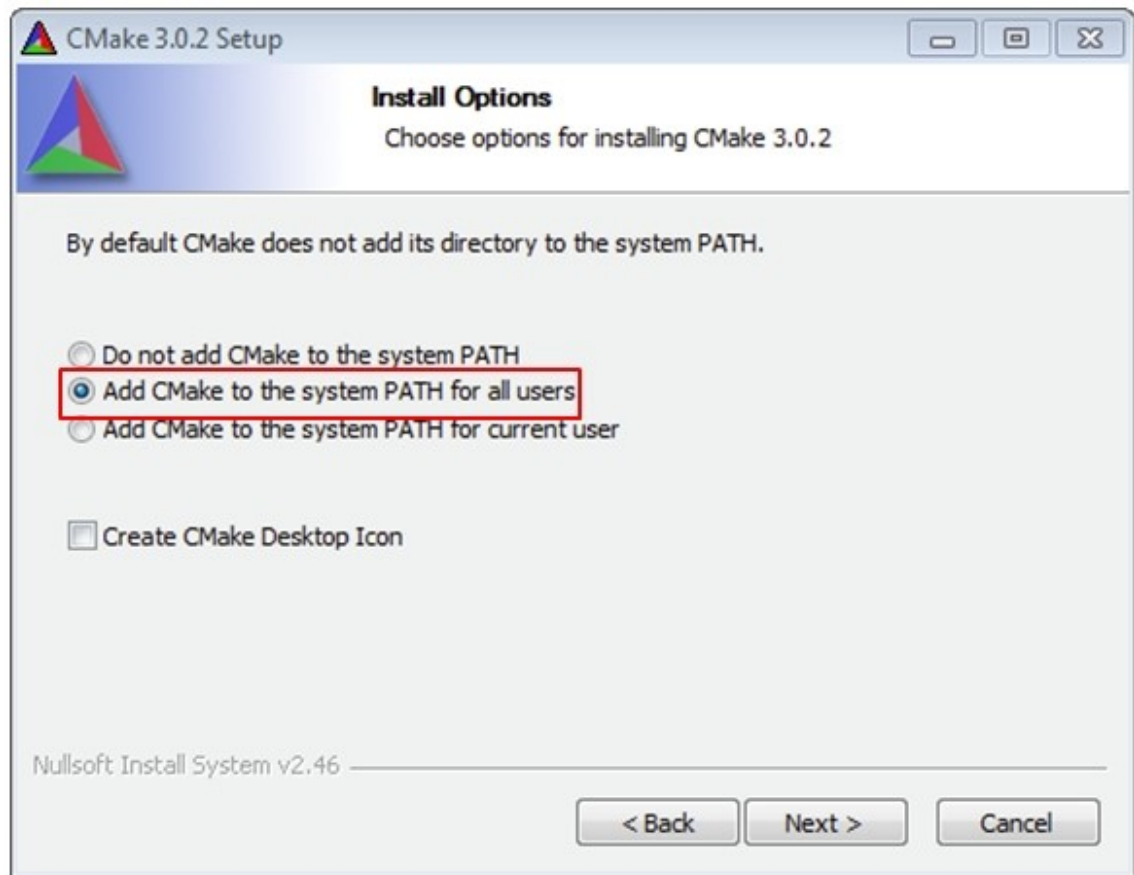
```
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>for %I in (.) do echo %~sI
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>echo C:\PROGRA~2\GNUTOO~1\82018~1
C:\PROGRA~2\GNUTOO~1\82018~1
```



## 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.

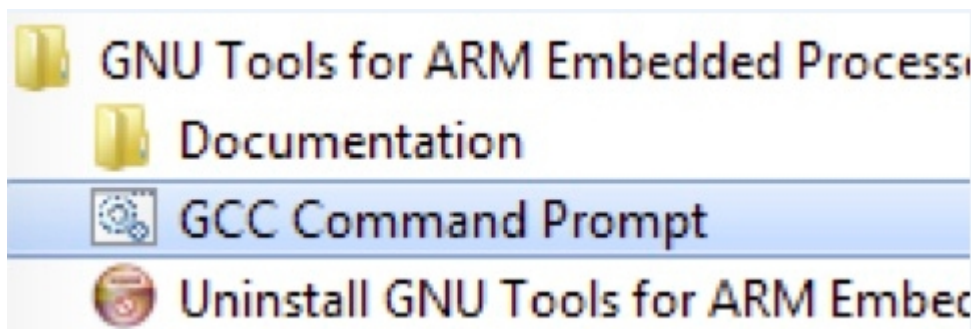
**Linux OS** It depends on the distributions of Linux Operation System. Here we use Ubuntu as an example.

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.

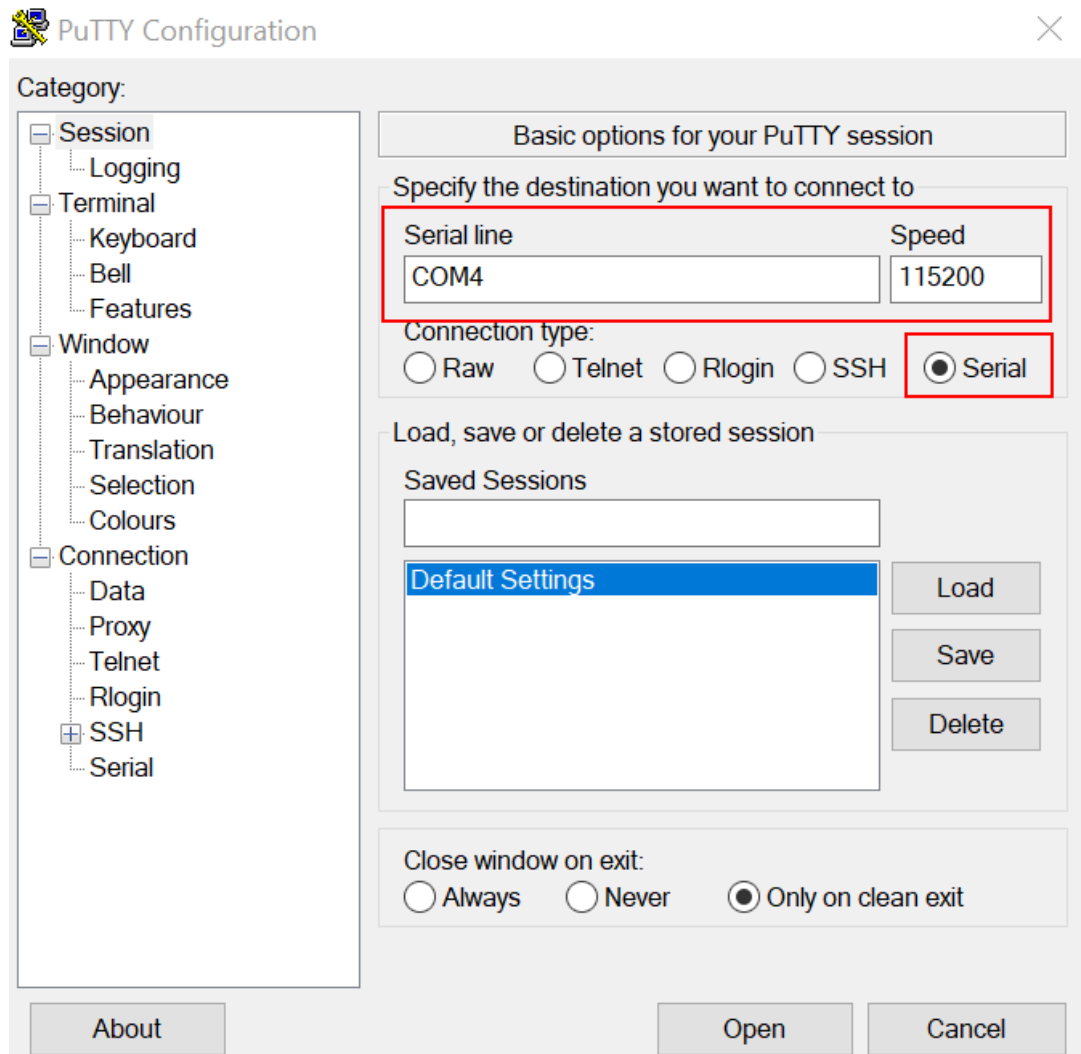
```
[ 84%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.0_FRDM-K64F
/devices/MK64F12/drivers/fsl_smc.c.obj
[ 92%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.0_FRDM-K64F
/devices/MK64F12/drivers/fsl_clock.c.obj
[100%] Linking C executable debug\hello_world.elf
[100%]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

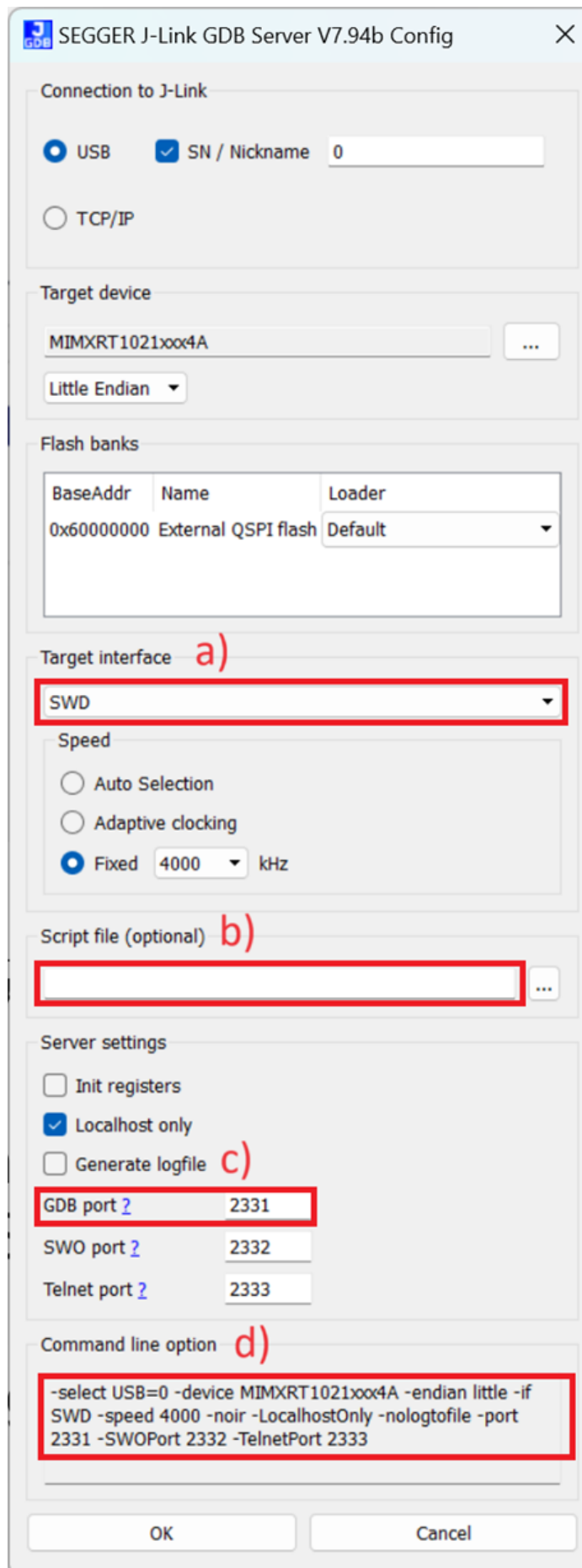


3. To launch the application, open the Windows **Start** menu and select **Programs > SEGGER > J-Link <version> J-Link GDB Server**.

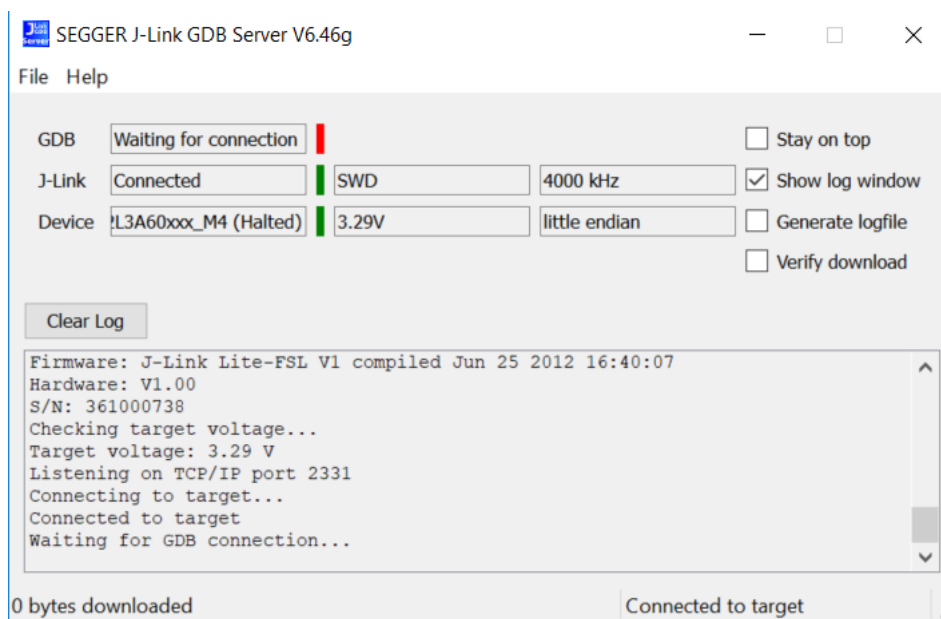
**Note:** It is assumed that the J-Link software is already installed.

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

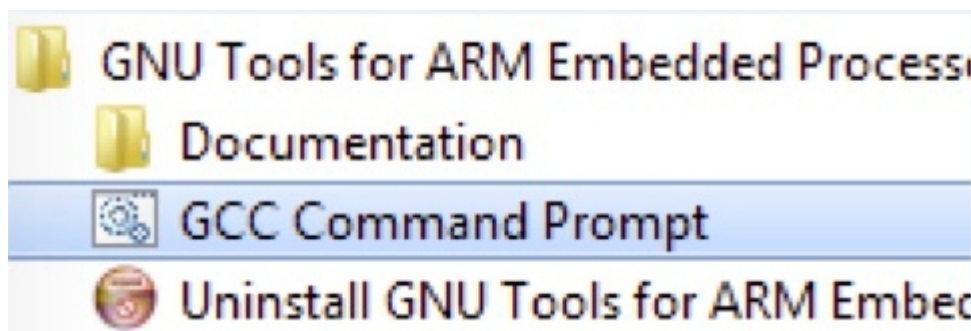
4. Make sure to check the following options.
  1. **Target interface:** The debug connection on board uses internal SWD signaling. In case of a wrong setting J-Link is unable to communicate with device under test.
  2. **Script file:** If required, a J-Link init script file can be used for board initialization.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

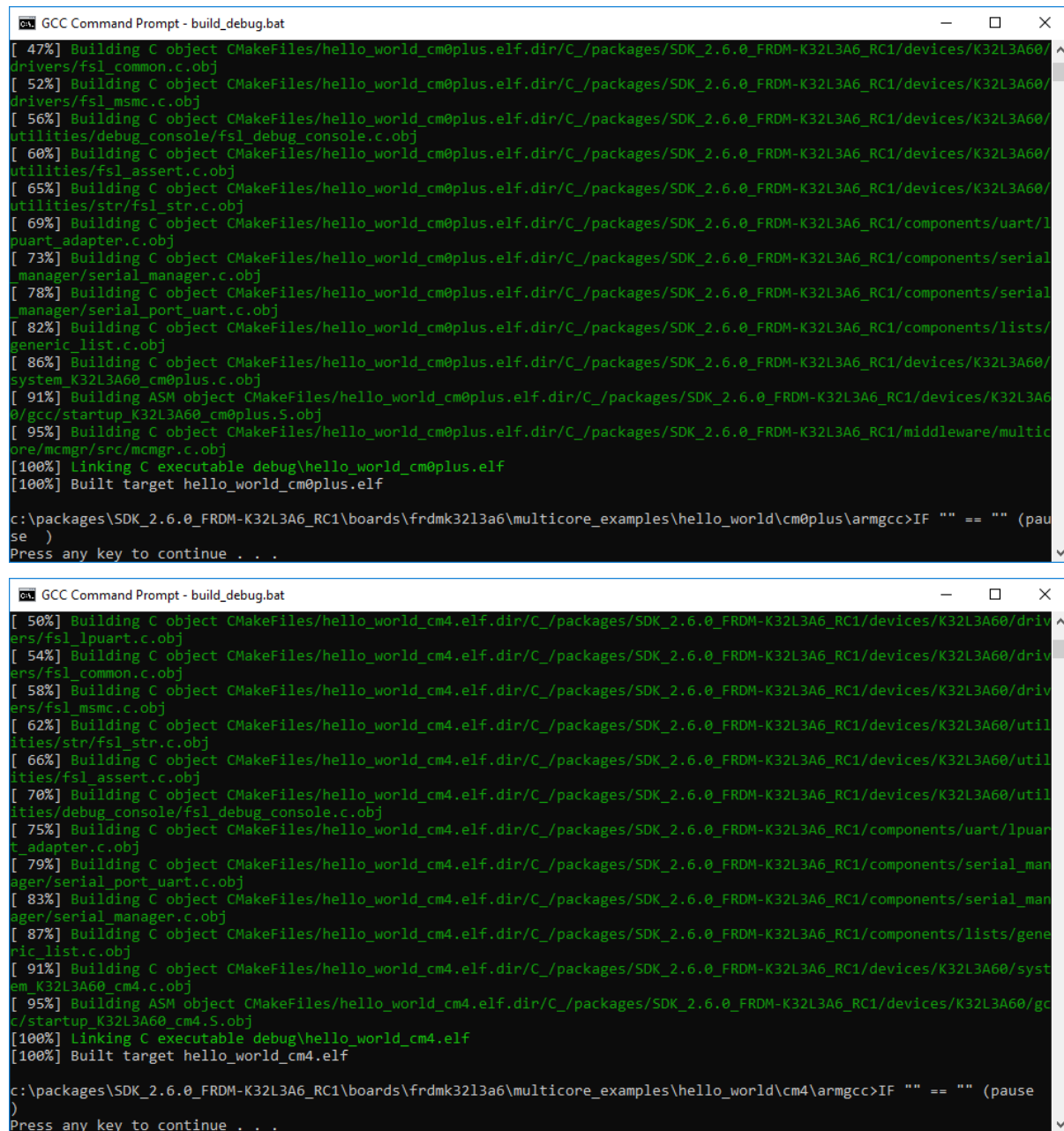
Begin with a simple dual-core version of the Hello World application. The multicore Hello World GCC build scripts are located in this folder:

```
<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 . . .

```

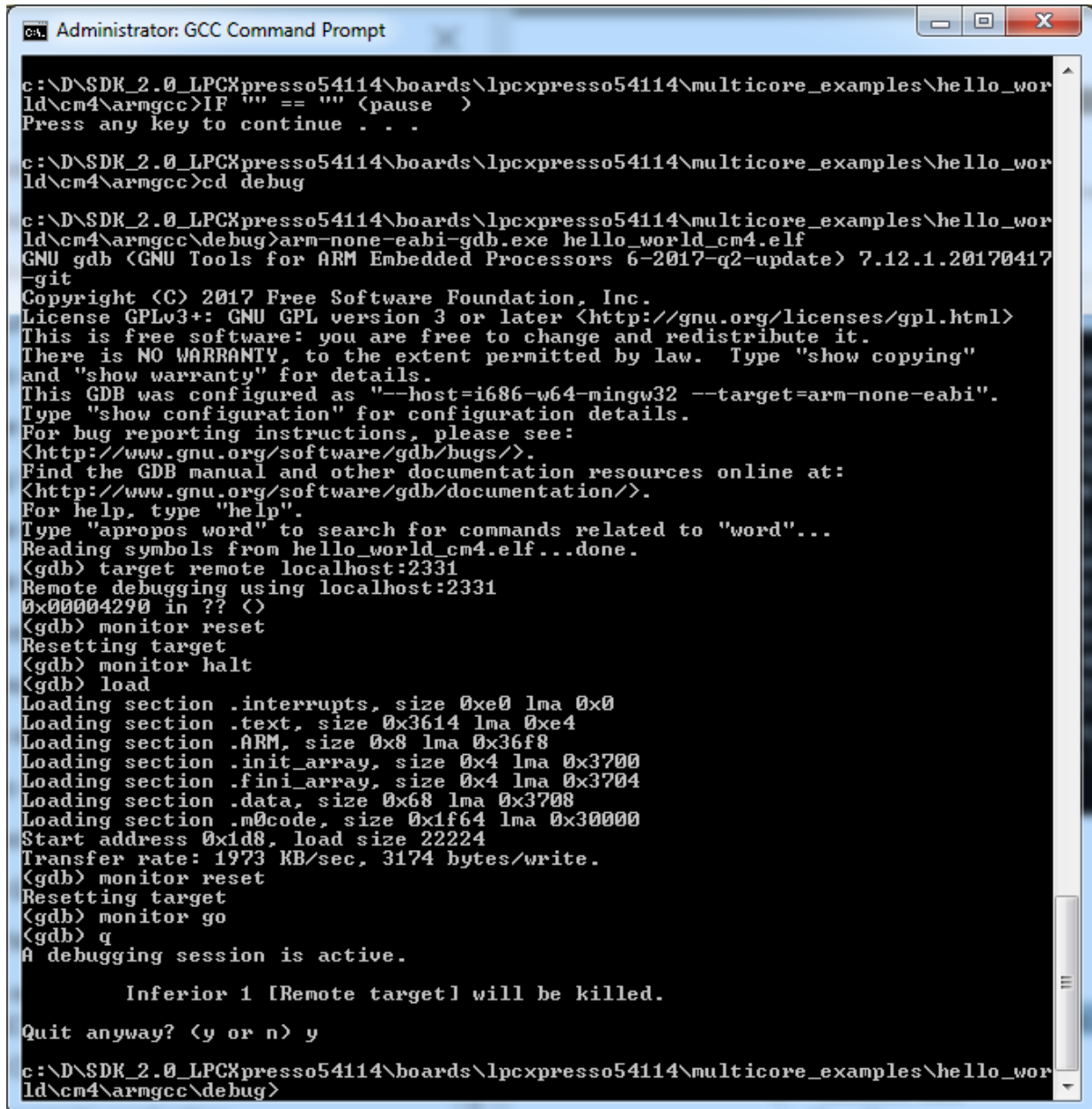
**Run a multicore example application** When running a multicore application, the same pre-requisites for J-Link/J-Link OpenSDA firmware, and the serial console as for the single-core application, applies, as described in **Run an example application**.

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

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



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



```

Administrator: GCC Command Prompt

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

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

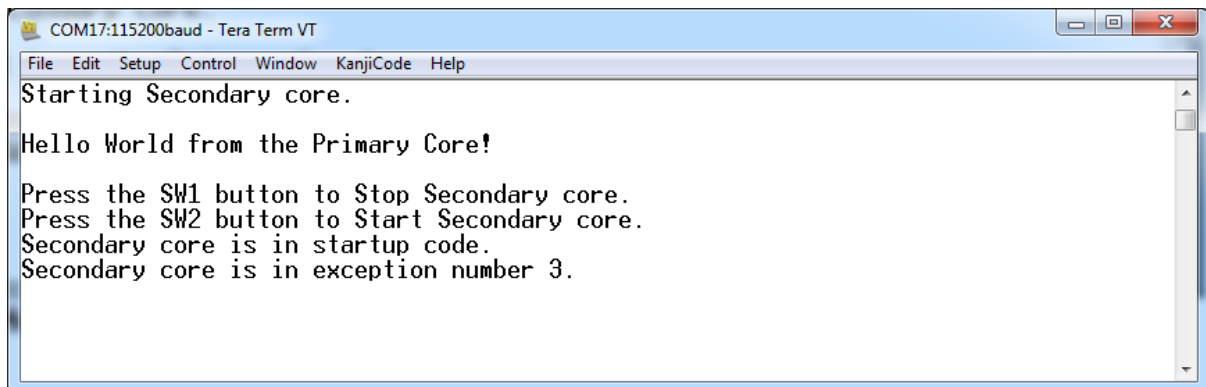
c:\D\SDK_2.0_LPCXpresso54114\boards\lpcxpresso54114\multicore_examples\hello_world_cm4\armgcc\debug>arm-none-eabi-gdb.exe hello_world_cm4.elf
GNU gdb (GNU Tools for ARM Embedded Processors 6-2017-q2-update) 7.12.1.20170417-git
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL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

    Inferior 1 [Remote target] will be killed.

Quit anyway? (y or n) y

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

```



```

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

Hello World from the Primary Core!

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

```

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

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

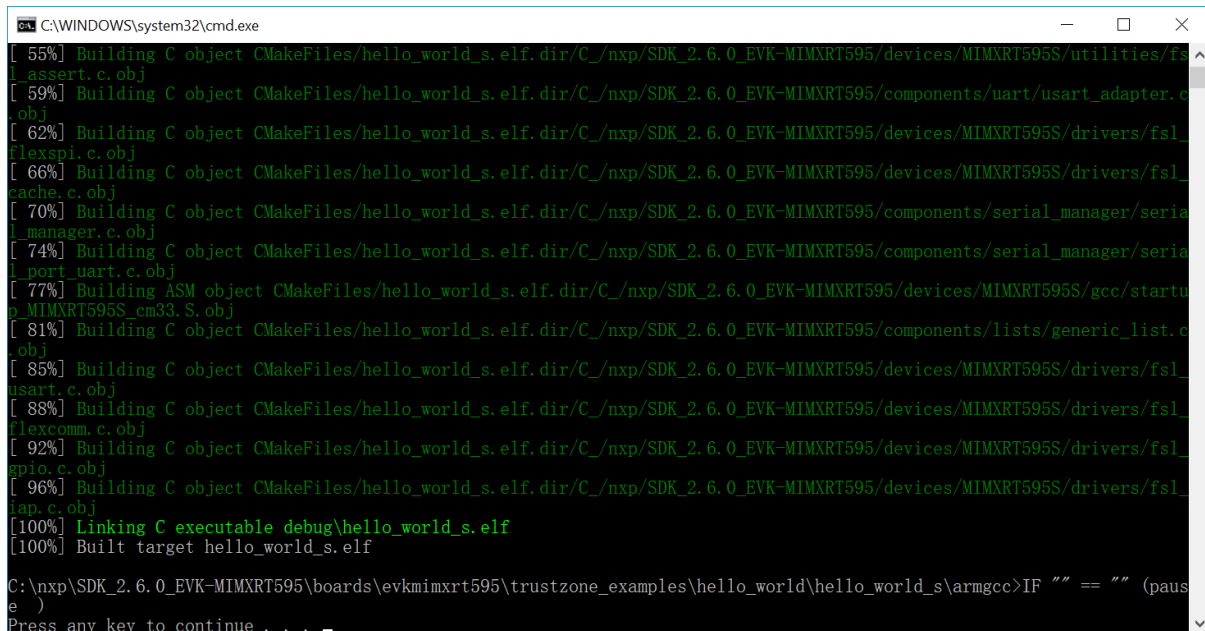
```
<install_dir>/boards/<board_name>/trustzone_examples/<application_name>/[<core_type>]/  
↪<application_name>_s/armgcc
```

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

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

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

Build both applications separately, following steps for single core examples as described in **Build an example application**. It is requested to build the application for the secure project first, because the non-secure project must know the secure project, since CMSE library is running the linker. It is not possible to finish the non-secure project linker with the secure project because the CMSE library is not ready.



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

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

```

C:\WINDOWS\system32\cmd.exe
[ 52%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/components/uart/usart_adapter.c.obj
[ 56%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/utilities/fsl_assert.c.obj
[ 60%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_flexspi.c.obj
[ 64%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_cache.c.obj
[ 68%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/components/serial_manager/serial_manager.c.obj
[ 72%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/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:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_usart.c.obj
[ 80%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/components/lists/generic_list.c.obj
[ 84%] Building ASM object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/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:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_flexcomm.c.obj
[ 92%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_gpio.c.obj
[ 96%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/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:\nxp\SDK_2.6.0_EVK-MIMXRT595\boards\evkmimxrt595\trustzone_examples\hello_world\hello_world_ns\armgcc>IF "" == "" (pause)
Press any key to continue . . .

```

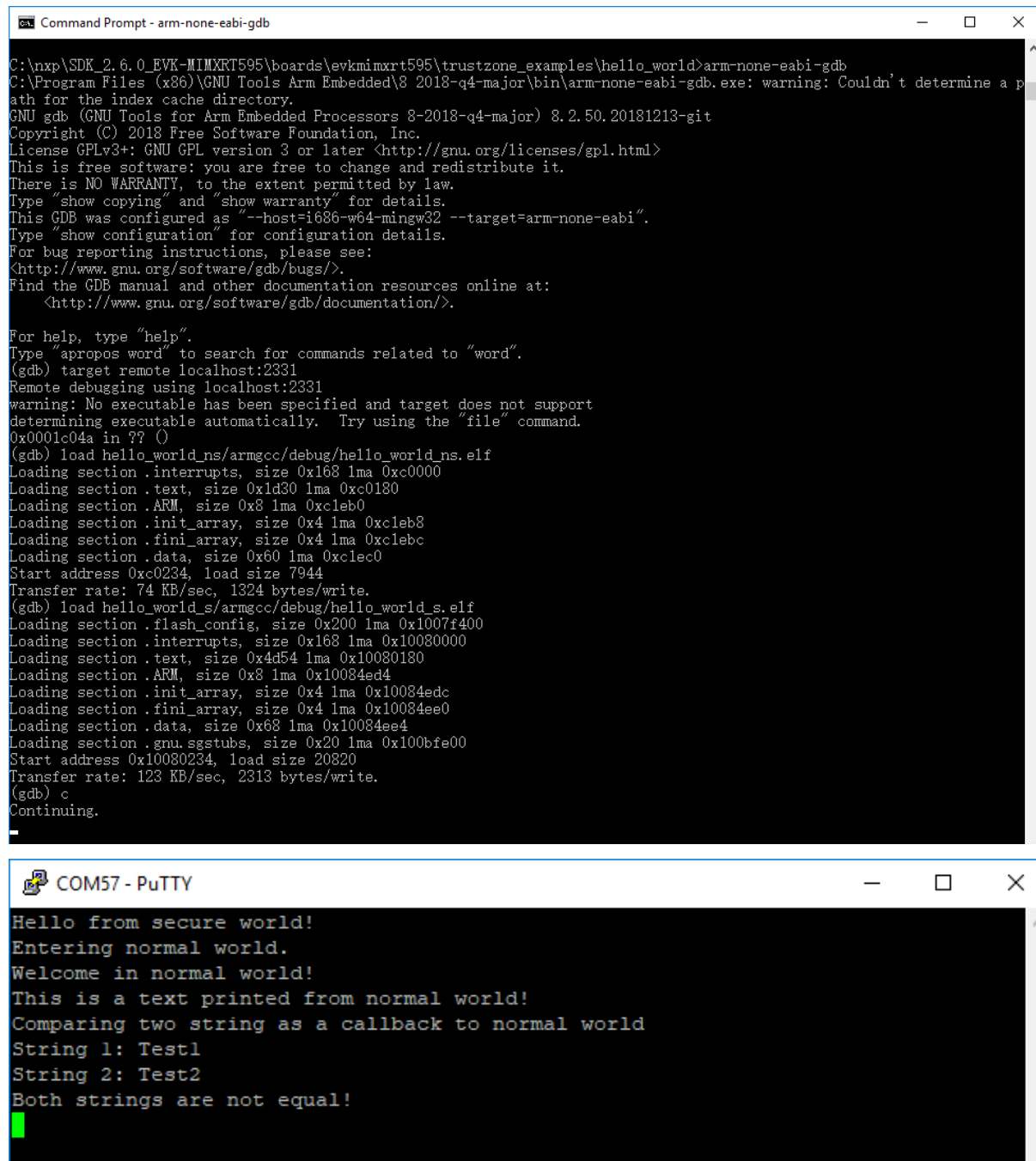
**Run a TrustZone example application** When running a TrustZone application, the same prerequisites for J-Link/J-Link OpenSDA firmware, and the serial console as for the single core application, apply, as described in **Run an example application**.

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

Then, run these commands:

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

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



The first screenshot shows a Windows Command Prompt window titled "Command Prompt - arm-none-eabi-gdb". The user has navigated to the directory `C:\nxp\SDK_2.6.0_EVK-MIMXRT595\boards\evkmimxrt595\trustzone_examples\hello_world` and executed `arm-none-eabi-gdb`. The GDB startup sequence is shown, including the version (8.2.50.20181213-git), copyright information, and the target configuration (`--host=i686-w64-mingw32 --target=arm-none-eabi`). The user enters `target remote localhost:2331`, and GDB attempts to connect. A warning is displayed: "warning: No executable has been specified and target does not support determining executable automatically. Try using the 'file' command." The user then enters `load hello_world_ns/armgcc/debug/hello_world_ns.elf`, and GDB loads the ELF file, displaying section information for `.interrupts`, `.text`, `.ARM`, `.init_array`, `.fini_array`, and `.data`. The user enters `c` to continue the execution.

The second screenshot shows a PuTTY window titled "COM57 - PuTTY". The output of the program is displayed: "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", and "Both strings are not equal!". A green cursor is visible at the end of the last line.

## MCUXpresso Config Tools

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

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

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

MCUXpresso Config Tools can be accessed in the following products:

- **Integrated** in the MCUXpresso IDE. Config tools are integrated with both compiler and debugger which makes it the easiest way to begin the development.
- **Standalone version** available for download from [www.nxp.com/mcuxpresso](http://www.nxp.com/mcuxpresso). Recommended for customers using IAR Embedded Workbench, Keil MDK  $\mu$ Vision, or Arm GCC.
- **Online version** available on [mcuxpresso.nxp.com](http://mcuxpresso.nxp.com). Recommended doing a quick evaluation of the processor or use the tool without installation.

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

## How to determine COM port

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

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

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

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

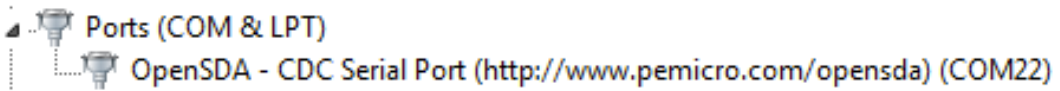
2. **Windows:** To determine the COM port open Device Manager in the Windows operating system. Click the **Start** menu and type **Device Manager** in the search bar.

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

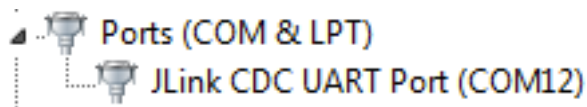
1. **CMSIS-DAP/mbd/DAPLink** interface:



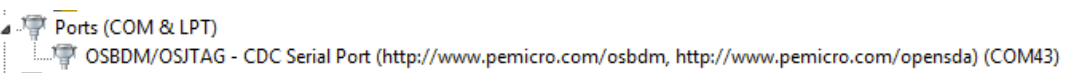
## 2. P&E Micro:



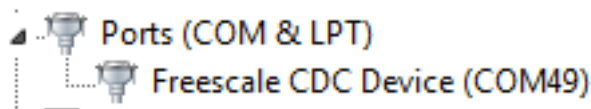
## 3. J-Link:



## 4. P&E Micro OSJTAG:



## 5. MRB-KW01:



## On-board Debugger

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

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

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

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

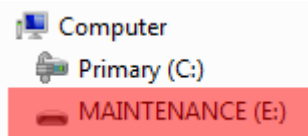
- J-Link: Download appropriate image from [www.segger.com/opensda.html](http://www.segger.com/opensda.html). Choose the appropriate J-Link binary based on the table in [Default debug interfaces](#). Any OpenSDA v1.0 interface should use the standard OpenSDA download (in other words, the one with no version).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



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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

## Default debug interfaces

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

| Hardware platform | Default debugger firmware | On-board debugger probe |
|-------------------|---------------------------|-------------------------|
| EVK-MCIMX7ULP     | N/A                       | N/A                     |
| EVK-MIMX8MM       | N/A                       | N/A                     |
| EVK-MIMX8MN       | N/A                       | N/A                     |
| EVK-MIMX8MNDDR3L  | N/A                       | N/A                     |
| EVK-MIMX8MP       | N/A                       | N/A                     |
| EVK-MIMX8MQ       | N/A                       | N/A                     |
| EVK-MIMX8ULP      | N/A                       | N/A                     |
| EVK-MIMXRT1010    | CMSIS-DAP                 | LPC-Link2               |
| EVK-MIMXRT1015    | CMSIS-DAP                 | LPC-Link2               |
| EVK-MIMXRT1020    | CMSIS-DAP                 | LPC-Link2               |
| EVK-MIMXRT1064    | CMSIS-DAP                 | LPC-Link2               |
| EVK-MIMXRT595     | CMSIS-DAP                 | LPC-Link2               |
| EVK-MIMXRT685     | CMSIS-DAP                 | LPC-Link2               |
| EVK9-MIMX8ULP     | N/A                       | N/A                     |
| EVKB-IMXRT1050    | CMSIS-DAP                 | LPC-Link2               |
| FRDM-K22F         | CMSIS-DAP                 | OpenSDA v2              |
| FRDM-K32L2A4S     | CMSIS-DAP                 | OpenSDA v2              |
| FRDM-K32L2B       | CMSIS-DAP                 | OpenSDA v2              |
| FRDM-K32L3A6      | CMSIS-DAP                 | OpenSDA v2              |
| FRDM-KE02Z40M     | P&E Micro                 | OpenSDA v1              |
| FRDM-KE15Z        | CMSIS-DAP                 | OpenSDA v2              |
| FRDM-KE16Z        | CMSIS-DAP                 | OpenSDA v2              |
| FRDM-KE17Z        | CMSIS-DAP                 | OpenSDA v2              |
| FRDM-KE17Z512     | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXA153      | CMSIS-DAP                 | MCU-Link                |
| FRDM-MCXA156      | CMSIS-DAP                 | MCU-Link                |
| FRDM-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

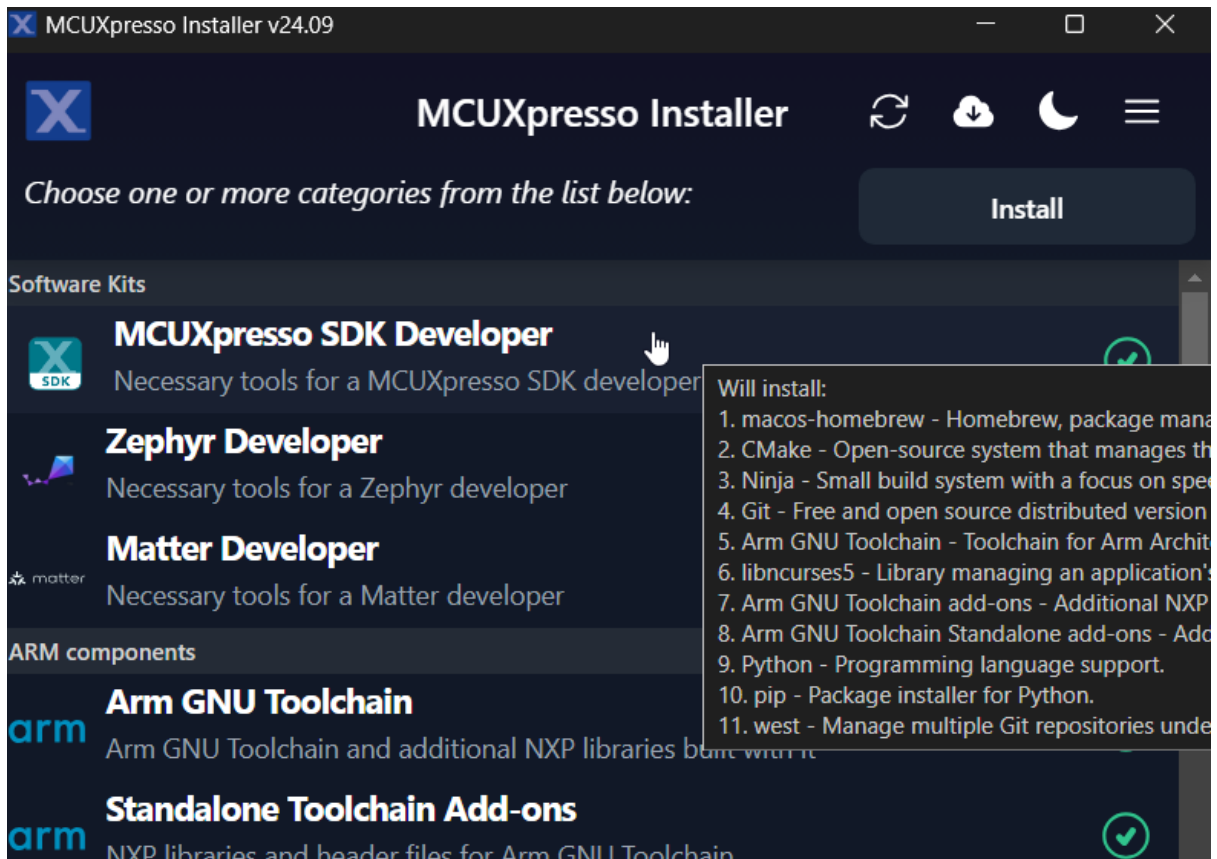
### 1.3.1 Getting Started with MCUXpresso SDK Repository

#### 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.

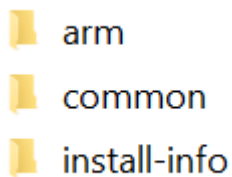
**Toolchain** MCUXpresso SDK supports all mainstream toolchains for embedded development. You can install your used or interested toolchains following the guides.

| 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>                                     |                             |
| Armlclang                    | <a href="#">Installing Arm Compiler for Embedded</a>                 |                             |
| Zephyr                       | <a href="#">Zephyr SDK</a>                                           |                             |
| Codewarrior                  | <a href="#">NXP CodeWarrior</a>                                      |                             |
| Xtensa                       | <a href="#">Tensilica Tools</a>                                      |                             |
| NXP S32Compiler RISC-V Zen-V | <a href="#">NXP Website</a>                                          |                             |

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

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

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



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



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

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

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

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

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

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

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

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

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

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

## Get MCUXpresso SDK Repo

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

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

# Update the west projects
cd mcuxpresso-sdk
west update

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

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

1. Navigate to the core directory:

```
cd mcuxsdk
```

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

```
python -m venv .venv

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

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

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

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

3. Install the required Python packages:

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

## Explore Contents

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

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

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

All the projects record in the **Manifest repo** are checked out to the folder `mcuxsdk/`, the layout of `mcuxsdk` folder is shown as below:

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

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

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

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

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

## Run a demo using MCUXpresso for VS Code

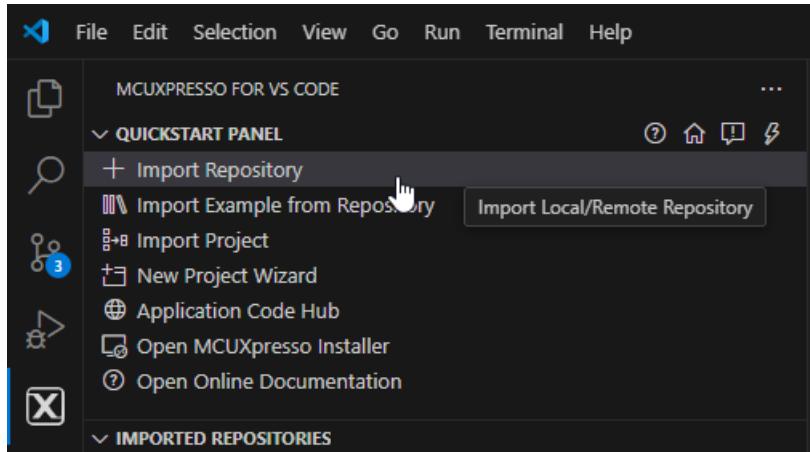
This section explains how to configure MCUXpresso for VS Code to build, run, and debug example applications. This guide uses the `hello_world` demo application as an example. However, these

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

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

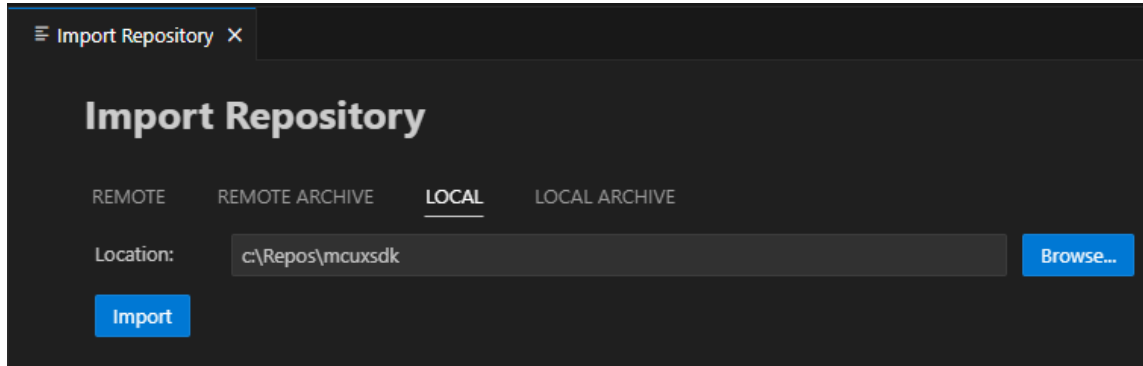
To build an example application:

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

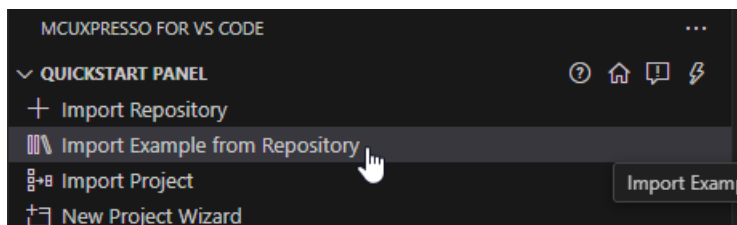


**Note:** You can import the SDK in several ways. Refer to [MCUXpresso for VS Code Wiki](#) for details.

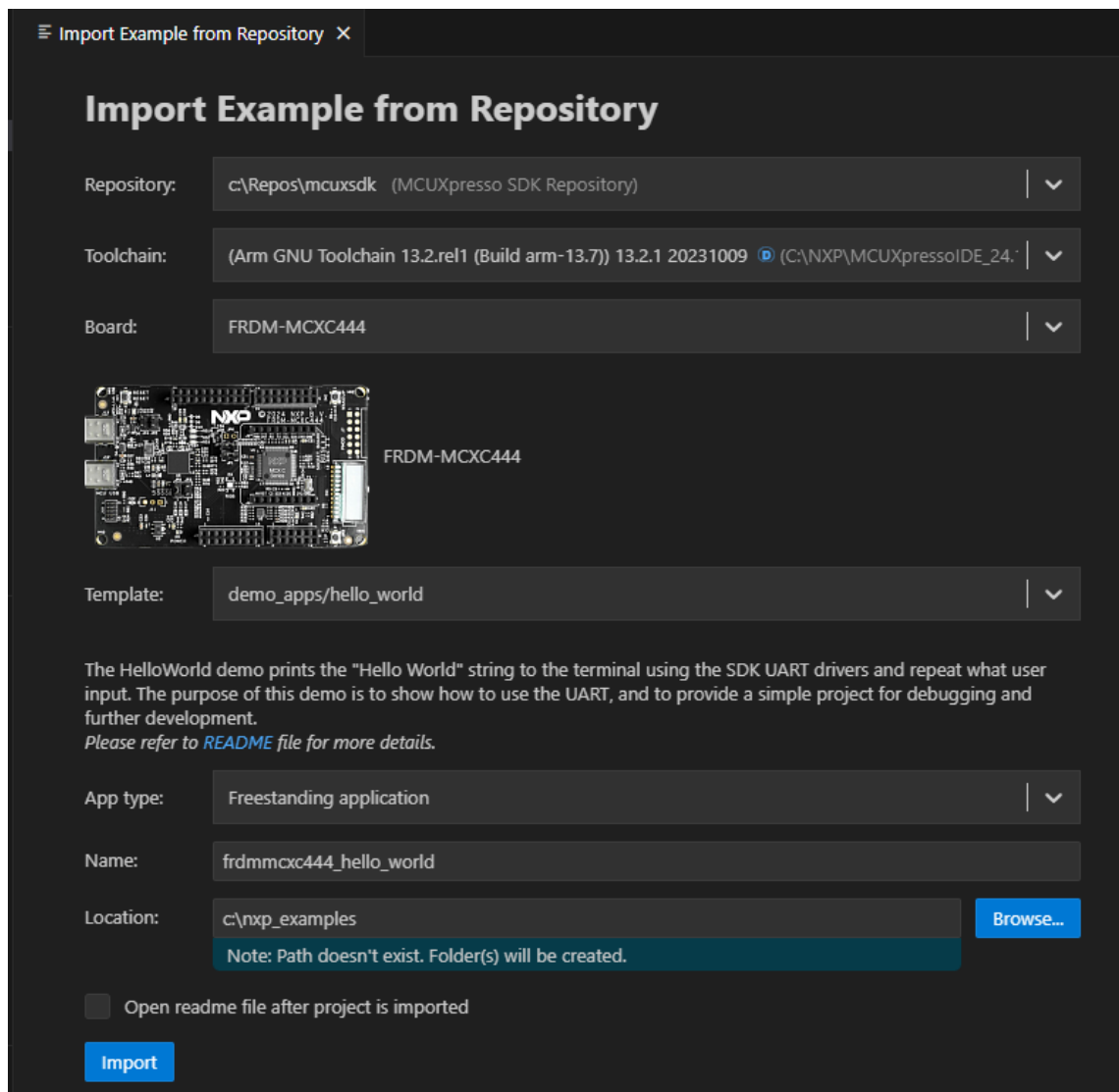
Select **Local** if you've already obtained the SDK as seen in [Get MCUXpresso SDK Repo](#). Select your location and click **Import**.



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



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



**Import Example from Repository**

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

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

Board: FRDM-MCXC444 | v

 FRDM-MCXC444

Template: demo\_apps/hello\_world | v

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

App type: Freestanding application | v

Name: frdmmcxc444\_hello\_world

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

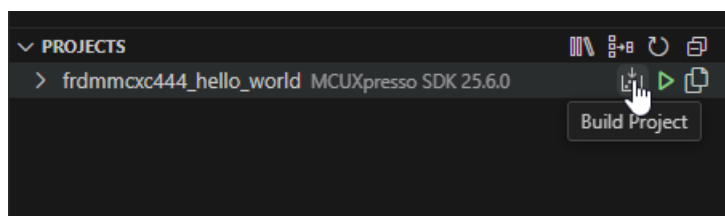
Note: Path doesn't exist. Folder(s) will be created.

☐ Open readme file after project is imported

[Import](#)

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

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



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

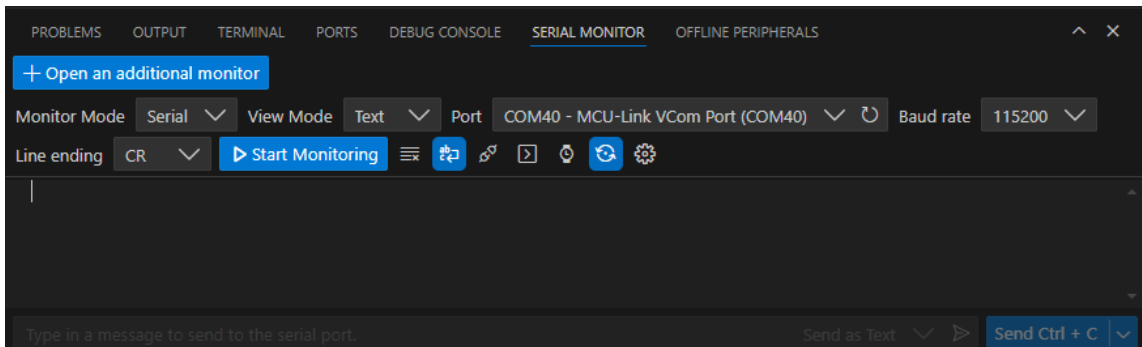
```

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

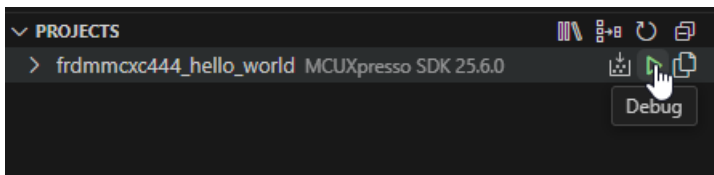
```

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

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



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



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

```

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

```

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

```

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

```

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

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

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

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

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

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

(continues on next page)



(continued from previous page)

```

↪ evkbnmimxrt1060]
INFO: [ 4]west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbnmimxrt1170 -Dcore_id=cm4]
INFO: [ 5]west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbnmimxrt1170 -Dcore_id=cm7]
INFO: [ 6]west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkcmimxrt1060]
INFO: [ 7]west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkmcimx7ulp]
...

```

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 evkbnmimxrt1170 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 evkbnmimxrt1170 --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

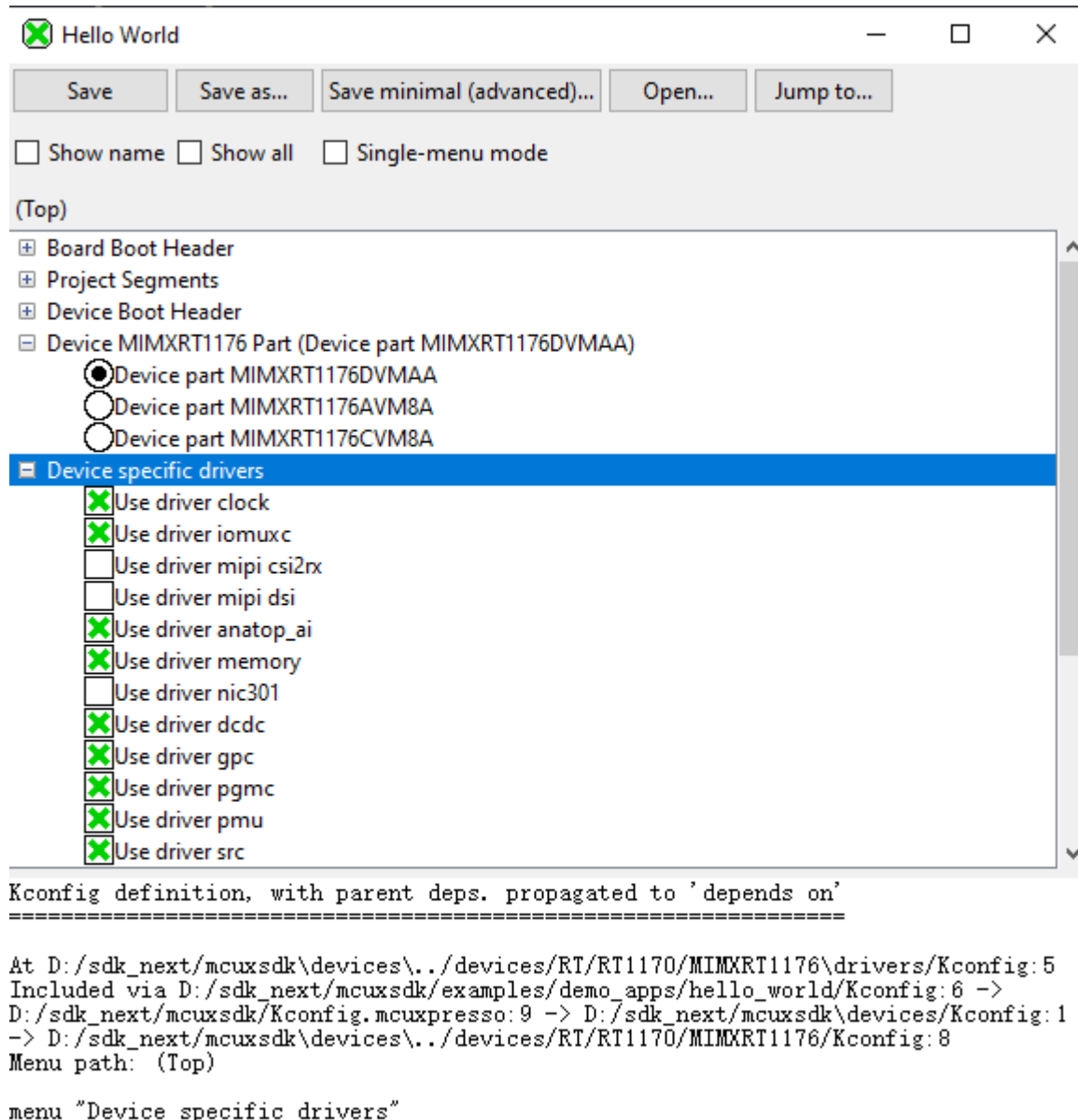
```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

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

2. Run guiconfig target

```
west build -t guiconfig
```

Then you will get the Kconfig GUI launched, like



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

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

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

Flash the hello\_world example:

```
west flash -r linkserver
```

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

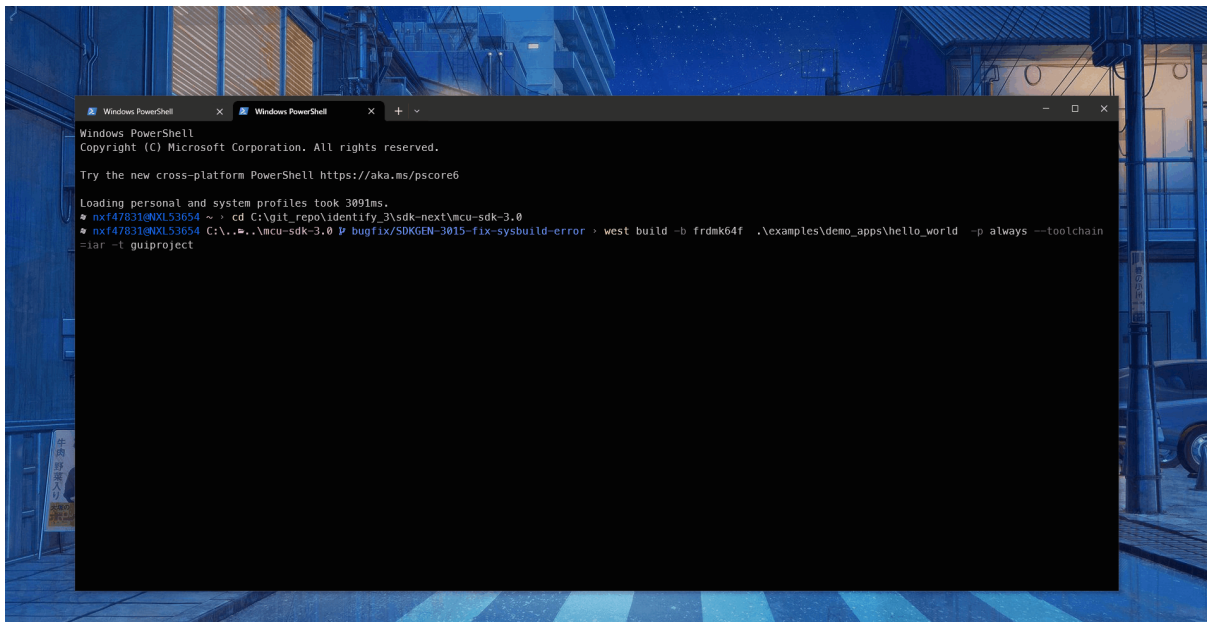
```
west debug -r linkserver
```

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

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

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

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



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

## 1.4 Release Notes

### 1.4.1 MCUXpresso SDK Release Notes

#### Overview

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

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

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

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

### MCUXpresso SDK

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

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

### Development tools

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

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

### Supported development systems

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

|                                      |                                                                            |                                                                            |                                                                            |                                                                                  |
|--------------------------------------|----------------------------------------------------------------------------|----------------------------------------------------------------------------|----------------------------------------------------------------------------|----------------------------------------------------------------------------------|
| De-<br>vel-<br>op-<br>ment<br>boards | MCU devices                                                                |                                                                            |                                                                            |                                                                                  |
| <b>LPCX-<br/>presso</b>              | LPC54005JBD100,<br>LPC54016JET100,<br>LPC54S005JBD100,<br>LPC54S016JET100, | LPC54005JET100,<br>LPC54016JET180,<br>LPC54S005JET100,<br>LPC54S016JET180, | LPC54016JBD100,<br>LPC54018JBD208,<br>LPC54S016JBD100,<br>LPC54S018JBD208, | LPC54016JBD208,<br>LPC54018JET180,<br>LPC54S016JBD208,<br><b>LPC54S018JET180</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.

**MCU Boot** MCU Boot (formerly KBOOT) NXP/Freescale proprietary loader

**coreHTTP** coreHTTP

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

**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".

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

**lwIP** The lwIP TCP/IP stack is pre-integrated with MCUXpresso SDK and runs on top of the MCUXpresso SDK Ethernet driver with Ethernet-capable devices/boards.

For details, see the *lwIP TCP/IP Stack and MCUXpresso SDK Integration User's Guide* (document MCUXSDKLWIPUG).

lwIP is a small independent implementation of the TCP/IP protocol suite.

**LVGL** LVGL Open Source Graphics Library

**llhttp** HTTP parser llhttp

**LittleFS** LittleFS filesystem stack

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

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

**emWin** The MCUXpresso SDK is pre-integrated with the SEGGER emWin GUI middleware. The AppWizard provides developers and designers with a flexible tool to create stunning user interface applications, without writing any code.

**AWS IoT** Amazon Web Service (AWS) IoT Core SDK.

## 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.

### Debug SRAM target with J-Link on MCUXpresso IDE issue

If using the J-Link to Debug SRAM target on the IDE, when clicking the Restart button, the debug fails. The issue is caused because the integrity of the SRAM image cannot be guaranteed after it has been executed, and the MCUXpresso IDE does not reload the code before restarting. Plain load images (images loaded from external SPI flash to SRAMX by the ROM bootloader then executed) cannot be debugged using MCUXpresso version 11.0.0. Code should be developed using the SRAM debug method, then programmed into flash once debugging is completed. For more information, see the Getting Started with MCUXpresso SDK User's Guide (document MCUXSDKG-SUG).

### Example freertos\_spi fails in MCUXpresso IDE

The example freertos\_spi fails in the MCUXpresso IDE environment.



### USB high-speed interrupt endpoint issue

If the user wants to use a high-speed interrupt endpoint, the maximum packet size should be 512 bytes.

### Example freertos\_spi fails in MCUXpresso IDE

The example freertos\_spi fails in the MCUXpresso IDE environment.

### The usart\_wakeup\_deepsleep doesn't work

The example doesn't work.

**Affected toolchains:** mcux **Affected platforms:** lpcxpresso54s018

## 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
- 

#### LPC\_ADC

###### [2.6.0]

- New Features
  - Added new feature macro to distinguish whether the GPADC\_CTRL0\_GPADC\_TSAMP control bit is on the device.

- Added new variable `extendSampleTimeNumber` to indicate the ADC extend sample time.
- Bugfix
  - Fixed the bug that incorrectly sets the `PASS_ENABLE` bit based on the sample time setting.

#### [2.5.3]

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

#### [2.5.2]

- Improvements
  - Integrated different sequence's sample time numbers into one variable.
- Bug Fixes
  - Fixed violation of MISRA C-2012 rule 20.9 .

#### [2.5.1]

- Bug Fixes
  - Fixed ADC conversion sequence priority misconfiguration issue in the `ADC_SetConvSeqAHighPriority()` and `ADC_SetConvSeqBHighPriority()` APIs.
- Improvements
  - Supported configuration ADC conversion sequence sampling time.

#### [2.5.0]

- Improvements
  - Add missing parameter tag of `ADC_DoOffsetCalibration()`.
- Bug Fixes
  - Removed a duplicated API with typo in name: `ADC_EnableShresholdCompareInterrupt()`.

#### [2.4.1]

- Bug Fixes
  - Enabled self-calibration after clock divider be changed to make sure the frequency update be taken.

#### [2.4.0]

- New Features
  - Added new API `ADC_DoOffsetCalibration()` which supports a specific operation frequency.
- Other Changes
  - Marked the `ADC_DoSelfCalibration(ADC_Type *base)` as deprecated.
- Bug Fixes

- Fixed the violations of MISRA C-2012 rules:
  - \* Rule 10.1 10.3 10.4 10.7 10.8 17.7.

### [2.3.2]

- Improvements
  - Added delay after enabling using the ADC GPADC\_CTRL0 LDO\_POWER\_EN bit for JN5189/QN9090.
- New Features
  - Added support for platforms which have only one ADC sequence control/result register.

### [2.3.1]

- Bug Fixes
  - Avoided writing ADC STARTUP register in ADC\_Init().
  - Fixed Coverity zero divider error in ADC\_DoSelfCalibration().

### [2.3.0]

- Improvements
  - Updated “ADC\_Init()” “ADC\_GetChannelConversionResult()” API and “adc\_resolution\_t” structure to match QN9090.
  - Added “ADC\_EnableTemperatureSensor” API.

### [2.2.1]

- Improvements
  - Added a brief delay in uSec after ADC calibration start.

### [2.2.0]

- Improvements
  - Updated “ADC\_DoSelfCalibration” API and “adc\_config\_t” structure to match LPC845.

### [2.1.0]

- Improvements
  - Renamed “ADC\_EnableShresholdCompareInterrupt” to “ADC\_EnableThresholdCompareInterrupt”.

### [2.0.0]

- Initial version.
-

## AES

### [2.0.3]

- Edit aes\_one\_block() function to be interrupt safe.

### [2.0.2]

- Fix MISRA-2012 issues.

### [2.0.1]

- Improvements
  - GCM constant time tag comparison.

### [2.0.0]

- Initial version.
- 

## CLOCK

### [2.3.3]

- Improvements
  - Added lost comments for some enumerations.

### [2.3.2]

- Bug Fixes
  - Fixed violation of MISRA C-2012 rule 5.7

### [2.3.1]

- Bug Fixes
  - Fixed MISRA C-2012 rule 10.1, rule 10.4, rule 10.8, rule 15.5 and so on.
  - Fixed IAR warning Pa082 for the clock driver.

### [2.3.0]

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

### [2.2.0]

- New Feature:
  - add new APIs including CLOCK\_GetEmcClkFreq and CLOCK\_GetMCanClkFreq due to removing some variables in enum clock\_name\_t

### [2.1.0]

- Bug Fix:
  - Fix flexcomm0-9 clock calculation.
  - Correct the return frequency of `CLOCK_GetFrgClkFreq`.
  - Fix the bug in function `CLOCK_GetPllConfig()` to refine the cache feature.
  - Update the code to suppress the incorrect configuration in `CLOCK_GetUsbPllOutFromSetup()`.
  - Fix C++ build errors in `CLOCK_GetClockAttachId()` and `CLOCK_AttachClk()`.
- New feature
  - Adding new API `CLOCK_DelayAtLeastUs()` implemented by DWT to allow users set delay in unit of microsecond.

### [2.0.4]

- Bug Fix:
  - Fix attach incorrect `attach_id`.

### [2.0.3]

- New Feature:
  - add get actual clock attach id api to allow users to obtain the actual clock source in target register.
- Bug Fix:
  - The attach clock and get actual clock attach id apis should check combination of two clock source.
- Optimization:
  - Make the judgement statments more clear.
  - Strengthen the compatibility of clock attach id.
  - Remove some unmeaningful definitions and add some useful ones to enhance readability.

### [2.0.2]

- Change `CLOCK_SetupFROClocking` from a macro to a function for different FRO setting address per different ROM version.

### [2.0.1]

- some minor fixes.

### [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.
- 

## CRC

### [2.1.1]

- Fix MISRA issue.

### [2.1.0]

- Add CRC\_WriteSeed function.

### [2.0.2]

- Fix MISRA issue.

### [2.0.1]

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

### [2.0.0]

- Initial version.
-

## CTIMER

### [2.3.3]

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

### [2.3.2]

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

### [2.3.1]

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

### [2.3.0]

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

### [2.2.2]

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

### [2.2.1]

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

### [2.2.0]

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

#### [2.1.0]

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

#### [2.0.3]

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

#### [2.0.2]

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

#### [2.0.1]

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

#### [2.0.0]

- Initial version.
- 

### LPC\_DMA

#### [2.5.3]

- Improvements
  - Add assert in DMA\_SetChannelXferConfig to prevent XFERCOUNT value overflow.

#### [2.5.2]

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

#### [2.5.1]

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

**[2.5.0]**

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

**[2.4.4]**

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

**[2.4.3]**

- Improvements
  - Added features FSL\_FEATURE\_DMA\_DESCRIPTOR\_ALIGN\_SIZEn/FSL\_FEATURE\_DMA0\_DESCRIPTOR\_ALIGN\_SIZEn to support the descriptor align size not constant in the two instances.

**[2.4.2]**

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

**[2.4.1]**

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

**[2.4.0]**

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

**[2.3.0]**

- Bug Fixes
  - Removed DMA\_HandleIRQ prototype definition from header file.
  - Added DMA\_IRQHandle prototype definition in header file.

#### [2.2.5]

- Improvements
  - Added new API DMA\_SetupChannelDescriptor to support configuring wrap descriptor.
  - Added wrap support in function DMA\_SubmitChannelTransfer.

#### [2.2.4]

- Bug Fixes
  - Fixed the issue that macro DMA\_CHANNEL\_CFER used wrong parameter to calculate DSTINC.

#### [2.2.3]

- Bug Fixes
  - Improved DMA driver Deinit function for correct logic order.
- Improvements
  - Added API DMA\_SubmitChannelTransferParameter to support creating head descriptor directly.
  - Added API DMA\_SubmitChannelDescriptor to support ping pong transfer.
  - Added macro DMA\_ALLOCATE\_HEAD\_DESCRIPTOR/DMA\_ALLOCATE\_LINK\_DESCRIPTOR to simplify DMA descriptor allocation.

#### [2.2.2]

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

#### [2.2.1]

- Bug Fixes
  - Fixed Coverity issue.

#### [2.2.0]

- Improvements
  - Changed API DMA\_SetupDMADescriptor to non-static.
  - Marked APIs below as deprecated.
    - \* DMA\_PrepareTransfer.
    - \* DMA\_Submit transfer.
  - Added new APIs as below:
    - \* DMA\_SetChannelConfig.
    - \* DMA\_PrepareChannelTransfer.
    - \* DMA\_InstallDescriptorMemory.
    - \* DMA\_SubmitChannelTransfer.
    - \* DMA\_SetChannelConfigValid.

- \* DMA\_DoChannelSoftwareTrigger.
- \* DMA\_LoadChannelTransferConfig.

#### [2.0.1]

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

#### [2.0.0]

- Initial version.
- 

### DMIC

#### [2.3.3]

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

#### [2.3.2]

- New Features
  - Supported 4 channels in driver.

#### [2.3.1]

- Bug Fixes
  - Fixed the issue that DMIC\_EnableChannelDma and DMIC\_EnableChannelFifo did not clean relevant bits.

#### [2.3.0]

- Improvements
  - Added new apis DMIC\_ResetChannelDecimator/DMIC\_EnableChannelGlobalSync/DMIC\_DisableChannel

#### [2.2.1]

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

#### [2.2.0]

- Bug Fixes
  - Corrected the usage of feature FSL\_FEATURE\_DMIC\_IO\_HAS\_NO\_BYPASS.

### [2.1.1]

- Improvements
  - Added feature FSL\_FEATURE\_DMIC\_HAS\_NO\_IOCFCG for IOCFCG register.

### [2.1.0]

- New Features
  - Added API DMIC\_EnableChannelInterrupt/DMIC\_EnableChannelDma to replace API DMIC\_SetOperationMode.
  - Added API DMIC\_SetIOCFCG and marked DMIC\_ConfigIO as deprecated.
  - Added API DMIC\_EnableChannelSignExtend to support sign extend feature.

### [2.0.5]

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

### [2.0.4]

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

### [2.0.3]

- New Features
- Supported linked transfer in DMIC DMA driver.
- Added new API DMIC\_EnableChannelFifo/DMIC\_DoFifoReset/DMIC\_InstallDMADescriptor.

### [2.0.2]

- New Features
  - Supported more channels in driver.

### [2.0.1]

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

### [2.0.0]

- Initial version.
-



## DMIC\_DMA

### [2.4.2]

- Bug Fixes
  - Fixed coverity High Impact finding

### [2.4.1]

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

### [2.4.0]

- Bug Fixes
  - Fixed the issue that DMIC\_TransferAbortReceiveDMA can not disable dmic and dma request issue.

### [2.3.1]

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

### [2.3.0]

- Refer DMIC driver change log 2.0.1 to 2.3.0
- 

## EMC

### [2.0.4]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.4, 10.8, 11.9, 14.2, 14.3, 14.4.

### [2.0.3]

- Improvements
  - Used SDK\_DelayAtLeastUs instead of for loop during the dynamic memory initialization.

### [2.0.3]

- Improvements
  - Replaced deprecated enumerator CLOCK\_GetFreq(kCLOCK EMC) with CLOCK\_GetEmcClkFreq().

### [2.0.2]

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

#### [2.0.1]

- Improvements
  - Added const for two BASE values.

#### [2.0.0]

- Initial version.
- 

### LPC\_ENET

#### [2.3.5]

- Bug Fixes
  - Fixed ENET\_GetMacAddr address byte order not matching ENET\_SetMacAddr.

#### [2.3.4]

- Bug Fixes
  - Fixed the issue that free wrong buffer address when one frame stores in multiple buffers and memory pool is not enough to allocate these buffers to receive one complete frame.
  - Fixed the issue that ENET\_DropFrame checks the buffer descriptor flag after it has been re-initialized.
  - Fixed the ENET\_GetRxFram FCS calculation issue.
  - Fixed the issue that there's no valid error type in the return structure when Rx error bit is set.

#### [2.3.3]

- Bug Fixes
  - Fixed the issue that ENET\_SetSMI uses wrong clock source to calculate the divisor.

#### [2.3.2]

- New features
  - Added hardware checksum acceleration support.
- Bug Fixes
  - Fixed the issue that enable/disable interrupt APIs miss part of configuration.

#### [2.3.1]

- Improvements
  - update ENET\_SetSYSControl to support mcx family.

**[2.3.0]**

- Improvements
  - Added MDIO access wrapper APIs for ease of use.

**[2.2.0]**

- Bug Fixes
  - Corrected the timestamp retrieving code in ReadFrame.
- New Features
  - Supported zero copy Rx with new APIs.
- Improvements
  - Removed 4 bytes CRC data in ReadFrame function, not give them to user.
  - Deleted previous timestamp rings which store Tx/Rx timestamp temporarily for further retrieving. Now get Rx timestamp directly with receiving frame API, and get Tx timestamp in Tx over interrupt handler callback.
  - Added channel parameter for the SendFrame function, let user to decide which kind of frame can be sent from specified channel.
  - Supported scattered Tx buffers and more Tx configurations in SendFrame which aren't integrated.
  - Adjusted the callback location in Tx reclaim function. When use multiple BDs for Tx, only last BD transmit over interrupt event calls the callback. It simplifies the usage of Tx reclaiming.
  - Added interrupt configuration in config parameter for ENET\_Init() to simplify the interrupt enable.
  - Changed the Tx/Rx descriptor name to common name rather than previous read format name which make user confused when driver uses it as write-back format.

**[2.1.5]**

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 3.1,5.8,8.4,8.6,10.1,10.3,10.4,10.6,10.8,11.6,11.9,12.2,14.4,15.6

**[2.1.4]**

- Bug Fixes
  - Fixed the MDC clock divider setting issue occurring when core clock range exceeds 150M.

**[2.1.3]**

- In ENET\_StartRXTx, updated to enable TX and RX at the same time to avoid issue where ENET module could not work under 10 M.
- Changed to use CLOCK\_GetCoreSysClkFreq() instead of SystemCoreClock to get accurate core clock.

### [2.1.2]

- Bug Fixes
  - Fixed ENET receive issue where it sometimes lost some unicast packets. The issue is caused by the program timing issue for writing MAC\_ADDR\_LOW and MAC\_ADDR\_HIGH.

### [2.1.1]

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

### [2.1.0]

- New Features
  - Added two APIs to set the ENET to ACCPET or reject the multicast frames.

### [2.0.0]

- Initial version.
- 

## FLEXCOMM

### [2.0.2]

- Bug Fixes
  - Fixed typos in FLEXCOMM15\_DriverIRQHandler().
  - Fixed MISRA issues.
    - \* Fixed rules 10.1, 10.3, 10.4, 10.7, 10.8, 11.3, 11.6, 11.8, 11.9, 13.5.
- Improvements
  - Added instance calculation in FLEXCOMM16\_DriverIRQHandler() to align with Flexcomm 14 and 15.

### [2.0.1]

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

### [2.0.0]

- Initial version.
- 

## FMEAS

### [2.1.1]

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

**[2.1.0]**

- Updated “FMEAS\_GetFrequency”, ”FMEAS\_StartMeasure”, ”FMEAS\_IsMeasureComplete” API and add definition to match ASYNC\_SYSCON.

**[2.0.0]**

- Initial version ported from LPCOpen.
- 

**GINT****[2.1.1]**

- Improvements
  - Added support for platforms with PORT\_POL and PORT\_ENA registers without arrays.

**[2.1.0]**

- Improvements
  - Updated for platforms which only has one port.

**[2.0.3]**

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

**[2.0.2]**

- Bug Fixes
  - Fixed issue for MISRA-2012 check.
    - \* Fixed rule 17.7.

**[2.0.1]**

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

**[2.0.0]**

- Initial version.
- 

**GPIO****[2.1.7]**

- Improvements
  - Enhanced GPIO\_PinInit to enable clock internally.

#### [2.1.6]

- Bug Fixes
  - Clear bit before set it within GPIO\_SetPinInterruptConfig() API.

#### [2.1.5]

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

#### [2.1.4]

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

#### [2.1.3]

- Improvements
  - Updated “GPIO\_PinInit” API. If it has DIRCLR and DIRSET registers, use them at set 1 or clean 0.

#### [2.1.2]

- Improvements
  - Removed deprecated APIs.

#### [2.1.1]

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

#### [2.1.0]

- New Features
  - Added GPIO initialize API.

#### [2.0.0]

- Initial version.
-

## I2C

### [2.3.3]

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

### [2.3.2]

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

### [2.3.1]

- Improvement
  - Before master transfer with transactional APIs, enable master function while disable slave function and vice versa for slave transfer to avoid the one affecting the other.
- Bug Fixes
  - Fixed bug in I2C\_SlaveEnable that the slave enable/disable should not affect the other register bits.

### [2.3.0]

- Improvement
  - Added new return codes kStatus\_I2C\_EventTimeout and kStatus\_I2C\_SclLowTimeout, and added the check for event timeout and SCL timeout in I2C master transfer.
  - Fixed bug in slave transfer that the address match event should be invoked before not after slave transmit/receive event.

### [2.2.0]

- New Features
  - Added enumeration `_i2c_status_flags` to include all previous master and slave status flags, and added missing status flags.
  - Modified `I2C_GetStatusFlags` to get all I2C flags.
  - Added API `I2C_ClearStatusFlags` to clear all clearable flags not just master flags.
  - Modified master transactional APIs to enable bus event timeout interrupt during transfer, to avoid glitch on bus causing transfer hangs indefinitely.
- Bug Fixes
  - Fixed bug that status flags and interrupt enable masks share the same enumerations by adding enumeration `_i2c_interrupt_enable` for all master and slave interrupt sources.

**[2.1.0]**

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

**[2.0.8]**

- Bug Fixes
  - Fixed `I2C_MasterSetBaudRate` issue that `MSTSCLOW` and `MSTSCHIGH` are incorrect when `MSTTIME` is odd.

**[2.0.7]**

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

**[2.0.6]**

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

**[2.0.5]**

- Bug Fixes
  - Fixed wrong assignment for `datasize` in `I2C_InitTransferStateMachineDMA`.
  - Fixed wrong working flow in `I2C_RunTransferStateMachineDMA` to ensure master can work in no start flag and no stop flag mode.



- Fixed wrong working flow in I2C\_RunTransferStateMachine and added kReceive-DataBeginState in \_i2c\_transfer\_states to ensure master can work in no start flag and no stop flag mode.
- Fixed wrong handle state in I2C\_MasterTransferDMAHandleIRQ. After all the data has been transfered or nak is returned, handle state should be changed to idle.
- Improvements
  - Rounded up the calculated divider value in I2C\_MasterSetBaudRate.

#### [2.0.4]

- Improvements
  - Updated the I2C\_WATI\_TIMEOUT macro to unified name I2C\_RETRY\_TIMES
  - Updated the “I2C\_MasterSetBaudRate” API to support baudrate configuration for feature QN9090.
- Bug Fixes
  - Fixed build warnning caused by uninitialized variable.
  - Fixed COVERITY issue of unchecked return value in I2C\_RTOS\_Transfer.

#### [2.0.3]

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

#### [2.0.2]

- Improvements
  - In slave IRQ:
    1. Changed slave receive process to first set the I2C\_SLVCTL\_SLVCONTINUE\_MASK to acknowledge the received data, then do data receive.
    2. Improved slave transmit process to set the I2C\_SLVCTL\_SLVCONTINUE\_MASK immediately after writing the data.

#### [2.0.1]

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

#### [2.0.0]

- Initial version.
- 

## I2S

#### [2.3.2]

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

### [2.3.1]

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

### [2.3.0]

- Improvements
  - Added api I2S\_InstallDMADescriptorMemory/I2S\_TransferSendLoopDMA/I2S\_TransferReceiveLoopDMA to support loop transfer.
  - Added api I2S\_EmptyTxFifo to support blocking flush tx fifo.
  - Updated api I2S\_TransferAbortDMA by removed the blocking flush tx fifo from this function.
- Bug Fixes
  - Removed the while loop in abort transfer function to fix the dead loop issue under specific user case.

### [2.2.2]

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

### [2.2.1]

- Improvements
  - Added feature FSL\_FEATURE\_FLEXCOMM\_INSTANCE\_I2S\_SUPPORT\_SECONDARY\_CHANNELn for the SOC has parts of instance support secondary channel.
- Bug Fixes
  - Added volatile statement for the state variable of i2s\_handle and enable the mainline channel pair before enable interrupt to avoid the issue of code excution reordering which may cause the interrupt generated unexpectedly.

### [2.2.0]

- Improvements
  - Added 8/16/24 bits mono data format transfer support in I2S driver.
  - Added new apis I2S\_SetBitClockRate.
- Bug Fixes
  - Fixed the PA082 build warning.
  - Fixed the sign-compare warning.
  - Fixed violations of the MISRA C-2012 rules 10.4, 10.8, 11.9, 10.1, 11.3, 13.5, 11.8, 10.3, 10.7.
  - Fixed the Operand don't affect result Coverity issue.

**[2.1.0]**

- Improvements
  - Added a feature for the FLEXCOMM which supports I2S and has interconnection with DMIC.
  - Used a feature to control PDMDATA instead of I2S\_CFG1\_PDMDATA.
  - Added member bytesPerFrame in i2s\_dma\_handle\_t, used for DMA transfer width configure, instead of using sizeof(uint32\_t) hardcoded.
  - Used the macro provided by DMA driver to define the I2S DMA descriptor.
- Bug Fixes
  - Fixed the issue that I2S DMA driver always generated duplicate callback.

**[2.0.3]**

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

**[2.0.2]**

- New Features
  - Added ENABLE\_IRQ handle after register I2S interrupt handle.

**[2.0.1]**

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

**[2.0.0]**

- Initial version.
- 

**I2S\_DMA****[2.3.3]**

- Bug Fixes
  - Fixed data size limit does not match the macro DMA\_MAX\_TRANSFER\_BYTES issue.

**[2.3.2]**

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

**[2.3.1]**

- Refer I2S driver change log 2.0.1 to 2.3.1
-

## IAP

### [2.0.7]

- Bug Fixes
  - Fixed IAP\_ReinvokeISP bug that can't support UART ISP auto baud detection.

### [2.0.6]

- Bug Fixes
  - Fixed IAP\_ReinvokeISP wrong parameter setting.

### [2.0.5]

- New Feature
  - Added support config flash memory access time.

### [2.0.4]

- Bug Fixes
  - Fixed the violations of MISRA 2012 rules 9.1

### [2.0.3]

- New Features
  - Added support for LPC 845's FAIM operation.
  - Added support for LPC 80x's fixed reference clock for flash controller.
  - Added support for LPC 5411x's Read UID command useless situation.
- Improvements
  - Improved the document and code structure.
- Bug Fixes
  - Fixed the violations of MISRA 2012 rules:
    - \* Rule 10.1 10.3 10.4 17.7

### [2.0.2]

- New Features
  - Added an API to read generated signature.
- Bug Fixes
  - Fixed the incorrect board support of IAP\_ExtendedFlashSignatureRead().

### [2.0.1]

- New Features
  - Added an API to read factory settings for some calibration registers.
- Improvements
  - Updated the size of result array in part APIs.

**[2.0.0]**

- Initial version.
- 

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

- Bug Fixes
  - Fixed CERT-C violations.

**[2.0.9]**

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

**[2.0.8]**

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

**[2.0.7]**

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

**[2.0.6]**

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

**[2.0.5]**

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

**[2.0.4]**

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

**[2.0.3]**

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

#### [2.0.2]

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

#### [2.0.1]

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

#### [2.0.0]

- Initial version.
- 

### IOCON

#### [2.2.0]

- Improvements
  - Removed duplicate macro definitions.
  - Renamed 'IOCON\_I2C\_SLEW' macro to 'IOCON\_I2C\_MODE' to match its companion 'IOCON\_GPIO\_MODE'. The original is kept as a deprecated symbol.

#### [2.1.2]

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

#### [2.1.1]

- Updated left shift format with mask value instead of a constant value to automatically adapt to all platforms.

#### [2.1.0]

- Added a new IOCON\_PinMuxSet() function with a feature IOCON\_ONE\_DIMENSION for LPC845MAX board.

#### [2.0.0]

- Initial version.
- 

### LPC\_LCDC

#### [2.0.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 3.1, 10.3, 10.4, 10.6, 10.7, 10.8, 14.4, 17.7.

**[2.0.1]**

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

**[2.0.0]**

- Initial version.
- 

**MCAN****[2.4.2]**

- Bug Fixes
  - Fixed MISRA issue rule-10.3, rule-10.6, rule-10.7 and rule-15.7.

**[2.4.1]**

- Bug Fixes
  - Fixed incorrect fifo1 status on message lost.

**[2.4.0]**

- Improvements
  - Add MCAN\_CalculateSpecifiedTimingValues() API to get CAN bit timing parameter with user-defined settings.
  - Add MCAN\_FDCalculateSpecifiedTimingValues() API to get CANFD bit timing parameter with user-defined settings.

**[2.3.2]**

- Bug Fixes
  - Fix MISRA C-2012 issue 10.1 and 10.4.

**[2.3.1]**

- Bug Fixes
  - Fixed the issue that MCAN\_TransferSendNonBlocking() API can't send remote frame.

**[2.3.0]**

- Improvements
  - Add MCAN\_SetMessageRamConfig() API to perform global message RAM configure.
  - Add MCAN\_EnterInitialMode() API.

## [2.2.0]

- Improvements
  - Add MCAN\_SetBaudRate/MCAN\_SetBaudRateFD APIs to make users easy to set CAN baud rate.

## [2.1.8]

- Bug Fixes
  - Add check FIFO status code in MCAN\_ReadRxFifo() to avoid read back empty frame and wrong trigger the FIFO index increase.

## [2.1.7]

- Bug Fixes
  - Fixed the clear error flags issue in MCAN\_TransferHandleIRQ() API.
  - Fixed the Solve Tx interrupt issue in MCAN\_TransferHandleIRQ() API which may abort the unhandled transfers.
  - Remove disable global tx interrupt from MCAN\_TransferAbortSend API.

## [2.1.6]

- Bug Fixes
  - Fixed the issue of writing 1 in the following functions.
    - MCAN\_TransmitAddRequest
    - MCAN\_TransmitCancelRequest
    - MCAN\_ClearRxBufferStatusFlag

## [2.1.5]

- Bug Fixes
  - Fix MISRA C-2012 issue.

## [2.1.4]

- Improvements
  - Updated improve timing APIs to make it can calculate the CiA recommended timing configuration.
  - Implement Transmitter Delay Compensation feature.
  - Modify the default baudRateFD value to 2M.
- Bug Fixes
  - Fixed the code error issue in MCAN\_ClearStatusFlag() to avoid clear all flags.



### [2.1.3]

- Bug Fixes
  - Fixed the code error issue and simplified the algorithm in improved timing APIs.
    - \* MCAN\_CalculateImprovedTimingValues
    - \* MCAN\_FDCalculateImprovedTimingValues

### [2.1.2]

- Bug Fixes
  - Fixed the non-divisible case in improved timing APIs.
    - \* MCAN\_CalculateImprovedTimingValues
    - \* MCAN\_FDCalculateImprovedTimingValues

### [2.1.1]

- Bug Fixes
  - MISRA C-2012 issue check.
    - \* Fixed rules, containing: rule-10.1, rule-10.3, rule-10.4, rule-10.6, rule-10.7, rule-10.8, rule-11.9, rule-14.4, rule-15.5, rule-15.6, rule-15.7, rule-17.7, rule-18.4, rule-2.2, rule-21.15, rule-5.8, rule-8.3.
    - \* Fixed the Coverity issue of BAD\_SHIFT in MCAN.
    - \* Fixed the issue of Pa082 warning.
    - \* Fixed the issue of dropping interrupt flags in handler function.

### [2.1.0]

- Bug Fixes
  - Fixed Coverity issue FORWARD\_NULL.
  - Fixed Clang issue.
  - Fixed legacy issue in the driver and changed default bus data baud rate for CANFD.
- Improvements
  - Implemented feature for improved timing configuration.

### [2.0.3]

- Improvements
  - Used memset to initialize the structure before using.
  - Added function definition comment in c file.
  - Updated source file license to SPDX BSD\_3.
  - Corrected capital mistake of Fifo and fifo.
  - Reset the MCAN module in LPC drivers after clock enable.

#### [2.0.2]

- Bug Fixes
  - Picked MISRA fixed in release 8 branch.
  - MISRA C 2012 fixed regarding FlexCAN and MCAN address update.
- Improvements
  - Implemented for delay/retry in MCAN driver.

#### [2.0.1]

- Improvements
  - LPC54608 chip did not support the FD feature, so added a feature macro for it.

#### [2.0.0]

- Initial version.
- 

### MRT

#### [2.0.5]

- Bug Fixes
  - Fixed CERT INT31-C violations.

#### [2.0.4]

- Improvements
  - Don't reset MRT when there is not system level MRT reset functions.

#### [2.0.3]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1 and 10.4.
  - Fixed the wrong count value assertion in MRT\_StartTimer API.

#### [2.0.2]

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

#### [2.0.1]

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

#### [2.0.0]

- Initial version.
-

## OTP

### [2.0.1]

- Bug Fixes
  - Fixed MISRA-C 2012 violations.

### [2.0.0]

- Initial version.
- 

## PINT

### [2.2.0]

- Fixed
  - Fixed the issue that clear interrupt flag when it's not handled. This causes events to be lost.
- Changed
  - Used one callback for one PINT instance. It's unnecessary to provide different callbacks for all PINT events.

### [2.1.13]

- Improvements
  - Added instance array for PINT to adapt more devices.
  - Used release reset instead of reset PINT which may clear other related registers out of PINT.

### [2.1.12]

- Bug Fixes
  - Fixed coverity issue.

### [2.1.11]

- Bug Fixes
  - Fixed MISRA C-2012 rule 10.7 violation.

### [2.1.10]

- New Features
  - Added the driver support for MCXN10 platform with combined interrupt handler.

### [2.1.9]

- Bug Fixes
  - Fixed MISRA-2012 rule 8.4.

#### [2.1.8]

- Bug Fixes
  - Fixed MISRA-2012 rule 10.1 rule 10.4 rule 10.8 rule 18.1 rule 20.9.

#### [2.1.7]

- Improvements
  - Added fully support for the SECPINT, making it can be used just like PINT.

#### [2.1.6]

- Bug Fixes
  - Fixed the bug of not enabling common pint clock when enabling security pint clock.

#### [2.1.5]

- Bug Fixes
  - Fixed issue for MISRA-2012 check.
    - \* Fixed rule 10.1 rule 10.3 rule 10.4 rule 10.8 rule 14.4.
  - Changed interrupt init order to make pin interrupt configuration more reasonable.

#### [2.1.4]

- Improvements
  - Added feature to control distinguish PINT/SECPINT relevant interrupt/clock configurations for PINT\_Init and PINT\_Deinit API.
  - Swapped the order of clearing PIN interrupt status flag and clearing pending NVIC interrupt in PINT\_EnableCallback and PINT\_EnableCallbackByIndex function.
- Bug Fixes
  - \* Fixed build issue caused by incorrect macro definitions.

#### [2.1.3]

- Bug fix:
  - Updated PINT\_PinInterruptClrStatus to clear PINT interrupt status when the bit is asserted and check whether was triggered by edge-sensitive mode.
  - Write 1 to IST corresponding bit will clear interrupt status only in edge-sensitive mode and will switch the active level for this pin in level-sensitive mode.
  - Fixed MISRA c-2012 rule 10.1, rule 10.6, rule 10.7.
  - Added FSL\_FEATURE\_SECPINT\_NUMBER\_OF\_CONNECTED\_OUTPUTS to distinguish IRQ relevant array definitions for SECPINT/PINT on lpc55s69 board.
  - Fixed PINT driver c++ build error and remove index offset operation.

#### [2.1.2]

- Improvement:
  - Improved way of initialization for SECPINT/PINT in PINT\_Init API.

**[2.1.1]**

- Improvement:
  - Enabled secure pint interrupt and add secure interrupt handle.

**[2.1.0]**

- Added PINT\_EnableCallbackByIndex/PINT\_DisableCallbackByIndex APIs to enable/disable callback by index.

**[2.0.2]**

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

**[2.0.1]**

- Bug fix:
  - Updated PINT driver to clear interrupt only in Edge sensitive.

**[2.0.0]**

- Initial version.
- 

**POWER****[2.1.0]**

- New features
  - Added BOD control APIs.

**[2.0.0]**

- initial version.
- 

**PUF****[2.2.0]**

- Add support for kPUF\_KeySlot4.
- Add new PUF\_ClearKey() function, that clears a desired PUF internal HW key register.

**[2.1.6]**

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

**[2.1.5]**

- Use common SDK delay in puf\_wait\_usec().

#### [2.1.4]

- Replace register uint32\_t ticksCount with volatile uint32\_t ticksCount in puf\_wait\_usec() to prevent optimization out delay loop.

#### [2.1.3]

- Fix MISRA C-2012 issue.

#### [2.1.2]

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

#### [2.1.1]

- Fix ARMGCC build warning .

#### [2.1.0]

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

#### [2.0.3]

- Fix MISRA C-2012 issue.

#### [2.0.2]

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

#### [2.0.1]

- Bug Fixes:
  - Fixed puf\_wait\_usec function optimization issue.

#### [2.0.0]

- Initial version.
- 

## RESET

#### [2.4.0]

- Improvements
  - Add RESET\_ReleasePeripheralReset API.

**[2.0.1]**

- Update component full\_name to “Reset Driver”.

**[2.0.0]**

- initial version.
- 

**RIT****[2.1.2]**

- Bug Fixes
  - Fixed CERT INT31-C violations.

**[2.1.1]**

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 11.9, 17.7.

**[2.1.0]**

- Bug Fixes
  - Fixed issue for wrong implementation of clearing counter API in RIT driver.

**[2.0.2]**

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

**[2.0.1]**

- Bug Fixes
  - Fixed incorrect comments of some APIs.

**[2.0.0]**

- Initial version.
- 

**RNG****[2.1.0]**

- Renamed function RNG\_GetRandomData() to RNG\_GetRandomDataWord(). Added function RNG\_GetRandomData() which discarding next 32 words after reading RNG register which results into better entropy, as is recommended in UM.
- API is aligned with other RNG driver, having similar functionality as other RNG/TRNG drivers.

## [2.0.0]

- Initial version.
- 

## RTC

### [2.2.0]

- New Features
  - Created new APIs for the RTC driver.
    - \* RTC\_EnableSubsecCounter
    - \* RTC\_GetSubsecValue

### [2.1.3]

- Bug Fixes
  - Fixed issue that RTC\_GetWakeupCount may return wrong value.

### [2.1.2]

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

### [2.1.1]

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

### [2.1.0]

- Bug Fixes
  - Created new APIs for the RTC driver.
    - \* RTC\_EnableTimer
    - \* RTC\_EnableWakeUpTimerInterruptFromDPD
    - \* RTC\_EnableAlarmTimerInterruptFromDPD
    - \* RTC\_EnableWakeupTimer
    - \* RTC\_GetEnabledWakeupTimer
    - \* RTC\_SetSecondsTimerMatch
    - \* RTC\_GetSecondsTimerMatch
    - \* RTC\_SetSecondsTimerCount
    - \* RTC\_GetSecondsTimerCount
  - deprecated legacy APIs for the RTC driver.
    - \* RTC\_StartTimer
    - \* RTC\_StopTimer
    - \* RTC\_EnableInterrupts



- \* RTC\_DisableInterrupts
- \* RTC\_GetEnabledInterrupts

#### [2.0.0]

- Initial version.
- 

### SCTIMER

#### [2.5.1]

- Bug Fixes
  - Fixed bug in SCTIMER\_SetupCaptureAction: When kSCTIMER\_Counter\_H is selected, events 12-15 and capture registers 12-15 CAPn\_H field can't be used.

#### [2.5.0]

- Improvements
  - Add SCTIMER\_GetCaptureValue API to get capture value in capture registers.

#### [2.4.9]

- Improvements
  - Supported platforms which don't have system level SCTIMER reset.

#### [2.4.8]

- Bug Fixes
  - Fixed the issue that the SCTIMER\_UpdatePwmDutycycle() can't writes MATCH\_H bit and RELOADn\_H.

#### [2.4.7]

- Bug Fixes
  - Fixed the issue that the SCTIMER\_UpdatePwmDutycycle() can't configure 100% duty cycle PWM.

#### [2.4.6]

- Bug Fixes
  - Fixed the issue where the H register was not written as a word along with the L register.
  - Fixed the issue that the SCTIMER\_SetCOUNTValue() is not configured with high 16 bits in unify mode.

#### [2.4.5]

- Bug Fixes
  - Fix SCT\_EV\_STATE\_STATEMSKn macro build error.

#### [2.4.4]

- Bug Fixes
  - Fix MISRA C-2012 issue 10.8.

#### [2.4.3]

- Bug Fixes
  - Fixed the wrong way of writing CAPCTRL and REGMODE registers in SCTIMER\_SetupCaptureAction.

#### [2.4.2]

- Bug Fixes
  - Fixed SCTIMER\_SetupPwm 100% duty cycle issue.

#### [2.4.1]

- Bug Fixes
  - Fixed the issue that MATCHn\_H bit and RELOADn\_H bit could not be written.

#### [2.4.0]

#### [2.3.0]

- Bug Fixes
  - Fixed the potential overflow issue of pulseperiod variable in SCTIMER\_SetupPwm/SCTIMER\_UpdatePwmDutycycle API.
  - Fixed the issue of SCTIMER\_CreateAndScheduleEvent API does not correctly work with 32 bit unified counter.
  - Fixed the issue of position of clear counter operation in SCTIMER\_Init API.
- Improvements
  - Update SCTIMER\_SetupPwm/SCTIMER\_UpdatePwmDutycycle to support generate 0% and 100% PWM signal.
  - Add SCTIMER\_SetupEventActiveDirection API to configure event activity direction.
  - Update SCTIMER\_StartTimer/SCTIMER\_StopTimer API to support start/stop low counter and high counter at the same time.
  - Add SCTIMER\_SetCounterState/SCTIMER\_GetCounterState API to write/read counter current state value.
  - Update APIs to make it meaningful.
    - \* SCTIMER\_SetEventInState
    - \* SCTIMER\_ClearEventInState
    - \* SCTIMER\_GetEventInState

### [2.2.0]

- Improvements
  - Updated for 16-bit register access.

### [2.1.3]

- Bug Fixes
  - Fixed the issue of uninitialized variables in SCTIMER\_SetupPwm.
  - Fixed the issue that the Low 16-bit and high 16-bit work independently in SCTIMER driver.
- Improvements
  - Added an enumerable macro of unify counter for user.
    - \* kSCTIMER\_Counter\_U
  - Created new APIs for the RTC driver.
    - \* SCTIMER\_SetupStateLdMethodAction
    - \* SCTIMER\_SetupNextStateActionwithLdMethod
    - \* SCTIMER\_SetCOUNTValue
    - \* SCTIMER\_GetCOUNTValue
    - \* SCTIMER\_SetEventInState
    - \* SCTIMER\_ClearEventInState
    - \* SCTIMER\_GetEventInState
  - Deprecated legacy APIs for the RTC driver.
    - \* SCTIMER\_SetupNextStateAction

### [2.1.2]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.3, 10.4, 10.6, 10.7, 11.9, 14.2 and 15.5.

### [2.1.1]

- Improvements
  - Updated the register and macro names to align with the header of devices.

### [2.1.0]

- Bug Fixes
  - Fixed issue where SCT application level Interrupt handler function is occupied by SCT driver.
  - Fixed issue where wrong value for INSYNC field inside SCTIMER\_Init function.
  - Fixed issue to change Default value for INSYNC field inside SCTIMER\_GetDefaultConfig.

### [2.0.1]

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

### [2.0.0]

- Initial version.
- 

## SDIF

### [2.1.0]

- Improvements
  - Removed redundant member endianMode in sdif\_config\_t.
  - Added error status check in function SDIF\_WaitCommandDone.
  - Fixed the read fifo data incomplete issue in interrupt non-dma mode.

### [2.0.15]

- Bug Fixes
  - Cleared the interrupt status before enable the interrupt to avoid interrupt generate unexpectedly.
  - Fixed the SDIF\_ReadDataPortBlocking blocking at wrong condition issue.
- Improvements
  - Enabled the functionality of timeout parameter in SDIF\_SendCommand.
  - Added the error recovery while sending sync clock command timeout.

### [2.0.14]

- Improvements
  - Used different status code for command and data interrupt callback.
- Bug Fixes
  - Fixed the DMA descriptor attribute field unreset when configuring the current transfer DMA descriptor issue which may cause the transfer terminate unexpected.

### [2.0.13]

- Improvements
  - Disabled redundant interrupt per different transfer request.
  - Disabled interrupt and reset command/data pointer in handle when transfer completes.
- Bug Fixes
  - Fixed the PA082 build warning.
  - Fixed violations of the MISRA C-2012 rules 14.4, 17.7, 10.4, 10.3, 10.8, 14.3, 10.1, 16.4, 15.7, 12.2, 11.3, 11.9.

**[2.0.12]**

- Bug Fixes
  - Fixed the issue that SDIF\_ConfigClockDelay didn't reset the delay field before write.
  - Removed useless fifo reset code in transfer function.
  - Fixed the divider overflow issue in function SDIF\_SetCardClock.

**[2.0.11]**

- Improvements
  - Added API SDIF\_GetEnabledInterruptStatus/SDIF\_GetEnabledDMAInterruptStatus and used in SDIF\_TransferHandleIRQ.
  - Removed useless members interruptFlags/dmaInterruptFlags in the sdif\_handle\_t.
  - Improved SDIF\_SendCommand with return success directly when timeout is 0.
  - Added timeout error check when sending update clock command in SDIF\_SetCardClock.
  - Removed START\_CMD status polling for normal command sending in SDIF\_TransferBlocking/SDIF\_TransferNonBlocking.
  - Disabled timeout parameter in function SDIF\_SendCommand.
- Bug Fixes
  - Added delay cycle for the default speed mode(400 K and 25 M) to fix the timing issue when different AHB clocks are configured.

**[2.0.10]**

- Bug Fixes
  - Fixed the issue that API SDIF\_EnableCardClock could not clear the clock enable bit.

**[2.0.9]**

- Bug Fixes
  - Fixed MDK 66-D warning.

**[2.0.8]**

- New Features
  - Added control macro to enable/disable the RESET and CLOCK code in current driver.
  - Disabled useless interrupt while DMA is used.
  - Updated SDIF driver for one instance support two cards.

**[2.0.7]**

- Bug Fixes
  - Enlarged the timeout value to avoid a command conflict issue.

#### [2.0.6]

- Bug Fixes
  - Removed `assert(srcClock_Hz <= FSL_FEATURE_SDIF_MAX_SOURCE_CLOCK)`.
  - Used hardware reset instead of software reset during initialization.

#### [2.0.5]

- New Features
  - Added non-word aligned data address and DMA descriptor address transfer support. Once one of the above addresses is not aligned, switch to host transfer mode.
- Bug Fixes
  - Fixed the issue that DMA suspended during initialization.
  - Removed useless `memset` function call.

#### [2.0.4]

- Improvements
  - Added `cardInserted/cardRemoved` callback function.
  - Added host base address/user data parameter for all call back functions.

#### [2.0.3]

- Improvements
  - Improved Clock Delay macro to allow the user to redefine and remove useless delay for clock below 25 MHz.

#### [2.0.2]

- Bug Fixes
  - Fixed the issue that the status flag could not be cleared entirely after transfer complete.

#### [2.0.1]

- New Features
  - Improved interrupt transfer callback.
- Bug Fixes
  - Added assert to limit the SDIF source clock below 52 MHz.

#### [2.0.0]

- Initial version.
-

## SHA

### [2.3.2]

- Add -O2 optimization for GCC to sha\_process\_message\_data\_master(), because without it the function hangs under some conditions.

### [2.3.1]

- Modified sha\_process\_message\_data\_master() to ensure that MEMCTRL will be written within 64 cycles of writing last word to INDATA as is mentioned in errata, even with different optimization levels.

### [2.3.0]

- Modified SHA\_Update to use blocking version of AHB Master mode when its available on chip. Added SHA\_UpdateNonBlocking() function which uses nonblocking AHB Master mode.
- Fixed incorrect calculation of SHA when calling SHA\_Update multiple times when is CPU used to load data.
- Added Reset into SHA\_ClkInit and SHA\_ClkDeinit function.

### [2.2.2]

- Modified SHA\_Finish function. While using pseudo DMA with maximum optimization, compiler optimize out condition. Which caused block in this function and did not check flag, which has been set in interrupt.

### [2.2.1]

- MISRA C-2012 issue fix.

### [2.2.0]

- Support MEMADDR pseudo DMA for loading input data in SHA\_Update function (LPCXpresso54018 and LPCXpresso54628).

### [2.1.1]

- MISRA C-2012 issue fixed: rule 10.3, 10.4, 11.9, 14.4, 16.4 and 17.7.

### [2.1.0]

- Updated “sha\_ldm\_stm\_16\_words” “sha\_one\_block” API to match QN9090. QN9090 has no ALIAS register.
- Added “SHA\_ClkInit” “SHA\_ClkInit”

### [2.0.0]

- Initial version.
-

## SPI

### [2.3.2]

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

### [2.3.1]

- Improvements
  - Changed SPI\_DUMMYDATA to 0x00.

### [2.3.0]

- Update version.

### [2.2.2]

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

### [2.2.1]

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

### [2.2.0]

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

### [2.1.1]

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

### [2.1.0]

- Bug Fixes
  - Fixed Coverity issue of incrementing null pointer in SPI\_TransferHandleIRQInternal.
  - Eliminated IAR Pa082 warnings.
  - Fixed MISRA issues.
    - \* Fixed rules 10.1, 10.3, 10.4, 10.7, 10.8, 11.3, 11.6, 11.8, 11.9, 13.5.
- New Features



- Modified the definition of SPI\_SSELPOL\_MASK to support the socs that have only 3 SSEL pins.

#### [2.0.4]

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

#### [2.0.3]

- Improvements
  - Added “SPI\_FIFO\_DEPTH(base)” with more definition.

#### [2.0.2]

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

#### [2.0.1]

- Changed the data buffer from uint32\_t to uint8\_t which matches the real applications for SPI DMA driver.
- Added dummy data setup API to allow users to configure the dummy data to be transferred.
- Added new APIs for half-duplex transfer function. Users can not only send and receive data by one API in polling/interrupt/DMA way, but choose either to transmit first or to receive first. Besides, the PCS pin can be configured as assert status in transmission (between transmit and receive) by setting the isPcsAssertInTransfer to true.

#### [2.0.0]

- Initial version.
- 

## SPI\_DMA

#### [2.2.2]

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

#### [2.2.1]

- Bug Fixes
  - Fixed MISRA 2012 11.6 issue..

#### [2.2.0]

- Improvements
    - Supported dataSize larger than 1024 data transmit.
- 

### SPI Flash Interface

#### [2.0.3]

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

#### [2.0.2]

- Bug Fixes
  - Fixed the command function set issue. After the command being set, there will be no wait for the CMD flag, as it may have been cleared by CS deassert.

#### [2.0.1]

- New Features
  - Added an API to read/write 1/2 Bytes data from/to SPIFI. This interface is useful for flash command, which only needs 1/2 Bytes data. The previous driver needed users to make sure of the minimum length being 4, which might cause issues in some flash commands.

#### [2.0.0]

- Initial version.
- 

### USART

#### [2.8.5]

- Bug Fixes
  - Fixed race condition during call of USART\_EnableTxDMA and USART\_EnableRxDMA.

#### [2.8.4]

- Bug Fixes
  - Fixed exclusive access in USART\_TransferReceiveNonBlocking and USART\_TransferSendNonBlocking.

#### [2.8.3]

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

**[2.8.2]**

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

**[2.8.1]**

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

**[2.8.0]**

- New Features
  - Added the rx timeout interrupts and status flags of bus status.
  - Added new rx timeout configuration item in usart\_config\_t.
  - Added API USART\_SetRxTimeoutConfig for rx timeout configuration.
- Improvements
  - When the calculated baudrate cannot meet user's configuration, lower OSR value is allowed to use.

**[2.7.0]**

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

**[2.6.0]**

- Improvements
  - Used separate data for TX and RX in usart\_transfer\_t.
- Bug Fixes
  - Fixed bug that when ring buffer is used, if some data is received in ring buffer first before calling USART\_TransferReceiveNonBlocking, the received data count returned by USART\_TransferGetReceiveCount is wrong.
- New Features
  - Added missing API USART\_TransferGetSendCountDMA get send count using DMA.

**[2.5.0]**

- New Features
  - Added APIs USART\_GetRxFifoCount/USART\_GetTxFifoCount to get rx/tx FIFO data count.
  - Added APIs USART\_SetRxFifoWatermark/USART\_SetTxFifoWatermark to set rx/tx FIFO water mark.
- Bug Fixes
  - Fixed DMA transfer blocking issue by enabling tx idle interrupt after DMA transmission finishes.

#### [2.4.0]

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

#### [2.3.1]

- Bug Fixes
  - Fixed bug that operation on `INTENSET`, `INTENCLR`, `FIFOINTENSET` and `FIFOINTENCLR` should use bitwise operation not ‘or’ operation.
  - Fixed bug that if rx interrupt occurs before TX interrupt is enabled and after `txDataSize` is configured, the data will be sent early by mistake, thus TX interrupt will be enabled after data is sent out.
- Improvements
  - Added check for baud rate’s accuracy that returns `kStatus_USART_BaudrateNotSupport` when the best achieved baud rate is not within 3% error of configured baud rate.

#### [2.3.0]

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

#### [2.2.0]

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

**[2.1.1]**

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

**[2.1.0]**

- New Features
  - Added features to allow users to configure the USART to synchronous transfer(master and slave) mode.
- Bug Fixes
  - Modified USART\_SetBaudRate to get more accurate configuration.

**[2.0.3]**

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

**[2.0.2]**

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

**[2.0.1]**

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

**[2.0.0]**

- Initial version.
- 

**USART\_DMA****[2.6.0]**

- Refer USART driver change log 2.0.1 to 2.6.0
-

## UTICK

### [2.0.5]

- Improvements
  - Improved for SOC RW610.

### [2.0.4]

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

### [2.0.3]

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

### [2.0.2]

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

### [2.0.1]

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

### [2.0.0]

- Initial version.
- 

## WWDT

### [2.1.10]

- Bug Fixes
  - Chek WWDT\_RSTS instead of FSL\_FEATURE\_WWDT\_HAS\_NO\_RESET to determine whether the peripheral can be reset.

### [2.1.9]

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

### [2.1.8]

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

**[2.1.7]**

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

**[2.1.5]**

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

**[2.1.4]**

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

**[2.1.3]**

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

**[2.1.2]**

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

**[2.1.1]**

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

**[2.1.0]**

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

## [2.0.0]

- Initial version.
- 

## 1.6 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.

[LPC54S018](#)

## 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 AWS IoT

[aws\\_iot](#)

### 1.7.3 FreeRTOS

[FreeRTOS](#)

### 1.7.4 lwIP

[lwip](#)

### 1.7.5 File systemFatfs

[fatfs](#)



# Chapter 2

## LPC54S018

### 2.1 AES: AES encryption decryption driver

*status\_t* AES\_SetKey(AES\_Type \*base, const uint8\_t \*key, size\_t keySize)

Sets AES key.

Sets AES key.

#### Parameters

- base – AES peripheral base address
- key – Input key to use for encryption or decryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.

#### Returns

Status from Set Key operation

*status\_t* AES\_EncryptEcb(AES\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, size\_t size)

Encrypts AES using the ECB block mode.

Encrypts AES using the ECB block mode.

#### Parameters

- base – AES peripheral base address
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.

#### Returns

Status from encrypt operation

*status\_t* AES\_DecryptEcb(AES\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, size\_t size)

Decrypts AES using the ECB block mode.

Decrypts AES using the ECB block mode.

#### Parameters

- base – AES peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plain text

- size – Size of input and output data in bytes. Must be multiple of 16 bytes.

**Returns**

Status from decrypt operation

```
status_t AES_EncryptCbc(AES_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, size_t  
size, const uint8_t iv[16])
```

Encrypts AES using CBC block mode.

**Parameters**

- base – AES peripheral base address
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.

**Returns**

Status from encrypt operation

```
status_t AES_DecryptCbc(AES_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, size_t  
size, const uint8_t iv[16])
```

Decrypts AES using CBC block mode.

**Parameters**

- base – AES peripheral base address
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.

**Returns**

Status from decrypt operation

```
status_t AES_EncryptCfb(AES_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, size_t  
size, const uint8_t iv[16])
```

Encrypts AES using CFB block mode.

**Parameters**

- base – AES peripheral base address
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input Initial vector to be used as the first input block.

**Returns**

Status from encrypt operation

```
status_t AES_DecryptCfb(AES_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, size_t  
size, const uint8_t iv[16])
```

Decrypts AES using CFB block mode.

**Parameters**

- base – AES peripheral base address
- ciphertext – Input cipher text to decrypt

- plaintext – **[out]** Output plain text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input Initial vector to be used as the first input block.

**Returns**

Status from decrypt operation

```
status_t AES_EncryptOfb(AES_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[16])
```

Encrypts AES using OFB block mode.

**Parameters**

- base – AES peripheral base address
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes.
- iv – Input Initial vector to be used as the first input block.

**Returns**

Status from encrypt operation

```
status_t AES_DecryptOfb(AES_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[16])
```

Decrypts AES using OFB block mode.

**Parameters**

- base – AES peripheral base address
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text
- size – Size of input and output data in bytes.
- iv – Input Initial vector to be used as the first input block.

**Returns**

Status from decrypt operation

```
status_t AES_CryptCtr(AES_Type *base, const uint8_t *input, uint8_t *output, size_t size, uint8_t counter[16], uint8_t counterlast[16], size_t *szLeft)
```

Encrypts or decrypts AES using CTR block mode.

Encrypts or decrypts AES using CTR block mode. AES CTR mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

**Parameters**

- base – AES peripheral base address
- input – Input data for CTR block mode
- output – **[out]** Output data for CTR block mode
- size – Size of input and output data in bytes
- counter – **[inout]** Input counter (updates on return)
- counterlast – **[out]** Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.

- `szLeft` – **[out]** Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used.

### Returns

Status from crypt operation

```
status_t AES_EncryptTagGcm(AES_Type *base, const uint8_t *plaintext, uint8_t *ciphertext,  
                           size_t size, const uint8_t *iv, size_t ivSize, const uint8_t *aad,  
                           size_t aadSize, uint8_t *tag, size_t tagSize)
```

Encrypts AES and tags using GCM block mode.

Encrypts AES and optionally tags using GCM block mode. If plaintext is NULL, only the GHASH is calculated and output in the 'tag' field.

### Parameters

- `base` – AES peripheral base address
- `plaintext` – Input plain text to encrypt
- `ciphertext` – **[out]** Output cipher text.
- `size` – Size of input and output data in bytes
- `iv` – Input initial vector
- `ivSize` – Size of the IV
- `aad` – Input additional authentication data
- `aadSize` – Input size in bytes of AAD
- `tag` – **[out]** Output hash tag. Set to NULL to skip tag processing.
- `tagSize` – Input size of the tag to generate, in bytes. Must be 4,8,12,13,14,15 or 16.

### Returns

Status from encrypt operation

```
status_t AES_DecryptTagGcm(AES_Type *base, const uint8_t *ciphertext, uint8_t *plaintext,  
                           size_t size, const uint8_t *iv, size_t ivSize, const uint8_t *aad,  
                           size_t aadSize, const uint8_t *tag, size_t tagSize)
```

Decrypts AES and authenticates using GCM block mode.

Decrypts AES and optionally authenticates using GCM block mode. If ciphertext is NULL, only the GHASH is calculated and compared with the received GHASH in 'tag' field.

### Parameters

- `base` – AES peripheral base address
- `ciphertext` – Input cipher text to decrypt
- `plaintext` – **[out]** Output plain text.
- `size` – Size of input and output data in bytes
- `iv` – Input initial vector
- `ivSize` – Size of the IV
- `aad` – Input additional authentication data
- `aadSize` – Input size in bytes of AAD
- `tag` – Input hash tag to compare. Set to NULL to skip tag processing.
- `tagSize` – Input size of the tag, in bytes. Must be 4, 8, 12, 13, 14, 15, or 16.

### Returns

Status from decrypt operation

AES\_BLOCK\_SIZE

AES block size in bytes

AES\_IV\_SIZE

AES Input Vector size in bytes

FSL\_AES\_DRIVER\_VERSION

Defines LPC AES driver version 2.0.3.

Change log:

- Version 2.0.3
  - Edit aes\_one\_block() function to be interrupt safe.
- Version 2.0.2
  - Fix MISRA-2012 issues
- Version 2.0.1
  - GCM constant time tag comparison
- Version 2.0.0
  - initial version

## 2.2 Clock Driver

enum \_clock\_ip\_name

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

*Values:*

enumerator kCLOCK\_IpInvalid

Invalid Ip Name.

enumerator kCLOCK\_Rom

Clock gate name: Rom.

enumerator kCLOCK\_Sram1

Clock gate name: Sram1.

enumerator kCLOCK\_Sram2

Clock gate name: Sram2.

enumerator kCLOCK\_Sram3

Clock gate name: Sram3.

enumerator kCLOCK\_Spifi

Clock gate name: Spifi.

enumerator kCLOCK\_InputMux

Clock gate name: InputMux.

enumerator kCLOCK\_Iocon

Clock gate name: Iocon.

enumerator kCLOCK\_Gpio0

Clock gate name: Gpio0.

enumerator kCLOCK\_Gpio1

Clock gate name: Gpio1.

enumerator kCLOCK\_Gpio2  
Clock gate name: Gpio2.

enumerator kCLOCK\_Gpio3  
Clock gate name: Gpio3.

enumerator kCLOCK\_Pint  
Clock gate name: Pint.

enumerator kCLOCK\_Gint  
Clock gate name: Gint.

enumerator kCLOCK\_Dma  
Clock gate name: Dma.

enumerator kCLOCK\_Crc  
Clock gate name: Crc.

enumerator kCLOCK\_Wwdt  
Clock gate name: Wwdt.

enumerator kCLOCK\_Rtc  
Clock gate name: Rtc.

enumerator kCLOCK\_Adc0  
Clock gate name: Adc0.

enumerator kCLOCK\_Mrt  
Clock gate name: Mrt.

enumerator kCLOCK\_Rit  
Clock gate name: Rit.

enumerator kCLOCK\_Sct0  
Clock gate name: Sct0.

enumerator kCLOCK\_Mcan0  
Clock gate name: Mcan0.

enumerator kCLOCK\_Mcan1  
Clock gate name: Mcan1.

enumerator kCLOCK\_Utick  
Clock gate name: Utick.

enumerator kCLOCK\_FlexComm0  
Clock gate name: FlexComm0.

enumerator kCLOCK\_FlexComm1  
Clock gate name: FlexComm1.

enumerator kCLOCK\_FlexComm2  
Clock gate name: FlexComm2.

enumerator kCLOCK\_FlexComm3  
Clock gate name: FlexComm3.

enumerator kCLOCK\_FlexComm4  
Clock gate name: FlexComm4.

enumerator kCLOCK\_FlexComm5  
Clock gate name: FlexComm5.

enumerator kCLOCK\_FlexComm6  
Clock gate name: FlexComm6.

enumerator kCLOCK\_FlexComm7  
Clock gate name: FlexComm7.

enumerator kCLOCK\_MinUart0  
Clock gate name: MinUart0.

enumerator kCLOCK\_MinUart1  
Clock gate name: MinUart1.

enumerator kCLOCK\_MinUart2  
Clock gate name: MinUart2.

enumerator kCLOCK\_MinUart3  
Clock gate name: MinUart3.

enumerator kCLOCK\_MinUart4  
Clock gate name: MinUart4.

enumerator kCLOCK\_MinUart5  
Clock gate name: MinUart5.

enumerator kCLOCK\_MinUart6  
Clock gate name: MinUart6.

enumerator kCLOCK\_MinUart7  
Clock gate name: MinUart7.

enumerator kCLOCK\_LSpi0  
Clock gate name: LSpi0.

enumerator kCLOCK\_LSpi1  
Clock gate name: LSpi1.

enumerator kCLOCK\_LSpi2  
Clock gate name: LSpi2.

enumerator kCLOCK\_LSpi3  
Clock gate name: LSpi3.

enumerator kCLOCK\_LSpi4  
Clock gate name: LSpi4.

enumerator kCLOCK\_LSpi5  
Clock gate name: LSpi5.

enumerator kCLOCK\_LSpi6  
Clock gate name: LSpi6.

enumerator kCLOCK\_LSpi7  
Clock gate name: LSpi7.

enumerator kCLOCK\_BI2c0  
Clock gate name: BI2c0.

enumerator kCLOCK\_BI2c1  
Clock gate name: BI2c1.

enumerator kCLOCK\_BI2c2  
Clock gate name: BI2c2.

enumerator kCLOCK\_BI2c3  
Clock gate name: BI2c3.

enumerator kCLOCK\_BI2c4  
Clock gate name: BI2c4.

enumerator kCLOCK\_BI2c5  
Clock gate name: BI2c5.

enumerator kCLOCK\_BI2c6  
Clock gate name: BI2c6.

enumerator kCLOCK\_BI2c7  
Clock gate name: BI2c7.

enumerator kCLOCK\_FlexI2s0  
Clock gate name: FlexI2s0.

enumerator kCLOCK\_FlexI2s1  
Clock gate name: FlexI2s1.

enumerator kCLOCK\_FlexI2s2  
Clock gate name: FlexI2s2.

enumerator kCLOCK\_FlexI2s3  
Clock gate name: FlexI2s3.

enumerator kCLOCK\_FlexI2s4  
Clock gate name: FlexI2s4.

enumerator kCLOCK\_FlexI2s5  
Clock gate name: FlexI2s5.

enumerator kCLOCK\_FlexI2s6  
Clock gate name: FlexI2s6.

enumerator kCLOCK\_FlexI2s7  
Clock gate name: FlexI2s7.

enumerator kCLOCK\_DMic  
Clock gate name: DMic.

enumerator kCLOCK\_Ct32b2  
Clock gate name: Ct32b2.

enumerator kCLOCK\_Usbd0  
Clock gate name: Usbd0.

enumerator kCLOCK\_Ct32b0  
Clock gate name: Ct32b0.

enumerator kCLOCK\_Ct32b1  
Clock gate name: Ct32b1.

enumerator kCLOCK\_BodyBias0  
Clock gate name: BodyBias0.

enumerator kCLOCK\_EzhArchB0  
Clock gate name: EzhArchB0.

enumerator kCLOCK\_Lcd  
Clock gate name: Lcd.



enumerator kCLOCK\_Sdio  
Clock gate name: Sdio.

enumerator kCLOCK\_Usbh1  
Clock gate name: Usbh1.

enumerator kCLOCK\_Usbd1  
Clock gate name: Usbd1.

enumerator kCLOCK\_UsbRam1  
Clock gate name: UsbRam1.

enumerator kCLOCK\_Emc  
Clock gate name: Emc.

enumerator kCLOCK\_Eth  
Clock gate name: Eth.

enumerator kCLOCK\_Gpio4  
Clock gate name: Gpio4.

enumerator kCLOCK\_Gpio5  
Clock gate name: Gpio5.

enumerator kCLOCK\_Aes  
Clock gate name: Aes.

enumerator kCLOCK\_Otp  
Clock gate name: Otp.

enumerator kCLOCK\_Rng  
Clock gate name: Rng.

enumerator kCLOCK\_FlexComm8  
Clock gate name: FlexComm8.

enumerator kCLOCK\_FlexComm9  
Clock gate name: FlexComm9.

enumerator kCLOCK\_MinUart8  
Clock gate name: MinUart8.

enumerator kCLOCK\_MinUart9  
Clock gate name: MinUart9.

enumerator kCLOCK\_LSpi8  
Clock gate name: LSpi8.

enumerator kCLOCK\_LSpi9  
Clock gate name: LSpi9.

enumerator kCLOCK\_BI2c8  
Clock gate name: BI2c8.

enumerator kCLOCK\_BI2c9  
Clock gate name: BI2c9.

enumerator kCLOCK\_FlexI2s8  
Clock gate name: FlexI2s8.

enumerator kCLOCK\_FlexI2s9  
Clock gate name: FlexI2s9.

enumerator kCLOCK\_UsbHmr0

Clock gate name: UsbHmr0.

enumerator kCLOCK\_UsbHsl0

Clock gate name: UsbHsl0.

enumerator kCLOCK\_Sha0

Clock gate name: Sha0.

enumerator kCLOCK\_SmartCard0

Clock gate name: SmartCard0.

enumerator kCLOCK\_SmartCard1

Clock gate name: SmartCard1.

enumerator kCLOCK\_FlexComm10

Clock gate name: FlexComm10.

enumerator kCLOCK\_Puf

Clock gate name: Puf.

enumerator kCLOCK\_Ct32b3

Clock gate name: Ct32b3.

enumerator kCLOCK\_Ct32b4

Clock gate name: Ct32b4.

enum \_clock\_name

Clock name used to get clock frequency.

*Values:*

enumerator kCLOCK\_CoreSysClk

Core/system clock (aka MAIN\_CLK)

enumerator kCLOCK\_BusClk

Bus clock (AHB clock)

enumerator kCLOCK\_ClockOut

CLOCKOUT

enumerator kCLOCK\_FroHf

FRO48/96

enumerator kCLOCK\_UsbPll

USB1 PLL

enumerator kCLOCK\_Mclk

MCLK

enumerator kCLOCK\_Fro12M

FRO12M

enumerator kCLOCK\_ExtClk

External Clock

enumerator kCLOCK\_PllOut

PLL Output

enumerator kCLOCK\_UsbClk

USB input

enumerator kCLOCK\_WdtOsc

Watchdog Oscillator

enumerator kCLOCK\_Frg

Frg Clock

enumerator kCLOCK\_AsyncApbClk

Async APB clock

enum \_async\_clock\_src

Clock source selections for the asynchronous APB clock.

*Values:*

enumerator kCLOCK\_AsyncMainClk

Main System clock

enumerator kCLOCK\_AsyncFro12Mhz

12MHz FRO

enumerator kCLOCK\_AsyncAudioPllClk

Async Audio PLL clock.

enumerator kCLOCK\_AsyncI2cClkFe6

Async I2C clock.

enum \_clock\_attach\_id

The enumerator of clock attach Id.

*Values:*

enumerator kSYSTICK\_DIV\_CLK\_to\_SYSTICKCLK

Attach SYSTICK\_DIV\_CLK to SYSTICKCLK.

enumerator kWDT\_OSC\_to\_SYSTICKCLK

Attach WDT\_OSC to SYSTICKCLK.

enumerator kOSC32K\_to\_SYSTICKCLK

Attach OSC32K to SYSTICKCLK.

enumerator kFRO12M\_to\_SYSTICKCLK

Attach FRO12M to SYSTICKCLK.

enumerator kNONE\_to\_SYSTICKCLK

Attach NONE to SYSTICKCLK.

enumerator kFRO12M\_to\_MAIN\_CLK

Attach FRO12M to MAIN\_CLK.

enumerator kEXT\_CLK\_to\_MAIN\_CLK

Attach EXT\_CLK to MAIN\_CLK.

enumerator kWDT\_OSC\_to\_MAIN\_CLK

Attach WDT\_OSC to MAIN\_CLK.

enumerator kFRO\_HF\_to\_MAIN\_CLK

Attach FRO\_HF to MAIN\_CLK.

enumerator kSYS\_PLL\_to\_MAIN\_CLK

Attach SYS\_PLL to MAIN\_CLK.

enumerator kOSC32K\_to\_MAIN\_CLK

Attach OSC32K to MAIN\_CLK.

enumerator kMAIN\_CLK\_to\_CLKOUT  
Attach MAIN\_CLK to CLKOUT.

enumerator kEXT\_CLK\_to\_CLKOUT  
Attach EXT\_CLK to CLKOUT.

enumerator kWDT\_OSC\_to\_CLKOUT  
Attach WDT\_OSC to CLKOUT.

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

enumerator kSYS\_PLL\_to\_CLKOUT  
Attach SYS\_PLL to CLKOUT.

enumerator kUSB\_PLL\_to\_CLKOUT  
Attach USB\_PLL to CLKOUT.

enumerator kAUDIO\_PLL\_to\_CLKOUT  
Attach AUDIO\_PLL to CLKOUT.

enumerator kOSC32K\_OSC\_to\_CLKOUT  
Attach OSC32K\_OSC to CLKOUT.

enumerator kFRO12M\_to\_SYS\_PLL  
Attach FRO12M to SYS\_PLL.

enumerator kEXT\_CLK\_to\_SYS\_PLL  
Attach EXT\_CLK to SYS\_PLL.

enumerator kWDT\_OSC\_to\_SYS\_PLL  
Attach WDT\_OSC to SYS\_PLL.

enumerator kOSC32K\_to\_SYS\_PLL  
Attach OSC32K to SYS\_PLL.

enumerator kNONE\_to\_SYS\_PLL  
Attach NONE to SYS\_PLL.

enumerator kFRO12M\_to\_AUDIO\_PLL  
Attach FRO12M to AUDIO\_PLL.

enumerator kEXT\_CLK\_to\_AUDIO\_PLL  
Attach EXT\_CLK to AUDIO\_PLL.

enumerator kNONE\_to\_AUDIO\_PLL  
Attach NONE to AUDIO\_PLL.

enumerator kMAIN\_CLK\_to\_SPIFI\_CLK  
Attach MAIN\_CLK to SPIFI\_CLK.

enumerator kSYS\_PLL\_to\_SPIFI\_CLK  
Attach SYS\_PLL to SPIFI\_CLK.

enumerator kUSB\_PLL\_to\_SPIFI\_CLK  
Attach USB\_PLL to SPIFI\_CLK.

enumerator kFRO\_HF\_to\_SPIFI\_CLK  
Attach FRO\_HF to SPIFI\_CLK.

enumerator kAUDIO\_PLL\_to\_SPIFI\_CLK  
Attach AUDIO\_PLL to SPIFI\_CLK.

enumerator kNONE\_to\_SPIFI\_CLK  
Attach NONE to SPIFI\_CLK.

enumerator kFRO\_HF\_to\_ADC\_CLK  
Attach FRO\_HF to ADC\_CLK.

enumerator kSYS\_PLL\_to\_ADC\_CLK  
Attach SYS\_PLL to ADC\_CLK.

enumerator kUSB\_PLL\_to\_ADC\_CLK  
Attach USB\_PLL to ADC\_CLK.

enumerator kAUDIO\_PLL\_to\_ADC\_CLK  
Attach AUDIO\_PLL to ADC\_CLK.

enumerator kNONE\_to\_ADC\_CLK  
Attach NONE to ADC\_CLK.

enumerator kFRO\_HF\_to\_USB0\_CLK  
Attach FRO\_HF to USB0\_CLK.

enumerator kSYS\_PLL\_to\_USB0\_CLK  
Attach SYS\_PLL to USB0\_CLK.

enumerator kUSB\_PLL\_to\_USB0\_CLK  
Attach USB\_PLL to USB0\_CLK.

enumerator kNONE\_to\_USB0\_CLK  
Attach NONE to USB0\_CLK.

enumerator kFRO\_HF\_to\_USB1\_CLK  
Attach FRO\_HF to USB1\_CLK.

enumerator kSYS\_PLL\_to\_USB1\_CLK  
Attach SYS\_PLL to USB1\_CLK.

enumerator kUSB\_PLL\_to\_USB1\_CLK  
Attach USB\_PLL to USB1\_CLK.

enumerator kNONE\_to\_USB1\_CLK  
Attach NONE to USB1\_CLK.

enumerator kFRO12M\_to\_FLEXCOMM0  
Attach FRO12M to FLEXCOMM0.

enumerator kFRO\_HF\_to\_FLEXCOMM0  
Attach FRO\_HF to FLEXCOMM0.

enumerator kAUDIO\_PLL\_to\_FLEXCOMM0  
Attach AUDIO\_PLL to FLEXCOMM0.

enumerator kMCLK\_to\_FLEXCOMM0  
Attach MCLK to FLEXCOMM0.

enumerator kFRG\_to\_FLEXCOMM0  
Attach FRG to FLEXCOMM0.

enumerator kNONE\_to\_FLEXCOMM0  
Attach NONE to FLEXCOMM0.

enumerator kFRO12M\_to\_FLEXCOMM1  
Attach FRO12M to FLEXCOMM1.

enumerator kFRO\_HF\_to\_FLEXCOMM1  
Attach FRO\_HF to FLEXCOMM1.

enumerator kAUDIO\_PLL\_to\_FLEXCOMM1  
Attach AUDIO\_PLL to FLEXCOMM1.

enumerator kMCLK\_to\_FLEXCOMM1  
Attach MCLK to FLEXCOMM1.

enumerator kFRG\_to\_FLEXCOMM1  
Attach FRG to FLEXCOMM1.

enumerator kNONE\_to\_FLEXCOMM1  
Attach NONE to FLEXCOMM1.

enumerator kFRO12M\_to\_FLEXCOMM2  
Attach FRO12M to FLEXCOMM2.

enumerator kFRO\_HF\_to\_FLEXCOMM2  
Attach FRO\_HF to FLEXCOMM2.

enumerator kAUDIO\_PLL\_to\_FLEXCOMM2  
Attach AUDIO\_PLL to FLEXCOMM2.

enumerator kMCLK\_to\_FLEXCOMM2  
Attach MCLK to FLEXCOMM2.

enumerator kFRG\_to\_FLEXCOMM2  
Attach FRG to FLEXCOMM2.

enumerator kNONE\_to\_FLEXCOMM2  
Attach NONE to FLEXCOMM2.

enumerator kFRO12M\_to\_FLEXCOMM3  
Attach FRO12M to FLEXCOMM3.

enumerator kFRO\_HF\_to\_FLEXCOMM3  
Attach FRO\_HF to FLEXCOMM3.

enumerator kAUDIO\_PLL\_to\_FLEXCOMM3  
Attach AUDIO\_PLL to FLEXCOMM3.

enumerator kMCLK\_to\_FLEXCOMM3  
Attach MCLK to FLEXCOMM3.

enumerator kFRG\_to\_FLEXCOMM3  
Attach FRG to FLEXCOMM3.

enumerator kNONE\_to\_FLEXCOMM3  
Attach NONE to FLEXCOMM3.

enumerator kFRO12M\_to\_FLEXCOMM4  
Attach FRO12M to FLEXCOMM4.

enumerator kFRO\_HF\_to\_FLEXCOMM4  
Attach FRO\_HF to FLEXCOMM4.

enumerator kAUDIO\_PLL\_to\_FLEXCOMM4  
Attach AUDIO\_PLL to FLEXCOMM4.

enumerator kMCLK\_to\_FLEXCOMM4  
Attach MCLK to FLEXCOMM4.

enumerator kFRG\_to\_FLEXCOMM4  
Attach FRG to FLEXCOMM4.

enumerator kNONE\_to\_FLEXCOMM4  
Attach NONE to FLEXCOMM4.

enumerator kFRO12M\_to\_FLEXCOMM5  
Attach FRO12M to FLEXCOMM5.

enumerator kFRO\_HF\_to\_FLEXCOMM5  
Attach FRO\_HF to FLEXCOMM5.

enumerator kAUDIO\_PLL\_to\_FLEXCOMM5  
Attach AUDIO\_PLL to FLEXCOMM5.

enumerator kMCLK\_to\_FLEXCOMM5  
Attach MCLK to FLEXCOMM5.

enumerator kFRG\_to\_FLEXCOMM5  
Attach FRG to FLEXCOMM5.

enumerator kNONE\_to\_FLEXCOMM5  
Attach NONE to FLEXCOMM5.

enumerator kFRO12M\_to\_FLEXCOMM6  
Attach FRO12M to FLEXCOMM6.

enumerator kFRO\_HF\_to\_FLEXCOMM6  
Attach FRO\_HF to FLEXCOMM6.

enumerator kAUDIO\_PLL\_to\_FLEXCOMM6  
Attach AUDIO\_PLL to FLEXCOMM6.

enumerator kMCLK\_to\_FLEXCOMM6  
Attach MCLK to FLEXCOMM6.

enumerator kFRG\_to\_FLEXCOMM6  
Attach FRG to FLEXCOMM6.

enumerator kNONE\_to\_FLEXCOMM6  
Attach NONE to FLEXCOMM6.

enumerator kFRO12M\_to\_FLEXCOMM7  
Attach FRO12M to FLEXCOMM7.

enumerator kFRO\_HF\_to\_FLEXCOMM7  
Attach FRO\_HF to FLEXCOMM7.

enumerator kAUDIO\_PLL\_to\_FLEXCOMM7  
Attach AUDIO\_PLL to FLEXCOMM7.

enumerator kMCLK\_to\_FLEXCOMM7  
Attach MCLK to FLEXCOMM7.

enumerator kFRG\_to\_FLEXCOMM7  
Attach FRG to FLEXCOMM7.

enumerator kNONE\_to\_FLEXCOMM7  
Attach NONE to FLEXCOMM7.

enumerator kFRO12M\_to\_FLEXCOMM8  
Attach FRO12M to FLEXCOMM8.

enumerator kFRO\_HF\_to\_FLEXCOMM8  
Attach FRO\_HF to FLEXCOMM8.

enumerator kAUDIO\_PLL\_to\_FLEXCOMM8  
Attach AUDIO\_PLL to FLEXCOMM8.

enumerator kMCLK\_to\_FLEXCOMM8  
Attach MCLK to FLEXCOMM8.

enumerator kFRG\_to\_FLEXCOMM8  
Attach FRG to FLEXCOMM8.

enumerator kNONE\_to\_FLEXCOMM8  
Attach NONE to FLEXCOMM8.

enumerator kFRO12M\_to\_FLEXCOMM9  
Attach FRO12M to FLEXCOMM9.

enumerator kFRO\_HF\_to\_FLEXCOMM9  
Attach FRO\_HF to FLEXCOMM9.

enumerator kAUDIO\_PLL\_to\_FLEXCOMM9  
Attach AUDIO\_PLL to FLEXCOMM9.

enumerator kMCLK\_to\_FLEXCOMM9  
Attach MCLK to FLEXCOMM9.

enumerator kFRG\_to\_FLEXCOMM9  
Attach FRG to FLEXCOMM9.

enumerator kNONE\_to\_FLEXCOMM9  
Attach NONE to FLEXCOMM9.

enumerator kMAIN\_CLK\_to\_FLEXCOMM10  
Attach MAIN\_CLK to FLEXCOMM10.

enumerator kSYS\_PLL\_to\_FLEXCOMM10  
Attach SYS\_PLL to FLEXCOMM10.

enumerator kUSB\_PLL\_to\_FLEXCOMM10  
Attach USB\_PLL to FLEXCOMM10.

enumerator kFRO\_HF\_to\_FLEXCOMM10  
Attach FRO\_HF to FLEXCOMM10.

enumerator kAUDIO\_PLL\_to\_FLEXCOMM10  
Attach AUDIO\_PLL to FLEXCOMM10.

enumerator kNONE\_to\_FLEXCOMM10  
Attach NONE to FLEXCOMM10.

enumerator kFRO\_HF\_to\_MCLK  
Attach FRO\_HF to MCLK.

enumerator kAUDIO\_PLL\_to\_MCLK  
Attach AUDIO\_PLL to MCLK.

enumerator kNONE\_to\_MCLK  
Attach NONE to MCLK.

enumerator kMAIN\_CLK\_to\_FRG  
Attach MAIN\_CLK to FRG.



enumerator kSYS\_PLL\_to\_FRG

Attach SYS\_PLL to FRG.

enumerator kFRO12M\_to\_FRG

Attach FRO12M to FRG.

enumerator kFRO\_HF\_to\_FRG

Attach FRO\_HF to FRG.

enumerator kNONE\_to\_FRG

Attach NONE to FRG.

enumerator kFRO12M\_to\_DMIC

Attach FRO12M to DMIC.

enumerator kFRO\_HF\_DIV\_to\_DMIC

Attach FRO\_HF\_DIV to DMIC.

enumerator kAUDIO\_PLL\_to\_DMIC

Attach AUDIO\_PLL to DMIC.

enumerator kMCLK\_to\_DMIC

Attach MCLK to DMIC.

enumerator kMAIN\_CLK\_to\_DMIC

Attach MAIN\_CLK to DMIC.

enumerator kWDT\_OSC\_to\_DMIC

Attach WDT\_OSC to DMIC.

enumerator kNONE\_to\_DMIC

Attach NONE to DMIC.

enumerator kMAIN\_CLK\_to\_SCT\_CLK

Attach MAIN\_CLK to SCT\_CLK.

enumerator kSYS\_PLL\_to\_SCT\_CLK

Attach SYS\_PLL to SCT\_CLK.

enumerator kFRO\_HF\_to\_SCT\_CLK

Attach FRO\_HF to SCT\_CLK.

enumerator kAUDIO\_PLL\_to\_SCT\_CLK

Attach AUDIO\_PLL to SCT\_CLK.

enumerator kNONE\_to\_SCT\_CLK

Attach NONE to SCT\_CLK.

enumerator kMAIN\_CLK\_to\_LCD\_CLK

Attach MAIN\_CLK to LCD\_CLK.

enumerator kLCDCLKIN\_to\_LCD\_CLK

Attach LCDCLKIN to LCD\_CLK.

enumerator kFRO\_HF\_to\_LCD\_CLK

Attach FRO\_HF to LCD\_CLK.

enumerator kNONE\_to\_LCD\_CLK

Attach NONE to LCD\_CLK.

enumerator kMAIN\_CLK\_to\_SDIO\_CLK

Attach MAIN\_CLK to SDIO\_CLK.

enumerator kSYS\_PLL\_to\_SDIO\_CLK  
Attach SYS\_PLL to SDIO\_CLK.

enumerator kUSB\_PLL\_to\_SDIO\_CLK  
Attach USB\_PLL to SDIO\_CLK.

enumerator kFRO\_HF\_to\_SDIO\_CLK  
Attach FRO\_HF to SDIO\_CLK.

enumerator kAUDIO\_PLL\_to\_SDIO\_CLK  
Attach AUDIO\_PLL to SDIO\_CLK.

enumerator kNONE\_to\_SDIO\_CLK  
Attach NONE to SDIO\_CLK.

enumerator kMAIN\_CLK\_to\_ASYNC\_APB  
Attach MAIN\_CLK to ASYNC\_APB.

enumerator kFRO12M\_to\_ASYNC\_APB  
Attach FRO12M to ASYNC\_APB.

enumerator kAUDIO\_PLL\_to\_ASYNC\_APB  
Attach AUDIO\_PLL to ASYNC\_APB.

enumerator kI2C\_CLK\_FC6\_to\_ASYNC\_APB  
Attach I2C\_CLK\_FC6 to ASYNC\_APB.

enumerator kNONE\_to\_NONE  
Attach NONE to NONE.

enum \_clock\_div\_name

Clock dividers.

*Values:*

enumerator kCLOCK\_DivSystickClk  
Systick Clock Divider.

enumerator kCLOCK\_DivArmTrClkDiv  
Arm Tr Clk Div Divider.

enumerator kCLOCK\_DivCan0Clk  
Can0 Clock Divider.

enumerator kCLOCK\_DivCan1Clk  
Can1 Clock Divider.

enumerator kCLOCK\_DivSmartCard0Clk  
Smart Card0 Clock Divider.

enumerator kCLOCK\_DivSmartCard1Clk  
Smart Card1 Clock Divider.

enumerator kCLOCK\_DivAhbClk  
Ahb Clock Divider.

enumerator kCLOCK\_DivClkOut  
Clk Out Divider.

enumerator kCLOCK\_DivFrohfClk  
Frohf Clock Divider.

enumerator kCLOCK\_DivSpifiClk

Spifi Clock Divider.

enumerator kCLOCK\_DivAdcAsyncClk

Adc Async Clock Divider.

enumerator kCLOCK\_DivUsb0Clk

Usb0 Clock Divider.

enumerator kCLOCK\_DivUsb1Clk

Usb1 Clock Divider.

enumerator kCLOCK\_DivFrg

Frg Divider.

enumerator kCLOCK\_DivDmicClk

Dmic Clock Divider.

enumerator kCLOCK\_DivMClk

I2S MCLK Clock Divider.

enumerator kCLOCK\_DivLcdClk

Lcd Clock Divider.

enumerator kCLOCK\_DivSctClk

Sct Clock Divider.

enumerator kCLOCK\_DivEmcClk

Emc Clock Divider.

enumerator kCLOCK\_DivSdioClk

Sdio clock divider.

enum \_pll\_error

PLL status definitions.

*Values:*

enumerator kStatus\_PLL\_Success

PLL operation was successful

enumerator kStatus\_PLL\_OutputTooLow

PLL output rate request was too low

enumerator kStatus\_PLL\_OutputTooHigh

PLL output rate request was too high

enumerator kStatus\_PLL\_InputTooLow

PLL input rate is too low

enumerator kStatus\_PLL\_InputTooHigh

PLL input rate is too high

enumerator kStatus\_PLL\_OutsideIntLimit

Requested output rate isn't possible

enumerator kStatus\_PLL\_CCOTooLow

Requested CCO rate isn't possible

enumerator kStatus\_PLL\_CCOTooHigh

Requested CCO rate isn't possible

enum `_clock_usb_src`

USB clock source definition.

*Values:*

enumerator `kCLOCK_UsbSrcFro`

Use FRO 96 or 48 MHz.

enumerator `kCLOCK_UsbSrcSystemPll`

Use System PLL output.

enumerator `kCLOCK_UsbSrcMainClock`

Use Main clock.

enumerator `kCLOCK_UsbSrcUsbPll`

Use USB PLL clock.

enumerator `kCLOCK_UsbSrcNone`

Use None, this may be selected in order to reduce power when no output is needed.

enum `_usb_pll_psel`

USB PDEL Divider.

*Values:*

enumerator `pSel_Divide_1`

enumerator `pSel_Divide_2`

enumerator `pSel_Divide_4`

enumerator `pSel_Divide_8`

typedef enum `_clock_ip_name` `clock_ip_name_t`

Clock gate name used for `CLOCK_EnableClock/CLOCK_DisableClock`.

typedef enum `_clock_name` `clock_name_t`

Clock name used to get clock frequency.

typedef enum `_async_clock_src` `async_clock_src_t`

Clock source selections for the asynchronous APB clock.

typedef enum `_clock_attach_id` `clock_attach_id_t`

The enumerator of clock attach Id.

typedef enum `_clock_div_name` `clock_div_name_t`

Clock dividers.

typedef struct `_pll_config` `pll_config_t`

PLL configuration structure.

This structure can be used to configure the settings for a PLL setup structure. Fill in the desired configuration for the PLL and call the PLL setup function to fill in a PLL setup structure.

typedef struct `_pll_setup` `pll_setup_t`

PLL setup structure This structure can be used to pre-build a PLL setup configuration at run-time and quickly set the PLL to the configuration. It can be populated with the PLL setup function. If powering up or waiting for PLL lock, the PLL input clock source should be configured prior to PLL setup.

typedef enum `_pll_error` `pll_error_t`

PLL status definitions.

```
typedef enum _clock_usb_src clock_usb_src_t
```

USB clock source definition.

```
typedef enum _usb_pll_psel usb_pll_psel
```

USB PDEL Divider.

```
typedef struct _usb_pll_setup usb_pll_setup_t
```

PLL setup structure This structure can be used to pre-build a USB PLL setup configuration at run-time and quickly set the usb PLL to the configuration. It can be populated with the USB PLL setup function. If powering up or waiting for USB PLL lock, the PLL input clock source should be configured prior to USB PLL setup.

```
static inline void CLOCK_EnableClock(clock_ip_name_t clk)
```

```
static inline void CLOCK_DisableClock(clock_ip_name_t clk)
```

```
void CLOCK_SetupFROClocking(uint32_t froFreq)
```

Initialize the Core clock to given frequency (12, 48 or 96 MHz), this API is implemented in ROM code. Turns on FRO and uses default CCO, if freq is 12000000, then high speed output is off, else high speed output is enabled. Usage: CLOCK\_SetupFROClocking(frequency), (frequency must be one of 12, 48 or 96 MHz) Note: Need to make sure ROM and OTP has power(PDRUNCFG0[17,29]= 0U) before calling this API since this API is implemented in ROM code and the FROHF TRIM value is stored in OTP.

#### Parameters

- froFreq – target fro frequency.

#### Returns

Nothing

```
void CLOCK_AttachClk(clock_attach_id_t connection)
```

Configure the clock selection muxes.

#### Parameters

- connection – : Clock to be configured.

#### Returns

Nothing

```
clock_attach_id_t CLOCK_GetClockAttachId(clock_attach_id_t attachId)
```

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

#### Parameters

- attachId – : Clock attach id to get.

#### Returns

Clock source value.

```
void CLOCK_SetClkDiv(clock_div_name_t div_name, uint32_t divided_by_value, bool reset)
```

Setup peripheral clock dividers.

#### Parameters

- div\_name – : Clock divider name
- divided\_by\_value – Value to be divided
- reset – : Whether to reset the divider counter.

#### Returns

Nothing

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

Return Frequency of selected clock.

**Returns**

Frequency of selected clock

uint32\_t CLOCK\_GetFro12MFreq(void)

Return Frequency of FRO 12MHz.

**Returns**

Frequency of FRO 12MHz

uint32\_t CLOCK\_GetClockOutClkFreq(void)

Return Frequency of ClockOut.

**Returns**

Frequency of ClockOut

uint32\_t CLOCK\_GetSpifiClkFreq(void)

Return Frequency of Spifi Clock.

**Returns**

Frequency of Spifi.

uint32\_t CLOCK\_GetAdcClkFreq(void)

Return Frequency of Adc Clock.

**Returns**

Frequency of Adc Clock.

uint32\_t CLOCK\_GetMCanClkFreq(uint32\_t MCanSel)

brief Return Frequency of MCAN Clock param MCanSel : 0U: MCAN0; 1U: MCAN1 return  
Frequency of MCAN Clock

uint32\_t CLOCK\_GetUsb0ClkFreq(void)

Return Frequency of Usb0 Clock.

**Returns**

Frequency of Usb0 Clock.

uint32\_t CLOCK\_GetUsb1ClkFreq(void)

Return Frequency of Usb1 Clock.

**Returns**

Frequency of Usb1 Clock.

uint32\_t CLOCK\_GetMclkClkFreq(void)

Return Frequency of MClk Clock.

**Returns**

Frequency of MClk Clock.

uint32\_t CLOCK\_GetSctClkFreq(void)

Return Frequency of SCTimer Clock.

**Returns**

Frequency of SCTimer Clock.

uint32\_t CLOCK\_GetSdioClkFreq(void)

Return Frequency of SDIO Clock.

**Returns**

Frequency of SDIO Clock.

uint32\_t CLOCK\_GetLcdClkFreq(void)

Return Frequency of LCD Clock.

**Returns**

Frequency of LCD Clock.

uint32\_t CLOCK\_GetLcdClkIn(void)

Return Frequency of LCD CLKIN Clock.

**Returns**

Frequency of LCD CLKIN Clock.

uint32\_t CLOCK\_GetExtClkFreq(void)

Return Frequency of External Clock.

**Returns**

Frequency of External Clock. If no external clock is used returns 0.

uint32\_t CLOCK\_GetWdtOscFreq(void)

Return Frequency of Watchdog Oscillator.

**Returns**

Frequency of Watchdog Oscillator

uint32\_t CLOCK\_GetFroHfFreq(void)

Return Frequency of High-Freq output of FRO.

**Returns**

Frequency of High-Freq output of FRO

uint32\_t CLOCK\_GetFrgClkFreq(void)

Return Frequency of frg.

**Returns**

Frequency of FRG

uint32\_t CLOCK\_GetDmicClkFreq(void)

Return Frequency of dmic.

**Returns**

Frequency of DMIC

uint32\_t CLOCK\_SetFRGClock(uint32\_t freq)

Set FRG Clk.

**Returns**

1: if set FRG CLK successfully. 0: if set FRG CLK fail.

uint32\_t CLOCK\_GetPllOutFreq(void)

Return Frequency of PLL.

**Returns**

Frequency of PLL

uint32\_t CLOCK\_GetUsbPllOutFreq(void)

Return Frequency of USB PLL.

**Returns**

Frequency of PLL

uint32\_t CLOCK\_GetAudioPllOutFreq(void)

Return Frequency of AUDIO PLL.

**Returns**

Frequency of PLL

uint32\_t CLOCK\_GetOsc32KFreq(void)

Return Frequency of 32kHz osc.

**Returns**

Frequency of 32kHz osc

uint32\_t CLOCK\_GetCoreSysClkFreq(void)

Return Frequency of Core System.

**Returns**

Frequency of Core System

uint32\_t CLOCK\_GetI2SMClkFreq(void)

Return Frequency of I2S MCLK Clock.

**Returns**

Frequency of I2S MCLK Clock

uint32\_t CLOCK\_GetFlexCommClkFreq(uint32\_t id)

Return Frequency of Flexcomm functional Clock.

**Returns**

Frequency of Flexcomm functional Clock

uint32\_t CLOCK\_GetFRGInputClock(void)

return FRG Clk

**Returns**

Frequency of FRG CLK.

\_\_STATIC\_INLINE async\_clock\_src\_t CLOCK\_GetAsyncApbClkSrc (void)

Return Asynchronous APB Clock source.

**Returns**

Asynchronous APB CLock source

uint32\_t CLOCK\_GetAsyncApbClkFreq(void)

Return Frequency of Asynchronous APB Clock.

**Returns**

Frequency of Asynchronous APB Clock Clock

\_\_STATIC\_INLINE uint32\_t CLOCK\_GetEmcClkFreq (void)

Return EMC source.

**Returns**

EMC source

uint32\_t CLOCK\_GetAudioPLLInClockRate(void)

Return Audio PLL input clock rate.

**Returns**

Audio PLL input clock rate

uint32\_t CLOCK\_GetSystemPLLInClockRate(void)

Return System PLL input clock rate.

**Returns**

System PLL input clock rate

uint32\_t CLOCK\_GetSystemPLLOutClockRate(bool recompute)

Return System PLL output clock rate.



---

**Note:** The PLL rate is cached in the driver in a variable as the rate computation function can take some time to perform. It is recommended to use 'false' with the 'recompute' parameter.

---

#### Parameters

- recompute – : Forces a PLL rate recomputation if true

#### Returns

System PLL output clock rate

uint32\_t CLOCK\_GetAudioPLLOutClockRate(bool recompute)

Return System AUDIO PLL output clock rate.

---

**Note:** The AUDIO PLL rate is cached in the driver in a variable as the rate computation function can take some time to perform. It is recommended to use 'false' with the 'recompute' parameter.

---

#### Parameters

- recompute – : Forces a AUDIO PLL rate recomputation if true

#### Returns

System AUDIO PLL output clock rate

uint32\_t CLOCK\_GetUsbPLLOutClockRate(bool recompute)

Return System USB PLL output clock rate.

---

**Note:** The USB PLL rate is cached in the driver in a variable as the rate computation function can take some time to perform. It is recommended to use 'false' with the 'recompute' parameter.

---

#### Parameters

- recompute – : Forces a USB PLL rate recomputation if true

#### Returns

System USB PLL output clock rate

\_\_STATIC\_INLINE void CLOCK\_SetBypassPLL (bool bypass)

Enables and disables PLL bypass mode.

bypass : true to bypass PLL (PLL output = PLL input, false to disable bypass)

#### Returns

System PLL output clock rate

\_\_STATIC\_INLINE bool CLOCK\_IsSystemPLLLocked (void)

Check if PLL is locked or not.

#### Returns

true if the PLL is locked, false if not locked

\_\_STATIC\_INLINE bool CLOCK\_IsUsbPLLLocked (void)

Check if USB PLL is locked or not.

#### Returns

true if the USB PLL is locked, false if not locked

\_\_STATIC\_INLINE bool CLOCK\_IsAudioPLLLocked (void)

Check if AUDIO PLL is locked or not.

**Returns**

true if the AUDIO PLL is locked, false if not locked

\_\_STATIC\_INLINE void CLOCK\_Enable\_SysOsc (bool enable)

Enables and disables SYS OSC.

enable : true to enable SYS OSC, false to disable SYS OSC

void CLOCK\_SetStoredPLLClockRate(uint32\_t rate)

Store the current PLL rate.

**Parameters**

- rate – Current rate of the PLL

**Returns**

Nothing

void CLOCK\_SetStoredAudioPLLClockRate(uint32\_t rate)

Store the current AUDIO PLL rate.

**Parameters**

- rate – Current rate of the PLL

**Returns**

Nothing

uint32\_t CLOCK\_GetSystemPLLOutFromSetup(*pll\_setup\_t* \*pSetup)

Return System PLL output clock rate from setup structure.

**Parameters**

- pSetup – : Pointer to a PLL setup structure

**Returns**

System PLL output clock rate the setup structure will generate

uint32\_t CLOCK\_GetAudioPLLOutFromSetup(*pll\_setup\_t* \*pSetup)

Return System AUDIO PLL output clock rate from setup structure.

**Parameters**

- pSetup – : Pointer to a PLL setup structure

**Returns**

System PLL output clock rate the setup structure will generate

uint32\_t CLOCK\_GetAudioPLLOutFromFractSetup(*pll\_setup\_t* \*pSetup)

Return System AUDIO PLL output clock rate from audio fractionl setup structure.

**Parameters**

- pSetup – : Pointer to a PLL setup structure

**Returns**

System PLL output clock rate the setup structure will generate

uint32\_t CLOCK\_GetUsbPLLOutFromSetup(const *usb\_pll\_setup\_t* \*pSetup)

Return System USB PLL output clock rate from setup structure.

**Parameters**

- pSetup – : Pointer to a PLL setup structure

**Returns**

System PLL output clock rate the setup structure will generate

---

*pll\_error\_t* CLOCK\_SetupPLLDData(*pll\_config\_t* \*pControl, *pll\_setup\_t* \*pSetup)

Set PLL output based on the passed PLL setup data.

---

**Note:** Actual frequency for setup may vary from the desired frequency based on the accuracy of input clocks, rounding, non-fractional PLL mode, etc.

---

#### Parameters

- pControl – : Pointer to populated PLL control structure to generate setup with
- pSetup – : Pointer to PLL setup structure to be filled

#### Returns

PLL\_ERROR\_SUCCESS on success, or PLL setup error code

*pll\_error\_t* CLOCK\_SetupAudioPLLDData(*pll\_config\_t* \*pControl, *pll\_setup\_t* \*pSetup)

Set AUDIO PLL output based on the passed AUDIO PLL setup data.

---

**Note:** Actual frequency for setup may vary from the desired frequency based on the accuracy of input clocks, rounding, non-fractional PLL mode, etc.

---

#### Parameters

- pControl – : Pointer to populated PLL control structure to generate setup with
- pSetup – : Pointer to PLL setup structure to be filled

#### Returns

PLL\_ERROR\_SUCCESS on success, or PLL setup error code

*pll\_error\_t* CLOCK\_SetupSystemPLLPrec(*pll\_setup\_t* \*pSetup, uint32\_t flagcfg)

Set PLL output from PLL setup structure (precise frequency)

---

**Note:** This function will power off the PLL, setup the PLL with the new setup data, and then optionally powerup the PLL, wait for PLL lock, and adjust system voltages to the new PLL rate. The function will not alter any source clocks (ie, main system clock) that may use the PLL, so these should be setup prior to and after exiting the function.

---

#### Parameters

- pSetup – : Pointer to populated PLL setup structure
- flagcfg – : Flag configuration for PLL config structure

#### Returns

PLL\_ERROR\_SUCCESS on success, or PLL setup error code

*pll\_error\_t* CLOCK\_SetupAudioPLLPrec(*pll\_setup\_t* \*pSetup, uint32\_t flagcfg)

Set AUDIO PLL output from AUDIOPLL setup structure (precise frequency)

---

**Note:** This function will power off the PLL, setup the PLL with the new setup data, and then optionally powerup the AUDIO PLL, wait for PLL lock, and adjust system voltages to the new AUDIOPLL rate. The function will not alter any source clocks (ie, main system clock) that may use the AUDIO PLL, so these should be setup prior to and after exiting the function.

---

**Parameters**

- pSetup – : Pointer to populated PLL setup structure
- flagcfg – : Flag configuration for PLL config structure

**Returns**

PLL\_ERROR\_SUCCESS on success, or PLL setup error code

*pll\_error\_t* CLOCK\_SetupAudioPLLPrecFract(*pll\_setup\_t* \*pSetup, uint32\_t flagcfg)

Set AUDIO PLL output from AUDIOPLL setup structure using the Audio Fractional divider register(precise frequency)

---

**Note:** This function will power off the PLL, setup the PLL with the new setup data, and then optionally powerup the AUDIO PLL, wait for PLL lock, and adjust system voltages to the new AUDIOPLL rate. The function will not alter any source clocks (ie, main system clock) that may use the AUDIO PLL, so these should be setup prior to and after exiting the function.

---

**Parameters**

- pSetup – : Pointer to populated PLL setup structure
- flagcfg – : Flag configuration for PLL config structure

**Returns**

PLL\_ERROR\_SUCCESS on success, or PLL setup error code

*pll\_error\_t* CLOCK\_SetPLLFreq(const *pll\_setup\_t* \*pSetup)

Set PLL output from PLL setup structure (precise frequency)

---

**Note:** This function will power off the PLL, setup the PLL with the new setup data, and then optionally powerup the PLL, wait for PLL lock, and adjust system voltages to the new PLL rate. The function will not alter any source clocks (ie, main system clock) that may use the PLL, so these should be setup prior to and after exiting the function.

---

**Parameters**

- pSetup – : Pointer to populated PLL setup structure

**Returns**

kStatus\_PLL\_Success on success, or PLL setup error code

*pll\_error\_t* CLOCK\_SetAudioPLLFreq(const *pll\_setup\_t* \*pSetup)

Set Audio PLL output from Audio PLL setup structure (precise frequency)

---

**Note:** This function will power off the PLL, setup the Audio PLL with the new setup data, and then optionally powerup the PLL, wait for Audio PLL lock, and adjust system voltages to the new PLL rate. The function will not alter any source clocks (ie, main system clock) that may use the Audio PLL, so these should be setup prior to and after exiting the function.

---

**Parameters**

- pSetup – : Pointer to populated PLL setup structure

**Returns**

kStatus\_PLL\_Success on success, or Audio PLL setup error code

*pll\_error\_t* CLOCK\_SetUsbPLLFreq(const *usb\_pll\_setup\_t* \*pSetup)

Set USB PLL output from USB PLL setup structure (precise frequency)

---

**Note:** This function will power off the USB PLL, setup the PLL with the new setup data, and then optionally powerup the USB PLL, wait for USB PLL lock, and adjust system voltages to the new USB PLL rate. The function will not alter any source clocks (ie, usb pll clock) that may use the USB PLL, so these should be setup prior to and after exiting the function.

---

#### Parameters

- pSetup – : Pointer to populated USB PLL setup structure

#### Returns

kStatus\_PLL\_Success on success, or USB PLL setup error code

void CLOCK\_SetupSystemPLLMult(uint32\_t multiply\_by, uint32\_t input\_freq)

Set PLL output based on the multiplier and input frequency.

---

**Note:** Unlike the Chip\_Clock\_SetupSystemPLLPrec() function, this function does not disable or enable PLL power, wait for PLL lock, or adjust system voltages. These must be done in the application. The function will not alter any source clocks (ie, main system clock) that may use the PLL, so these should be setup prior to and after exiting the function.

---

#### Parameters

- multiply\_by – : multiplier
- input\_freq – : Clock input frequency of the PLL

#### Returns

Nothing

static inline void CLOCK\_DisableUsbDevicefs0Clock(*clock\_ip\_name\_t* clk)

Disable USB clock.

Disable USB clock.

bool CLOCK\_EnableUsbfs0DeviceClock(*clock\_usb\_src\_t* src, uint32\_t freq)

Enable USB Device FS clock.

#### Parameters

- src – : clock source
- freq – clock frequency Enable USB Device Full Speed clock.

bool CLOCK\_EnableUsbfs0HostClock(*clock\_usb\_src\_t* src, uint32\_t freq)

Enable USB HOST FS clock.

#### Parameters

- src – : clock source
- freq – clock frequency Enable USB HOST Full Speed clock.

void CLOCK\_SetStoredUsbPLLClockRate(uint32\_t rate)

Set the current Usb PLL Rate.

bool CLOCK\_EnableUsbhs0DeviceClock(*clock\_usb\_src\_t* src, uint32\_t freq)

Enable USB Device HS clock.

#### Parameters

- `src` – : clock source
- `freq` – clock frequency Enable USB Device High Speed clock.

`bool CLOCK_EnableUsbhs0HostClock(clock_usb_src_t src, uint32_t freq)`

Enable USB HOST HS clock.

#### Parameters

- `src` – : clock source
- `freq` – clock frequency Enable USB HOST High Speed clock.

`FSL_CLOCK_DRIVER_VERSION`

CLOCK driver version 2.3.3.

`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.

---

`CLOCK_USR_CFG_PLL_CONFIG_CACHE_COUNT`

User-defined the size of cache for `CLOCK_PllGetConfig()` function.

Once define this MACRO to be non-zero value, `CLOCK_PllGetConfig()` function would cache the recent calculation and accelerate the execution to get the right settings.

`SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY`

`ADC_CLOCKS`

Clock ip name array for ADC.

`ROM_CLOCKS`

Clock ip name array for ROM.

`SRAM_CLOCKS`

Clock ip name array for SRAM.

`FLASH_CLOCKS`

Clock ip name array for FLASH.

`FMC_CLOCKS`

Clock ip name array for FMC.

`EEPROM_CLOCKS`

Clock ip name array for EEPROM.

`SPIFI_CLOCKS`

Clock ip name array for SPIFI.

`INPUTMUX_CLOCKS`

Clock ip name array for INPUTMUX.

`IOCON_CLOCKS`

Clock ip name array for IOCON.

`GPIO_CLOCKS`

Clock ip name array for GPIO.

PINT\_CLOCKS

Clock ip name array for PINT.

GINT\_CLOCKS

Clock ip name array for GINT.

DMA\_CLOCKS

Clock ip name array for DMA.

CRC\_CLOCKS

Clock ip name array for CRC.

WWDT\_CLOCKS

Clock ip name array for WWDT.

RTC\_CLOCKS

Clock ip name array for RTC.

ADC0\_CLOCKS

Clock ip name array for ADC0.

MRT\_CLOCKS

Clock ip name array for MRT.

RIT\_CLOCKS

Clock ip name array for RIT.

SCT\_CLOCKS

Clock ip name array for SCT0.

MCAN\_CLOCKS

Clock ip name array for MCAN.

UTICK\_CLOCKS

Clock ip name array for UTICK.

FLEXCOMM\_CLOCKS

Clock ip name array for FLEXCOMM.

LPUART\_CLOCKS

Clock ip name array for LPUART.

BI2C\_CLOCKS

Clock ip name array for BI2C.

LPSI\_CLOCKS

Clock ip name array for LSPI.

FLEXI2S\_CLOCKS

Clock ip name array for FLEXI2S.

DMIC\_CLOCKS

Clock ip name array for DMIC.

CTIMER\_CLOCKS

Clock ip name array for CT32B.

LCD\_CLOCKS

Clock ip name array for LCD.

SDIO\_CLOCKS

Clock ip name array for SDIO.

USBRAM\_CLOCKS

Clock ip name array for USBRAM.

EMC\_CLOCKS

Clock ip name array for EMC.

ETH\_CLOCKS

Clock ip name array for ETH.

AES\_CLOCKS

Clock ip name array for AES.

OTP\_CLOCKS

Clock ip name array for OTP.

RNG\_CLOCKS

Clock ip name array for RNG.

USBHMR0\_CLOCKS

Clock ip name array for USBHMR0.

USBHSL0\_CLOCKS

Clock ip name array for USBHSL0.

SHA0\_CLOCKS

Clock ip name array for SHA0.

SMARTCARD\_CLOCKS

Clock ip name array for SMARTCARD.

USBD\_CLOCKS

Clock ip name array for USBD.

USBH\_CLOCKS

Clock ip name array for USBH.

CLK\_GATE\_REG\_OFFSET\_SHIFT

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

CLK\_GATE\_REG\_OFFSET\_MASK

CLK\_GATE\_BIT\_SHIFT\_SHIFT

CLK\_GATE\_BIT\_SHIFT\_MASK

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

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

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

AHB\_CLK\_CTRL0

AHB\_CLK\_CTRL1

AHB\_CLK\_CTRL2

ASYNC\_CLK\_CTRL0

CLK\_ATTACH\_ID(mux, sel, pos)

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

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



MUX\_A\_OFFSET(**mux**)  
MUX\_A(**mux**, **sel**)  
MUX\_B(**mux**, **sel**, **selector**)  
GET\_ID\_ITEM(**connection**)  
GET\_ID\_NEXT\_ITEM(**connection**)  
GET\_ID\_ITEM\_MUX(**connection**)  
GET\_ID\_ITEM\_SEL(**connection**)  
GET\_ID\_SELECTOR(**connection**)  
CLR\_ID\_ITEM\_SEL(**connection**)  
CM\_STICKCLKSEL  
CM\_MAINCLKSELA  
CM\_MAINCLKSELB  
CM\_CLKOUTCLKSELA  
CM\_SYSPLLCLKSEL  
CM\_AUDPLLCLKSEL  
CM\_SPIFICLKSEL  
CM\_ADCASYNCCLKSEL  
CM\_USB0CLKSEL  
CM\_USB1CLKSEL  
CM\_FXCOMCLKSEL0  
CM\_FXCOMCLKSEL1  
CM\_FXCOMCLKSEL2  
CM\_FXCOMCLKSEL3  
CM\_FXCOMCLKSEL4  
CM\_FXCOMCLKSEL5  
CM\_FXCOMCLKSEL6  
CM\_FXCOMCLKSEL7  
CM\_FXCOMCLKSEL8  
CM\_FXCOMCLKSEL9  
CM\_FXCOMCLKSEL10  
CM\_MCLKCLKSEL  
CM\_FRGCLKSEL  
CM\_DMICCLKSEL

CM\_SCTCLKSEL

CM\_LCDCLKSEL

CM\_SDIOCLKSEL

CM\_ASYNCAPB

PLL\_CONFIGFLAG\_USEINRATE

PLL configuration structure flags for 'flags' field These flags control how the PLL configuration function sets up the PLL setup structure.

When the PLL\_CONFIGFLAG\_USEINRATE flag is selected, the 'InputRate' field in the configuration structure must be assigned with the expected PLL frequency. If the PLL\_CONFIGFLAG\_USEINRATE is not used, 'InputRate' is ignored in the configuration function and the driver will determine the PLL rate from the currently selected PLL source. This flag might be used to configure the PLL input clock more accurately when using the WDT oscillator or a more dynamic CLKIN source.

When the PLL\_CONFIGFLAG\_FORCENOFRACT flag is selected, the PLL hardware for the automatic bandwidth selection, Spread Spectrum (SS) support, and fractional M-divider are not used.

Flag to use InputRate in PLL configuration structure for setup

PLL\_CONFIGFLAG\_FORCENOFRACT

Force non-fractional output mode, PLL output will not use the fractional, automatic bandwidth, or \ SS hardware

PLL\_SETUPFLAG\_POWERUP

PLL setup structure flags for 'flags' field These flags control how the PLL setup function sets up the PLL.

Setup will power on the PLL after setup

PLL\_SETUPFLAG\_WAITLOCK

Setup will wait for PLL lock, implies the PLL will be powered on

PLL\_SETUPFLAG\_ADGVOLT

Optimize system voltage for the new PLL rate

uint32\_t desiredRate

Desired PLL rate in Hz

uint32\_t inputRate

PLL input clock in Hz, only used if PLL\_CONFIGFLAG\_USEINRATE flag is set

uint32\_t flags

PLL configuration flags, Or'ed value of PLL\_CONFIGFLAG\_\* definitions

uint32\_t pllctrl

PLL control register SYSPLLCTRL

uint32\_t pllndec

PLL NDEC register SYSPLLNDEC

uint32\_t pllpdec

PLL PDEC register SYSPLLPDEC

uint32\_t pllmdec

PLL MDEC registers SYSPLLPDEC

uint32\_t pllRate

Actual PLL rate

uint32\_t audpllfrac

only audio PLL has this function

uint32\_t flags

PLL setup flags, Or'ed value of PLL\_SETUPFLAG\_\* definitions

uint8\_t msel

USB PLL control register msel:1U-256U

uint8\_t psel

USB PLL control register psel:only support inter 1U 2U 4U 8U

uint8\_t nsel

USB PLL control register nsel:only support inter 1U 2U 3U 4U

bool direct

USB PLL CCO output control

bool bypass

USB PLL input clock bypass control

bool fbssel

USB PLL integer mode and non-integer mode control

uint32\_t inputRate

USB PLL input rate

struct \_\_pll\_config

*#include <fsl\_clock.h>* PLL configuration structure.

This structure can be used to configure the settings for a PLL setup structure. Fill in the desired configuration for the PLL and call the PLL setup function to fill in a PLL setup structure.

struct \_\_pll\_setup

*#include <fsl\_clock.h>* PLL setup structure This structure can be used to pre-build a PLL setup configuration at run-time and quickly set the PLL to the configuration. It can be populated with the PLL setup function. If powering up or waiting for PLL lock, the PLL input clock source should be configured prior to PLL setup.

struct \_\_usb\_pll\_setup

*#include <fsl\_clock.h>* USB PLL setup structure This structure can be used to pre-build a USB PLL setup configuration at run-time and quickly set the USB PLL to the configuration. It can be populated with the USB PLL setup function. If powering up or waiting for USB PLL lock, the PLL input clock source should be configured prior to USB PLL setup.

## 2.3 CRC: Cyclic Redundancy Check Driver

FSL\_CRC\_DRIVER\_VERSION

CRC driver version. Version 2.1.1.

Current version: 2.1.1

Change log:

- Version 2.0.0
  - initial version

- Version 2.0.1
  - add explicit type cast when writing to WR\_DATA
- Version 2.0.2
  - Fix MISRA issue
- Version 2.1.0
  - Add CRC\_WriteSeed function
- Version 2.1.1
  - Fix MISRA issue

enum *\_crc\_polynomial*  
CRC polynomials to use.

*Values:*

enumerator *kCRC\_Polynomial\_CRC\_CCITT*  
 $x^{16}+x^{12}+x^5+1$

enumerator *kCRC\_Polynomial\_CRC\_16*  
 $x^{16}+x^{15}+x^2+1$

enumerator *kCRC\_Polynomial\_CRC\_32*  
 $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

typedef enum *\_crc\_polynomial* *crc\_polynomial\_t*  
CRC polynomials to use.

typedef struct *\_crc\_config* *crc\_config\_t*  
CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

void *CRC\_Init*(CRC\_Type \*base, const *crc\_config\_t* \*config)  
Enables and configures the CRC peripheral module.

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

#### Parameters

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

static inline void *CRC\_Deinit*(CRC\_Type \*base)  
Disables the CRC peripheral module.

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

#### Parameters

- base – CRC peripheral address.

void *CRC\_Reset*(CRC\_Type \*base)  
resets CRC peripheral module.

#### Parameters

- base – CRC peripheral address.

void CRC\_WriteSeed(CRC\_Type \*base, uint32\_t seed)

Write seed to CRC peripheral module.

#### Parameters

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

void CRC\_GetDefaultConfig(*crc\_config\_t* \*config)

Loads default values to CRC protocol configuration structure.

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

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

#### Parameters

- config – CRC protocol configuration structure

void CRC\_GetConfig(CRC\_Type \*base, *crc\_config\_t* \*config)

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

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

#### Parameters

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

void CRC\_WriteData(CRC\_Type \*base, const uint8\_t \*data, size\_t dataSize)

Writes data to the CRC module.

Writes input data buffer bytes to CRC data register.

#### Parameters

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

static inline uint32\_t CRC\_Get32bitResult(CRC\_Type \*base)

Reads 32-bit checksum from the CRC module.

Reads CRC data register.

#### Parameters

- base – CRC peripheral address.

#### Returns

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

static inline uint16\_t CRC\_Get16bitResult(CRC\_Type \*base)

Reads 16-bit checksum from the CRC module.

Reads CRC data register.

#### Parameters

- base – CRC peripheral address.

**Returns**

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

CRC\_DRIVER\_USE\_CRC16\_CCITT\_FALSE\_AS\_DEFAULT

Default configuration structure filled by CRC\_GetDefaultConfig(). Uses CRC-16/CCITT-FALSE as default.

struct \_\_crc\_config

#include <fsl\_crc.h> CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

**Public Members**

crc\_polynomial\_t polynomial

CRC polynomial.

bool reverseIn

Reverse bits on input.

bool complementIn

Perform 1's complement on input.

bool reverseOut

Reverse bits on output.

bool complementOut

Perform 1's complement on output.

uint32\_t seed

Starting checksum value.

## 2.4 CTIMER: Standard counter/timers

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

Un-gates the clock and configures the peripheral for basic operation.

---

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

---

**Parameters**

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

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

Gates the timer clock.

**Parameters**

- base – Ctimer peripheral base address

void CTIMER\_GetDefaultConfig(*ctimer\_config\_t* \*config)

Fills in the timers configuration structure with the default settings.

The default values are:

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

### Parameters

- config – Pointer to the user configuration structure.

*status\_t* CTIMER\_SetupPwmPeriod(CTIMER\_Type \*base, const *ctimer\_match\_t* pwmPeriodChannel, *ctimer\_match\_t* matchChannel, uint32\_t pwmPeriod, uint32\_t pulsePeriod, bool enableInt)

Configures the PWM signal parameters.

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

---

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

---

### Parameters

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

### Returns

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

*status\_t* CTIMER\_SetupPwm(CTIMER\_Type \*base, const *ctimer\_match\_t* pwmPeriodChannel, *ctimer\_match\_t* matchChannel, uint8\_t dutyCyclePercent, uint32\_t pwmFreq\_Hz, uint32\_t srcClock\_Hz, bool enableInt)

Configures the PWM signal parameters.

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

---

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

---

### Parameters

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

- `pwmFreq_Hz` – PWM signal frequency in Hz
- `srcClock_Hz` – Timer counter clock in Hz
- `enableInt` – Enable interrupt when the timer value reaches the match value of the PWM pulse, if it is 0 then no interrupt will be generated.

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

Updates the pulse period of an active PWM signal.

#### Parameters

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

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

Updates the duty cycle of an active PWM signal.

---

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

---

#### Parameters

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

#### Returns

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

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

Enables the selected Timer interrupts.

#### Parameters

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

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

Disables the selected Timer interrupts.

#### Parameters

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

```
static inline uint32_t CTIMER_GetEnabledInterrupts(CTIMER_Type *base)
```

Gets the enabled Timer interrupts.

#### Parameters



- base – Ctimer peripheral base address

**Returns**

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

```
static inline uint32_t CTIMER_GetStatusFlags(CTIMER_Type *base)
```

Gets the Timer status flags.

**Parameters**

- base – Ctimer peripheral base address

**Returns**

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

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

Clears the Timer status flags.

**Parameters**

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

```
static inline void CTIMER_StartTimer(CTIMER_Type *base)
```

Starts the Timer counter.

**Parameters**

- base – Ctimer peripheral base address

```
static inline void CTIMER_StopTimer(CTIMER_Type *base)
```

Stops the Timer counter.

**Parameters**

- base – Ctimer peripheral base address

FSL\_CTIMER\_DRIVER\_VERSION

Version 2.3.3

```
enum _ctimer_capture_channel
```

List of Timer capture channels.

*Values:*

```
enumerator kCTIMER_Capture_0
```

Timer capture channel 0

```
enumerator kCTIMER_Capture_1
```

Timer capture channel 1

```
enumerator kCTIMER_Capture_3
```

Timer capture channel 3

```
enum _ctimer_capture_edge
```

List of capture edge options.

*Values:*

```
enumerator kCTIMER_Capture_RiseEdge
```

Capture on rising edge

enumerator kCTIMER\_Capture\_FallEdge

Capture on falling edge

enumerator kCTIMER\_Capture\_BothEdge

Capture on rising and falling edge

enum \_ctimer\_match

List of Timer match registers.

*Values:*

enumerator kCTIMER\_Match\_0

Timer match register 0

enumerator kCTIMER\_Match\_1

Timer match register 1

enumerator kCTIMER\_Match\_2

Timer match register 2

enumerator kCTIMER\_Match\_3

Timer match register 3

enum \_ctimer\_external\_match

List of external match.

*Values:*

enumerator kCTIMER\_External\_Match\_0

External match 0

enumerator kCTIMER\_External\_Match\_1

External match 1

enumerator kCTIMER\_External\_Match\_2

External match 2

enumerator kCTIMER\_External\_Match\_3

External match 3

enum \_ctimer\_match\_output\_control

List of output control options.

*Values:*

enumerator kCTIMER\_Output\_NoAction

No action is taken

enumerator kCTIMER\_Output\_Clear

Clear the EM bit/output to 0

enumerator kCTIMER\_Output\_Set

Set the EM bit/output to 1

enumerator kCTIMER\_Output\_Toggle

Toggle the EM bit/output

enum \_ctimer\_timer\_mode

List of Timer modes.

*Values:*

enumerator kCTIMER\_TimerMode

enumerator kCTIMER\_IncreaseOnRiseEdge

enumerator kCTIMER\_IncreaseOnFallEdge

enumerator kCTIMER\_IncreaseOnBothEdge

enum \_ctimer\_interrupt\_enable

List of Timer interrupts.

*Values:*

enumerator kCTIMER\_Match0InterruptEnable

Match 0 interrupt

enumerator kCTIMER\_Match1InterruptEnable

Match 1 interrupt

enumerator kCTIMER\_Match2InterruptEnable

Match 2 interrupt

enumerator kCTIMER\_Match3InterruptEnable

Match 3 interrupt

enum \_ctimer\_status\_flags

List of Timer flags.

*Values:*

enumerator kCTIMER\_Match0Flag

Match 0 interrupt flag

enumerator kCTIMER\_Match1Flag

Match 1 interrupt flag

enumerator kCTIMER\_Match2Flag

Match 2 interrupt flag

enumerator kCTIMER\_Match3Flag

Match 3 interrupt flag

enum ctimer\_callback\_type\_t

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

*Values:*

enumerator kCTIMER\_SingleCallback

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

enumerator kCTIMER\_MultipleCallback

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

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

List of Timer capture channels.

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

List of capture edge options.

typedef enum \_ctimer\_match ctimer\_match\_t

List of Timer match registers.

```
typedef enum _ctimer_external_match ctimer_external_match_t
```

List of external match.

```
typedef enum _ctimer_match_output_control ctimer_match_output_control_t
```

List of output control options.

```
typedef enum _ctimer_timer_mode ctimer_timer_mode_t
```

List of Timer modes.

```
typedef enum _ctimer_interrupt_enable ctimer_interrupt_enable_t
```

List of Timer interrupts.

```
typedef enum _ctimer_status_flags ctimer_status_flags_t
```

List of Timer flags.

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

```
typedef struct _ctimer_match_config ctimer_match_config_t
```

Match configuration.

This structure holds the configuration settings for each match register.

```
typedef struct _ctimer_config ctimer_config_t
```

Timer configuration structure.

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

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

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

Setup the match register.

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

#### Parameters

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

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

Get the status of output match.

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

#### Parameters

- `base` – Ctimer peripheral base address
- `matchChannel` – External match channel, user can obtain the status of multiple match channels at the same time by using the logic of “|” enumeration `ctimer_external_match_t`

#### Returns

The mask of external match channel status flags. Users need to use the `_ctimer_external_match` type to decode the return variables.

```
void CTIMER_SetupCapture(CTIMER_Type *base, ctimer_capture_channel_t capture,
                        ctimer_capture_edge_t edge, bool enableInt)
```

Setup the capture.

#### Parameters

- base – Ctimer peripheral base address
- capture – Capture channel to configure
- edge – Edge on the channel that will trigger a capture
- enableInt – Flag to enable channel interrupts, if enabled then the registered call back is called upon capture

```
static inline uint32_t CTIMER_GetTimerCountValue(CTIMER_Type *base)
```

Get the timer count value from TC register.

#### Parameters

- base – Ctimer peripheral base address.

#### Returns

return the timer count value.

```
void CTIMER_RegisterCallBack(CTIMER_Type *base, ctimer_callback_t *cb_func,
                            ctimer_callback_type_t cb_type)
```

Register callback.

This function configures CTimer Callback in following modes:

- Single Callback: cb\_func should be pointer to callback function pointer For example: `ctimer_callback_t ctimer_callback = pwm_match_callback; CTIMER_RegisterCallBack(CTIMER, &ctimer_callback, kCTIMER_SingleCallback);`
- Multiple Callback: cb\_func should be pointer to array of callback function pointers Each element corresponds to Interrupt Flag in IR register. For example: `ctimer_callback_t ctimer_callback_table[] = { ctimer_match0_callback, NULL, NULL, ctimer_match3_callback, NULL, NULL, NULL, NULL}; CTIMER_RegisterCallBack(CTIMER, &ctimer_callback_table[0], kCTIMER_MultipleCallback);`

#### Parameters

- base – Ctimer peripheral base address
- cb\_func – Pointer to callback function pointer
- cb\_type – callback function type, singular or multiple

```
static inline void CTIMER_Reset(CTIMER_Type *base)
```

Reset the counter.

The timer counter and prescale counter are reset on the next positive edge of the APB clock.

#### Parameters

- base – Ctimer peripheral base address

```
static inline void CTIMER_SetPrescale(CTIMER_Type *base, uint32_t prescale)
```

Setup the timer prescale value.

Specifies the maximum value for the Prescale Counter.

#### Parameters

- base – Ctimer peripheral base address
- prescale – Prescale value

```
static inline uint32_t CTIMER_GetCaptureValue(CTIMER_Type *base, ctimer_capture_channel_t
                                              capture)
```

Get capture channel value.

Get the counter/timer value on the corresponding capture channel.

**Parameters**

- base – Ctimer peripheral base address
- capture – Select capture channel

**Returns**

The timer count capture value.

```
static inline void CTIMER_EnableResetMatchChannel(CTIMER_Type *base, ctimer_match_t
                                                  match, bool enable)
```

Enable reset match channel.

Set the specified match channel reset operation.

**Parameters**

- base – Ctimer peripheral base address
- match – match channel used
- enable – Enable match channel reset operation.

```
static inline void CTIMER_EnableStopMatchChannel(CTIMER_Type *base, ctimer_match_t
                                                  match, bool enable)
```

Enable stop match channel.

Set the specified match channel stop operation.

**Parameters**

- base – Ctimer peripheral base address.
- match – match channel used.
- enable – Enable match channel stop operation.

```
static inline void CTIMER_EnableMatchChannelReload(CTIMER_Type *base, ctimer_match_t
                                                  match, bool enable)
```

Enable reload channel falling edge.

Enable the specified match channel reload match shadow value.

**Parameters**

- base – Ctimer peripheral base address.
- match – match channel used.
- enable – Enable .

```
static inline void CTIMER_EnableRisingEdgeCapture(CTIMER_Type *base,
                                                  ctimer_capture_channel_t capture, bool
                                                  enable)
```

Enable capture channel rising edge.

Sets the specified capture channel for rising edge capture.

**Parameters**

- base – Ctimer peripheral base address.
- capture – capture channel used.
- enable – Enable rising edge capture.

```
static inline void CTIMER_EnableFallingEdgeCapture(CTIMER_Type *base,
                                                    ctimer_capture_channel_t capture, bool
                                                    enable)
```

Enable capture channel falling edge.

Sets the specified capture channel for falling edge capture.

#### Parameters

- base – Ctimer peripheral base address.
- capture – capture channel used.
- enable – Enable falling edge capture.

```
static inline void CTIMER_SetShadowValue(CTIMER_Type *base, ctimer_match_t match,
                                          uint32_t matchvalue)
```

Set the specified match shadow channel.

#### Parameters

- base – Ctimer peripheral base address.
- match – match channel used.
- matchvalue – Reload the value of the corresponding match register.

```
struct _ctimer_match_config
```

*#include <fsl\_ctimer.h>* Match configuration.

This structure holds the configuration settings for each match register.

#### Public Members

uint32\_t matchValue

This is stored in the match register

bool enableCounterReset

true: Match will reset the counter false: Match will not reset the counter

bool enableCounterStop

true: Match will stop the counter false: Match will not stop the counter

ctimer\_match\_output\_control\_t outControl

Action to be taken on a match on the EM bit/output

bool outPinInitState

Initial value of the EM bit/output

bool enableInterrupt

true: Generate interrupt upon match false: Do not generate interrupt on match

```
struct _ctimer_config
```

*#include <fsl\_ctimer.h>* Timer configuration structure.

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

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

### Public Members

*ctimer\_timer\_mode\_t* mode

Timer mode

*ctimer\_capture\_channel\_t* input

Input channel to increment the timer; used only in timer modes that rely on this input signal to increment TC

uint32\_t prescale

Prescale value

## 2.5 DMA: Direct Memory Access Controller Driver

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

Initializes DMA peripheral.

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

### Parameters

- base – DMA peripheral base address.

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

Deinitializes DMA peripheral.

This function gates the DMA clock.

### Parameters

- base – DMA peripheral base address.

void DMA\_InstallDescriptorMemory(DMA\_Type \*base, void \*addr)

Install DMA descriptor memory.

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

### Parameters

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

static inline bool DMA\_ChannelIsActive(DMA\_Type \*base, uint32\_t channel)

Return whether DMA channel is processing transfer.

### Parameters

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

### Returns

True for active state, false otherwise.

static inline bool DMA\_ChannelIsBusy(DMA\_Type \*base, uint32\_t channel)

Return whether DMA channel is busy.

### Parameters

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



**Returns**

True for busy state, false otherwise.

static inline void DMA\_EnableChannelInterrupts(DMA\_Type \*base, uint32\_t channel)

Enables the interrupt source for the DMA transfer.

**Parameters**

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

static inline void DMA\_DisableChannelInterrupts(DMA\_Type \*base, uint32\_t channel)

Disables the interrupt source for the DMA transfer.

**Parameters**

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

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

Enable DMA channel.

**Parameters**

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

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

Disable DMA channel.

**Parameters**

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

static inline void DMA\_EnableChannelPeriphRq(DMA\_Type \*base, uint32\_t channel)

Set PERIPHREQEN of channel configuration register.

**Parameters**

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

static inline void DMA\_DisableChannelPeriphRq(DMA\_Type \*base, uint32\_t channel)

Get PERIPHREQEN value of channel configuration register.

**Parameters**

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

**Returns**

True for enabled PeriphRq, false for disabled.

void DMA\_ConfigureChannelTrigger(DMA\_Type \*base, uint32\_t channel, dma\_channel\_trigger\_t \*trigger)

Set trigger settings of DMA channel.

**Deprecated:**

Do not use this function. It has been superseded by DMA\_SetChannelConfig.

**Parameters**

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

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

set channel config.

This function provide a interface to configure channel configuration registers.

#### Parameters

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

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

DMA channel xfer transfer configurations.

#### Parameters

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

#### Returns

The value of xfer config

```
uint32_t DMA_GetRemainingBytes(DMA_Type *base, uint32_t channel)
```

Gets the remaining bytes of the current DMA descriptor transfer.

#### Parameters

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

#### Returns

The number of bytes which have not been transferred yet.

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

Set priority of channel configuration register.

#### Parameters

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

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

Get priority of channel configuration register.

**Parameters**

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

**Returns**

Channel priority value.

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

Set channel configuration valid.

**Parameters**

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

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

Do software trigger for the channel.

**Parameters**

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

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

Load channel transfer configurations.

**Parameters**

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

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

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

*Deprecated:*

Do not use this function. It has been superceded by DMA\_SetupDescriptor.

**Parameters**

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

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

setup dma descriptor

Note: This function do not support configure wrap descriptor.

**Parameters**

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

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

setup dma channel descriptor

Note: This function support configure wrap descriptor.

#### Parameters

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

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

load channel transfer decriptor.

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

- a. for the polling transfer, application can allocate a local descriptor memory table to prepare a descriptor firstly and then call this api to load the configured descriptor to driver descriptor table.

```
DMA_Init(DMA0);  
DMA_EnableChannel(DMA0, DEMO_DMA_CHANNEL);  
DMA_SetupDescriptor(desc, xferCfg, s_srcBuffer, &s_destBuffer[0], NULL);  
DMA_LoadChannelDescriptor(DMA0, DEMO_DMA_CHANNEL, (dma_descriptor_t *)desc);  
DMA_DoChannelSoftwareTrigger(DMA0, DEMO_DMA_CHANNEL);  
while(DMA_ChannelIsBusy(DMA0, DEMO_DMA_CHANNEL))  
{  
}
```

#### Parameters

- base – DMA base address.
- channel – DMA channel.
- descriptor – configured DMA descriptor.

```
void DMA_AbortTransfer(dma_handle_t *handle)
```

Abort running transfer by handle.

This function aborts DMA transfer specified by handle.

#### Parameters

- handle – DMA handle pointer.

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

Creates the DMA handle.

This function is called if using transaction API for DMA. This function initializes the internal state of DMA handle.

#### Parameters

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

void DMA\_SetCallback(*dma\_handle\_t* \*handle, *dma\_callback* callback, void \*userData)

Installs a callback function for the DMA transfer.

This callback is called in DMA IRQ handler. Use the callback to do something after the current major loop transfer completes.

#### Parameters

- handle – DMA handle pointer.
- callback – DMA callback function pointer.
- userData – Parameter for callback function.

void DMA\_PrepareTransfer(*dma\_transfer\_config\_t* \*config, void \*srcAddr, void \*dstAddr, uint32\_t byteWidth, uint32\_t transferBytes, *dma\_transfer\_type\_t* type, void \*nextDesc)

Prepares the DMA transfer structure.

#### Deprecated:

Do not use this function. It has been superceded by DMA\_PrepareChannelTransfer. This function prepares the transfer configuration structure according to the user input.

---

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

---

#### Parameters

- config – The user configuration structure of type *dma\_transfer\_t*.
- srcAddr – DMA transfer source address.
- dstAddr – DMA transfer destination address.
- byteWidth – DMA transfer destination address width(bytes).
- transferBytes – DMA transfer bytes to be transferred.
- type – DMA transfer type.
- nextDesc – Chain custom descriptor to transfer.

void DMA\_PrepareChannelTransfer(*dma\_channel\_config\_t* \*config, void \*srcStartAddr, void \*dstStartAddr, uint32\_t xferCfg, *dma\_transfer\_type\_t* type, *dma\_channel\_trigger\_t* \*trigger, void \*nextDesc)

Prepare channel transfer configurations.

This function used to prepare channel transfer configurations.

### Parameters

- config – Pointer to DMA channel transfer configuration structure.
- srcStartAddr – source start address.
- dstStartAddr – destination start address.
- xferCfg – xfer configuration, user can reference DMA\_CHANNEL\_XFER about to how to get xferCfg value.
- type – transfer type.
- trigger – DMA channel trigger configurations.
- nextDesc – address of next descriptor.

*status\_t* DMA\_SubmitTransfer(*dma\_handle\_t* \*handle, *dma\_transfer\_config\_t* \*config)

Submits the DMA transfer request.

### Deprecated:

Do not use this function. It has been superceded by DMA\_SubmitChannelTransfer.

This function submits the DMA transfer request according to the transfer configuration structure. If the user submits the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time.

### Parameters

- handle – DMA handle pointer.
- config – Pointer to DMA transfer configuration structure.

### Return values

- kStatus\_DMA\_Success – It means submit transfer request succeed.
- kStatus\_DMA\_QueueFull – It means TCD queue is full. Submit transfer request is not allowed.
- kStatus\_DMA\_Busy – It means the given channel is busy, need to submit request later.

*void* DMA\_SubmitChannelTransferParameter(*dma\_handle\_t* \*handle, *uint32\_t* xferCfg, *void* \*srcStartAddr, *void* \*dstStartAddr, *void* \*nextDesc)

Submit channel transfer paramter directly.

This function used to configure channel head descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, it is useful for the case:

- a. for the single transfer, application doesn't need to allocate descriptor table, the head descriptor can be used for it.

```
DMA_SetChannelConfig(base, channel, trigger, isPeriph);
DMA_CreateHandle(handle, base, channel)
DMA_SubmitChannelTransferParameter(handle, DMA_CHANNEL_XFER(reload, clrTrig,
↪intA, intB, width, srcInc, dstInc,
bytes), srcStartAddr, dstStartAddr, NULL);
DMA_StartTransfer(handle)
```

- b. for the linked transfer, application should responsible for link descriptor, for example, if 4 transfer is required, then application should prepare three descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```

define link descriptor table in application with macro
DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc[3]);

DMA_SetupDescriptor(nextDesc0, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
DMA_SetupDescriptor(nextDesc1, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc2);
DMA_SetupDescriptor(nextDesc2, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, NULL);
DMA_SetChannelConfig(base, channel, trigger, isPeriph);
DMA_CreateHandle(handle, base, channel)
DMA_SubmitChannelTransferParameter(handle, DMA_CHANNEL_XFER(reload, clrTrig,
↪ intA, intB, width, srcInc, dstInc,
bytes), srcStartAddr, dstStartAddr, nextDesc0);
DMA_StartTransfer(handle);

```

### Parameters

- handle – Pointer to DMA handle.
- xferCfg – xfer configuration, user can reference DMA\_CHANNEL\_XFER about to how to get xferCfg value.
- srcStartAddr – source start address.
- dstStartAddr – destination start address.
- nextDesc – address of next descriptor.

void DMA\_SubmitChannelDescriptor(*dma\_handle\_t* \*handle, *dma\_descriptor\_t* \*descriptor)

Submit channel descriptor.

This function used to configure channel head descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, this function is typical for the ping pong case:

- for the ping pong case, application should responsible for the descriptor, for example, application should prepare two descriptor table with macro.

```

define link descriptor table in application with macro
DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc[2]);

DMA_SetupDescriptor(nextDesc0, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
DMA_SetupDescriptor(nextDesc1, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc0);
DMA_SetChannelConfig(base, channel, trigger, isPeriph);
DMA_CreateHandle(handle, base, channel)
DMA_SubmitChannelDescriptor(handle, nextDesc0);
DMA_StartTransfer(handle);

```

### Parameters

- handle – Pointer to DMA handle.
- descriptor – descriptor to submit.

*status\_t* DMA\_SubmitChannelTransfer(*dma\_handle\_t* \*handle, *dma\_channel\_config\_t* \*config)

Submits the DMA channel transfer request.

This function submits the DMA transfer request according to the transfer configuration structure. If the user submits the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time. It is used for the case:

- a. for the single transfer, application doesn't need to allocate descriptor table, the head descriptor can be used for it.

```
DMA_CreateHandle(handle, base, channel)
DMA_PrepareChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,trigger,NULL);
DMA_SubmitChannelTransfer(handle, config)
DMA_StartTransfer(handle)
```

- b. for the linked transfer, application should responsible for link descriptor, for example, if 4 transfer is required, then application should prepare three descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```
define link descriptor table in application with macro
DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc);
DMA_SetupDescriptor(nextDesc0, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
DMA_SetupDescriptor(nextDesc1, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc2);
DMA_SetupDescriptor(nextDesc2, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, NULL);
DMA_CreateHandle(handle, base, channel)
DMA_PrepareChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,trigger,
↪ nextDesc0);
DMA_SubmitChannelTransfer(handle, config)
DMA_StartTransfer(handle)
```

- c. for the ping pong case, application should responsible for link descriptor, for example, application should prepare two descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```
define link descriptor table in application with macro
DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc);

DMA_SetupDescriptor(nextDesc0, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
DMA_SetupDescriptor(nextDesc1, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc0);
DMA_CreateHandle(handle, base, channel)
DMA_PrepareChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,trigger,
↪ nextDesc0);
DMA_SubmitChannelTransfer(handle, config)
DMA_StartTransfer(handle)
```

## Parameters

- handle – DMA handle pointer.
- config – Pointer to DMA transfer configuration structure.

## Return values



- `kStatus_DMA_Success` – It means submit transfer request succeed.
- `kStatus_DMA_QueueFull` – It means TCD queue is full. Submit transfer request is not allowed.
- `kStatus_DMA_Busy` – It means the given channel is busy, need to submit request later.

`void DMA_StartTransfer(dma_handle_t *handle)`

DMA start transfer.

This function enables the channel request. User can call this function after submitting the transfer request. It will trigger transfer start with software trigger only when hardware trigger is not used.

#### Parameters

- `handle` – DMA handle pointer.

`void DMA_IRQHandle(DMA_Type *base)`

DMA IRQ handler for descriptor transfer complete.

This function clears the channel major interrupt flag and call the callback function if it is not NULL.

#### Parameters

- `base` – DMA base address.

`FSL_DMA_DRIVER_VERSION`

DMA driver version.

Version 2.5.3.

`_dma_transfer_status` DMA transfer status

*Values:*

enumerator `kStatus_DMA_Busy`

Channel is busy and can't handle the transfer request.

`_dma_addr_interleave_size` dma address interleave size

*Values:*

enumerator `kDMA_AddressInterleave0xWidth`

dma source/destination address no interleave

enumerator `kDMA_AddressInterleave1xWidth`

dma source/destination address interleave 1xwidth

enumerator `kDMA_AddressInterleave2xWidth`

dma source/destination address interleave 2xwidth

enumerator `kDMA_AddressInterleave4xWidth`

dma source/destination address interleave 3xwidth

`_dma_transfer_width` dma transfer width

*Values:*

enumerator `kDMA_Transfer8BitWidth`

dma channel transfer bit width is 8 bit

enumerator kDMA\_\_Transfer16BitWidth  
dma channel transfer bit width is 16 bit

enumerator kDMA\_\_Transfer32BitWidth  
dma channel transfer bit width is 32 bit

enum \_\_dma\_\_priority

DMA channel priority.

*Values:*

enumerator kDMA\_\_ChannelPriority0  
Highest channel priority - priority 0

enumerator kDMA\_\_ChannelPriority1  
Channel priority 1

enumerator kDMA\_\_ChannelPriority2  
Channel priority 2

enumerator kDMA\_\_ChannelPriority3  
Channel priority 3

enumerator kDMA\_\_ChannelPriority4  
Channel priority 4

enumerator kDMA\_\_ChannelPriority5  
Channel priority 5

enumerator kDMA\_\_ChannelPriority6  
Channel priority 6

enumerator kDMA\_\_ChannelPriority7  
Lowest channel priority - priority 7

enum \_\_dma\_\_int

DMA interrupt flags.

*Values:*

enumerator kDMA\_\_IntA  
DMA interrupt flag A

enumerator kDMA\_\_IntB  
DMA interrupt flag B

enumerator kDMA\_\_IntError  
DMA interrupt flag error

enum \_\_dma\_\_trigger\_\_type

DMA trigger type.

*Values:*

enumerator kDMA\_\_NoTrigger  
Trigger is disabled

enumerator kDMA\_\_LowLevelTrigger  
Low level active trigger

enumerator kDMA\_\_HighLevelTrigger  
High level active trigger

enumerator kDMA\_FallingEdgeTrigger

Falling edge active trigger

enumerator kDMA\_RisingEdgeTrigger

Rising edge active trigger

\_dma\_burst\_size DMA burst size

*Values:*

enumerator kDMA\_BurstSize1

burst size 1 transfer

enumerator kDMA\_BurstSize2

burst size 2 transfer

enumerator kDMA\_BurstSize4

burst size 4 transfer

enumerator kDMA\_BurstSize8

burst size 8 transfer

enumerator kDMA\_BurstSize16

burst size 16 transfer

enumerator kDMA\_BurstSize32

burst size 32 transfer

enumerator kDMA\_BurstSize64

burst size 64 transfer

enumerator kDMA\_BurstSize128

burst size 128 transfer

enumerator kDMA\_BurstSize256

burst size 256 transfer

enumerator kDMA\_BurstSize512

burst size 512 transfer

enumerator kDMA\_BurstSize1024

burst size 1024 transfer

enum \_dma\_trigger\_burst

DMA trigger burst.

*Values:*

enumerator kDMA\_SingleTransfer

Single transfer

enumerator kDMA\_LevelBurstTransfer

Burst transfer driven by level trigger

enumerator kDMA\_EdgeBurstTransfer1

Perform 1 transfer by edge trigger

enumerator kDMA\_EdgeBurstTransfer2

Perform 2 transfers by edge trigger

enumerator kDMA\_EdgeBurstTransfer4

Perform 4 transfers by edge trigger

enumerator kDMA\_EdgeBurstTransfer8  
Perform 8 transfers by edge trigger

enumerator kDMA\_EdgeBurstTransfer16  
Perform 16 transfers by edge trigger

enumerator kDMA\_EdgeBurstTransfer32  
Perform 32 transfers by edge trigger

enumerator kDMA\_EdgeBurstTransfer64  
Perform 64 transfers by edge trigger

enumerator kDMA\_EdgeBurstTransfer128  
Perform 128 transfers by edge trigger

enumerator kDMA\_EdgeBurstTransfer256  
Perform 256 transfers by edge trigger

enumerator kDMA\_EdgeBurstTransfer512  
Perform 512 transfers by edge trigger

enumerator kDMA\_EdgeBurstTransfer1024  
Perform 1024 transfers by edge trigger

enum \_dma\_burst\_wrap  
DMA burst wrapping.

*Values:*

enumerator kDMA\_NoWrap  
Wrapping is disabled

enumerator kDMA\_SrcWrap  
Wrapping is enabled for source

enumerator kDMA\_DstWrap  
Wrapping is enabled for destination

enumerator kDMA\_SrcAndDstWrap  
Wrapping is enabled for source and destination

enum \_dma\_transfer\_type  
DMA transfer type.

*Values:*

enumerator kDMA\_MemoryToMemory  
Transfer from memory to memory (increment source and destination)

enumerator kDMA\_PeripheralToMemory  
Transfer from peripheral to memory (increment only destination)

enumerator kDMA\_MemoryToPeripheral  
Transfer from memory to peripheral (increment only source)

enumerator kDMA\_StaticToStatic  
Peripheral to static memory (do not increment source or destination)

typedef struct *\_dma\_descriptor* dma\_descriptor\_t  
DMA descriptor structure.

typedef struct *\_dma\_xfercfg* dma\_xfercfg\_t  
DMA transfer configuration.

typedef enum *\_dma\_priority* dma\_priority\_t

DMA channel priority.

typedef enum *\_dma\_irq* dma\_irq\_t

DMA interrupt flags.

typedef enum *\_dma\_trigger\_type* dma\_trigger\_type\_t

DMA trigger type.

typedef enum *\_dma\_trigger\_burst* dma\_trigger\_burst\_t

DMA trigger burst.

typedef enum *\_dma\_burst\_wrap* dma\_burst\_wrap\_t

DMA burst wrapping.

typedef enum *\_dma\_transfer\_type* dma\_transfer\_type\_t

DMA transfer type.

typedef struct *\_dma\_channel\_trigger* dma\_channel\_trigger\_t

DMA channel trigger.

typedef struct *\_dma\_channel\_config* dma\_channel\_config\_t

DMA channel trigger.

typedef struct *\_dma\_transfer\_config* dma\_transfer\_config\_t

DMA transfer configuration.

typedef void (\*dma\_callback)(struct *\_dma\_handle* \*handle, void \*userData, bool transferDone, uint32\_t intmode)

Define Callback function for DMA.

typedef struct *\_dma\_handle* dma\_handle\_t

DMA transfer handle structure.

DMA\_MAX\_TRANSFER\_COUNT

DMA max transfer size.

FSL\_FEATURE\_DMA\_NUMBER\_OF\_CHANNELS<sub>n</sub>(x)

DMA channel numbers.

FSL\_FEATURE\_DMA\_MAX\_CHANNELS

FSL\_FEATURE\_DMA\_ALL\_CHANNELS

FSL\_FEATURE\_DMA\_LINK\_DESCRIPTOR\_ALIGN\_SIZE

DMA head link descriptor table align size.

DMA\_ALLOCATE\_HEAD\_DESCRIPTOR(name, number)

DMA head descriptor table allocate macro To simplify user interface, this macro will help allocate descriptor memory, user just need to provide the name and the number for the allocate descriptor.

#### Parameters

- name – Allocate decriptor name.
- number – Number of descriptor to be allocated.

DMA\_ALLOCATE\_HEAD\_DESCRIPTOR\_AT\_NONCACHEABLE(name, number)

DMA head descriptor table allocate macro at noncacheable section To simplify user interface, this macro will help allocate descriptor memory at noncacheable section, user just need to provide the name and the number for the allocate descriptor.

#### Parameters

- name – Allocate decriptor name.
- number – Number of descriptor to be allocated.

DMA\_ALLOCATE\_LINK\_DESCRIPTOR(name, number)

DMA link descriptor table allocate macro To simplify user interface, this macro will help allocate descriptor memory, user just need to provide the name and the number for the allocate descriptor.

#### Parameters

- name – Allocate decriptor name.
- number – Number of descriptor to be allocated.

DMA\_ALLOCATE\_LINK\_DESCRIPTOR\_AT\_NONCACHEABLE(name, number)

DMA link descriptor table allocate macro at noncacheable section To simplify user interface, this macro will help allocate descriptor memory at noncacheable section, user just need to provide the name and the number for the allocate descriptor.

#### Parameters

- name – Allocate decriptor name.
- number – Number of descriptor to be allocated.

DMA\_ALLOCATE\_DATA\_TRANSFER\_BUFFER(name, width)

DMA transfer buffer address need to align with the transfer width.

DMA\_CHANNEL\_GROUP(channel)

DMA\_CHANNEL\_INDEX(base, channel)

DMA\_COMMON\_REG\_GET(base, channel, reg)

DMA linked descriptor address algin size.

DMA\_COMMON\_CONST\_REG\_GET(base, channel, reg)

DMA\_COMMON\_REG\_SET(base, channel, reg, value)

DMA\_DESCRIPTOR\_END\_ADDRESS(start, inc, bytes, width)

DMA descriptor end address calculate.

#### Parameters

- start – start address
- inc – address interleave size
- bytes – transfer bytes
- width – transfer width

DMA\_CHANNEL\_XFER(reload, clrTrig, intA, intB, width, srcInc, dstInc, bytes)

struct \_dma\_descriptor

*#include <fsl\_dma.h>* DMA descriptor structure.

#### Public Members

volatile uint32\_t xfercfg

Transfer configuration

void \*srcEndAddr

Last source address of DMA transfer

```
void *dstEndAddr
    Last destination address of DMA transfer
void *linkToNextDesc
    Address of next DMA descriptor in chain
struct __dma_xfercfg
    #include <fsl_dma.h> DMA transfer configuration.
```

### Public Members

```
bool valid
    Descriptor is ready to transfer
bool reload
    Reload channel configuration register after current descriptor is exhausted
bool swtrig
    Perform software trigger. Transfer if fired when 'valid' is set
bool clrtrig
    Clear trigger
bool intA
    Raises IRQ when transfer is done and set IRQA status register flag
bool intB
    Raises IRQ when transfer is done and set IRQB status register flag
uint8_t byteWidth
    Byte width of data to transfer
uint8_t srcInc
    Increment source address by 'srcInc' x 'byteWidth'
uint8_t dstInc
    Increment destination address by 'dstInc' x 'byteWidth'
uint16_t transferCount
    Number of transfers
struct __dma_channel_trigger
    #include <fsl_dma.h> DMA channel trigger.
```

### Public Members

```
dma_trigger_type_t type
    Select hardware trigger as edge triggered or level triggered.
dma_trigger_burst_t burst
    Select whether hardware triggers cause a single or burst transfer.
dma_burst_wrap_t wrap
    Select wrap type, source wrap or dest wrap, or both.
struct __dma_channel_config
    #include <fsl_dma.h> DMA channel trigger.
```

**Public Members**

`void *srcStartAddr`  
Source data address

`void *dstStartAddr`  
Destination data address

`void *nextDesc`  
Chain custom descriptor

`uint32_t xferCfg`  
channel transfer configurations

`dma_channel_trigger_t *trigger`  
DMA trigger type

`bool isPeriph`  
select the request type

`struct _dma_transfer_config`  
*#include <fsl\_dma.h>* DMA transfer configuration.

**Public Members**

`uint8_t *srcAddr`  
Source data address

`uint8_t *dstAddr`  
Destination data address

`uint8_t *nextDesc`  
Chain custom descriptor

`dma_xfercfg_t xfercfg`  
Transfer options

`bool isPeriph`  
DMA transfer is driven by peripheral

`struct _dma_handle`  
*#include <fsl\_dma.h>* DMA transfer handle structure.

**Public Members**

`dma_callback` callback  
Callback function. Invoked when transfer of descriptor with interrupt flag finishes

`void *userData`  
Callback function parameter

`DMA_Type *base`  
DMA peripheral base address

`uint8_t channel`  
DMA channel number



## 2.6 DMIC: Digital Microphone

### 2.7 DMIC DMA Driver

```
status_t DMIC_TransferCreateHandleDMA(DMIC_Type *base, dmic_dma_handle_t *handle,
                                     dmic_dma_transfer_callback_t callback, void
                                     *userData, dma_handle_t *rxDmaHandle)
```

Initializes the DMIC handle which is used in transactional functions.

#### Parameters

- base – DMIC peripheral base address.
- handle – Pointer to *dmic\_dma\_handle\_t* structure.
- callback – Callback function.
- userData – User data.
- rxDmaHandle – User-requested DMA handle for RX DMA transfer.

```
status_t DMIC_TransferReceiveDMA(DMIC_Type *base, dmic_dma_handle_t *handle,
                                 dmic_transfer_t *xfer, uint32_t channel)
```

Receives data using DMA.

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

#### Parameters

- base – USART peripheral base address.
- handle – Pointer to *usart\_dma\_handle\_t* structure.
- xfer – DMIC DMA transfer structure. See *dmic\_transfer\_t*.
- channel – DMIC start channel number.

#### Return values

*kStatus\_Success* –

```
void DMIC_TransferAbortReceiveDMA(DMIC_Type *base, dmic_dma_handle_t *handle)
```

Aborts the received data using DMA.

This function aborts the received data using DMA.

#### Parameters

- base – DMIC peripheral base address
- handle – Pointer to *dmic\_dma\_handle\_t* structure

```
status_t DMIC_TransferGetReceiveCountDMA(DMIC_Type *base, dmic_dma_handle_t *handle,
                                           uint32_t *count)
```

Get the number of bytes that have been received.

This function gets the number of bytes that have been received.

#### Parameters

- base – DMIC peripheral base address.
- handle – DMIC handle pointer.
- count – Receive bytes count.

#### Return values

- *kStatus\_NoTransferInProgress* – No receive in progress.

- `kStatus_InvalidArgument` – Parameter is invalid.
- `kStatus_Success` – Get successfully through the parameter count;

```
void DMIC__InstallDMADescriptorMemory(dmic_dma_handle_t *handle, void *linkAddr, size_t linkNum)
```

Install DMA descriptor memory.

This function used to register DMA descriptor memory for linked transfer, a typical case is ping pong transfer which will request more than one DMA descriptor memory space, it should be called after `DMIC_TransferCreateHandleDMA`. User should be take care about the address of DMA descriptor pool which required align with 16BYTE at least.

#### Parameters

- `handle` – Pointer to DMA channel transfer handle.
- `linkAddr` – DMA link descriptor address.
- `linkNum` – DMA link descriptor number.

```
FSL_DMIC_DMA_DRIVER_VERSION
```

DMIC DMA driver version 2.4.2.

```
typedef struct _dmic_transfer dmic_transfer_t
```

DMIC transfer structure.

```
typedef struct _dmic_dma_handle dmic_dma_handle_t
```

```
typedef void (*dmic_dma_transfer_callback_t)(DMIC_Type *base, dmic_dma_handle_t *handle, status_t status, void *userData)
```

DMIC transfer callback function.

```
struct _dmic_transfer
```

`#include <fsl_dmic_dma.h>` DMIC transfer structure.

#### Public Members

```
void *data
```

The buffer of data to be transfer.

```
uint8_t dataWidth
```

DMIC support 16bit/32bit

```
size_t dataSize
```

The byte count to be transfer.

```
uint8_t dataAddrInterleaveSize
```

destination address interleave size

```
struct _dmic_transfer *linkTransfer
```

use to support link transfer

```
struct _dmic_dma_handle
```

`#include <fsl_dmic_dma.h>` DMIC DMA handle.

#### Public Members

```
DMIC_Type *base
```

DMIC peripheral base address.

*dma\_handle\_t* \*rxDmaHandle  
The DMA RX channel used.

*dmic\_dma\_transfer\_callback\_t* callback  
Callback function.

void \*userData  
DMIC callback function parameter.

size\_t transferSize  
Size of the data to receive.

volatile uint8\_t state  
Internal state of DMIC DMA transfer

uint32\_t channel  
DMIC channel used.

bool isChannelValid  
DMIC channel initialization flag

*dma\_descriptor\_t* \*desLink  
descriptor pool pointer

size\_t linkNum  
number of descriptor in descriptors pool

## 2.8 DMIC Driver

uint32\_t DMIC\_GetInstance(DMIC\_Type \*base)  
Get the DMIC instance from peripheral base address.

### Parameters

- base – DMIC peripheral base address.

### Returns

DMIC instance.

void DMIC\_Init(DMIC\_Type \*base)  
Turns DMIC Clock on.

### Parameters

- base – : DMIC base

### Returns

Nothing

void DMIC\_DeInit(DMIC\_Type \*base)  
Turns DMIC Clock off.

### Parameters

- base – : DMIC base

### Returns

Nothing

void DMIC\_ConfigIO(DMIC\_Type \*base, *dmic\_io\_t* config)  
Configure DMIC io.

**Deprecated:**

Do not use this function. It has been superceded by DMIC\_SetIOCFG

**Parameters**

- base – : The base address of DMIC interface
- config – : DMIC io configuration

**Returns**

Nothing

```
static inline void DMIC_SetIOCFG(DMIC_Type *base, uint32_t sel)
```

Stereo PDM select.

**Parameters**

- base – : The base address of DMIC interface
- sel – : Reference dmic\_io\_t, can be a single or combination value of dmic\_io\_t.

**Returns**

Nothing

```
void DMIC_SetOperationMode(DMIC_Type *base, operation_mode_t mode)
```

Set DMIC operating mode.

**Deprecated:**

Do not use this function. It has been superceded by DMIC\_EnableChannelInterrupt, DMIC\_EnableChannelDma.

**Parameters**

- base – : The base address of DMIC interface
- mode – : DMIC mode

**Returns**

Nothing

```
void DMIC_Use2fs(DMIC_Type *base, bool use2fs)
```

Configure Clock scaling.

**Parameters**

- base – : The base address of DMIC interface
- use2fs – : clock scaling

**Returns**

Nothing

```
void DMIC_CfgChannelDc(DMIC_Type *base, dmic_channel_t channel, dc_removal_t  
dc_cut_level, uint32_t post_dc_gain_reduce, bool saturate16bit)
```

Configure DMIC channel.

**Parameters**

- base – : The base address of DMIC interface
- channel – : DMIC channel
- dc\_cut\_level – : dc\_removal\_t, Cut off Frequency
- post\_dc\_gain\_reduce – : Fine gain adjustment in the form of a number of bits to downshift.

- saturate16bit – : If selects 16-bit saturation.

```
void DMIC_ConfigChannel(DMIC_Type *base, dmic_channel_t channel, stereo_side_t side,  
                       dmic_channel_config_t *channel_config)
```

Configure DMIC channel.

#### Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel
- side – : *stereo\_side\_t*, choice of left or right
- channel\_config – : Channel configuration

#### Returns

Nothing

```
void DMIC_EnableChannel(DMIC_Type *base, uint32_t channelmask)
```

Enable a particular channel.

#### Parameters

- base – : The base address of DMIC interface
- channelmask – reference *\_dmic\_channel\_mask*

#### Returns

Nothing

```
void DMIC_FifoChannel(DMIC_Type *base, uint32_t channel, uint32_t trig_level, uint32_t  
                     enable, uint32_t resetn)
```

Configure fifo settings for DMIC channel.

#### Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel
- trig\_level – : FIFO trigger level
- enable – : FIFO level
- resetn – : FIFO reset

#### Returns

Nothing

```
static inline void DMIC_EnableChannelInterrupt(DMIC_Type *base, dmic_channel_t channel,  
                                                bool enable)
```

Enable a particular channel interrupt request.

#### Parameters

- base – : The base address of DMIC interface
- channel – : Channel selection
- enable – : true is enable, false is disable

```
static inline void DMIC_EnableChannelDma(DMIC_Type *base, dmic_channel_t channel, bool  
                                         enable)
```

Enable a particular channel dma request.

#### Parameters

- base – : The base address of DMIC interface
- channel – : Channel selection

- enable – : true is enable, false is disable

```
static inline void DMIC__EnableChannelFifo(DMIC_Type *base, dmic_channel_t channel, bool enable)
```

Enable a particular channel fifo.

#### Parameters

- base – : The base address of DMIC interface
- channel – : Channel selection
- enable – : true is enable, false is disable

```
static inline void DMIC__DoFifoReset(DMIC_Type *base, dmic_channel_t channel)
```

Channel fifo reset.

#### Parameters

- base – : The base address of DMIC interface
- channel – : Channel selection

```
static inline uint32_t DMIC__FifoGetStatus(DMIC_Type *base, uint32_t channel)
```

Get FIFO status.

#### Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel

#### Returns

FIFO status

```
static inline void DMIC__FifoClearStatus(DMIC_Type *base, uint32_t channel, uint32_t mask)
```

Clear FIFO status.

#### Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel
- mask – : Bits to be cleared

#### Returns

FIFO status

```
static inline uint32_t DMIC__FifoGetData(DMIC_Type *base, uint32_t channel)
```

Get FIFO data.

#### Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel

#### Returns

FIFO data

```
static inline uint32_t DMIC__FifoGetAddress(DMIC_Type *base, uint32_t channel)
```

Get FIFO address.

#### Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel

#### Returns

FIFO data

```
void DMIC_EnableIntCallback(DMIC_Type *base, dmic_callback_t cb)
```

Enable callback.

This function enables the interrupt for the selected DMIC peripheral. The callback function is not enabled until this function is called.

**Parameters**

- *base* – Base address of the DMIC peripheral.
- *cb* – callback Pointer to store callback function.

**Return values**

None. –

```
void DMIC_DisableIntCallback(DMIC_Type *base, dmic_callback_t cb)
```

Disable callback.

This function disables the interrupt for the selected DMIC peripheral.

**Parameters**

- *base* – Base address of the DMIC peripheral.
- *cb* – callback Pointer to store callback function..

**Return values**

None. –

```
static inline void DMIC_SetGainNoiseEstHwvad(DMIC_Type *base, uint32_t value)
```

Sets the gain value for the noise estimator.

**Parameters**

- *base* – DMIC base pointer
- *value* – gain value for the noise estimator.

**Return values**

None. –

```
static inline void DMIC_SetGainSignalEstHwvad(DMIC_Type *base, uint32_t value)
```

Sets the gain value for the signal estimator.

**Parameters**

- *base* – DMIC base pointer
- *value* – gain value for the signal estimator.

**Return values**

None. –

```
static inline void DMIC_SetFilterCtrlHwvad(DMIC_Type *base, uint32_t value)
```

Sets the hwvad filter cutoff frequency parameter.

**Parameters**

- *base* – DMIC base pointer
- *value* – cut off frequency value.

**Return values**

None. –

```
static inline void DMIC_SetInputGainHwvad(DMIC_Type *base, uint32_t value)
```

Sets the input gain of hwvad.

**Parameters**

- *base* – DMIC base pointer

- value – input gain value for hwvad.

**Return values**

None. –

static inline void DMIC\_CtrlClrIntrHwvad(DMIC\_Type \*base, bool st10)

Clears hwvad internal interrupt flag.

**Parameters**

- base – DMIC base pointer
- st10 – bit value.

**Return values**

None. –

static inline void DMIC\_FilterResetHwvad(DMIC\_Type \*base, bool rstt)

Resets hwvad filters.

**Parameters**

- base – DMIC base pointer
- rstt – Reset bit value.

**Return values**

None. –

static inline uint16\_t DMIC\_GetNoiseEnvlpEst(DMIC\_Type \*base)

Gets the value from output of the filter z7.

**Parameters**

- base – DMIC base pointer

**Return values**

output – of filter z7.

void DMIC\_HwvadEnableIntCallback(DMIC\_Type \*base, *dmic\_hwvad\_callback\_t* vadcb)

Enable hwvad callback.

This function enables the hwvad interrupt for the selected DMIC peripheral. The callback function is not enabled until this function is called.

**Parameters**

- base – Base address of the DMIC peripheral.
- vadcb – callback Pointer to store callback function.

**Return values**

None. –

void DMIC\_HwvadDisableIntCallback(DMIC\_Type \*base, *dmic\_hwvad\_callback\_t* vadcb)

Disable callback.

This function disables the hwvad interrupt for the selected DMIC peripheral.

**Parameters**

- base – Base address of the DMIC peripheral.
- vadcb – callback Pointer to store callback function..

**Return values**

None. –

FSL\_DMIC\_DRIVER\_VERSION

DMIC driver version 2.3.3.



\_dmic\_status DMIC transfer status.

*Values:*

enumerator kStatus\_DMIC\_Busy

DMIC is busy

enumerator kStatus\_DMIC\_Idle

DMIC is idle

enumerator kStatus\_DMIC\_OverRunError

DMIC over run Error

enumerator kStatus\_DMIC\_UnderRunError

DMIC under run Error

enum \_operation\_mode

DMIC different operation modes.

*Values:*

enumerator kDMIC\_OperationModeInterrupt

Interrupt mode

enumerator kDMIC\_OperationModeDma

DMA mode

enum \_stereo\_side

DMIC left/right values.

*Values:*

enumerator kDMIC\_Left

Left Stereo channel

enumerator kDMIC\_Right

Right Stereo channel

enum pdm\_div\_t

DMIC Clock pre-divider values.

*Values:*

enumerator kDMIC\_PdmDiv1

DMIC pre-divider set in divide by 1

enumerator kDMIC\_PdmDiv2

DMIC pre-divider set in divide by 2

enumerator kDMIC\_PdmDiv3

DMIC pre-divider set in divide by 3

enumerator kDMIC\_PdmDiv4

DMIC pre-divider set in divide by 4

enumerator kDMIC\_PdmDiv6

DMIC pre-divider set in divide by 6

enumerator kDMIC\_PdmDiv8

DMIC pre-divider set in divide by 8

enumerator kDMIC\_PdmDiv12

DMIC pre-divider set in divide by 12

enumerator kDMIC\_PdmDiv16

DMIC pre-divider set in divide by 16

enumerator kDMIC\_PdmDiv24

DMIC pre-divider set in divide by 24

enumerator kDMIC\_PdmDiv32

DMIC pre-divider set in divide by 32

enumerator kDMIC\_PdmDiv48

DMIC pre-divider set in divide by 48

enumerator kDMIC\_PdmDiv64

DMIC pre-divider set in divide by 64

enumerator kDMIC\_PdmDiv96

DMIC pre-divider set in divide by 96

enumerator kDMIC\_PdmDiv128

DMIC pre-divider set in divide by 128

enum \_compensation

Pre-emphasis Filter coefficient value for 2FS and 4FS modes.

*Values:*

enumerator kDMIC\_CompValueZero

Compensation 0

enumerator kDMIC\_CompValueNegativePoint16

Compensation -0.16

enumerator kDMIC\_CompValueNegativePoint15

Compensation -0.15

enumerator kDMIC\_CompValueNegativePoint13

Compensation -0.13

enum \_dc\_removal

DMIC DC filter control values.

*Values:*

enumerator kDMIC\_DcNoRemove

Flat response no filter

enumerator kDMIC\_DcCut155

Cut off Frequency is 155 Hz

enumerator kDMIC\_DcCut78

Cut off Frequency is 78 Hz

enumerator kDMIC\_DcCut39

Cut off Frequency is 39 Hz

enum \_dmic\_io

DMIC IO configuration.

*Values:*

enumerator kDMIC\_PdmDual

Two separate pairs of PDM wires

enumerator kDMIC\_PdmStereo  
Stereo data0

enumerator kDMIC\_PdmBypass  
Clk Bypass clocks both channels

enumerator kDMIC\_PdmBypassClk0  
Clk Bypass clocks only channel0

enumerator kDMIC\_PdmBypassClk1  
Clk Bypass clocks only channel1

enum \_dmic\_channel  
DMIC Channel number.  
*Values:*

enumerator kDMIC\_Channel0  
DMIC channel 0

enumerator kDMIC\_Channel1  
DMIC channel 1

enumerator kDMIC\_ChannelMAX  
Maximum number of DMIC channels

\_dmic\_channel\_mask DMIC Channel mask.  
*Values:*

enumerator kDMIC\_EnableChannel0  
DMIC channel 0 mask

enumerator kDMIC\_EnableChannel1  
DMIC channel 1 mask

enum \_dmic\_phy\_sample\_rate  
DMIC and decimator sample rates.  
*Values:*

enumerator kDMIC\_PhyFullSpeed  
Decimator gets one sample per each chosen clock edge of PDM interface

enumerator kDMIC\_PhyHalfSpeed  
PDM clock to Microphone is halved, decimator receives each sample twice

typedef enum *\_operation\_mode* operation\_mode\_t  
DMIC different operation modes.

typedef enum *\_stereo\_side* stereo\_side\_t  
DMIC left/right values.

typedef enum *\_compensation* compensation\_t  
Pre-emphasis Filter coefficient value for 2FS and 4FS modes.

typedef enum *\_dc\_removal* dc\_removal\_t  
DMIC DC filter control values.

typedef enum *\_dmic\_io* dmic\_io\_t  
DMIC IO configuration.

```
typedef enum _dmic_channel dmic_channel_t
    DMIC Channel number.
typedef enum _dmic_phy_sample_rate dmic_phy_sample_rate_t
    DMIC and decimator sample rates.
typedef struct _dmic_channel_config dmic_channel_config_t
    DMIC Channel configuration structure.
typedef void (*dmic_callback_t)(void)
    DMIC Callback function.
typedef void (*dmic_hwvad_callback_t)(void)
    HWVAD Callback function.
struct _dmic_channel_config
    #include <fsl_dmic.h> DMIC Channel configuration structure.
```

### Public Members

```
pdm_div_t divhfelk
    DMIC Clock pre-divider values
uint32_t osr
    oversampling rate(CIC decimation rate) for PCM
uint32_t gainshift
    4FS PCM data gain control
compensation_t preac2coef
    Pre-emphasis Filter coefficient value for 2FS
compensation_t preac4coef
    Pre-emphasis Filter coefficient value for 4FS
dc_removal_t dc_cut_level
    DMIC DC filter control values.
uint32_t post_dc_gain_reduce
    Fine gain adjustment in the form of a number of bits to downshift
dmic_phy_sample_rate_t sample_rate
    DMIC and decimator sample rates
bool saturate16bit
    Selects 16-bit saturation. 0 means results roll over if out range and do not saturate. 1
    means if the result overflows, it saturates at 0xFFFF for positive overflow and 0x8000
    for negative overflow.
```

## 2.9 EMC: External Memory Controller Driver

```
void EMC_Init(EMC_Type *base, emc_basic_config_t *config)
```

Initializes the basic for EMC. This function ungates the EMC clock, initializes the emc system configure and enable the EMC module. This function must be called in the first step to initialize the external memory.

### Parameters

- base – EMC peripheral base address.

- config – The EMC basic configuration.

```
void EMC_DynamicMemInit(EMC_Type *base, emc_dynamic_timing_config_t *timing,  
                        emc_dynamic_chip_config_t *config, uint32_t totalChips)
```

Initializes the dynamic memory controller. This function initializes the dynamic memory controller in external memory controller. This function must be called after EMC\_Init and before accessing the external dynamic memory.

#### Parameters

- base – EMC peripheral base address.
- timing – The timing and latency for dynamica memory controller setting. It shall be used for all dynamica memory chips, threfore the worst timing value for all used chips must be given.
- config – The EMC dynamic memory controller chip independent configuration pointer. This configuration pointer is actually pointer to a configuration array. the array number depends on the “totalChips”.
- totalChips – The total dynamic memory chip numbers been used or the length of the “emc\_dynamic\_chip\_config\_t” type memory.

```
void EMC_StaticMemInit(EMC_Type *base, uint32_t *extWait_Ns, emc_static_chip_config_t  
                      *config, uint32_t totalChips)
```

Initializes the static memory controller. This function initializes the static memory controller in external memory controller. This function must be called after EMC\_Init and before accessing the external static memory.

#### Parameters

- base – EMC peripheral base address.
- extWait\_Ns – The extended wait timeout or the read/write transfer time. This is common for all static memory chips and set with NULL if not required.
- config – The EMC static memory controller chip independent configuration pointer. This configuration pointer is actually pointer to a configuration array. the array number depends on the “totalChips”.
- totalChips – The total static memory chip numbers been used or the length of the “emc\_static\_chip\_config\_t” type memory.

```
void EMC_Deinit(EMC_Type *base)
```

Deinitializes the EMC module and gates the clock. This function gates the EMC controller clock. As a result, the EMC module doesn’t work after calling this function.

#### Parameters

- base – EMC peripheral base address.

```
static inline void EMC_Enable(EMC_Type *base, bool enable)
```

Enables/disables the EMC module.

#### Parameters

- base – EMC peripheral base address.
- enable – True enable EMC module, false disable.

```
static inline void EMC_EnableDynamicMemControl(EMC_Type *base, bool enable)
```

Enables/disables the EMC Dynaimc memory controller.

#### Parameters

- base – EMC peripheral base address.

- enable – True enable EMC dynamic memory controller, false disable.

static inline void EMC\_MirrorChipAddr(EMC\_Type \*base, bool enable)

Enables/disables the EMC address mirror. Enable the address mirror the EMC\_CS1 is mirrored to both EMC\_CS0 and EMC\_DYCS0 memory areas. Disable the address mirror enables EMC\_CS0 and EMC\_DYCS0 memory to be accessed.

#### Parameters

- base – EMC peripheral base address.
- enable – True enable the address mirror, false disable the address mirror.

static inline void EMC\_EnterSelfRefreshCommand(EMC\_Type \*base, bool enable)

Enter the self-refresh mode for dynamic memory controller. This function provided self-refresh mode enter or exit for application.

#### Parameters

- base – EMC peripheral base address.
- enable – True enter the self-refresh mode, false to exit self-refresh and enter the normal mode.

static inline bool EMC\_IsInSelfrefreshMode(EMC\_Type \*base)

Get the operating mode of the EMC. This function can be used to get the operating mode of the EMC.

#### Parameters

- base – EMC peripheral base address.

#### Returns

The EMC in self-refresh mode if true, else in normal mode.

static inline void EMC\_EnterLowPowerMode(EMC\_Type \*base, bool enable)

Enter/exit the low-power mode.

#### Parameters

- base – EMC peripheral base address.
- enable – True Enter the low-power mode, false exit low-power mode and return to normal mode.

FSL\_EMC\_DRIVER\_VERSION

EMC driver version.

enum \_\_emc\_static\_memwidth

Define EMC memory width for static memory device.

*Values:*

enumerator kEMC\_8BitWidth

8 bit memory width.

enumerator kEMC\_16BitWidth

16 bit memory width.

enumerator kEMC\_32BitWidth

32 bit memory width.

enum \_\_emc\_static\_special\_config

Define EMC static configuration.

*Values:*

enumerator kEMC\_\_AsynchronousPageEnable  
Enable the asynchronous page mode. page length four.

enumerator kEMC\_\_ActiveHighChipSelect  
Chip select active high.

enumerator kEMC\_\_ByteLaneStateAllLow  
Reads/writes the respective value bits in BLS3:0 are low.

enumerator kEMC\_\_ExtWaitEnable  
Extended wait enable.

enumerator kEMC\_\_BufferEnable  
Buffer enable.

enum \_\_emc\_\_dynamic\_\_device  
EMC dynamic memory device.

*Values:*

enumerator kEMC\_\_Sdram  
Dynamic memory device: SDRAM.

enumerator kEMC\_\_Lpsdram  
Dynamic memory device: Low-power SDRAM.

enum \_\_emc\_\_dynamic\_\_read  
EMC dynamic read strategy.

*Values:*

enumerator kEMC\_\_NoDelay  
No delay.

enumerator kEMC\_\_Cmddelay  
Command delayed strategy, using EMCCLKDELAY.

enumerator kEMC\_\_CmdDelayPulseOneclk  
Command delayed strategy pulse one clock cycle using EMCCLKDELAY.

enumerator kEMC\_\_CmddelayPulsetwoclk  
Command delayed strategy pulse two clock cycle using EMCCLKDELAY.

enum \_\_emc\_\_endian\_\_mode  
EMC endian mode.

*Values:*

enumerator kEMC\_\_LittleEndian  
Little endian mode.

enumerator kEMC\_\_BigEndian  
Big endian mode.

enum \_\_emc\_\_fbclk\_\_src  
EMC Feedback clock input source select.

*Values:*

enumerator kEMC\_\_IntloopbackEmcclk  
Use the internal loop back from EMC\_CLK output.

enumerator kEMC\_\_EMCFbclkInput  
Use the external EMC\_FBCLK input.

typedef enum *\_emc\_static\_memwidth* emc\_static\_memwidth\_t

Define EMC memory width for static memory device.

typedef enum *\_emc\_static\_special\_config* emc\_static\_special\_config\_t

Define EMC static configuration.

typedef enum *\_emc\_dynamic\_device* emc\_dynamic\_device\_t

EMC dynamic memory device.

typedef enum *\_emc\_dynamic\_read* emc\_dynamic\_read\_t

EMC dynamic read strategy.

typedef enum *\_emc\_endian\_mode* emc\_endian\_mode\_t

EMC endian mode.

typedef enum *\_emc\_fbclk\_src* emc\_fbclk\_src\_t

EMC Feedback clock input source select.

typedef struct *\_emc\_dynamic\_timing\_config* emc\_dynamic\_timing\_config\_t

EMC dynamic timing/delay configure structure.

typedef struct *\_emc\_dynamic\_chip\_config* emc\_dynamic\_chip\_config\_t

EMC dynamic memory controller independent chip configuration structure. Please take refer to the address mapping table in the RM in EMC chapter when you set the “devAddrMap”. Choose the right Bit 14 Bit12 ~ Bit 7 group in the table according to the bus width/banks/row/column length for you device. Set devAddrMap with the value make up with the seven bits (bit14 bit12 ~ bit 7) and inset the bit 13 with 0. for example, if the bit 14 and bit12 ~ bit7 is 1000001 is choosen according to the 32bit high-performance bus width with 2 banks, 11 row lwngh, 8 column length. Set devAddrMap with 0x81.

typedef struct *\_emc\_static\_chip\_config* emc\_static\_chip\_config\_t

EMC static memory controller independent chip configuration structure.

typedef struct *\_emc\_basic\_config* emc\_basic\_config\_t

EMC module basic configuration structure.

Defines the static memory controller configure structure and uses the EMC\_Init() function to make necessary initializations.

EMC\_STATIC\_MEMDEV\_NUM

Define the chip numbers for dynamic and static memory devices.

EMC\_DYNAMIC\_MEMDEV\_NUM

EMC\_ADDRMAP\_SHIFT

EMC\_ADDRMAP\_MASK

EMC\_ADDRMAP(x)

EMC\_HZ\_ONEMHZ

EMC\_MILLISECS\_ONESEC

EMC\_SDRAM\_MODE\_CL\_SHIFT

EMC\_SDRAM\_MODE\_CL\_MASK

EMC\_SDRAM\_NOP\_DELAY\_US

EDMA\_SDRAM NOP command wait us.

EMC\_SDRAM\_PRECHARGE\_DELAY\_US

EDMA\_SDRAM precharge command wait us.



EMC\_SDRAM\_AUTO\_REFRESH\_DELAY\_US

EDMA\_SDRAM auto refresh wait us.

struct \_emc\_dynamic\_timing\_config

#include <fsl\_emc.h> EMC dynamic timing/delay configure structure.

### Public Members

uint32\_t refreshPeriod\_Nanosec

The refresh period in unit of nanosecond.

uint32\_t tRp\_Ns

Precharge command period in unit of nanosecond.

uint32\_t tRas\_Ns

Active to precharge command period in unit of nanosecond.

uint32\_t tSrex\_Ns

Self-refresh exit time in unit of nanosecond.

uint32\_t tApr\_Ns

Last data out to active command time in unit of nanosecond.

uint32\_t tDal\_Ns

Data-in to active command in unit of nanosecond.

uint32\_t tWr\_Ns

Write recovery time in unit of nanosecond.

uint32\_t tRc\_Ns

Active to active command period in unit of nanosecond.

uint32\_t tRfc\_Ns

Auto-refresh period and auto-refresh to active command period in unit of nanosecond.

uint32\_t tXsr\_Ns

Exit self-refresh to active command time in unit of nanosecond.

uint32\_t tRrd\_Ns

Active bank A to active bank B latency in unit of nanosecond.

uint8\_t tMrd\_Nclk

Load mode register to active command time in unit of EMCCLK cycles.

struct \_emc\_dynamic\_chip\_config

#include <fsl\_emc.h> EMC dynamic memory controller independent chip configuration structure. Please take refer to the address mapping table in the RM in EMC chapter when you set the “devAddrMap”. Choose the right Bit 14 Bit12 ~ Bit 7 group in the table according to the bus width/banks/row/column length for you device. Set devAddrMap with the value make up with the seven bits (bit14 bit12 ~ bit 7) and inset the bit 13 with 0. for example, if the bit 14 and bit12 ~ bit7 is 1000001 is choosen according to the 32bit high-performance bus width with 2 banks, 11 row lwngh, 8 column length. Set devAddrMap with 0x81.

### Public Members

uint8\_t chipIndex

Chip Index, range from 0 ~ EMC\_DYNAMIC\_MEMDEV\_NUM - 1.

emc\_dynamic\_device\_t dynamicDevice

All chips shall use the same device setting. mixed use are not supported.

uint8\_t rAS\_Nclk

Active to read/write delay tRCD.

uint16\_t sdramModeReg

Sdram mode register setting.

uint16\_t sdramExtModeReg

Used for low-power sdram device. The extended mode register.

uint8\_t devAddrMap

dynamic device address mapping, choose the address mapping for your specific device.

struct \_\_emc\_static\_chip\_config

#include <fsl\_emc.h> EMC static memory controller independent chip configuration structure.

### Public Members

emc\_static\_memwidth\_t memWidth

Memory width.

uint32\_t specailConfig

Static configuration,a logical OR of “emc\_static\_special\_config\_t”.

uint32\_t tWaitWriteEn\_Ns

The delay form chip select to write enable in unit of nanosecond.

uint32\_t tWaitOutEn\_Ns

The delay from chip selcet to output enable in unit of nanosecond.

uint32\_t tWaitReadNoPage\_Ns

In No-page mode, the delay from chip select to read access in unit of nanosecond.

uint32\_t tWaitReadPage\_Ns

In page mode, the read after the first read wait states in unit of nanosecond.

uint32\_t tWaitWrite\_Ns

The delay from chip select to write access in unit of nanosecond.

uint32\_t tWaitTurn\_Ns

The Bus turn-around time in unit of nanosecond.

struct \_\_emc\_basic\_config

#include <fsl\_emc.h> EMC module basic configuration structure.

Defines the static memory controller configure structure and uses the EMC\_Init() function to make necessary initializations.

### Public Members

emc\_endian\_mode\_t endian

Endian mode .

emc\_fbclk\_src\_t fbClkSrc

The feedback clock source.

uint8\_t emcClkDiv

EMC\_CLK = AHB\_CLK / (emc\_clkDiv + 1).

## 2.10 FLEXCOMM: FLEXCOMM Driver

### 2.11 FLEXCOMM Driver

FSL\_FLEXCOMM\_DRIVER\_VERSION

FlexCOMM driver version 2.0.2.

enum FLEXCOMM\_PERIPH\_T

FLEXCOMM peripheral modes.

*Values:*

enumerator FLEXCOMM\_PERIPH\_NONE

No peripheral

enumerator FLEXCOMM\_PERIPH\_USART

USART peripheral

enumerator FLEXCOMM\_PERIPH\_SPI

SPI Peripheral

enumerator FLEXCOMM\_PERIPH\_I2C

I2C Peripheral

enumerator FLEXCOMM\_PERIPH\_I2S\_TX

I2S TX Peripheral

enumerator FLEXCOMM\_PERIPH\_I2S\_RX

I2S RX Peripheral

typedef void (\*flexcomm\_irq\_handler\_t)(void \*base, void \*handle)

Typedef for interrupt handler.

IRQn\_Type const kFlexcommIrqs[]

Array with IRQ number for each FLEXCOMM module.

uint32\_t FLEXCOMM\_GetInstance(void \*base)

Returns instance number for FLEXCOMM module with given base address.

status\_t FLEXCOMM\_Init(void \*base, FLEXCOMM\_PERIPH\_T periph)

Initializes FLEXCOMM and selects peripheral mode according to the second parameter.

void FLEXCOMM\_SetIRQHandler(void \*base, flexcomm\_irq\_handler\_t handler, void \*flexcommHandle)

Sets IRQ handler for given FLEXCOMM module. It is used by drivers register IRQ handler according to FLEXCOMM mode.

## 2.12 FMEAS: Frequency Measure Driver

static inline void FMEAS\_StartMeasure(FMEAS\_SYSCON\_Type \*base)

Starts a frequency measurement cycle.

### Parameters

- base – : SYSCON peripheral base address.

static inline bool FMEAS\_IsMeasureComplete(*FMEAS\_SYSCON\_Type* \*base)

Indicates when a frequency measurement cycle is complete.

**Parameters**

- base – : SYSCON peripheral base address.

**Returns**

true if a measurement cycle is active, otherwise false.

uint32\_t FMEAS\_GetFrequency(*FMEAS\_SYSCON\_Type* \*base, uint32\_t refClockRate)

Returns the computed value for a frequency measurement cycle.

**Parameters**

- base – : SYSCON peripheral base address.
- refClockRate – : Reference clock rate used during the frequency measurement cycle.

**Returns**

Frequency in Hz.

FSL\_FMEAS\_DRIVER\_VERSION

Defines LPC Frequency Measure driver version 2.1.1.

typedef SYSCON\_Type FMEAS\_SYSCON\_Type

FMEAS\_SYSCON\_FREQMECTRL\_CAPVAL\_MASK

FMEAS\_SYSCON\_FREQMECTRL\_CAPVAL\_SHIFT

FMEAS\_SYSCON\_FREQMECTRL\_CAPVAL

FMEAS\_SYSCON\_FREQMECTRL\_PROG\_MASK

FMEAS\_SYSCON\_FREQMECTRL\_PROG\_SHIFT

FMEAS\_SYSCON\_FREQMECTRL\_PROG

## 2.13 GINT: Group GPIO Input Interrupt Driver

FSL\_GINT\_DRIVER\_VERSION

Driver version.

enum \_gint\_comb

GINT combine inputs type.

*Values:*

enumerator kGINT\_CombineOr

A grouped interrupt is generated when any one of the enabled inputs is active

enumerator kGINT\_CombineAnd

A grouped interrupt is generated when all enabled inputs are active

enum \_gint\_trig

GINT trigger type.

*Values:*

enumerator kGINT\_TrigEdge

Edge triggered based on polarity

enumerator kGINT\_TrigLevel  
Level triggered based on polarity

enum \_\_gint\_port

Values:

enumerator kGINT\_Port0

enumerator kGINT\_Port1

typedef enum *\_\_gint\_comb* gint\_comb\_t  
GINT combine inputs type.

typedef enum *\_\_gint\_trig* gint\_trig\_t  
GINT trigger type.

typedef enum *\_\_gint\_port* gint\_port\_t

typedef void (\*gint\_cb\_t)(void)  
GINT Callback function.

void GINT\_Init(GINT\_Type \*base)  
Initialize GINT peripheral.

This function initializes the GINT peripheral and enables the clock.

#### Parameters

- base – Base address of the GINT peripheral.

#### Return values

None. –

void GINT\_SetCtrl(GINT\_Type \*base, *gint\_comb\_t* comb, *gint\_trig\_t* trig, *gint\_cb\_t* callback)  
Setup GINT peripheral control parameters.

This function sets the control parameters of GINT peripheral.

#### Parameters

- base – Base address of the GINT peripheral.
- comb – Controls if the enabled inputs are logically ORed or ANDed for interrupt generation.
- trig – Controls if the enabled inputs are level or edge sensitive based on polarity.
- callback – This function is called when configured group interrupt is generated.

#### Return values

None. –

void GINT\_GetCtrl(GINT\_Type \*base, *gint\_comb\_t* \*comb, *gint\_trig\_t* \*trig, *gint\_cb\_t* \*callback)  
Get GINT peripheral control parameters.

This function returns the control parameters of GINT peripheral.

#### Parameters

- base – Base address of the GINT peripheral.
- comb – Pointer to store combine input value.
- trig – Pointer to store trigger value.
- callback – Pointer to store callback function.

**Return values**

None. –

```
void GINT__ConfigPins(GINT_Type *base, gint_port_t port, uint32_t polarityMask, uint32_t enableMask)
```

Configure GINT peripheral pins.

This function enables and controls the polarity of enabled pin(s) of a given port.

**Parameters**

- base – Base address of the GINT peripheral.
- port – Port number.
- polarityMask – Each bit position selects the polarity of the corresponding enabled pin. 0 = The pin is active LOW. 1 = The pin is active HIGH.
- enableMask – Each bit position selects if the corresponding pin is enabled or not. 0 = The pin is disabled. 1 = The pin is enabled.

**Return values**

None. –

```
void GINT__GetConfigPins(GINT_Type *base, gint_port_t port, uint32_t *polarityMask, uint32_t *enableMask)
```

Get GINT peripheral pin configuration.

This function returns the pin configuration of a given port.

**Parameters**

- base – Base address of the GINT peripheral.
- port – Port number.
- polarityMask – Pointer to store the polarity mask Each bit position indicates the polarity of the corresponding enabled pin. 0 = The pin is active LOW. 1 = The pin is active HIGH.
- enableMask – Pointer to store the enable mask. Each bit position indicates if the corresponding pin is enabled or not. 0 = The pin is disabled. 1 = The pin is enabled.

**Return values**

None. –

```
void GINT__EnableCallback(GINT_Type *base)
```

Enable callback.

This function enables the interrupt for the selected GINT peripheral. Although the pin(s) are monitored as soon as they are enabled, the callback function is not enabled until this function is called.

**Parameters**

- base – Base address of the GINT peripheral.

**Return values**

None. –

```
void GINT__DisableCallback(GINT_Type *base)
```

Disable callback.

This function disables the interrupt for the selected GINT peripheral. Although the pins are still being monitored but the callback function is not called.

**Parameters**

- base – Base address of the peripheral.

**Return values**

None. –

```
static inline void GINT_ClrStatus(GINT_Type *base)
```

Clear GINT status.

This function clears the GINT status bit.

**Parameters**

- base – Base address of the GINT peripheral.

**Return values**

None. –

```
static inline uint32_t GINT_GetStatus(GINT_Type *base)
```

Get GINT status.

This function returns the GINT status.

**Parameters**

- base – Base address of the GINT peripheral.

**Return values**

status – = 0 No group interrupt request. = 1 Group interrupt request active.

```
void GINT_Deinit(GINT_Type *base)
```

Deinitialize GINT peripheral.

This function disables the GINT clock.

**Parameters**

- base – Base address of the GINT peripheral.

**Return values**

None. –

## 2.14 I2C: Inter-Integrated Circuit Driver

### 2.15 I2C DMA Driver

```
void I2C_MasterTransferCreateHandleDMA(I2C_Type *base, i2c_master_dma_handle_t *handle,  
                                       i2c_master_dma_transfer_callback_t callback, void  
                                       *userData, dma_handle_t *dmaHandle)
```

Init the I2C handle which is used in transactional functions.

**Parameters**

- base – I2C peripheral base address
- handle – pointer to i2c\_master\_dma\_handle\_t structure
- callback – pointer to user callback function
- userData – user param passed to the callback function
- dmaHandle – DMA handle pointer

```
status_t I2C_MasterTransferDMA(I2C_Type *base, i2c_master_dma_handle_t *handle,  
                               i2c_master_transfer_t *xfer)
```

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

#### Parameters

- base – I2C peripheral base address
- handle – pointer to i2c\_master\_dma\_handle\_t structure
- xfer – pointer to transfer structure of i2c\_master\_transfer\_t

#### Return values

- kStatus\_Success – 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.

```
status_t I2C_MasterTransferGetCountDMA(I2C_Type *base, i2c_master_dma_handle_t *handle,  
                                       size_t *count)
```

Get master transfer status during a dma non-blocking transfer.

#### Parameters

- base – I2C peripheral base address
- handle – pointer to i2c\_master\_dma\_handle\_t structure
- count – Number of bytes transferred so far by the non-blocking transaction.

```
void I2C_MasterTransferAbortDMA(I2C_Type *base, i2c_master_dma_handle_t *handle)
```

Abort a master dma non-blocking transfer in a early time.

#### Parameters

- base – I2C peripheral base address
- handle – pointer to i2c\_master\_dma\_handle\_t structure

```
FSL_I2C_DMA_DRIVER_VERSION
```

I2C DMA driver version.

```
typedef struct _i2c_master_dma_handle i2c_master_dma_handle_t
```

I2C master dma handle typedef.

```
typedef void (*i2c_master_dma_transfer_callback_t)(I2C_Type *base, i2c_master_dma_handle_t  
*handle, status_t status, void *userData)
```

I2C master dma transfer callback typedef.

```
typedef void (*flexcomm_i2c_dma_master_irq_handler_t)(I2C_Type *base,  
i2c_master_dma_handle_t *handle)
```

Typedef for master dma handler.

```
I2C_MAX_DMA_TRANSFER_COUNT
```

Maximum length of single DMA transfer (determined by capability of the DMA engine)

```
struct _i2c_master_dma_handle
```

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



### Public Members

uint8\_t state

Transfer state machine current state.

uint32\_t transferCount

Indicates progress of the transfer

uint32\_t remainingBytesDMA

Remaining byte count to be transferred using DMA.

uint8\_t \*buf

Buffer pointer for current state.

bool checkAddrNack

Whether to check the nack signal is detected during addressing.

*dma\_handle\_t* \*dmaHandle

The DMA handler used.

*i2c\_master\_transfer\_t* transfer

Copy of the current transfer info.

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

Callback function called after dma transfer finished.

void \*userData

Callback parameter passed to callback function.

## 2.16 I2C Driver

FSL\_I2C\_DRIVER\_VERSION

I2C driver version.

I2C status return codes.

*Values:*

enumerator kStatus\_I2C\_Busy

The master is already performing a transfer.

enumerator kStatus\_I2C\_Idle

The slave driver is idle.

enumerator kStatus\_I2C\_Nak

The slave device sent a NAK in response to a byte.

enumerator kStatus\_I2C\_InvalidParameter

Unable to proceed due to invalid parameter.

enumerator kStatus\_I2C\_BitError

Transferred bit was not seen on the bus.

enumerator kStatus\_I2C\_ArbitrationLost

Arbitration lost error.

enumerator kStatus\_I2C\_NoTransferInProgress

Attempt to abort a transfer when one is not in progress.

enumerator kStatus\_I2C\_DmaRequestFail

DMA request failed.

enumerator kStatus\_I2C\_StartStopError

Start and stop error.

enumerator kStatus\_I2C\_UnexpectedState

Unexpected state.

enumerator kStatus\_I2C\_Timeout

Timeout when waiting for I2C master/slave pending status to set to continue transfer.

enumerator kStatus\_I2C\_Addr\_Nak

NAK received for Address

enumerator kStatus\_I2C\_EventTimeout

Timeout waiting for bus event.

enumerator kStatus\_I2C\_SclLowTimeout

Timeout SCL signal remains low.

enum \_i2c\_status\_flags

I2C status flags.

---

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

---

*Values:*

enumerator kI2C\_MasterPendingFlag

The I2C module is waiting for software interaction. bit 0

enumerator kI2C\_MasterArbitrationLostFlag

The arbitration of the bus was lost. There was collision on the bus. bit 4

enumerator kI2C\_MasterStartStopErrorFlag

There was an error during start or stop phase of the transaction. bit 6

enumerator kI2C\_MasterIdleFlag

The I2C master idle status. bit 5

enumerator kI2C\_MasterRxReadyFlag

The I2C master rx ready status. bit 1

enumerator kI2C\_MasterTxReadyFlag

The I2C master tx ready status. bit 2

enumerator kI2C\_MasterAddrNackFlag

The I2C master address nack status. bit 7

enumerator kI2C\_MasterDataNackFlag

The I2C master data nack status. bit 3

enumerator kI2C\_SlavePendingFlag

The I2C module is waiting for software interaction. bit 8

enumerator kI2C\_SlaveNotStretching

Indicates whether the slave is currently stretching clock (0 = yes, 1 = no). bit 11

enumerator kI2C\_SlaveSelected

Indicates whether the slave is selected by an address match. bit 14

enumerator kI2C\_SaveDeselected

Indicates that slave was previously deselected (deselect event took place, w1c). bit 15

enumerator kI2C\_SlaveAddressedFlag

One of the I2C slave's 4 addresses is matched. bit 22

enumerator kI2C\_SlaveReceiveFlag

Slave receive data available. bit 9

enumerator kI2C\_SlaveTransmitFlag

Slave data can be transmitted. bit 10

enumerator kI2C\_SlaveAddress0MatchFlag

Slave address0 match. bit 20

enumerator kI2C\_SlaveAddress1MatchFlag

Slave address1 match. bit 12

enumerator kI2C\_SlaveAddress2MatchFlag

Slave address2 match. bit 13

enumerator kI2C\_SlaveAddress3MatchFlag

Slave address3 match. bit 21

enumerator kI2C\_MonitorReadyFlag

The I2C monitor ready interrupt. bit 16

enumerator kI2C\_MonitorOverflowFlag

The monitor data overrun interrupt. bit 17

enumerator kI2C\_MonitorActiveFlag

The monitor is active. bit 18

enumerator kI2C\_MonitorIdleFlag

The monitor idle interrupt. bit 19

enumerator kI2C\_EventTimeoutFlag

The bus event timeout interrupt. bit 24

enumerator kI2C\_SclTimeoutFlag

The SCL timeout interrupt. bit 25

enumerator kI2C\_MasterAllClearFlags

enumerator kI2C\_SlaveAllClearFlags

enumerator kI2C\_CommonAllClearFlags

enum \_i2c\_interrupt\_enable

I2C interrupt enable.

---

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

---

*Values:*

enumerator kI2C\_MasterPendingInterruptEnable

The I2C master communication pending interrupt.

enumerator kI2C\_MasterArbitrationLostInterruptEnable

The I2C master arbitration lost interrupt.

enumerator kI2C\_MasterStartStopErrorInterruptEnable

The I2C master start/stop timing error interrupt.

enumerator kI2C\_SlavePendingInterruptEnable

The I2C slave communication pending interrupt.

enumerator kI2C\_SlaveNotStretchingInterruptEnable

The I2C slave not stretching interrupt, deep-sleep mode can be entered only when this interrupt occurs.

enumerator kI2C\_SlaveDeselectedInterruptEnable

The I2C slave deselection interrupt.

enumerator kI2C\_MonitorReadyInterruptEnable

The I2C monitor ready interrupt.

enumerator kI2C\_MonitorOverflowInterruptEnable

The monitor data overrun interrupt.

enumerator kI2C\_MonitorIdleInterruptEnable

The monitor idle interrupt.

enumerator kI2C\_EventTimeoutInterruptEnable

The bus event timeout interrupt.

enumerator kI2C\_SclTimeoutInterruptEnable

The SCL timeout interrupt.

enumerator kI2C\_MasterAllInterruptEnable

enumerator kI2C\_SlaveAllInterruptEnable

enumerator kI2C\_CommonAllInterruptEnable

I2C\_RETRY\_TIMES

Retry times for waiting flag.

I2C\_MASTER\_TRANSMIT\_IGNORE\_LAST\_NACK

Whether to ignore the nack signal of the last byte during master transmit.

I2C\_STAT\_MSTCODE\_IDLE

Master Idle State Code

I2C\_STAT\_MSTCODE\_RXREADY

Master Receive Ready State Code

I2C\_STAT\_MSTCODE\_TXREADY

Master Transmit Ready State Code

I2C\_STAT\_MSTCODE\_NACKADR

Master NACK by slave on address State Code

I2C\_STAT\_MSTCODE\_NACKDAT

Master NACK by slave on data State Code

I2C\_STAT\_SLVST\_ADDR

I2C\_STAT\_SLVST\_RX

I2C\_STAT\_SLVST\_TX

## 2.17 I2C Master Driver

`void I2C_MasterGetDefaultConfig(i2c_master_config_t *masterConfig)`

Provides a default configuration for the I2C master peripheral.

This function provides the following default configuration for the I2C master peripheral:

```
masterConfig->enableMaster      = true;
masterConfig->baudRate__Bps     = 100000U;
masterConfig->enableTimeout     = false;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with `I2C_MasterInit()`.

### Parameters

- `masterConfig` – **[out]** User provided configuration structure for default values. Refer to `i2c_master_config_t`.

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

Initializes the I2C master peripheral.

This function enables the peripheral clock and initializes the I2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

### Parameters

- `base` – The I2C peripheral base address.
- `masterConfig` – User provided peripheral configuration. Use `I2C_MasterGetDefaultConfig()` to get a set of defaults that you can override.
- `srcClock_Hz` – Frequency in Hertz of the I2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

`void I2C_MasterDeinit(I2C_Type *base)`

Deinitializes the I2C master peripheral.

This function disables the I2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

### Parameters

- `base` – The I2C peripheral base address.

`uint32_t I2C_GetInstance(I2C_Type *base)`

Returns an instance number given a base address.

If an invalid base address is passed, debug builds will assert. Release builds will just return instance number 0.

### Parameters

- `base` – The I2C peripheral base address.

### Returns

I2C instance number starting from 0.

`static inline void I2C_MasterReset(I2C_Type *base)`

Performs a software reset.

Restores the I2C master peripheral to reset conditions.

### Parameters

- `base` – The I2C peripheral base address.

static inline void I2C\_MasterEnable(I2C\_Type \*base, bool enable)

Enables or disables the I2C module as master.

#### Parameters

- `base` – The I2C peripheral base address.
- `enable` – Pass true to enable or false to disable the specified I2C as master.

uint32\_t I2C\_GetStatusFlags(I2C\_Type \*base)

Gets the I2C status flags.

A bit mask with the state of all I2C status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

#### See also:

`_i2c_status_flags`.

#### Parameters

- `base` – The I2C peripheral base address.

#### Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I2C\_ClearStatusFlags(I2C\_Type \*base, uint32\_t statusMask)

Clears the I2C status flag state.

Refer to `kI2C_CommonAllClearStatusFlags`, `kI2C_MasterAllClearStatusFlags` and `kI2C_SlaveAllClearStatusFlags` to see the clearable flags. Attempts to clear other flags has no effect.

#### See also:

`_i2c_status_flags`, `_i2c_master_status_flags` and `_i2c_slave_status_flags`.

#### Parameters

- `base` – The I2C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of the members in `kI2C_CommonAllClearStatusFlags`, `kI2C_MasterAllClearStatusFlags` and `kI2C_SlaveAllClearStatusFlags`. You may pass the result of a previous call to `I2C_GetStatusFlags()`.

static inline void I2C\_MasterClearStatusFlags(I2C\_Type \*base, uint32\_t statusMask)

Clears the I2C master status flag state.

#### Deprecated:

Do not use this function. It has been superseded by `I2C_ClearStatusFlags`. The following status register flags can be cleared:

- `kI2C_MasterArbitrationLostFlag`
- `kI2C_MasterStartStopErrorFlag`

Attempts to clear other flags has no effect.

**See also:**

`_i2c_status_flags`.

**Parameters**

- `base` – The I2C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i2c_status_flags` enumerators OR'd together. You may pass the result of a previous call to `I2C_GetStatusFlags()`.

```
static inline void I2C_EnableInterrupts(I2C_Type *base, uint32_t interruptMask)
```

Enables the I2C interrupt requests.

**Parameters**

- `base` – The I2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_i2c_interrupt_enable` for the set of constants that should be OR'd together to form the bit mask.

```
static inline void I2C_DisableInterrupts(I2C_Type *base, uint32_t interruptMask)
```

Disables the I2C interrupt requests.

**Parameters**

- `base` – The I2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_i2c_interrupt_enable` for the set of constants that should be OR'd together to form the bit mask.

```
static inline uint32_t I2C_GetEnabledInterrupts(I2C_Type *base)
```

Returns the set of currently enabled I2C interrupt requests.

**Parameters**

- `base` – The I2C peripheral base address.

**Returns**

A bitmask composed of `_i2c_interrupt_enable` enumerators OR'd together to indicate the set of enabled interrupts.

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

Sets the I2C bus frequency for master transactions.

The I2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

**Parameters**

- `base` – The I2C peripheral base address.
- `srcClock_Hz` – I2C functional clock frequency in Hertz.
- `baudRate_Bps` – Requested bus frequency in bits per second.

```
void I2C_MasterSetTimeoutValue(I2C_Type *base, uint8_t timeout_Ms, uint32_t srcClock_Hz)
```

Sets the I2C bus timeout value.

If the SCL signal remains low or bus does not have event longer than the timeout value, `kI2C_SclTimeoutFlag` or `kI2C_EventTimeoutFlag` is set. This can indicate the bus is held by slave or any fault occurs to the I2C module.

**Parameters**

- `base` – The I2C peripheral base address.
- `timeout_Ms` – Timeout value in millisecond.
- `srcClock_Hz` – I2C functional clock frequency in Hertz.

`static inline bool I2C_MasterGetBusIdleState(I2C_Type *base)`

Returns whether the bus is idle.

Requires the master mode to be enabled.

#### Parameters

- `base` – The I2C peripheral base address.

#### Return values

- `true` – Bus is busy.
- `false` – Bus is idle.

`status_t I2C_MasterStart(I2C_Type *base, uint8_t address, i2c_direction_t direction)`

Sends a START on the I2C bus.

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

#### Parameters

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

#### Return values

- `kStatus_Success` – Successfully send the start signal.
- `kStatus_I2C_Busy` – Current bus is busy.

`status_t I2C_MasterStop(I2C_Type *base)`

Sends a STOP signal on the I2C bus.

#### Return values

- `kStatus_Success` – Successfully send the stop signal.
- `kStatus_I2C_Timeout` – Send stop signal failed, timeout.

`static inline status_t I2C_MasterRepeatedStart(I2C_Type *base, uint8_t address, i2c_direction_t direction)`

Sends a REPEATED START on the I2C bus.

#### Parameters

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

#### Return values

- `kStatus_Success` – Successfully send the start signal.
- `kStatus_I2C_Busy` – Current bus is busy but not occupied by current I2C master.



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

Performs a polling send transfer on the I2C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns *kStatus\_I2C\_Nak*.

#### Parameters

- *base* – The I2C peripheral base address.
- *txBuff* – The pointer to the data to be transferred.
- *txSize* – The length in bytes of the data to be transferred.
- *flags* – Transfer control flag to control special behavior like suppressing start or stop, for normal transfers use *kI2C\_TransferDefaultFlag*

#### Return values

- *kStatus\_\_Success* – Data was sent successfully.
- *kStatus\_I2C\_Busy* – Another master is currently utilizing the bus.
- *kStatus\_I2C\_Nak* – The slave device sent a NAK in response to a byte.
- *kStatus\_I2C\_ArbitrationLost* – Arbitration lost error.

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

Performs a polling receive transfer on the I2C bus.

#### Parameters

- *base* – The I2C peripheral base address.
- *rxBuff* – The pointer to the data to be transferred.
- *rxSize* – The length in bytes of the data to be transferred.
- *flags* – Transfer control flag to control special behavior like suppressing start or stop, for normal transfers use *kI2C\_TransferDefaultFlag*

#### Return values

- *kStatus\_\_Success* – Data was received successfully.
- *kStatus\_I2C\_Busy* – Another master is currently utilizing the bus.
- *kStatus\_I2C\_Nak* – The slave device sent a NAK in response to a byte.
- *kStatus\_I2C\_ArbitrationLost* – Arbitration lost error.

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

Performs a master polling transfer on the I2C bus.

---

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

---

#### Parameters

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

#### Return values

- *kStatus\_\_Success* – Successfully complete the data transmission.
- *kStatus\_I2C\_Busy* – Previous transmission still not finished.

- `kStatus_I2C_Timeout` – Transfer error, wait signal timeout.
- `kStatus_I2C_ArbitrationLost` – Transfer error, arbitration lost.
- `kStatus_I2C_Nak` – Transfer error, receive NAK during transfer.
- `kStatus_I2C_Addr_Nak` – Transfer error, receive NAK during addressing.

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

Creates a new handle for the I2C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I2C_MasterTransferAbort()` API shall be called.

#### Parameters

- `base` – The I2C peripheral base address.
- `handle` – **[out]** Pointer to the I2C master driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

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

Performs a non-blocking transaction on the I2C bus.

#### Parameters

- `base` – The I2C peripheral base address.
- `handle` – Pointer to the I2C master driver handle.
- `xfer` – The pointer to the transfer descriptor.

#### Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I2C_Busy` – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

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

Returns number of bytes transferred so far.

#### Parameters

- `base` – The I2C peripheral base address.
- `handle` – Pointer to the I2C master driver handle.
- `count` – **[out]** Number of bytes transferred so far by the non-blocking transaction.

#### Return values

- `kStatus_Success` –
- `kStatus_I2C_Busy` –

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

Terminates a non-blocking I2C master transmission early.

---

**Note:** It is not safe to call this function from an IRQ handler that has a higher priority than the I2C peripheral's IRQ priority.

---

**Parameters**

- base – The I2C peripheral base address.
- handle – Pointer to the I2C master driver handle.

**Return values**

- kStatus\_Success – A transaction was successfully aborted.
- kStatus\_I2C\_Timeout – Timeout during polling for flags.

void I2C\_MasterTransferHandleIRQ(I2C\_Type \*base, i2c\_master\_handle\_t \*handle)  
Reusable routine to handle master interrupts.

---

**Note:** This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

---

**Parameters**

- base – The I2C peripheral base address.
- handle – Pointer to the I2C master driver handle.

enum \_i2c\_direction  
Direction of master and slave transfers.

*Values:*

enumerator kI2C\_Write  
Master transmit.

enumerator kI2C\_Read  
Master receive.

enum \_i2c\_master\_transfer\_flags  
Transfer option flags.

---

**Note:** These enumerations are intended to be OR'd together to form a bit mask of options for the \_i2c\_master\_transfer::flags field.

---

*Values:*

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

enumerator kI2C\_TransferNoStartFlag  
Don't send a start condition, address, and sub address

enumerator kI2C\_TransferRepeatedStartFlag  
Send a repeated start condition

enumerator kI2C\_TransferNoStopFlag  
Don't send a stop condition.

enum \_i2c\_transfer\_states  
States for the state machine used by transactional APIs.

*Values:*

enumerator kIdleState

enumerator kTransmitSubaddrState

enumerator kTransmitDataState

enumerator kReceiveDataBeginState

enumerator kReceiveDataState

enumerator kReceiveLastDataState

enumerator kStartState

enumerator kStopState

enumerator kWaitForCompletionState

typedef enum *\_i2c\_direction* i2c\_direction\_t

Direction of master and slave transfers.

typedef struct *\_i2c\_master\_config* i2c\_master\_config\_t

Structure with settings to initialize the I2C master module.

This structure holds configuration settings for the I2C peripheral. To initialize this structure to reasonable defaults, call the I2C\_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef struct *\_i2c\_master\_transfer* i2c\_master\_transfer\_t

I2C master transfer typedef.

typedef struct *\_i2c\_master\_handle* i2c\_master\_handle\_t

I2C master handle typedef.

typedef void (\*i2c\_master\_transfer\_callback\_t)(I2C\_Type \*base, i2c\_master\_handle\_t \*handle, status\_t completionStatus, void \*userData)

Master completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to I2C\_MasterTransferCreateHandle().

#### **Param base**

The I2C peripheral base address.

#### **Param completionStatus**

Either kStatus\_Success or an error code describing how the transfer completed.

#### **Param userData**

Arbitrary pointer-sized value passed from the application.

struct *\_i2c\_master\_config*

*#include <fsl\_i2c.h>* Structure with settings to initialize the I2C master module.

This structure holds configuration settings for the I2C peripheral. To initialize this structure to reasonable defaults, call the I2C\_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

### **Public Members**

bool enableMaster

Whether to enable master mode.

uint32\_t baudRate\_Bps

Desired baud rate in bits per second.

bool enableTimeout

Enable internal timeout function.

uint8\_t timeout\_Ms

Event timeout and SCL low timeout value.

struct \_i2c\_master\_transfer

*#include <fsl\_i2c.h>* Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the I2C\_MasterTransferNonBlocking() API.

### Public Members

uint32\_t flags

Bit mask of options for the transfer. See enumeration \_i2c\_master\_transfer\_flags for available options. Set to 0 or kI2C\_TransferDefaultFlag for normal transfers.

uint8\_t slaveAddress

The 7-bit slave address.

i2c\_direction\_t direction

Either kI2C\_Read or kI2C\_Write.

uint32\_t subaddress

Sub address. Transferred MSB first.

size\_t subaddressSize

Length of sub address to send in bytes. Maximum size is 4 bytes.

void \*data

Pointer to data to transfer.

size\_t dataSize

Number of bytes to transfer.

struct \_i2c\_master\_handle

*#include <fsl\_i2c.h>* Driver handle for master non-blocking APIs.

---

**Note:** The contents of this structure are private and subject to change.

---

### Public Members

uint8\_t state

Transfer state machine current state.

uint32\_t transferCount

Indicates progress of the transfer

uint32\_t remainingBytes

Remaining byte count in current state.

uint8\_t \*buf

Buffer pointer for current state.

`bool checkAddrNack`

Whether to check the nack signal is detected during addressing.

`i2c_master_transfer_t transfer`

Copy of the current transfer info.

`i2c_master_transfer_callback_t completionCallback`

Callback function pointer.

`void *userData`

Application data passed to callback.

## 2.18 I2C Slave Driver

`void I2C_SlaveGetDefaultConfig(i2c_slave_config_t *slaveConfig)`

Provides a default configuration for the I2C slave peripheral.

This function provides the following default configuration for the I2C slave peripheral:

```
slaveConfig->enableSlave = true;
slaveConfig->address0.disable = false;
slaveConfig->address0.address = 0u;
slaveConfig->address1.disable = true;
slaveConfig->address2.disable = true;
slaveConfig->address3.disable = true;
slaveConfig->busSpeed = kI2C_SlaveStandardMode;
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with `I2C_SlaveInit()`. Be sure to override at least the `address0.address` member of the configuration structure with the desired slave address.

### Parameters

- `slaveConfig` – **[out]** User provided configuration structure that is set to default values. Refer to `i2c_slave_config_t`.

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

Initializes the I2C slave peripheral.

This function enables the peripheral clock and initializes the I2C slave peripheral as described by the user provided configuration.

### Parameters

- `base` – The I2C peripheral base address.
- `slaveConfig` – User provided peripheral configuration. Use `I2C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- `srcClock_Hz` – Frequency in Hertz of the I2C functional clock. Used to calculate CLKDIV value to provide enough data setup time for master when slave stretches the clock.

`void I2C_SlaveSetAddress(I2C_Type *base, i2c_slave_address_register_t addressRegister, uint8_t address, bool addressDisable)`

Configures Slave Address n register.

This function writes new value to Slave Address register.

### Parameters

- `base` – The I2C peripheral base address.

- `addressRegister` – The module supports multiple address registers. The parameter determines which one shall be changed.
- `address` – The slave address to be stored to the address register for matching.
- `addressDisable` – Disable matching of the specified address register.

`void I2C_SlaveDeinit(I2C_Type *base)`

Deinitializes the I2C slave peripheral.

This function disables the I2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

#### Parameters

- `base` – The I2C peripheral base address.

`static inline void I2C_SlaveEnable(I2C_Type *base, bool enable)`

Enables or disables the I2C module as slave.

#### Parameters

- `base` – The I2C peripheral base address.
- `enable` – True to enable or false to disable.

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

Clears the I2C status flag state.

The following status register flags can be cleared:

- slave deselected flag

Attempts to clear other flags has no effect.

#### See also:

`_i2c_slave_flags`.

#### Parameters

- `base` – The I2C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i2c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `I2C_SlaveGetStatusFlags()`.

`status_t I2C_SlaveWriteBlocking(I2C_Type *base, const uint8_t *txBuff, size_t txSize)`

Performs a polling send transfer on the I2C bus.

The function executes blocking address phase and blocking data phase.

#### Parameters

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

#### Returns

`kStatus_Success` Data has been sent.

#### Returns

`kStatus_Fail` Unexpected slave state (master data write while master read from slave is expected).

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

Performs a polling receive transfer on the I2C bus.

The function executes blocking address phase and blocking data phase.

#### Parameters

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

#### Returns

kStatus\_Success Data has been received.

#### Returns

kStatus\_Fail Unexpected slave state (master data read while master write to slave is expected).

void I2C\_SlaveTransferCreateHandle(I2C\_Type \*base, i2c\_slave\_handle\_t \*handle, i2c\_slave\_transfer\_callback\_t callback, void \*userData)

Creates a new handle for the I2C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the I2C\_SlaveTransferAbort() API shall be called.

#### Parameters

- base – The I2C peripheral base address.
- handle – **[out]** Pointer to the I2C slave driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

*status\_t* I2C\_SlaveTransferNonBlocking(I2C\_Type \*base, i2c\_slave\_handle\_t \*handle, uint32\_t eventMask)

Starts accepting slave transfers.

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

If no slave Tx transfer is busy, a master read from slave request invokes kI2C\_SlaveTransmitEvent callback. If no slave Rx transfer is busy, a master write to slave request invokes kI2C\_SlaveReceiveEvent callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of i2c\_slave\_transfer\_event\_t enumerators for the events you wish to receive. The kI2C\_SlaveTransmitEvent and kI2C\_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the kI2C\_SlaveAllEvents constant is provided as a convenient way to enable all events.

#### Parameters

- base – The I2C peripheral base address.
- handle – Pointer to i2c\_slave\_handle\_t structure which stores the transfer state.



- `eventMask` – Bit mask formed by OR'ing together `i2c_slave_transfer_event_t` enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and `kI2C_SlaveAllEvents` to enable all events.

#### Return values

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

`status_t I2C_SlaveSetSendBuffer(I2C_Type *base, volatile i2c_slave_transfer_t *transfer, const void *txData, size_t txSize, uint32_t eventMask)`

Starts accepting master read from slave requests.

The function can be called in response to `kI2C_SlaveTransmitEvent` callback to start a new slave Tx transfer from within the transfer callback.

The set of events received by the callback is customizable. To do so, set the `eventMask` parameter to the OR'd combination of `i2c_slave_transfer_event_t` enumerators for the events you wish to receive. The `kI2C_SlaveTransmitEvent` and `kI2C_SlaveReceiveEvent` events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the `kI2C_SlaveAllEvents` constant is provided as a convenient way to enable all events.

#### Parameters

- `base` – The I2C peripheral base address.
- `transfer` – Pointer to `i2c_slave_transfer_t` structure.
- `txData` – Pointer to data to send to master.
- `txSize` – Size of `txData` in bytes.
- `eventMask` – Bit mask formed by OR'ing together `i2c_slave_transfer_event_t` enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and `kI2C_SlaveAllEvents` to enable all events.

#### Return values

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

`status_t I2C_SlaveSetReceiveBuffer(I2C_Type *base, volatile i2c_slave_transfer_t *transfer, void *rxData, size_t rxSize, uint32_t eventMask)`

Starts accepting master write to slave requests.

The function can be called in response to `kI2C_SlaveReceiveEvent` callback to start a new slave Rx transfer from within the transfer callback.

The set of events received by the callback is customizable. To do so, set the `eventMask` parameter to the OR'd combination of `i2c_slave_transfer_event_t` enumerators for the events you wish to receive. The `kI2C_SlaveTransmitEvent` and `kI2C_SlaveReceiveEvent` events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the `kI2C_SlaveAllEvents` constant is provided as a convenient way to enable all events.

#### Parameters

- `base` – The I2C peripheral base address.

- `transfer` – Pointer to `i2c_slave_transfer_t` structure.
- `rxData` – Pointer to data to store data from master.
- `rxSize` – Size of `rxData` in bytes.
- `eventMask` – Bit mask formed by OR'ing together `i2c_slave_transfer_event_t` enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and `kI2C_SlaveAllEvents` to enable all events.

**Return values**

- `kStatus__Success` – Slave transfers were successfully started.
- `kStatus__I2C__Busy` – Slave transfers have already been started on this handle.

```
static inline uint32_t I2C_SlaveGetReceivedAddress(I2C_Type *base, volatile i2c_slave_transfer_t *transfer)
```

Returns the slave address sent by the I2C master.

This function should only be called from the address match event callback `kI2C_SlaveAddressMatchEvent`.

**Parameters**

- `base` – The I2C peripheral base address.
- `transfer` – The I2C slave transfer.

**Returns**

The 8-bit address matched by the I2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

```
void I2C_SlaveTransferAbort(I2C_Type *base, i2c_slave_handle_t *handle)
```

Aborts the slave non-blocking transfers.

---

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

---

**Parameters**

- `base` – The I2C peripheral base address.
- `handle` – Pointer to `i2c_slave_handle_t` structure which stores the transfer state.

**Return values**

- `kStatus__Success` –
- `kStatus__I2C__Idle` –

```
status_t I2C_SlaveTransferGetCount(I2C_Type *base, i2c_slave_handle_t *handle, size_t *count)
```

Gets the slave transfer remaining bytes during a interrupt non-blocking transfer.

**Parameters**

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

**Return values**

- `kStatus__InvalidArgument` – count is Invalid.
- `kStatus__Success` – Successfully return the count.

void I2C\_SlaveTransferHandleIRQ(I2C\_Type \*base, i2c\_slave\_handle\_t \*handle)

Reusable routine to handle slave interrupts.

---

**Note:** This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

---

### Parameters

- base – The I2C peripheral base address.
- handle – Pointer to i2c\_slave\_handle\_t structure which stores the transfer state.

enum \_i2c\_slave\_address\_register

I2C slave address register.

*Values:*

enumerator kI2C\_SlaveAddressRegister0

Slave Address 0 register.

enumerator kI2C\_SlaveAddressRegister1

Slave Address 1 register.

enumerator kI2C\_SlaveAddressRegister2

Slave Address 2 register.

enumerator kI2C\_SlaveAddressRegister3

Slave Address 3 register.

enum \_i2c\_slave\_address\_qual\_mode

I2C slave address match options.

*Values:*

enumerator kI2C\_QualModeMask

The SLVQUAL0 field (qualAddress) is used as a logical mask for matching address0.

enumerator kI2C\_QualModeExtend

The SLVQUAL0 (qualAddress) field is used to extend address 0 matching in a range of addresses.

enum \_i2c\_slave\_bus\_speed

I2C slave bus speed options.

*Values:*

enumerator kI2C\_SlaveStandardMode

enumerator kI2C\_SlaveFastMode

enumerator kI2C\_SlaveFastModePlus

enumerator kI2C\_SlaveHsMode

enum \_i2c\_slave\_transfer\_event

Set of events sent to the callback for non blocking slave transfers.

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

---

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

---

*Values:*

enumerator `kI2C_SlaveAddressMatchEvent`

Received the slave address after a start or repeated start.

enumerator `kI2C_SlaveTransmitEvent`

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator `kI2C_SlaveReceiveEvent`

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

enumerator `kI2C_SlaveCompletionEvent`

All data in the active transfer have been consumed.

enumerator `kI2C_SlaveDeselectedEvent`

The slave function has become deselected (SLVSEL flag changing from 1 to 0).

enumerator `kI2C_SlaveAllEvents`

Bit mask of all available events.

enum `_i2c_slave_fsm`

I2C slave software finite state machine states.

*Values:*

enumerator `kI2C_SlaveFsmAddressMatch`

enumerator `kI2C_SlaveFsmReceive`

enumerator `kI2C_SlaveFsmTransmit`

typedef enum `_i2c_slave_address_register` `i2c_slave_address_register_t`

I2C slave address register.

typedef struct `_i2c_slave_address` `i2c_slave_address_t`

Data structure with 7-bit Slave address and Slave address disable.

typedef enum `_i2c_slave_address_qual_mode` `i2c_slave_address_qual_mode_t`

I2C slave address match options.

typedef enum `_i2c_slave_bus_speed` `i2c_slave_bus_speed_t`

I2C slave bus speed options.

typedef struct `_i2c_slave_config` `i2c_slave_config_t`

Structure with settings to initialize the I2C slave module.

This structure holds configuration settings for the I2C slave peripheral. To initialize this structure to reasonable defaults, call the `I2C_SlaveGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum `_i2c_slave_transfer_event` `i2c_slave_transfer_event_t`

Set of events sent to the callback for non blocking slave transfers.

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

---

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

---

`typedef struct _i2c_slave_handle i2c_slave_handle_t`  
I2C slave handle typedef.

`typedef struct _i2c_slave_transfer i2c_slave_transfer_t`  
I2C slave transfer structure.

`typedef void (*i2c_slave_transfer_callback_t)(I2C_Type *base, volatile i2c_slave_transfer_t *transfer, void *userData)`  
Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the `I2C_SlaveSetCallback()` function after you have created a handle.

**Param base**

Base address for the I2C instance on which the event occurred.

**Param transfer**

Pointer to transfer descriptor containing values passed to and/or from the callback.

**Param userData**

Arbitrary pointer-sized value passed from the application.

`typedef enum _i2c_slave_fsm i2c_slave_fsm_t`  
I2C slave software finite state machine states.

`typedef void (*flexcomm_i2c_master_irq_handler_t)(I2C_Type *base, i2c_master_handle_t *handle)`

Typedef for master interrupt handler.

`typedef void (*flexcomm_i2c_slave_irq_handler_t)(I2C_Type *base, i2c_slave_handle_t *handle)`  
Typedef for slave interrupt handler.

`struct _i2c_slave_address`  
`#include <fsl_i2c.h>` Data structure with 7-bit Slave address and Slave address disable.

**Public Members**

`uint8_t address`  
7-bit Slave address SLVADR.

`bool addressDisable`  
Slave address disable SADISABLE.

`struct _i2c_slave_config`  
`#include <fsl_i2c.h>` Structure with settings to initialize the I2C slave module.

This structure holds configuration settings for the I2C slave peripheral. To initialize this structure to reasonable defaults, call the `I2C_SlaveGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

**Public Members**

`i2c_slave_address_t address0`  
Slave's 7-bit address and disable.

*i2c\_slave\_address\_t* address1

Alternate slave 7-bit address and disable.

*i2c\_slave\_address\_t* address2

Alternate slave 7-bit address and disable.

*i2c\_slave\_address\_t* address3

Alternate slave 7-bit address and disable.

*i2c\_slave\_address\_qual\_mode\_t* qualMode

Qualify mode for slave address 0.

uint8\_t qualAddress

Slave address qualifier for address 0.

*i2c\_slave\_bus\_speed\_t* busSpeed

Slave bus speed mode. If the slave function stretches SCL to allow for software response, it must provide sufficient data setup time to the master before releasing the stretched clock. This is accomplished by inserting one clock time of CLKDIV at that point. The busSpeed value is used to configure CLKDIV such that one clock time is greater than the tSU;DAT value noted in the I2C bus specification for the I2C mode that is being used. If the busSpeed mode is unknown at compile time, use the longest data setup time kI2C\_SlaveStandardMode (250 ns)

bool enableSlave

Enable slave mode.

struct *\_i2c\_slave\_transfer*

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

## Public Members

*i2c\_slave\_handle\_t* \*handle

Pointer to handle that contains this transfer.

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

Reason the callback is being invoked.

uint8\_t receivedAddress

Matching address send by master. 7-bits plus R/nW bit0

uint32\_t eventMask

Mask of enabled events.

uint8\_t \*rxData

Transfer buffer for receive data

const uint8\_t \*txData

Transfer buffer for transmit data

size\_t txSize

Transfer size

size\_t rxSize

Transfer size

size\_t transferredCount

Number of bytes transferred during this transfer.

*status\_t* completionStatus

Success or error code describing how the transfer completed. Only applies for *kI2C\_SlaveCompletionEvent*.

struct *\_i2c\_slave\_handle*

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

---

**Note:** The contents of this structure are private and subject to change.

---

### Public Members

volatile *i2c\_slave\_transfer\_t* transfer

I2C slave transfer.

volatile bool isBusy

Whether transfer is busy.

volatile *i2c\_slave\_fsm\_t* slaveFsm

slave transfer state machine.

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

Callback function called at transfer event.

void \*userData

Callback parameter passed to callback.

## 2.19 I2S: I2S Driver

### 2.20 I2S DMA Driver

void I2S\_TxTransferCreateHandleDMA(I2S\_Type \*base, *i2s\_dma\_handle\_t* \*handle, *dma\_handle\_t* \*dmaHandle, *i2s\_dma\_transfer\_callback\_t* callback, void \*userData)

Initializes handle for transfer of audio data.

#### Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- dmaHandle – pointer to dma handle structure.
- callback – function to be called back when transfer is done or fails.
- userData – pointer to data passed to callback.

*status\_t* I2S\_TxTransferSendDMA(I2S\_Type \*base, *i2s\_dma\_handle\_t* \*handle, *i2s\_transfer\_t* transfer)

Begins or queue sending of the given data.

#### Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- transfer – data buffer.

**Return values**

- `kStatus__Success` –
- `kStatus_I2S_Busy` – if all queue slots are occupied with unsent buffers.

`void I2S_TransferAbortDMA(I2S_Type *base, i2s_dma_handle_t *handle)`

Aborts transfer of data.

**Parameters**

- `base` – I2S base pointer.
- `handle` – pointer to handle structure.

`void I2S_RxTransferCreateHandleDMA(I2S_Type *base, i2s_dma_handle_t *handle, dma_handle_t *dmaHandle, i2s_dma_transfer_callback_t callback, void *userData)`

Initializes handle for reception of audio data.

**Parameters**

- `base` – I2S base pointer.
- `handle` – pointer to handle structure.
- `dmaHandle` – pointer to dma handle structure.
- `callback` – function to be called back when transfer is done or fails.
- `userData` – pointer to data passed to callback.

`status_t I2S_RxTransferReceiveDMA(I2S_Type *base, i2s_dma_handle_t *handle, i2s_transfer_t transfer)`

Begins or queue reception of data into given buffer.

**Parameters**

- `base` – I2S base pointer.
- `handle` – pointer to handle structure.
- `transfer` – data buffer.

**Return values**

- `kStatus__Success` –
- `kStatus_I2S_Busy` – if all queue slots are occupied with buffers which are not full.

`void I2S_DMACallback(dma_handle_t *handle, void *userData, bool transferDone, uint32_t tcDs)`

Invoked from DMA interrupt handler.

**Parameters**

- `handle` – pointer to DMA handle structure.
- `userData` – argument for user callback.
- `transferDone` – if transfer was done.
- `tcDs` –

`void I2S_TransferInstallLoopDMADescriptorMemory(i2s_dma_handle_t *handle, void *dmaDescriptorAddr, size_t dmaDescriptorNum)`

Install DMA descriptor memory for loop transfer only.

This function used to register DMA descriptor memory for the i2s loop dma transfer.



It must be called before I2S\_TransferSendLoopDMA/I2S\_TransferReceiveLoopDMA and after I2S\_RxTransferCreateHandleDMA/I2S\_TxTransferCreateHandleDMA.

User should be take care about the address of DMA descriptor pool which required align with 16BYTE at least.

#### Parameters

- handle – Pointer to i2s DMA transfer handle.
- dmaDescriptorAddr – DMA descriptor start address.
- dmaDescriptorNum – DMA descriptor number.

*status\_t* I2S\_TransferSendLoopDMA(I2S\_Type \*base, i2s\_dma\_handle\_t \*handle, i2s\_transfer\_t \*xfer, uint32\_t loopTransferCount)

Send link transfer data using DMA.

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

This function support loop transfer, such as A->B->...->A, the loop transfer chain will be converted into a chain of descriptor and submit to dma. Application must be aware of that the more counts of the loop transfer, then more DMA descriptor memory required, user can use function I2S\_InstallDMADescriptorMemory to register the dma descriptor memory.

As the DMA support maximum 1024 transfer count, so application must be aware of that this transfer function support maximum 1024 samples in each transfer, otherwise assert error or error status will be returned. Once the loop transfer start, application can use function I2S\_TransferAbortDMA to stop the loop transfer.

#### Parameters

- base – I2S peripheral base address.
- handle – Pointer to usart\_dma\_handle\_t structure.
- xfer – I2S DMA transfer structure. See i2s\_transfer\_t.
- loopTransferCount – loop count

#### Return values

kStatus\_Success –

*status\_t* I2S\_TransferReceiveLoopDMA(I2S\_Type \*base, i2s\_dma\_handle\_t \*handle, i2s\_transfer\_t \*xfer, uint32\_t loopTransferCount)

Receive link transfer data using DMA.

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

This function support loop transfer, such as A->B->...->A, the loop transfer chain will be converted into a chain of descriptor and submit to dma. Application must be aware of that the more counts of the loop transfer, then more DMA descriptor memory required, user can use function I2S\_InstallDMADescriptorMemory to register the dma descriptor memory.

As the DMA support maximum 1024 transfer count, so application must be aware of that this transfer function support maximum 1024 samples in each transfer, otherwise assert error or error status will be returned. Once the loop transfer start, application can use function I2S\_TransferAbortDMA to stop the loop transfer.

#### Parameters

- base – I2S peripheral base address.
- handle – Pointer to usart\_dma\_handle\_t structure.
- xfer – I2S DMA transfer structure. See i2s\_transfer\_t.

- loopTransferCount – loop count

**Return values**

kStatus\_\_Success –

FSL\_I2S\_DMA\_DRIVER\_VERSION

I2S DMA driver version 2.3.3.

typedef struct *i2s\_dma\_handle* i2s\_dma\_handle\_t

Members not to be accessed / modified outside of the driver.

typedef void (\*i2s\_dma\_transfer\_callback\_t)(I2S\_Type \*base, *i2s\_dma\_handle\_t* \*handle, *status\_t* completionStatus, void \*userData)

Callback function invoked from DMA API on completion.

**Param base**

I2S base pointer.

**Param handle**

pointer to I2S transaction.

**Param completionStatus**

status of the transaction.

**Param userData**

optional pointer to user arguments data.

struct *i2s\_dma\_handle*

*#include <fsl\_i2s\_dma.h>* i2s dma handle

**Public Members**

uint32\_t state

Internal state of I2S DMA transfer

uint8\_t bytesPerFrame

bytes per frame

*i2s\_dma\_transfer\_callback\_t* completionCallback

Callback function pointer

void \*userData

Application data passed to callback

*dma\_handle\_t* \*dmaHandle

DMA handle

volatile *i2s\_transfer\_t* i2sQueue[(4U)]

Transfer queue storing transfer buffers

volatile uint8\_t queueUser

Queue index where user's next transfer will be stored

volatile uint8\_t queueDriver

Queue index of buffer actually used by the driver

*dma\_descriptor\_t* \*i2sLoopDMADescriptor

descriptor pool pointer

size\_t i2sLoopDMADescriptorNum

number of descriptor in descriptors pool

## 2.21 I2S Driver

`void I2S_TxInit(I2S_Type *base, const i2s_config_t *config)`

Initializes the FLEXCOMM peripheral for I2S transmit functionality.

Ungates the FLEXCOMM clock and configures the module for I2S transmission using a configuration structure. The configuration structure can be custom filled or set with default values by `I2S_TxGetDefaultConfig()`.

---

**Note:** This API should be called at the beginning of the application to use the I2S driver.

---

### Parameters

- `base` – I2S base pointer.
- `config` – pointer to I2S configuration structure.

`void I2S_RxInit(I2S_Type *base, const i2s_config_t *config)`

Initializes the FLEXCOMM peripheral for I2S receive functionality.

Ungates the FLEXCOMM clock and configures the module for I2S receive using a configuration structure. The configuration structure can be custom filled or set with default values by `I2S_RxGetDefaultConfig()`.

---

**Note:** This API should be called at the beginning of the application to use the I2S driver.

---

### Parameters

- `base` – I2S base pointer.
- `config` – pointer to I2S configuration structure.

`void I2S_TxGetDefaultConfig(i2s_config_t *config)`

Sets the I2S Tx configuration structure to default values.

This API initializes the configuration structure for use in `I2S_TxInit()`. The initialized structure can remain unchanged in `I2S_TxInit()`, or it can be modified before calling `I2S_TxInit()`. Example:

```
i2s_config_t config;
I2S_TxGetDefaultConfig(&config);
```

Default values:

```
config->masterSlave = kI2S_MasterSlaveNormalMaster;
config->mode = kI2S_ModeI2sClassic;
config->rightLow = false;
config->leftJust = false;
config->pdmData = false;
config->sckPol = false;
config->wsPol = false;
config->divider = 1;
config->oneChannel = false;
config->dataLength = 16;
config->frameLength = 32;
config->position = 0;
config->watermark = 4;
config->txEmptyZero = true;
config->pack48 = false;
```

### Parameters

- config – pointer to I2S configuration structure.

void I2S\_RxGetDefaultConfig(*i2s\_config\_t* \*config)

Sets the I2S Rx configuration structure to default values.

This API initializes the configuration structure for use in I2S\_RxInit(). The initialized structure can remain unchanged in I2S\_RxInit(), or it can be modified before calling I2S\_RxInit(). Example:

```
i2s_config_t config;  
I2S_RxGetDefaultConfig(&config);
```

Default values:

```
config->masterSlave = kI2S_MasterSlaveNormalSlave;  
config->mode = kI2S_ModeI2sClassic;  
config->rightLow = false;  
config->leftJust = false;  
config->pdmData = false;  
config->sckPol = false;  
config->wsPol = false;  
config->divider = 1;  
config->oneChannel = false;  
config->dataLength = 16;  
config->frameLength = 32;  
config->position = 0;  
config->watermark = 4;  
config->txEmptyZero = false;  
config->pack48 = false;
```

### Parameters

- config – pointer to I2S configuration structure.

void I2S\_Deinit(I2S\_Type \*base)

De-initializes the I2S peripheral.

This API gates the FLEXCOMM clock. The I2S module can't operate unless I2S\_TxInit or I2S\_RxInit is called to enable the clock.

### Parameters

- base – I2S base pointer.

void I2S\_SetBitClockRate(I2S\_Type \*base, uint32\_t sourceClockHz, uint32\_t sampleRate, uint32\_t bitWidth, uint32\_t channelNumbers)

Transmitter/Receiver bit clock rate configurations.

### Parameters

- base – SAI base pointer.
- sourceClockHz – bit clock source frequency.
- sampleRate – audio data sample rate.
- bitWidth – audio data bitWidth.
- channelNumbers – audio channel numbers.

void I2S\_TxTransferCreateHandle(I2S\_Type \*base, *i2s\_handle\_t* \*handle, *i2s\_transfer\_callback\_t* callback, void \*userData)

Initializes handle for transfer of audio data.

### Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- callback – function to be called back when transfer is done or fails.
- userData – pointer to data passed to callback.

*status\_t* I2S\_TxTransferNonBlocking(I2S\_Type \*base, *i2s\_handle\_t* \*handle, *i2s\_transfer\_t* transfer)

Begins or queue sending of the given data.

#### Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- transfer – data buffer.

#### Return values

- kStatus\_\_Success –
- kStatus\_I2S\_Busy – if all queue slots are occupied with unsent buffers.

void I2S\_TxTransferAbort(I2S\_Type \*base, *i2s\_handle\_t* \*handle)

Aborts sending of data.

#### Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.

void I2S\_RxTransferCreateHandle(I2S\_Type \*base, *i2s\_handle\_t* \*handle, *i2s\_transfer\_callback\_t* callback, void \*userData)

Initializes handle for reception of audio data.

#### Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- callback – function to be called back when transfer is done or fails.
- userData – pointer to data passed to callback.

*status\_t* I2S\_RxTransferNonBlocking(I2S\_Type \*base, *i2s\_handle\_t* \*handle, *i2s\_transfer\_t* transfer)

Begins or queue reception of data into given buffer.

#### Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- transfer – data buffer.

#### Return values

- kStatus\_\_Success –
- kStatus\_I2S\_Busy – if all queue slots are occupied with buffers which are not full.

void I2S\_RxTransferAbort(I2S\_Type \*base, *i2s\_handle\_t* \*handle)

Aborts receiving of data.

#### Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.

*status\_t* I2S\_TransferGetCount(I2S\_Type \*base, *i2s\_handle\_t* \*handle, *size\_t* \*count)

Returns number of bytes transferred so far.

#### Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- count – **[out]** number of bytes transferred so far by the non-blocking transaction.

#### Return values

- kStatus\_Success –
- kStatus\_NoTransferInProgress – there is no non-blocking transaction currently in progress.

*status\_t* I2S\_TransferGetErrorCount(I2S\_Type \*base, *i2s\_handle\_t* \*handle, *size\_t* \*count)

Returns number of buffer underruns or overruns.

#### Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- count – **[out]** number of transmit errors encountered so far by the non-blocking transaction.

#### Return values

- kStatus\_Success –
- kStatus\_NoTransferInProgress – there is no non-blocking transaction currently in progress.

static inline void I2S\_Enable(I2S\_Type \*base)

Enables I2S operation.

#### Parameters

- base – I2S base pointer.

void I2S\_EnableSecondaryChannel(I2S\_Type \*base, *uint32\_t* channel, bool oneChannel, *uint32\_t* position)

Enables I2S secondary channel.

#### Parameters

- base – I2S base pointer.
- channel – secondary channel number, reference `_i2s_secondary_channel`.
- oneChannel – true is treated as single channel, functionality left channel for this pair.
- position – define the location within the frame of the data, should not bigger than 0x1FFU.

static inline void I2S\_DisableSecondaryChannel(I2S\_Type \*base, *uint32\_t* channel)

Disables I2S secondary channel.

#### Parameters

- base – I2S base pointer.
- channel – secondary channel number, reference `_i2s_secondary_channel`.

static inline void I2S\_Disable(I2S\_Type \*base)

Disables I2S operation.

#### Parameters

- base – I2S base pointer.

static inline void I2S\_EnableInterrupts(I2S\_Type \*base, uint32\_t interruptMask)

Enables I2S FIFO interrupts.

#### Parameters

- base – I2S base pointer.
- interruptMask – bit mask of interrupts to enable. See `i2s_flags_t` for the set of constants that should be OR'd together to form the bit mask.

static inline void I2S\_DisableInterrupts(I2S\_Type \*base, uint32\_t interruptMask)

Disables I2S FIFO interrupts.

#### Parameters

- base – I2S base pointer.
- interruptMask – bit mask of interrupts to enable. See `i2s_flags_t` for the set of constants that should be OR'd together to form the bit mask.

static inline uint32\_t I2S\_GetEnabledInterrupts(I2S\_Type \*base)

Returns the set of currently enabled I2S FIFO interrupts.

#### Parameters

- base – I2S base pointer.

#### Returns

A bitmask composed of `i2s_flags_t` enumerators OR'd together to indicate the set of enabled interrupts.

*status\_t* I2S\_EmptyTxFifo(I2S\_Type \*base)

Flush the valid data in TX fifo.

#### Parameters

- base – I2S base pointer.

#### Returns

`kStatus_Fail` empty TX fifo failed, `kStatus_Success` empty tx fifo success.

void I2S\_TxHandleIRQ(I2S\_Type \*base, *i2s\_handle\_t* \*handle)

Invoked from interrupt handler when transmit FIFO level decreases.

#### Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.

void I2S\_RxHandleIRQ(I2S\_Type \*base, *i2s\_handle\_t* \*handle)

Invoked from interrupt handler when receive FIFO level decreases.

#### Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.

FSL\_I2S\_DRIVER\_VERSION

I2S driver version 2.3.2.

\_i2s\_status I2S status codes.

*Values:*

enumerator kStatus\_I2S\_BufferComplete

Transfer from/into a single buffer has completed

enumerator kStatus\_I2S\_Done

All buffers transfers have completed

enumerator kStatus\_I2S\_Busy

Already performing a transfer and cannot queue another buffer

enum \_i2s\_flags

I2S flags.

---

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

---

*Values:*

enumerator kI2S\_TxErrorFlag

TX error interrupt

enumerator kI2S\_TxLevelFlag

TX level interrupt

enumerator kI2S\_RxErrorFlag

RX error interrupt

enumerator kI2S\_RxLevelFlag

RX level interrupt

enum \_i2s\_master\_slave

Master / slave mode.

*Values:*

enumerator kI2S\_MasterSlaveNormalSlave

Normal slave

enumerator kI2S\_MasterSlaveWsSyncMaster

WS synchronized master

enumerator kI2S\_MasterSlaveExtSckMaster

Master using existing SCK

enumerator kI2S\_MasterSlaveNormalMaster

Normal master

enum \_i2s\_mode

I2S mode.

*Values:*

enumerator kI2S\_ModeI2sClassic

I2S classic mode

enumerator kI2S\_ModeDspWs50

DSP mode, WS having 50% duty cycle



enumerator kI2S\_ModeDspWsShort  
DSP mode, WS having one clock long pulse

enumerator kI2S\_ModeDspWsLong  
DSP mode, WS having one data slot long pulse

\_i2s\_secondary\_channel I2S secondary channel.

*Values:*

enumerator kI2S\_SecondaryChannel1  
secondary channel 1

enumerator kI2S\_SecondaryChannel2  
secondary channel 2

enumerator kI2S\_SecondaryChannel3  
secondary channel 3

typedef enum \_i2s\_flags i2s\_flags\_t  
I2S flags.

---

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

---

typedef enum \_i2s\_master\_slave i2s\_master\_slave\_t  
Master / slave mode.

typedef enum \_i2s\_mode i2s\_mode\_t  
I2S mode.

typedef struct \_i2s\_config i2s\_config\_t  
I2S configuration structure.

typedef struct \_i2s\_transfer i2s\_transfer\_t  
Buffer to transfer from or receive audio data into.

typedef struct \_i2s\_handle i2s\_handle\_t  
Transactional state of the initialized transfer or receive I2S operation.

typedef void (\*i2s\_transfer\_callback\_t)(I2S\_Type \*base, i2s\_handle\_t \*handle, status\_t  
completionStatus, void \*userData)  
Callback function invoked from transactional API on completion of a single buffer transfer.

**Param base**  
I2S base pointer.

**Param handle**  
pointer to I2S transaction.

**Param completionStatus**  
status of the transaction.

**Param userData**  
optional pointer to user arguments data.

I2S\_NUM\_BUFFERS  
Number of buffers .

struct \_i2s\_config  
*#include <fsl\_i2s.h>* I2S configuration structure.

**Public Members**

*i2s\_master\_slave\_t* masterSlave

Master / slave configuration

*i2s\_mode\_t* mode

I2S mode

bool rightLow

Right channel data in low portion of FIFO

bool leftJust

Left justify data in FIFO

bool pdmData

Data source is the D-Mic subsystem

bool sckPol

SCK polarity

bool wsPol

WS polarity

uint16\_t divider

Flexcomm function clock divider (1 - 4096)

bool oneChannel

true mono, false stereo

uint8\_t dataLength

Data length (4 - 32)

uint16\_t frameLength

Frame width (4 - 512)

uint16\_t position

Data position in the frame

uint8\_t watermark

FIFO trigger level

bool txEmptyZero

Transmit zero when buffer becomes empty or last item

bool pack48

Packing format for 48-bit data (false - 24 bit values, true - alternating 32-bit and 16-bit values)

struct *\_i2s\_transfer*

*#include <fsl\_i2s.h>* Buffer to transfer from or receive audio data into.

**Public Members**

uint8\_t \*data

Pointer to data buffer.

size\_t dataSize

Buffer size in bytes.

struct *\_i2s\_handle*

*#include <fsl\_i2s.h>* Members not to be accessed / modified outside of the driver.

### Public Members

volatile uint32\_t state

State of transfer

*i2s\_transfer\_callback\_t* completionCallback

Callback function pointer

void \*userData

Application data passed to callback

bool oneChannel

true mono, false stereo

uint8\_t dataLength

Data length (4 - 32)

bool pack48

Packing format for 48-bit data (false - 24 bit values, true - alternating 32-bit and 16-bit values)

uint8\_t watermark

FIFO trigger level

bool useFifo48H

When dataLength 17-24: true use FIFOWR48H, false use FIFOWR

volatile *i2s\_transfer\_t* i2sQueue[(4U)]

Transfer queue storing transfer buffers

volatile uint8\_t queueUser

Queue index where user's next transfer will be stored

volatile uint8\_t queueDriver

Queue index of buffer actually used by the driver

volatile uint32\_t errorCount

Number of buffer underruns/overruns

volatile uint32\_t transferCount

Number of bytes transferred

## 2.22 IAP: In Application Programming Driver

*status\_t* IAP\_ReadPartID(uint32\_t \*partID)

Read part identification number.

This function is used to read the part identification number.

### Parameters

- partID – Address to store the part identification number.

### Return values

kStatus\_IAP\_Success – Api has been executed successfully.

*status\_t* IAP\_ReadBootCodeVersion(uint32\_t \*bootCodeVersion)

Read boot code version number.

This function is used to read the boot code version number.

note Boot code version is two 32-bit words. Word 0 is the major version, word 1 is the minor version.

**Parameters**

- `bootCodeVersion` – Address to store the boot code version.

**Return values**

`kStatus_IAP_Success` – Api has been executed successfully.

`void IAP_ReinvokeISP(uint8_t ispType, uint32_t *status)`

Reinvoke ISP.

This function is used to invoke the boot loader in ISP mode. It maps boot vectors and configures the peripherals for ISP.

note The error response will be returned when IAP is disabled or an invalid ISP type selection appears. The call won't return unless an error occurs, so there can be no status code.

**Parameters**

- `ispType` – ISP type selection.
- `status` – store the possible status.

**Return values**

`kStatus_IAP_ReinvokeISPConfig` – reinvoke configuration error.

`status_t IAP_ReadUniqueID(uint32_t *uniqueID)`

Read unique identification.

This function is used to read the unique id.

**Parameters**

- `uniqueID` – store the uniqueID.

**Return values**

`kStatus_IAP_Success` – Api has been executed successfully.

`FSL_IAP_DRIVER_VERSION`

iap status codes.

*Values:*

enumerator `kStatus_IAP_Success`

Api is executed successfully

enumerator `kStatus_IAP_InvalidCommand`

Invalid command

enumerator `kStatus_IAP_SrcAddrError`

Source address is not on word boundary

enumerator `kStatus_IAP_DstAddrError`

Destination address is not on a correct boundary

enumerator `kStatus_IAP_SrcAddrNotMapped`

Source address is not mapped in the memory map

enumerator `kStatus_IAP_DstAddrNotMapped`

Destination address is not mapped in the memory map

enumerator kStatus\_IAP\_CountError

Byte count is not multiple of 4 or is not a permitted value

enumerator kStatus\_IAP\_InvalidSector

Sector/page number is invalid or end sector/page number is greater than start sector/page number

enumerator kStatus\_IAP\_SectorNotblank

One or more sectors are not blank

enumerator kStatus\_IAP\_NotPrepared

Command to prepare sector for write operation has not been executed

enumerator kStatus\_IAP\_CompareError

Destination and source memory contents do not match

enumerator kStatus\_IAP\_Busy

Flash programming hardware interface is busy

enumerator kStatus\_IAP\_ParamError

Insufficient number of parameters or invalid parameter

enumerator kStatus\_IAP\_AddrError

Address is not on word boundary

enumerator kStatus\_IAP\_AddrNotMapped

Address is not mapped in the memory map

enumerator kStatus\_IAP\_NoPower

Flash memory block is powered down

enumerator kStatus\_IAP\_NoClock

Flash memory block or controller is not clocked

enumerator kStatus\_IAP\_ReinvokeISPConfig

Reinvoke configuration error

enum \_iap\_commands

iap command codes.

*Values:*

enumerator kIapCmd\_IAP\_ReadFactorySettings

Read the factory settings

enumerator kIapCmd\_IAP\_PrepareSectorforWrite

Prepare Sector for write

enumerator kIapCmd\_IAP\_CopyRamToFlash

Copy RAM to flash

enumerator kIapCmd\_IAP\_EraseSector

Erase Sector

enumerator kIapCmd\_IAP\_BlankCheckSector

Blank check sector

enumerator kIapCmd\_IAP\_ReadPartId

Read part id

enumerator kIapCmd\_IAP\_Read\_BootromVersion

Read bootrom version

enumerator kIapCmd\_IAP\_Compare  
Compare

enumerator kIapCmd\_IAP\_ReinvokeISP  
Reinvoke ISP

enumerator kIapCmd\_IAP\_ReadUid  
Read Uid

enumerator kIapCmd\_IAP\_ErasePage  
Erase Page

enumerator kIapCmd\_IAP\_ReadSignature  
Read Signature

enumerator kIapCmd\_IAP\_ExtendedReadSignature  
Extended Read Signature

enumerator kIapCmd\_IAP\_ReadEEPROMPage  
Read EEPROM page

enumerator kIapCmd\_IAP\_WriteEEPROMPage  
Write EEPROM page

enum \_flash\_access\_time  
Flash memory access time.  
*Values:*

enumerator kFlash\_IAP\_OneSystemClockTime

enumerator kFlash\_IAP\_TwoSystemClockTime  
1 system clock flash access time

enumerator kFlash\_IAP\_ThreeSystemClockTime  
2 system clock flash access time

## 2.23 INPUTMUX: Input Multiplexing Driver

enum \_inputmux\_connection\_t  
INPUTMUX connections type.  
*Values:*

enumerator kINPUTMUX\_SctGpi0ToSct0  
SCT INMUX.

enumerator kINPUTMUX\_SctGpi1ToSct0

enumerator kINPUTMUX\_SctGpi2ToSct0

enumerator kINPUTMUX\_SctGpi3ToSct0

enumerator kINPUTMUX\_SctGpi4ToSct0

enumerator kINPUTMUX\_SctGpi5ToSct0

enumerator kINPUTMUX\_SctGpi6ToSct0

enumerator kINPUTMUX\_SctGpi7ToSct0

enumerator kINPUTMUX\_\_T0Out0ToSct0  
enumerator kINPUTMUX\_\_T1Out0ToSct0  
enumerator kINPUTMUX\_\_T2Out0ToSct0  
enumerator kINPUTMUX\_\_T3Out0ToSct0  
enumerator kINPUTMUX\_\_T4Out0ToSct0  
enumerator kINPUTMUX\_\_AdcThcmpIrqToSct0  
enumerator kINPUTMUX\_\_GpioIntBmatchToSct0  
enumerator kINPUTMUX\_\_Usb0FrameToggleToSct0  
enumerator kINPUTMUX\_\_Usb1FrameToggleToSct0  
enumerator kINPUTMUX\_\_ArmTxevToSct0  
enumerator kINPUTMUX\_\_DebugHaltedToSct0  
enumerator kINPUTMUX\_\_SmartCard0TxActivreToSct0  
enumerator kINPUTMUX\_\_SmartCard0RxActivreToSct0  
enumerator kINPUTMUX\_\_SmartCard1TxActivreToSct0  
enumerator kINPUTMUX\_\_SmartCard1RxActivreToSct0  
enumerator kINPUTMUX\_\_I2s6SclkToSct0  
enumerator kINPUTMUX\_\_I2s7clkToSct0

Frequency measure.

enumerator kINPUTMUX\_\_MainOscToFreqmeas  
enumerator kINPUTMUX\_\_Fro12MhzToFreqmeas  
enumerator kINPUTMUX\_\_Fro96MhzToFreqmeas  
enumerator kINPUTMUX\_\_WdtOscToFreqmeas  
enumerator kINPUTMUX\_\_32KhzOscToFreqmeas  
enumerator kINPUTMUX\_\_MainClkToFreqmeas  
enumerator kINPUTMUX\_\_FreqmeGpioClk\_a  
enumerator kINPUTMUX\_\_FreqmeGpioClk\_b

Pin Interrupt.

enumerator kINPUTMUX\_\_GpioPort0Pin0ToPintsel  
enumerator kINPUTMUX\_\_GpioPort0Pin1ToPintsel  
enumerator kINPUTMUX\_\_GpioPort0Pin2ToPintsel  
enumerator kINPUTMUX\_\_GpioPort0Pin3ToPintsel  
enumerator kINPUTMUX\_\_GpioPort0Pin4ToPintsel  
enumerator kINPUTMUX\_\_GpioPort0Pin5ToPintsel  
enumerator kINPUTMUX\_\_GpioPort0Pin6ToPintsel

enumerator kINPUTMUX\_GpioPort0Pin7ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin8ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin9ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin10ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin11ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin12ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin13ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin14ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin15ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin16ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin17ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin18ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin19ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin20ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin21ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin22ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin23ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin24ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin25ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin26ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin27ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin28ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin29ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin30ToPintsel  
enumerator kINPUTMUX\_GpioPort0Pin31ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin0ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin1ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin2ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin3ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin4ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin5ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin6ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin7ToPintsel



enumerator kINPUTMUX\_GpioPort1Pin8ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin9ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin10ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin11ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin12ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin13ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin14ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin15ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin16ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin17ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin18ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin19ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin20ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin21ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin22ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin23ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin24ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin25ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin26ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin27ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin28ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin29ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin30ToPintsel  
enumerator kINPUTMUX\_GpioPort1Pin31ToPintsel

DMA ITRIG.

enumerator kINPUTMUX\_Adc0SeqaIrqToDma  
enumerator kINPUTMUX\_Adc0SeqbIrqToDma  
enumerator kINPUTMUX\_Sct0DmaReq0ToDma  
enumerator kINPUTMUX\_Sct0DmaReq1ToDma  
enumerator kINPUTMUX\_PinInt0ToDma  
enumerator kINPUTMUX\_PinInt1ToDma  
enumerator kINPUTMUX\_PinInt2ToDma  
enumerator kINPUTMUX\_PinInt3ToDma

enumerator kINPUTMUX\_Ctimer0M0ToDma  
enumerator kINPUTMUX\_Ctimer0M1ToDma  
enumerator kINPUTMUX\_Ctimer1M0ToDma  
enumerator kINPUTMUX\_Ctimer1M1ToDma  
enumerator kINPUTMUX\_Ctimer2M0ToDma  
enumerator kINPUTMUX\_Ctimer2M1ToDma  
enumerator kINPUTMUX\_Ctimer3M0ToDma  
enumerator kINPUTMUX\_Ctimer3M1ToDma  
enumerator kINPUTMUX\_Ctimer4M0ToDma  
enumerator kINPUTMUX\_Ctimer4M1ToDma  
enumerator kINPUTMUX\_Otrig0ToDma  
enumerator kINPUTMUX\_Otrig1ToDma  
enumerator kINPUTMUX\_Otrig2ToDma  
enumerator kINPUTMUX\_Otrig3ToDma  
enumerator AES256\_INPUT\_DMA\_REQ  
enumerator AES256\_OUTPUT\_DMA\_REQ  
DMA OTRIG.  
enumerator kINPUTMUX\_DmaFlexcomm0RxTrigoutToTriginChannels  
enumerator kINPUTMUX\_DmaFlexcomm0TxTrigoutToTriginChannels  
enumerator kINPUTMUX\_DmaFlexcomm1RxTrigoutToTriginChannels  
enumerator kINPUTMUX\_DmaFlexcomm1TxTrigoutToTriginChannels  
enumerator kINPUTMUX\_DmaFlexcomm2RxTrigoutToTriginChannels  
enumerator kINPUTMUX\_DmaFlexcomm2TxTrigoutToTriginChannels  
enumerator kINPUTMUX\_DmaFlexcomm3RxTrigoutToTriginChannels  
enumerator kINPUTMUX\_DmaFlexcomm3TxTrigoutToTriginChannels  
enumerator kINPUTMUX\_DmaFlexcomm4RxTrigoutToTriginChannels  
enumerator kINPUTMUX\_DmaFlexcomm4TxTrigoutToTriginChannels  
enumerator kINPUTMUX\_DmaFlexcomm5RxTrigoutToTriginChannels  
enumerator kINPUTMUX\_DmaFlexcomm5TxTrigoutToTriginChannels  
enumerator kINPUTMUX\_DmaFlexcomm6RxTrigoutToTriginChannels  
enumerator kINPUTMUX\_DmaFlexcomm6TxTrigoutToTriginChannels  
enumerator kINPUTMUX\_DmaFlexcomm7RxTrigoutToTriginChannels  
enumerator kINPUTMUX\_DmaFlexcomm7TxTrigoutToTriginChannels

```
enumerator kINPUTMUX_DmaDmic0Ch0TrigoutToTriginChannels
enumerator kINPUTMUX_DmaDmic0Ch1TrigoutToTriginChannels
enumerator kINPUTMUX_DmaSpifi0TrigoutToTriginChannels
enumerator kINPUTMUX_DmaSha_TrigoutToTriginChannels
enumerator kINPUTMUX_DmaFlexcomm8RxTrigoutToTriginChannels
enumerator kINPUTMUX_DmaFlexcomm8TxTrigoutToTriginChannels
enumerator kINPUTMUX_DmaFlexcomm9RxTrigoutToTriginChannels
enumerator kINPUTMUX_DmaFlexcomm9TxTrigoutToTriginChannels
enumerator kINPUTMUX_DmaSmartcard0RxTrigoutToTriginChannels
enumerator kINPUTMUX_DmaSmartcard0TxTrigoutToTriginChannels
enumerator kINPUTMUX_DmaSmartcard1RxTrigoutToTriginChannels
enumerator kINPUTMUX_DmaSmartcard1TxTrigoutToTriginChannels
enumerator kINPUTMUX_DmaFlexcomm10RxTrigoutToTriginChannels
enumerator kINPUTMUX_DmaFlexcomm10TxTrigoutToTriginChannels
```

```
typedef enum inputmux_connection_t inputmux_connection_t
    INPUTMUX connections type.
```

```
SCT0_PMUX_ID
```

Periphinmux IDs.

```
PINTSEL_PMUX_ID
```

```
DMA_TRIG0_PMUX_ID
```

```
DMA_OTRIG_PMUX_ID
```

```
FREQMEAS_PMUX_ID
```

```
PMUX_SHIFT
```

```
FSL_INPUTMUX_DRIVER_VERSION
```

Group interrupt driver version for SDK.

```
void INPUTMUX_Init(void *base)
```

Initialize INPUTMUX peripheral.

This function enables the INPUTMUX clock.

#### Parameters

- base – Base address of the INPUTMUX peripheral.

#### Return values

None. –

```
void INPUTMUX_AttachSignal(void *base, uint16_t index, inputmux_connection_t connection)
```

Attaches a signal.

This function attaches multiplexed signals from INPUTMUX to target signals. For example, to attach GPIO PORT0 Pin 5 to PINT peripheral, do the following:

```
INPUTMUX_AttachSignal(INPUTMUX, 2, kINPUTMUX_GpioPort0Pin5ToPintsel);
```

In this example, INTMUX has 8 registers for PINT, PINT\_SELO~PINT\_SEL7. With parameter index specified as 2, this function configures register PINT\_SEL2.

#### Parameters

- base – Base address of the INPUTMUX peripheral.
- index – The serial number of destination register in the group of INPUTMUX registers with same name.
- connection – Applies signal from source signals collection to target signal.

#### Return values

None. –

```
void INPUTMUX_Deinit(void *base)
```

Deinitialize INPUTMUX peripheral.

This function disables the INPUTMUX clock.

#### Parameters

- base – Base address of the INPUTMUX peripheral.

#### Return values

None. –

## 2.24 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 UART status codes.

enumerator kStatusGroup\_LPSCI  
Group number for LPSCI status codes.

enumerator kStatusGroup\_LPUART  
Group number for LPUART status codes.

enumerator kStatusGroup\_SPI  
Group number for SPI status code.

enumerator kStatusGroup\_XRDC  
Group number for XRDC status code.

enumerator kStatusGroup\_SEMA42  
Group number for SEMA42 status code.

enumerator kStatusGroup\_SDHC  
Group number for SDHC status code

enumerator kStatusGroup\_SDMMC  
Group number for SDMMC status code

enumerator kStatusGroup\_SAI  
Group number for SAI status code

enumerator kStatusGroup\_MCG  
Group number for MCG status codes.

enumerator kStatusGroup\_SCG  
Group number for SCG status codes.

enumerator kStatusGroup\_SDSPI  
Group number for SDSPI status codes.

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

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

enumerator kStatusGroup\_FLASHIAP  
Group number for FLASHIAP status codes

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

enumerator kStatusGroup\_I2S  
Group number for I2S status codes

enumerator kStatusGroup\_IUART  
Group number for IUART status codes

enumerator kStatusGroup\_CSI  
Group number for CSI status codes

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

enumerator kStatusGroup\_SDRAMC  
Group number for SDRAMC status codes.

enumerator kStatusGroup\_POWER  
Group number for POWER status codes.

enumerator kStatusGroup\_ENET  
Group number for ENET status codes.

enumerator kStatusGroup\_PHY  
Group number for PHY status codes.

enumerator kStatusGroup\_TRGMUX  
Group number for TRGMUX status codes.

enumerator kStatusGroup\_SMARTCARD  
Group number for SMARTCARD status codes.

enumerator kStatusGroup\_LMEM  
Group number for LMEM status codes.

enumerator kStatusGroup\_QSPI  
Group number for QSPI status codes.

enumerator kStatusGroup\_DMA  
Group number for DMA status codes.

enumerator kStatusGroup\_EDMA  
Group number for EDMA status codes.

enumerator kStatusGroup\_DMAMGR  
Group number for DMAMGR status codes.

enumerator kStatusGroup\_FLEXCAN  
Group number for FlexCAN status codes.

enumerator kStatusGroup\_LTC  
Group number for LTC status codes.

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

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

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

enumerator kStatusGroup\_DMIC  
Group number for DMIC status codes.

enumerator kStatusGroup\_SDIF  
Group number for SDIF status codes.

enumerator kStatusGroup\_SPIFI  
Group number for SPIFI status codes.



enumerator kStatusGroup\_OTP  
Group number for OTP status codes.

enumerator kStatusGroup\_MCAN  
Group number for MCAN status codes.

enumerator kStatusGroup\_CAAM  
Group number for CAAM status codes.

enumerator kStatusGroup\_ECSPI  
Group number for ECSPI status codes.

enumerator kStatusGroup\_USDHC  
Group number for USDHC status codes.

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

enumerator kStatusGroup\_DCP  
Group number for DCP status codes.

enumerator kStatusGroup\_MSCAN  
Group number for MSCAN status codes.

enumerator kStatusGroup\_ESAI  
Group number for ESAI status codes.

enumerator kStatusGroup\_FLEXSPI  
Group number for FLEXSPI status codes.

enumerator kStatusGroup\_MMDC  
Group number for MMDC status codes.

enumerator kStatusGroup\_PDM  
Group number for MIC status codes.

enumerator kStatusGroup\_SDMA  
Group number for SDMA status codes.

enumerator kStatusGroup\_ICS  
Group number for ICS status codes.

enumerator kStatusGroup\_SPDIF  
Group number for SPDIF status codes.

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

enumerator kStatusGroup\_HASHCRYPT  
Group number for Hashcrypt status codes

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

enumerator kStatusGroup\_I3C  
Group number for I3C status codes

enumerator kStatusGroup\_LPC\_I2C\_1  
Group number for LPC\_I2C\_1 status codes.

enumerator kStatusGroup\_NOTIFIER  
Group number for NOTIFIER status codes.

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

enumerator `kStatusGroup__SEMC`  
Group number for SEMC status codes.

enumerator `kStatusGroup__ApplicationRangeStart`  
Starting number for application groups.

enumerator `kStatusGroup__IAP`  
Group number for IAP status codes

enumerator `kStatusGroup__SFA`  
Group number for SFA status codes

enumerator `kStatusGroup__SPC`  
Group number for SPC status codes.

enumerator `kStatusGroup__PUF`  
Group number for PUF status codes.

enumerator `kStatusGroup__TOUCH_PANEL`  
Group number for touch panel status codes

enumerator `kStatusGroup__VBAT`  
Group number for VBAT status codes

enumerator `kStatusGroup__XSPI`  
Group number for XSPI status codes

enumerator `kStatusGroup__PNGDEC`  
Group number for PNGDEC status codes

enumerator `kStatusGroup__JPEGDEC`  
Group number for JPEGDEC status codes

enumerator `kStatusGroup__AUDMIX`  
Group number for AUDMIX status codes

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

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

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

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

enumerator `kStatusGroup__HAL_I2C`  
Group number for HAL I2C status codes.

enumerator `kStatusGroup__HAL_FLASH`  
Group number for HAL FLASH status codes.

enumerator `kStatusGroup__HAL_PWM`  
Group number for HAL PWM status codes.

enumerator `kStatusGroup__HAL_RNG`  
Group number for HAL RNG status codes.

enumerator kStatusGroup\_HAL\_I2S  
Group number for HAL I2S status codes.

enumerator kStatusGroup\_HAL\_ADC\_SENSOR  
Group number for HAL ADC SENSOR status codes.

enumerator kStatusGroup\_TIMERMANAGER  
Group number for TiMER MANAGER status codes.

enumerator kStatusGroup\_SERIALMANAGER  
Group number for SERIAL MANAGER status codes.

enumerator kStatusGroup\_LED  
Group number for LED status codes.

enumerator kStatusGroup\_BUTTON  
Group number for BUTTON status codes.

enumerator kStatusGroup\_EXTERN\_EEPROM  
Group number for EXTERN EEPROM status codes.

enumerator kStatusGroup\_SHELL  
Group number for SHELL status codes.

enumerator kStatusGroup\_MEM\_MANAGER  
Group number for MEM MANAGER status codes.

enumerator kStatusGroup\_LIST  
Group number for List status codes.

enumerator kStatusGroup\_OSA  
Group number for OSA status codes.

enumerator kStatusGroup\_COMMON\_TASK  
Group number for Common task status codes.

enumerator kStatusGroup\_MSG  
Group number for messaging status codes.

enumerator kStatusGroup\_SDK\_OCOTP  
Group number for OCOTP status codes.

enumerator kStatusGroup\_SDK\_FLEXSPINOR  
Group number for FLEXSPINOR status codes.

enumerator kStatusGroup\_CODEC  
Group number for codec status codes.

enumerator kStatusGroup\_ASRC  
Group number for codec status ASRC.

enumerator kStatusGroup\_OTFAD  
Group number for codec status codes.

enumerator kStatusGroup\_SDIOSLV  
Group number for SDIOSLV status codes.

enumerator kStatusGroup\_MECC  
Group number for MECC status codes.

enumerator kStatusGroup\_ENET\_QOS  
Group number for ENET\_QOS status codes.

enumerator kStatusGroup\_LOG  
Group number for LOG status codes.

enumerator kStatusGroup\_I3CBUS  
Group number for I3CBUS status codes.

enumerator kStatusGroup\_QSCI  
Group number for QSCI status codes.

enumerator kStatusGroup\_ELEMU  
Group number for ELEMU status codes.

enumerator kStatusGroup\_QUEUEDSPI  
Group number for QSPI status codes.

enumerator kStatusGroup\_POWER\_MANAGER  
Group number for POWER\_MANAGER status codes.

enumerator kStatusGroup\_IPED  
Group number for IPED status codes.

enumerator kStatusGroup\_ELS\_PKC  
Group number for ELS PKC status codes.

enumerator kStatusGroup\_CSS\_PKC  
Group number for CSS PKC status codes.

enumerator kStatusGroup\_HOSTIF  
Group number for HOSTIF status codes.

enumerator kStatusGroup\_CLIF  
Group number for CLIF status codes.

enumerator kStatusGroup\_BMA  
Group number for BMA status codes.

enumerator kStatusGroup\_NETC  
Group number for NETC status codes.

enumerator kStatusGroup\_ELE  
Group number for ELE status codes.

enumerator kStatusGroup\_GLIKEY  
Group number for GLIKEY status codes.

enumerator kStatusGroup\_AON\_POWER  
Group number for AON\_POWER status codes.

enumerator kStatusGroup\_AON\_COMMON  
Group number for AON\_COMMON status codes.

enumerator kStatusGroup\_ENDAT3  
Group number for ENDAT3 status codes.

enumerator kStatusGroup\_HIPERFACE  
Group number for HIPERFACE status codes.

enumerator kStatusGroup\_NPX  
Group number for NPX status codes.

enumerator kStatusGroup\_ELA\_CSEC  
Group number for ELA\_CSEC status codes.

enumerator kStatusGroup\_FLEXIO\_T\_FORMAT

Group number for T-format status codes.

enumerator kStatusGroup\_FLEXIO\_A\_FORMAT

Group number for A-format status codes.

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().

`void EnableDeepSleepIRQ(IRQn_Type interrupt)`

Enable specific interrupt for wake-up from deep-sleep mode.

Enable the interrupt for wake-up from deep sleep mode. Some interrupts are typically used in sleep mode only and will not occur during deep-sleep mode because relevant clocks are stopped. However, it is possible to enable those clocks (significantly increasing power consumption in the reduced power mode), making these wake-ups possible.

---

**Note:** This function also enables the interrupt in the NVIC (EnableIRQ() is called internally).

---

#### Parameters

- `interrupt` – The IRQ number.

`void DisableDeepSleepIRQ(IRQn_Type interrupt)`

Disable specific interrupt for wake-up from deep-sleep mode.

Disable the interrupt for wake-up from deep sleep mode. Some interrupts are typically used in sleep mode only and will not occur during deep-sleep mode because relevant clocks are stopped. However, it is possible to enable those clocks (significantly increasing power consumption in the reduced power mode), making these wake-ups possible.

---

**Note:** This function also disables the interrupt in the NVIC (DisableIRQ() is called internally).

---

#### Parameters

- `interrupt` – The IRQ number.

`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.

## 2.25 ADC: 12-bit SAR Analog-to-Digital Converter Driver

void ADC\_Init(ADC\_Type \*base, const adc\_config\_t \*config)

Initialize the ADC module.

### Parameters

- base – ADC peripheral base address.
- config – Pointer to configuration structure, see to adc\_config\_t.

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

Deinitialize the ADC module.

### Parameters

- base – ADC peripheral base address.

void ADC\_GetDefaultConfig(adc\_config\_t \*config)

Gets an available pre-defined settings for initial configuration.

This function initializes the initial configuration structure with an available settings. The default values are:

```
config->clockMode = kADC_ClockSynchronousMode;
config->clockDividerNumber = 0U;
config->resolution = kADC_Resolution12bit;
config->enableBypassCalibration = false;
config->sampleTimeNumber = 0U;
config->extendSampleTimeNumber = kADC_ExtendSampleTimeNotUsed;
```

### Parameters

- config – Pointer to configuration structure.

bool ADC\_DoSelfCalibration(ADC\_Type \*base)

Do the hardware self-calibration.

*Deprecated:*

Do not use this function. It has been superceded by ADC\_DoOffsetCalibration.

To calibrate the ADC, set the ADC clock to 500 kHz. In order to achieve the specified ADC accuracy, the A/D converter must be recalibrated, at a minimum, following every chip reset before initiating normal ADC operation.

**Parameters**

- `base` – ADC peripheral base address.

**Return values**

- `true` – Calibration succeed.
- `false` – Calibration failed.

`bool ADC_DoOffsetCalibration(ADC_Type *base, uint32_t frequency)`

Do the hardware offset-calibration.

To calibrate the ADC, set the ADC clock to no more then 30 MHz. In order to achieve the specified ADC accuracy, the A/D converter must be recalibrated, at a minimum, following every chip reset before initiating normal ADC operation.

**Parameters**

- `base` – ADC peripheral base address.
- `frequency` – The clock frequency that ADC operates at.

**Return values**

- `true` – Calibration succeed.
- `false` – Calibration failed.

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

Enable the conversion sequence A.

In order to avoid spuriously triggering the sequence, the trigger to conversion sequence should be ready before the sequence is ready. when the sequence is disabled, the trigger would be ignored. Also, it is suggested to disable the sequence during changing the sequence's setting.

**Parameters**

- `base` – ADC peripheral base address.
- `enable` – Switcher to enable the feature or not.

`void ADC_SetConvSeqAConfig(ADC_Type *base, const adc_conv_seq_config_t *config)`

Configure the conversion sequence A.

**Parameters**

- `base` – ADC peripheral base address.
- `config` – Pointer to configuration structure, see to *adc\_conv\_seq\_config\_t*.

`static inline void ADC_DoSoftwareTriggerConvSeqA(ADC_Type *base)`

Do trigger the sequence's conversion by software.

**Parameters**

- `base` – ADC peripheral base address.

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

Enable the burst conversion of sequence A.

Enable the burst mode would cause the conversion sequence to be continuously cycled through. Other triggers would be ignored while this mode is enabled. Repeated conversions could be halted by disabling this mode. And the sequence currently in process will be completed before conversions are terminated. Note that a new sequence could begin just before the burst mode is disabled.

**Parameters**

- base – ADC peripheral base address.
- enable – Switcher to enable this feature.

static inline void ADC\_SetConvSeqAHighPriority(ADC\_Type \*base)

Set the high priority for conversion sequence A.

**Parameters**

- base – ADC peripheral bass address.

static inline void ADC\_EnableConvSeqB(ADC\_Type \*base, bool enable)

Enable the conversion sequence B.

In order to avoid spuriously triggering the sequence, the trigger to conversion sequence should be ready before the sequence is ready. when the sequence is disabled, the trigger would be ignored. Also, it is suggested to disable the sequence during changing the sequence's setting.

**Parameters**

- base – ADC peripheral base address.
- enable – Switcher to enable the feature or not.

void ADC\_SetConvSeqBConfig(ADC\_Type \*base, const *adc\_conv\_seq\_config\_t* \*config)

Configure the conversion sequence B.

**Parameters**

- base – ADC peripheral base address.
- config – Pointer to configuration structure, see to *adc\_conv\_seq\_config\_t*.

static inline void ADC\_DoSoftwareTriggerConvSeqB(ADC\_Type \*base)

Do trigger the sequence's conversion by software.

**Parameters**

- base – ADC peripheral base address.

static inline void ADC\_EnableConvSeqBBurstMode(ADC\_Type \*base, bool enable)

Enable the burst conversion of sequence B.

Enable the burst mode would cause the conversion sequence to be continuously cycled through. Other triggers would be ignored while this mode is enabled. Repeated conversions could be halted by disabling this mode. And the sequence currently in process will be completed before cnversions are terminated. Note that a new sequence could begin just before the burst mode is disabled.

**Parameters**

- base – ADC peripheral base address.
- enable – Switcher to enable this feature.

static inline void ADC\_SetConvSeqBHighPriority(ADC\_Type \*base)

Set the high priority for conversion sequence B.

**Parameters**

- base – ADC peripheral bass address.

bool ADC\_GetConvSeqAGlobalConversionResult(ADC\_Type \*base, *adc\_result\_info\_t* \*info)

Get the global ADC conversion infomation of sequence A.

**Parameters**

- `base` – ADC peripheral base address.
- `info` – Pointer to information structure, see to `adc_result_info_t`;

**Return values**

- `true` – The conversion result is ready.
- `false` – The conversion result is not ready yet.

`bool ADC_GetConvSeqBGlobalConversionResult(ADC_Type *base, adc_result_info_t *info)`

Get the global ADC conversion information of sequence B.

**Parameters**

- `base` – ADC peripheral base address.
- `info` – Pointer to information structure, see to `adc_result_info_t`;

**Return values**

- `true` – The conversion result is ready.
- `false` – The conversion result is not ready yet.

`bool ADC_GetChannelConversionResult(ADC_Type *base, uint32_t channel, adc_result_info_t *info)`

Get the channel's ADC conversion completed under each conversion sequence.

**Parameters**

- `base` – ADC peripheral base address.
- `channel` – The indicated channel number.
- `info` – Pointer to information structure, see to `adc_result_info_t`;

**Return values**

- `true` – The conversion result is ready.
- `false` – The conversion result is not ready yet.

`static inline void ADC_SetThresholdPair0(ADC_Type *base, uint32_t lowValue, uint32_t highValue)`

Set the threshold pair 0 with low and high value.

**Parameters**

- `base` – ADC peripheral base address.
- `lowValue` – LOW threshold value.
- `highValue` – HIGH threshold value.

`static inline void ADC_SetThresholdPair1(ADC_Type *base, uint32_t lowValue, uint32_t highValue)`

Set the threshold pair 1 with low and high value.

**Parameters**

- `base` – ADC peripheral base address.
- `lowValue` – LOW threshold value. The available value is with 12-bit.
- `highValue` – HIGH threshold value. The available value is with 12-bit.

`static inline void ADC_SetChannelWithThresholdPair0(ADC_Type *base, uint32_t channelMask)`

Set given channels to apply the threshold pair 0.

**Parameters**

- `base` – ADC peripheral base address.

- channelMask – Indicated channels' mask.

static inline void ADC\_SetChannelWithThresholdPair1(ADC\_Type \*base, uint32\_t channelMask)

Set given channels to apply the threshold pare 1.

#### Parameters

- base – ADC peripheral base address.
- channelMask – Indicated channels' mask.

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

Enable interrupts for conversion sequences.

#### Parameters

- base – ADC peripheral base address.
- mask – Mask of interrupt mask value for global block except each channel, see to `_adc_interrupt_enable`.

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

Disable interrupts for conversion sequence.

#### Parameters

- base – ADC peripheral base address.
- mask – Mask of interrupt mask value for global block except each channel, see to `_adc_interrupt_enable`.

static inline void ADC\_EnableThresholdCompareInterrupt(ADC\_Type \*base, uint32\_t channel,  
*adc\_threshold\_interrupt\_mode\_t mode*)

Enable the interrupt of threshold compare event for each channel.

#### Parameters

- base – ADC peripheral base address.
- channel – Channel number.
- mode – Interrupt mode for threshold compare event, see to `adc_threshold_interrupt_mode_t`.

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

Get status flags of ADC module.

#### Parameters

- base – ADC peripheral base address.

#### Returns

Mask of status flags of module, see to `_adc_status_flags`.

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

Clear status flags of ADC module.

#### Parameters

- base – ADC peripheral base address.
- mask – Mask of status flags of module, see to `_adc_status_flags`.

FSL\_ADC\_DRIVER\_VERSION

ADC driver version 2.6.0.

enum `_adc_status_flags`

Flags.

Values:

enumerator kADC\_ThresholdCompareFlagOnChn0

Threshold comparison event on Channel 0.

enumerator kADC\_ThresholdCompareFlagOnChn1

Threshold comparison event on Channel 1.

enumerator kADC\_ThresholdCompareFlagOnChn2

Threshold comparison event on Channel 2.

enumerator kADC\_ThresholdCompareFlagOnChn3

Threshold comparison event on Channel 3.

enumerator kADC\_ThresholdCompareFlagOnChn4

Threshold comparison event on Channel 4.

enumerator kADC\_ThresholdCompareFlagOnChn5

Threshold comparison event on Channel 5.

enumerator kADC\_ThresholdCompareFlagOnChn6

Threshold comparison event on Channel 6.

enumerator kADC\_ThresholdCompareFlagOnChn7

Threshold comparison event on Channel 7.

enumerator kADC\_ThresholdCompareFlagOnChn8

Threshold comparison event on Channel 8.

enumerator kADC\_ThresholdCompareFlagOnChn9

Threshold comparison event on Channel 9.

enumerator kADC\_ThresholdCompareFlagOnChn10

Threshold comparison event on Channel 10.

enumerator kADC\_ThresholdCompareFlagOnChn11

Threshold comparison event on Channel 11.

enumerator kADC\_OverrunFlagForChn0

Mirror the OVERRUN status flag from the result register for ADC channel 0.

enumerator kADC\_OverrunFlagForChn1

Mirror the OVERRUN status flag from the result register for ADC channel 1.

enumerator kADC\_OverrunFlagForChn2

Mirror the OVERRUN status flag from the result register for ADC channel 2.

enumerator kADC\_OverrunFlagForChn3

Mirror the OVERRUN status flag from the result register for ADC channel 3.

enumerator kADC\_OverrunFlagForChn4

Mirror the OVERRUN status flag from the result register for ADC channel 4.

enumerator kADC\_OverrunFlagForChn5

Mirror the OVERRUN status flag from the result register for ADC channel 5.

enumerator kADC\_OverrunFlagForChn6

Mirror the OVERRUN status flag from the result register for ADC channel 6.

enumerator kADC\_OverrunFlagForChn7

Mirror the OVERRUN status flag from the result register for ADC channel 7.

enumerator kADC\_OverrunFlagForChn8

Mirror the OVERRUN status flag from the result register for ADC channel 8.

enumerator kADC\_OverrunFlagForChn9

Mirror the OVERRUN status flag from the result register for ADC channel 9.

enumerator kADC\_OverrunFlagForChn10

Mirror the OVERRUN status flag from the result register for ADC channel 10.

enumerator kADC\_OverrunFlagForChn11

Mirror the OVERRUN status flag from the result register for ADC channel 11.

enumerator kADC\_GlobalOverrunFlagForSeqA

Mirror the global OVERRUN status flag for conversion sequence A.

enumerator kADC\_GlobalOverrunFlagForSeqB

Mirror the global OVERRUN status flag for conversion sequence B.

enumerator kADC\_ConvSeqAInterruptFlag

Sequence A interrupt/DMA trigger.

enumerator kADC\_ConvSeqBInterruptFlag

Sequence B interrupt/DMA trigger.

enumerator kADC\_ThresholdCompareInterruptFlag

Threshold comparison interrupt flag.

enumerator kADC\_OverrunInterruptFlag

Overrun interrupt flag.

enum \_adc\_interrupt\_enable

Interrupts.

---

**Note:** Not all the interrupt options are listed here

---

*Values:*

enumerator kADC\_ConvSeqAInterruptEnable

Enable interrupt upon completion of each individual conversion in sequence A, or entire sequence.

enumerator kADC\_ConvSeqBInterruptEnable

Enable interrupt upon completion of each individual conversion in sequence B, or entire sequence.

enumerator kADC\_OverrunInterruptEnable

Enable the detection of an overrun condition on any of the channel data registers will cause an overrun interrupt/DMA trigger.

enum \_adc\_clock\_mode

Define selection of clock mode.

*Values:*

enumerator kADC\_ClockSynchronousMode

The ADC clock would be derived from the system clock based on “clockDividerNumber”.

enumerator kADC\_ClockAsynchronousMode

The ADC clock would be based on the SYSCON block’s divider.

enum \_adc\_resolution

Define selection of resolution.

*Values:*

enumerator kADC\_Resolution6bit  
6-bit resolution.

enumerator kADC\_Resolution8bit  
8-bit resolution.

enumerator kADC\_Resolution10bit  
10-bit resolution.

enumerator kADC\_Resolution12bit  
12-bit resolution.

enum \_adc\_voltage\_range  
Define range of the analog supply voltage VDDA.

*Values:*

enumerator kADC\_HighVoltageRange

enumerator kADC\_LowVoltageRange

enum \_adc\_trigger\_polarity  
Define selection of polarity of selected input trigger for conversion sequence.

*Values:*

enumerator kADC\_TriggerPolarityNegativeEdge  
A negative edge launches the conversion sequence on the trigger(s).

enumerator kADC\_TriggerPolarityPositiveEdge  
A positive edge launches the conversion sequence on the trigger(s).

enum \_adc\_priority  
Define selection of conversion sequence's priority.

*Values:*

enumerator kADC\_PriorityLow  
This sequence would be preempted when another sequence is started.

enumerator kADC\_PriorityHigh  
This sequence would preempt other sequence even when it is started.

enum \_adc\_seq\_interrupt\_mode  
Define selection of conversion sequence's interrupt.

*Values:*

enumerator kADC\_InterruptForEachConversion  
The sequence interrupt/DMA trigger will be set at the end of each individual ADC conversion inside this conversion sequence.

enumerator kADC\_InterruptForEachSequence  
The sequence interrupt/DMA trigger will be set when the entire set of this sequence conversions completes.

enum \_adc\_threshold\_compare\_status  
Define status of threshold compare result.

*Values:*

enumerator kADC\_ThresholdCompareInRange  
LOW threshold <= conversion value <= HIGH threshold.



enumerator kADC\_ThresholdCompareBelowRange  
conversion value < LOW threshold.

enumerator kADC\_ThresholdCompareAboveRange  
conversion value > HIGH threshold.

enum \_adc\_threshold\_crossing\_status  
Define status of threshold crossing detection result.

*Values:*

enumerator kADC\_ThresholdCrossingNoDetected  
No threshold Crossing detected.

enumerator kADC\_ThresholdCrossingDownward  
Downward Threshold Crossing detected.

enumerator kADC\_ThresholdCrossingUpward  
Upward Threshold Crossing Detected.

enum \_adc\_threshold\_interrupt\_mode  
Define interrupt mode for threshold compare event.

*Values:*

enumerator kADC\_ThresholdInterruptDisabled  
Threshold comparison interrupt is disabled.

enumerator kADC\_ThresholdInterruptOnOutside  
Threshold comparison interrupt is enabled on outside threshold.

enumerator kADC\_ThresholdInterruptOnCrossing  
Threshold comparison interrupt is enabled on crossing threshold.

enum \_adc\_inforesultshift  
Define the info result mode of different resolution.

*Values:*

enumerator kADC\_Resolution12bitInfoResultShift  
Info result shift of Resolution12bit.

enumerator kADC\_Resolution10bitInfoResultShift  
Info result shift of Resolution10bit.

enumerator kADC\_Resolution8bitInfoResultShift  
Info result shift of Resolution8bit.

enumerator kADC\_Resolution6bitInfoResultShift  
Info result shift of Resolution6bit.

enum \_adc\_tempsensor\_common\_mode  
Define common modes for Temperature sensor.

*Values:*

enumerator kADC\_HighNegativeOffsetAdded  
Temperature sensor common mode: high negative offset added.

enumerator kADC\_IntermediateNegativeOffsetAdded  
Temperature sensor common mode: intermediate negative offset added.

enumerator kADC\_NoOffsetAdded  
Temperature sensor common mode: no offset added.

enumerator kADC\_LowPositiveOffsetAdded

Temperature sensor common mode: low positive offset added.

enum \_adc\_second\_control

Define source impedance modes for GPADC control.

*Values:*

enumerator kADC\_Impedance621Ohm

Extend ADC sampling time according to source impedance 1: 0.621 kOhm.

enumerator kADC\_Impedance55kOhm

Extend ADC sampling time according to source impedance 20 (default): 55 kOhm.

enumerator kADC\_Impedance87kOhm

Extend ADC sampling time according to source impedance 31: 87 kOhm.

enumerator kADC\_NormalFunctionalMode

TEST mode: Normal functional mode.

enumerator kADC\_MultiplexeTestMode

TEST mode: Multiplexer test mode.

enumerator kADC\_ADCInUnityGainMode

TEST mode: ADC in unity gain mode.

typedef enum \_adc\_clock\_mode adc\_clock\_mode\_t

Define selection of clock mode.

typedef enum \_adc\_resolution adc\_resolution\_t

Define selection of resolution.

typedef enum \_adc\_voltage\_range adc\_vdda\_range\_t

Define range of the analog supply voltage VDDA.

typedef enum \_adc\_trigger\_polarity adc\_trigger\_polarity\_t

Define selection of polarity of selected input trigger for conversion sequence.

typedef enum \_adc\_priority adc\_priority\_t

Define selection of conversion sequence's priority.

typedef enum \_adc\_seq\_interrupt\_mode adc\_seq\_interrupt\_mode\_t

Define selection of conversion sequence's interrupt.

typedef enum \_adc\_threshold\_compare\_status adc\_threshold\_compare\_status\_t

Define status of threshold compare result.

typedef enum \_adc\_threshold\_crossing\_status adc\_threshold\_crossing\_status\_t

Define status of threshold crossing detection result.

typedef enum \_adc\_threshold\_interrupt\_mode adc\_threshold\_interrupt\_mode\_t

Define interrupt mode for threshold compare event.

typedef enum \_adc\_inforeresultshift adc\_inforeresult\_t

Define the info result mode of different resolution.

typedef enum \_adc\_tempsensor\_common\_mode adc\_tempsensor\_common\_mode\_t

Define common modes for Temperature sensor.

typedef enum \_adc\_second\_control adc\_second\_control\_t

Define source impedance modes for GPADC control.

```
typedef struct _adc_config adc_config_t
```

Define structure for configuring the block.

```
typedef struct _adc_conv_seq_config adc_conv_seq_config_t
```

Define structure for configuring conversion sequence.

```
typedef struct _adc_result_info adc_result_info_t
```

Define structure of keeping conversion result information.

```
struct _adc_config
```

*#include <fsl\_adc.h>* Define structure for configuring the block.

### Public Members

*adc\_clock\_mode\_t* clockMode

Select the clock mode for ADC converter.

*uint32\_t* clockDividerNumber

This field is only available when using kADC\_ClockSynchronousMode for “clockMode” field. The divider would be plused by 1 based on the value in this field. The available range is in 8 bits.

*adc\_resolution\_t* resolution

Select the conversion bits.

*bool* enableBypassCalibration

By default, a calibration cycle must be performed each time the chip is powered-up. Re-calibration may be warranted periodically - especially if operating conditions have changed. To enable this option would avoid the need to calibrate if offset error is not a concern in the application.

*uint32\_t* sampleTimeNumber

By default, with value as “0U”, the sample period would be 2.5 ADC clocks. Then, to plus the “sampleTimeNumber” value here. The available value range is in 3 bits.

*bool* enableLowPowerMode

If disable low-power mode, ADC remains activated even when no conversions are requested. If enable low-power mode, The ADC is automatically powered-down when no conversions are taking place.

*adc\_vdda\_range\_t* voltageRange

Configure the ADC for the appropriate operating range of the analog supply voltage VDDA. Failure to set the area correctly causes the ADC to return incorrect conversion results.

```
struct _adc_conv_seq_config
```

*#include <fsl\_adc.h>* Define structure for configuring conversion sequence.

### Public Members

*uint32\_t* channelMask

Selects which one or more of the ADC channels will be sampled and converted when this sequence is launched. The masked channels would be involved in current conversion sequence, beginning with the lowest-order. The available range is in 12-bit.

*uint32\_t* triggerMask

Selects which one or more of the available hardware trigger sources will cause this conversion sequence to be initiated. The available range is 6-bit.

*adc\_trigger\_polarity\_t* triggerPolarity

Select the trigger to launch conversion sequence.

bool enableSyncBypass

To enable this feature allows the hardware trigger input to bypass synchronization flip-flop stages and therefore shorten the time between the trigger input signal and the start of a conversion.

bool enableSingleStep

When enabling this feature, a trigger will launch a single conversion on the next channel in the sequence instead of the default response of launching an entire sequence of conversions.

*adc\_seq\_interrupt\_mode\_t* interruptMode

Select the interrupt/DMA trigger mode.

struct *\_adc\_result\_info*

*#include <fsl\_adc.h>* Define structure of keeping conversion result information.

### Public Members

uint32\_t result

Keep the conversion data value.

*adc\_threshold\_compare\_status\_t* thresholdCompareStatus

Keep the threshold compare status.

*adc\_threshold\_crossing\_status\_t* thresholdCorssingStatus

Keep the threshold crossing status.

uint32\_t channelNumber

Keep the channel number for this conversion.

bool overrunFlag

Keep the status whether the conversion is overrun or not.

## 2.26 ENET: Ethernet Driver

void ENET\_GetDefaultConfig(*enet\_config\_t* \*config)

Gets the ENET default configuration structure.

The purpose of this API is to get the default ENET configure structure for ENET\_Init(). User may use the initialized structure unchanged in ENET\_Init(), or modify some fields of the structure before calling ENET\_Init(). Example:

```
enet_config_t config;
ENET_GetDefaultConfig(&config);
```

### Parameters

- config – The ENET mac controller configuration structure pointer.

void ENET\_Init(ENET\_Type \*base, const *enet\_config\_t* \*config, uint8\_t \*macAddr, uint32\_t refclkSrc\_Hz)

Initializes the ENET module.

This function ungates the module clock and initializes it with the ENET basic configuration.

**Note:** As our transactional transmit API use the zero-copy transmit buffer. So there are two thing we emphasize here:

- a. Tx buffer free/requeue for application should be done in the Tx interrupt handler. Please set callback: kENET\_TxIntEvent with Tx buffer free/requeue process APIs.
  - b. The Tx interrupt is forced to open.
- 

#### Parameters

- base – ENET peripheral base address.
- config – ENET mac configuration structure pointer. The “enet\_config\_t” type mac configuration return from ENET\_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.
- macAddr – ENET mac address of Ethernet device. This MAC address should be provided.
- refclkSrc\_Hz – ENET input reference clock.

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

Deinitializes the ENET module.

This function gates the module clock and disables the ENET module.

#### Parameters

- base – ENET peripheral base address.

status\_t ENET\_DescriptorInit(ENET\_Type \*base, enet\_config\_t \*config, enet\_buffer\_config\_t \*bufferConfig)

Initialize for all ENET descriptors.

---

**Note:** This function finishes all Tx/Rx descriptors initialization. The descriptor initialization should be called after ENET\_Init().

---

#### Parameters

- base – ENET peripheral base address.
- config – The configuration for ENET.
- bufferConfig – All buffers configuration.

status\_t ENET\_RxBufferAllocAll(ENET\_Type \*base, enet\_handle\_t \*handle)

Allocates Rx buffers for all BDs. It's used for zero copy Rx. In zero copy Rx case, Rx buffers are dynamic. This function will populate initial buffers in all BDs for receiving. Then ENET\_GetRxFrame() is used to get Rx frame with zero copy, it will allocate new buffer to replace the buffer in BD taken by application, application should free those buffers after they're used.

---

**Note:** This function should be called after ENET\_CreateHandler() and buffer allocating callback function should be ready.

---

#### Parameters

- base – ENET peripheral base address.

- `handle` – The ENET handler structure. This is the same handler pointer used in the `ENET_Init`.

`void ENET_RxBufferFreeAll(ENET_Type *base, enet_handle_t *handle)`

Frees Rx buffers in all BDs. It's used for zero copy Rx. In zero copy Rx case, Rx buffers are dynamic. This function will free left buffers in all BDs.

#### Parameters

- `base` – ENET peripheral base address.
- `handle` – The ENET handler structure. This is the same handler pointer used in the `ENET_Init`.

`void ENET_StartRxTx(ENET_Type *base, uint8_t txRingNum, uint8_t rxRingNum)`

Starts the ENET Tx/Rx. This function enable the Tx/Rx and starts the Tx/Rx DMA. This shall be set after ENET initialization and before starting to receive the data.

---

**Note:** This must be called after all the ENET initialization. And should be called when the ENET receive/transmit is required.

---

#### Parameters

- `base` – ENET peripheral base address.
- `rxRingNum` – The number of the used Rx rings. It shall not be larger than the `ENET_RING_NUM_MAX(2)`. If the `ringNum` is set with 1, the ring 0 will be used.
- `txRingNum` – The number of the used Tx rings. It shall not be larger than the `ENET_RING_NUM_MAX(2)`. If the `ringNum` is set with 1, the ring 0 will be used.

`void ENET_SetISRHandler(ENET_Type *base, enet_isr_t ISRHandler)`

Set the second level IRQ handler.

#### Parameters

- `base` – ENET peripheral base address.
- `ISRHandler` – The handler to install.

`static inline void ENET_SetMII(ENET_Type *base, enet_mii_speed_t speed, enet_mii_duplex_t duplex)`

Sets the ENET MII speed and duplex.

This API is provided to dynamically change the speed and dulpex for MAC.

#### Parameters

- `base` – ENET peripheral base address.
- `speed` – The speed of the RMII mode.
- `duplex` – The duplex of the RMII mode.

`void ENET_SetSMI(ENET_Type *base)`

Sets the ENET SMI(serial management interface)- MII management interface.

#### Parameters

- `base` – ENET peripheral base address.

static inline bool ENET\_IsSMIBusy(ENET\_Type \*base)

Checks if the SMI is busy.

**Parameters**

- base – ENET peripheral base address.

**Returns**

The status of MII Busy status.

static inline uint16\_t ENET\_ReadSMIData(ENET\_Type \*base)

Reads data from the PHY register through SMI interface.

**Parameters**

- base – ENET peripheral base address.

**Returns**

The data read from PHY

void ENET\_StartSMIWrite(ENET\_Type \*base, uint8\_t phyAddr, uint8\_t regAddr, uint16\_t data)

Sends the MDIO IEEE802.3 Clause 22 format write command.

**Parameters**

- base – ENET peripheral base address.
- phyAddr – The PHY address.
- regAddr – The PHY register.
- data – The data written to PHY.

void ENET\_StartSMIRead(ENET\_Type \*base, uint8\_t phyAddr, uint8\_t regAddr)

Sends the MDIO IEEE802.3 Clause 22 format read command.

**Parameters**

- base – ENET peripheral base address.
- phyAddr – The PHY address.
- regAddr – The PHY register.

status\_t ENET\_MDIOWrite(ENET\_Type \*base, uint8\_t phyAddr, uint8\_t regAddr, uint16\_t data)

MDIO write with IEEE802.3 Clause 22 format.

**Parameters**

- base – ENET peripheral base address.
- phyAddr – The PHY address.
- regAddr – The PHY register.
- data – The data written to PHY.

**Returns**

kStatus\_Success MDIO access succeeds.

**Returns**

kStatus\_Timeout MDIO access timeout.

status\_t ENET\_MDIORead(ENET\_Type \*base, uint8\_t phyAddr, uint8\_t regAddr, uint16\_t \*pData)

MDIO read with IEEE802.3 Clause 22 format.

**Parameters**

- base – ENET peripheral base address.
- phyAddr – The PHY address.

- regAddr – The PHY register.
- pData – The data read from PHY.

**Returns**

kStatus\_Success MDIO access succeeds.

**Returns**

kStatus\_Timeout MDIO access timeout.

uint32\_t ENET\_GetInstance(ENET\_Type \*base)

Get the ENET instance from peripheral base address.

**Parameters**

- base – ENET peripheral base address.

**Returns**

ENET instance.

static inline void ENET\_SetMacAddr(ENET\_Type \*base, uint8\_t \*macAddr)

Sets the ENET module Mac address.

**Parameters**

- base – ENET peripheral base address.
- macAddr – The six-byte Mac address pointer. The pointer is allocated by application and input into the API.

void ENET\_GetMacAddr(ENET\_Type \*base, uint8\_t \*macAddr)

Gets the ENET module Mac address.

**Parameters**

- base – ENET peripheral base address.
- macAddr – The six-byte Mac address pointer. The pointer is allocated by application and input into the API.

static inline void ENET\_AcceptAllMulticast(ENET\_Type \*base)

Enable ENET device to accept all multicast frames.

**Parameters**

- base – ENET peripheral base address.

static inline void ENET\_RejectAllMulticast(ENET\_Type \*base)

ENET device reject to accept all multicast frames.

**Parameters**

- base – ENET peripheral base address.

void ENET\_EnterPowerDown(ENET\_Type \*base, uint32\_t \*wakeFilter)

Set the MAC to enter into power down mode. the remote power wake up frame and magic frame can wake up the ENET from the power down mode.

**Parameters**

- base – ENET peripheral base address.
- wakeFilter – The wakeFilter provided to configure the wake up frame filter. Set the wakeFilter to NULL is not required. But if you have the filter requirement, please make sure the wakeFilter pointer shall be eight continuous 32-bits configuration.



```
static inline void ENET_ExitPowerDown(ENET_Type *base)
```

Set the MAC to exit power down mode. Exit from the power down mode and recover to normal work mode.

#### Parameters

- base – ENET peripheral base address.

```
void ENET_EnableInterrupts(ENET_Type *base, uint32_t mask)
```

Enables the ENET DMA and MAC interrupts.

This function enables the ENET interrupt according to the provided mask. The mask is a logical OR of `enet_dma_interrupt_enable_t` and `enet_mac_interrupt_enable_t`. For example, to enable the dma and mac interrupt, do the following.

```
ENET_EnableInterrupts(ENET, kENET_DmaRx | kENET_DmaTx | kENET_MacPmt);
```

#### Parameters

- base – ENET peripheral base address.
- mask – ENET interrupts to enable. This is a logical OR of both enumeration :: `enet_dma_interrupt_enable_t` and `enet_mac_interrupt_enable_t`.

```
void ENET_DisableInterrupts(ENET_Type *base, uint32_t mask)
```

Disables the ENET DMA and MAC interrupts.

This function disables the ENET interrupt according to the provided mask. The mask is a logical OR of `enet_dma_interrupt_enable_t` and `enet_mac_interrupt_enable_t`. For example, to disable the dma and mac interrupt, do the following.

```
ENET_DisableInterrupts(ENET, kENET_DmaRx | kENET_DmaTx | kENET_MacPmt);
```

#### Parameters

- base – ENET peripheral base address.
- mask – ENET interrupts to disables. This is a logical OR of both enumeration :: `enet_dma_interrupt_enable_t` and `enet_mac_interrupt_enable_t`.

```
static inline uint32_t ENET_GetDmaInterruptStatus(ENET_Type *base, uint8_t channel)
```

Gets the ENET DMA interrupt status flag.

#### Parameters

- base – ENET peripheral base address.
- channel – The DMA Channel. Shall not be larger than `ENET_RING_NUM_MAX`.

#### Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration :: `enet_dma_interrupt_enable_t`.

```
static inline void ENET_ClearDmaInterruptStatus(ENET_Type *base, uint8_t channel, uint32_t mask)
```

Clear the ENET DMA interrupt status flag.

#### Parameters

- base – ENET peripheral base address.
- channel – The DMA Channel. Shall not be larger than `ENET_RING_NUM_MAX`.
- mask – The event status of the interrupt source. This is the logical OR of members of the enumeration :: `enet_dma_interrupt_enable_t`.

```
static inline uint32_t ENET_GetMacInterruptStatus(ENET_Type *base)
```

Gets the ENET MAC interrupt status flag.

#### Parameters

- base – ENET peripheral base address.

#### Returns

The event status of the interrupt source. Use the enum in `enet_mac_interrupt_enable_t` and right shift `ENET_MACINT_ENUM_OFFSET` to mask the returned value to get the exact interrupt status.

```
void ENET_ClearMacInterruptStatus(ENET_Type *base, uint32_t mask)
```

Clears the ENET mac interrupt events status flag.

This function clears enabled ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See the `enet_mac_interrupt_enable_t`. For example, to clear the TX frame interrupt and RX frame interrupt, do the following.

```
ENET_ClearMacInterruptStatus(ENET, kENET_MacPmt);
```

#### Parameters

- base – ENET peripheral base address.
- mask – ENET interrupt source to be cleared. This is the logical OR of members of the enumeration :: `enet_mac_interrupt_enable_t`.

```
static inline bool ENET_IsTxDescriptorDmaOwn(enet_tx_bd_struct_t *txDesc)
```

Get the Tx descriptor DMA Own flag.

#### Parameters

- txDesc – The given Tx descriptor.

#### Return values

True – the dma own Tx descriptor, false application own Tx descriptor.

```
void ENET_SetupTxDescriptor(enet_tx_bd_struct_t *txDesc, void *buffer1, uint32_t bytes1, void  
                           *buffer2, uint32_t bytes2, uint32_t framelen, bool intEnable, bool  
                           tsEnable, enet_desc_flag_t flag, uint8_t slotNum)
```

Setup a given Tx descriptor. This function is a low level functional API to setup or prepare a given Tx descriptor.

---

**Note:** This must be called after all the ENET initialization. And should be called when the ENET receive/transmit is required. Transmit buffers are ‘zero-copy’ buffers, so the buffer must remain in memory until the packet has been fully transmitted. The buffers should be free or requeued in the transmit interrupt irq handler.

---

#### Parameters

- txDesc – The given Tx descriptor.
- buffer1 – The first buffer address in the descriptor.
- bytes1 – The bytes in the first buffer.
- buffer2 – The second buffer address in the descriptor.
- bytes2 – The bytes in the second buffer.
- framelen – The length of the frame to be transmitted.
- intEnable – Interrupt enable flag.

- tsEnable – The timestamp enable.
- flag – The flag of this Tx descriptor, see “enet\_desc\_flag\_t”.
- slotNum – The slot num used for AV mode only.

```
static inline void ENET_UpdateTxDescriptorTail(ENET_Type *base, uint8_t channel, uint32_t txDescTailAddrAlign)
```

Update the Tx descriptor tail pointer. This function is a low level functional API to update the the Tx descriptor tail. This is called after you setup a new Tx descriptor to update the tail pointer to make the new descriptor accessible by DMA.

#### Parameters

- base – ENET peripheral base address.
- channel – The Tx DMA channel.
- txDescTailAddrAlign – The new Tx tail pointer address.

```
static inline void ENET_UpdateRxDescriptorTail(ENET_Type *base, uint8_t channel, uint32_t rxDescTailAddrAlign)
```

Update the Rx descriptor tail pointer. This function is a low level functional API to update the the Rx descriptor tail. This is called after you setup a new Rx descriptor to update the tail pointer to make the new descriptor accessible by DMA and to anouse the Rx poll command for DMA.

#### Parameters

- base – ENET peripheral base address.
- channel – The Rx DMA channel.
- rxDescTailAddrAlign – The new Rx tail pointer address.

```
static inline uint32_t ENET_GetRxDescriptor(enet_rx_bd_struct_t *rxDesc)
```

Gets the context in the ENET Rx descriptor. This function is a low level functional API to get the the status flag from a given Rx descriptor.

---

**Note:** This must be called after all the ENET initilization. And should be called when the ENET receive/transmit is required.

---

#### Parameters

- rxDesc – The given Rx descriptor.

#### Return values

The – RDES3 regions for write-back format Rx buffer descriptor.

```
void ENET_UpdateRxDescriptor(enet_rx_bd_struct_t *rxDesc, void *buffer1, void *buffer2, bool intEnable, bool doubleBuffEnable)
```

Updates the buffers and the own status for a given Rx descriptor. This function is a low level functional API to Updates the buffers and the own status for a given Rx descriptor.

---

**Note:** This must be called after all the ENET initilization. And should be called when the ENET receive/transmit is required.

---

#### Parameters

- rxDesc – The given Rx descriptor.
- buffer1 – The first buffer address in the descriptor.

- `buffer2` – The second buffer address in the descriptor.
- `intEnable` – Interrupt enable flag.
- `doubleBuffEnable` – The double buffer enable flag.

```
void ENET__CreateHandler(ENET_Type *base, enet_handle_t *handle, enet_config_t *config,  
                        enet_buffer_config_t *bufferConfig, enet_callback_t callback, void  
                        *userData)
```

Create ENET Handler.

This is a transactional API and it's provided to store all datas which are needed during the whole transactional process. This API should not be used when you use functional APIs to do data Tx/Rx. This is funtion will store many data/flag for transactional use.

#### Parameters

- `base` – ENET peripheral base address.
- `handle` – ENET handler.
- `config` – ENET configuration.
- `bufferConfig` – ENET buffer configuration.
- `callback` – The callback function.
- `userData` – The application data.

```
status_t ENET__GetRxFramSize(ENET_Type *base, enet_handle_t *handle, uint32_t *length,  
                             uint8_t channel)
```

Gets the size of the read frame. This function gets a received frame size from the ENET buffer descriptors.

---

**Note:** The FCS of the frame is automatically removed by MAC and the size is the length without the FCS. After calling `ENET__GetRxFramSize`, `ENET_ReadFrame()` should be called to update the receive buffers If the result is not “`kStatus_ENET_RxFramEmpty`”.

---

#### Parameters

- `base` – ENET peripheral base address.
- `handle` – The ENET handler structure. This is the same handler pointer used in the `ENET_Init`.
- `length` – The length of the valid frame received.
- `channel` – The DMAC channel for the Rx.

#### Return values

- `kStatus_ENET_RxFramEmpty` – No frame received. Should not call `ENET_ReadFrame` to read frame.
- `kStatus_ENET_RxFramError` – Data error happens. `ENET_ReadFrame` should be called with NULL data and NULL length to update the receive buffers.
- `kStatus_Success` – Receive a frame Successfully then the `ENET_ReadFrame` should be called with the right data buffer and the captured data length input.

```
status_t ENET__ReadFrame(ENET_Type *base, enet_handle_t *handle, uint8_t *data, uint32_t  
                        length, uint8_t channel, enet_ptp_time_t *timestamp)
```

Reads a frame from the ENET device. This function reads a frame from the ENET DMA

descriptors. The `ENET_GetRxFrameSize` should be used to get the size of the prepared data buffer. For example use Rx dma channel 0:

```
uint32_t length;
enet_handle_t g_handle;
Comment: Get the received frame size firstly.
status = ENET_GetRxFrameSize(&g_handle, &length, 0);
if (length != 0)
{
    Comment: Allocate memory here with the size of "length"
    uint8_t *data = memory allocate interface;
    if (!data)
    {
        ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0);
    }
    else
    {
        status = ENET_ReadFrame(ENET, &g_handle, data, length, 0);
    }
}
else if (status == kStatus_ENET_RxFrameError)
{
    Comment: Update the received buffer when a error frame is received.
    ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0);
}
```

### Parameters

- base – ENET peripheral base address.
- handle – The ENET handler structure. This is the same handler pointer used in the `ENET_Init`.
- data – The data buffer provided by user to store the frame which memory size should be at least "length".
- length – The size of the data buffer which is still the length of the received frame.
- channel – The Rx DMA channel. Shall not be larger than 2.
- timestamp – The timestamp address to store received timestamp.

### Returns

The execute status, successful or failure.

*status\_t* `ENET_GetRxFrame(ENET_Type *base, enet_handle_t *handle, enet_rx_frame_struct_t *rxFrame, uint8_t channel)`

Receives one frame in specified BD ring with zero copy.

This function will use the user-defined allocate and free callback. Every time application gets one frame through this function, driver will allocate new buffers for the BDs whose buffers have been taken by application.

---

**Note:** This function will drop current frame and update related BDs as available for DMA if new buffers allocating fails. Application must provide a memory pool including at least BD number + 1 buffers(+2 if enable double buffer) to make this function work normally.

---

### Parameters

- base – ENET peripheral base address.

- `handle` – The ENET handler pointer. This is the same handler pointer used in the `ENET_Init`.
- `rxFrame` – The received frame information structure provided by user.
- `channel` – The Rx DMA channel. Shall not be larger than 2.

#### Return values

- `kStatus_Success` – Succeed to get one frame and allocate new memory for Rx buffer.
- `kStatus_ENET_RxFrameEmpty` – There's no Rx frame in the BD.
- `kStatus_ENET_RxFrameError` – There's issue in this receiving. In this function, issue frame will be dropped.
- `kStatus_ENET_RxFrameDrop` – There's no new buffer memory for BD, dropped this frame.

`status_t` `ENET_SendFrame`(`ENET_Type` \*`base`, `enet_handle_t` \*`handle`, `enet_tx_frame_struct_t` \*`txFrame`, `uint8_t` `channel`)

Transmits an ENET frame.

---

**Note:** The CRC is automatically appended to the data. Input the data to send without the CRC. This API uses input buffer for Tx, application should reclaim the buffer after Tx is over.

---

#### Parameters

- `base` – ENET peripheral base address.
- `handle` – The ENET handler pointer. This is the same handler pointer used in the `ENET_Init`.
- `txFrame` – The Tx frame structure.
- `channel` – Channel to send the frame, same with queue index.

#### Return values

- `kStatus_Success` – Send frame succeed.
- `kStatus_ENET_TxFrameBusy` – Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity. The waiting mechanism is recommended to be added after each call return with `kStatus_ENET_TxFrameBusy`. Also need to pay attention to reclaim Tx frame after Tx is over.
- `kStatus_ENET_TxFrameOverLen` – Transmit frame length exceeds the 0x3FFF limit defined by the driver.

`void` `ENET_ReclaimTxDescriptor`(`ENET_Type` \*`base`, `enet_handle_t` \*`handle`, `uint8_t` `channel`)

Reclaim Tx descriptors. This function is used to update the Tx descriptor status and store the Tx timestamp when the 1588 feature is enabled. This is called by the transmit interrupt IRQ handler after the complete of a frame transmission.

#### Parameters

- `base` – ENET peripheral base address.
- `handle` – The ENET handler pointer. This is the same handler pointer used in the `ENET_Init`.
- `channel` – The Tx DMA channel.

```
void ENET_IRQHandler(ENET_Type *base, enet_handle_t *handle)
```

The ENET IRQ handler.

#### Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer.

```
FSL_ENET_DRIVER_VERSION
```

Defines the driver version.

```
ENET_RXDESCRIP_RD_BUFF1VALID_MASK
```

Defines for read format.

Buffer1 address valid.

```
ENET_RXDESCRIP_RD_BUFF2VALID_MASK
```

Buffer2 address valid.

```
ENET_RXDESCRIP_RD_IOC_MASK
```

Interrupt enable on complete.

```
ENET_RXDESCRIP_RD_OWN_MASK
```

Own bit.

```
ENET_RXDESCRIP_WR_ERR_MASK
```

Defines for write back format.

```
ENET_RXDESCRIP_WR_PYLOAD_MASK
```

```
ENET_RXDESCRIP_WR_PTPMSGTYPE_MASK
```

```
ENET_RXDESCRIP_WR_PTPTYPE_MASK
```

```
ENET_RXDESCRIP_WR_PTPVERSION_MASK
```

```
ENET_RXDESCRIP_WR_PTPTSA_MASK
```

```
ENET_RXDESCRIP_WR_PACKETLEN_MASK
```

```
ENET_RXDESCRIP_WR_ERRSUM_MASK
```

```
ENET_RXDESCRIP_WR_TYPE_MASK
```

```
ENET_RXDESCRIP_WR_DE_MASK
```

```
ENET_RXDESCRIP_WR_RE_MASK
```

```
ENET_RXDESCRIP_WR_OE_MASK
```

```
ENET_RXDESCRIP_WR_RS0V_MASK
```

```
ENET_RXDESCRIP_WR_RS1V_MASK
```

```
ENET_RXDESCRIP_WR_RS2V_MASK
```

```
ENET_RXDESCRIP_WR_LD_MASK
```

```
ENET_RXDESCRIP_WR_FD_MASK
```

```
ENET_RXDESCRIP_WR_CTXT_MASK
```

```
ENET_RXDESCRIP_WR_OWN_MASK
```

ENET\_TXDESCRIP\_RD\_BL1\_MASK

Defines for read format.

ENET\_TXDESCRIP\_RD\_BL2\_MASK

ENET\_TXDESCRIP\_RD\_BL1(n)

ENET\_TXDESCRIP\_RD\_BL2(n)

ENET\_TXDESCRIP\_RD\_TTSE\_MASK

ENET\_TXDESCRIP\_RD\_IOC\_MASK

ENET\_TXDESCRIP\_RD\_FL\_MASK

ENET\_TXDESCRIP\_RD\_FL(n)

ENET\_TXDESCRIP\_RD\_CIC(n)

ENET\_TXDESCRIP\_RD\_TSE\_MASK

ENET\_TXDESCRIP\_RD\_SLOT(n)

ENET\_TXDESCRIP\_RD\_SAIC(n)

ENET\_TXDESCRIP\_RD\_CPC(n)

ENET\_TXDESCRIP\_RD\_LDFD(n)

ENET\_TXDESCRIP\_RD\_LD\_MASK

ENET\_TXDESCRIP\_RD\_FD\_MASK

ENET\_TXDESCRIP\_RD\_CTXT\_MASK

ENET\_TXDESCRIP\_RD\_OWN\_MASK

ENET\_TXDESCRIP\_WB\_TTSS\_MASK

Defines for write back format.

ENET\_ABNORM\_INT\_MASK

ENET\_NORM\_INT\_MASK

ENET\_FRAME\_MAX\_FRAMELEN

Default maximum ethernet frame size.

ENET\_FCS\_LEN

Ethernet Rx frame FCS length.

ENET\_ADDR\_ALIGNMENT

Recommended ethernet buffer alignment.

ENET\_BUFF\_ALIGNMENT

Receive buffer alignment shall be 4bytes-aligned.

ENET\_RING\_NUM\_MAX

The maximum number of Tx/Rx descriptor rings.

ENET\_MTL\_RXFIFOSIZE

The Rx fifo size.

ENET\_MTL\_TXFIFOSIZE

The Tx fifo size.



ENET\_MACINT\_ENUM\_OFFSET

The offset for mac interrupt in enum type.

ENET\_FRAME\_TX\_LEN\_LIMITATION

The Tx frame length software limitation.

ENET\_FRAME\_RX\_ERROR\_BITS(x)

The Rx frame error bits field.

Defines the status return codes for transaction.

*Values:*

enumerator kStatus\_ENET\_InitMemoryFail

Status code 4000. Init failed since buffer memory was not enough.

enumerator kStatus\_ENET\_RxFrameError

Status code 4001. A frame received but data error occurred.

enumerator kStatus\_ENET\_RxFrameFail

Status code 4002. Failed to receive a frame.

enumerator kStatus\_ENET\_RxFrameEmpty

Status code 4003. No frame arrived.

enumerator kStatus\_ENET\_RxFrameDrop

Status code 4004. Rx frame was dropped since there's no buffer memory.

enumerator kStatus\_ENET\_TxFrameBusy

Status code 4005. There were no resources for Tx operation.

enumerator kStatus\_ENET\_TxFrameFail

Status code 4006. Transmit frame failed.

enumerator kStatus\_ENET\_TxFrameOverLen

Status code 4007. Failed to send an oversize frame.

enum \_enet\_mii\_mode

Defines the MII/RMII mode for data interface between the MAC and the PHY.

*Values:*

enumerator kENET\_MiiMode

MII mode for data interface.

enumerator kENET\_RmiiMode

RMII mode for data interface.

enum \_enet\_mii\_speed

Defines the 10/100 Mbps speed for the MII data interface.

*Values:*

enumerator kENET\_MiiSpeed10M

Speed 10 Mbps.

enumerator kENET\_MiiSpeed100M

Speed 100 Mbps.

enum \_enet\_mii\_duplex

Defines the half or full duplex for the MII data interface.

*Values:*

enumerator kENET\_MiiHalfDuplex

Half duplex mode.

enumerator kENET\_MiiFullDuplex

Full duplex mode.

enum \_enet\_mii\_normal\_opcode

Define the MII opcode for normal MDIO\_CLAUSES\_22 Frame.

*Values:*

enumerator kENET\_MiiWriteFrame

Write frame operation for a valid MII management frame.

enumerator kENET\_MiiReadFrame

Read frame operation for a valid MII management frame.

enum \_enet\_dma\_burstlen

Define the DMA maximum transmit burst length.

*Values:*

enumerator kENET\_BurstLen1

DMA burst length 1.

enumerator kENET\_BurstLen2

DMA burst length 2.

enumerator kENET\_BurstLen4

DMA burst length 4.

enumerator kENET\_BurstLen8

DMA burst length 8.

enumerator kENET\_BurstLen16

DMA burst length 16.

enumerator kENET\_BurstLen32

DMA burst length 32.

enumerator kENET\_BurstLen64

DMA burst length 64. eight times enabled.

enumerator kENET\_BurstLen128

DMA burst length 128. eight times enabled.

enumerator kENET\_BurstLen256

DMA burst length 256. eight times enabled.

enum \_enet\_desc\_flag

Define the flag for the descriptor.

*Values:*

enumerator kENET\_MiddleFlag

It's a middle descriptor of the frame.

enumerator kENET\_LastFlagOnly

It's the last descriptor of the frame.

enumerator kENET\_FirstFlagOnly

It's the first descriptor of the frame.

enumerator kENET\_FirstLastFlag

It's the first and last descriptor of the frame.

enum \_enet\_systime\_op

Define the system time adjust operation control.

*Values:*

enumerator kENET\_SystimeAdd

System time add to.

enumerator kENET\_SystimeSubtract

System time subtract.

enum \_enet\_ts\_rollover\_type

Define the system time rollover control.

*Values:*

enumerator kENET\_BinaryRollover

System time binary rollover.

enumerator kENET\_DigitalRollover

System time digital rollover.

enum \_enet\_special\_config

Defines some special configuration for ENET.

These control flags are provided for special user requirements. Normally, there is no need to set these control flags for ENET initialization. But if you have some special requirements, set the flags to specialControl in the enet\_config\_t.

---

**Note:** “kENET\_StoreAndForward” is recommended to be set when the ENET\_PTP1588FEATURE\_REQUIRED is defined or else the timestamp will be messed up when the overflow happens.

---

*Values:*

enumerator kENET\_DescDoubleBuffer

The double buffer is used in the Tx/Rx descriptor.

enumerator kENET\_StoreAndForward

The Rx/Tx store and forward enable.

enumerator kENET\_PromiscuousEnable

The promiscuous enabled.

enumerator kENET\_FlowControlEnable

The flow control enabled.

enumerator kENET\_BroadCastRxDisable

The broadcast disabled.

enumerator kENET\_MulticastAllEnable

All multicast are passed.

enumerator kENET\_8023AS2KPacket

8023as support for 2K packets.

enumerator kENET\_RxChecksumOffloadEnable

The Rx checksum offload enabled.

enum \_enet\_dma\_interrupt\_enable

List of DMA interrupts supported by the ENET interrupt. This enumeration uses one-hot encoding to allow a logical OR of multiple members.

*Values:*

enumerator kENET\_DmaTx

Tx interrupt.

enumerator kENET\_DmaTxStop

Tx stop interrupt.

enumerator kENET\_DmaTxBuffUnavail

Tx buffer unavailable.

enumerator kENET\_DmaRx

Rx interrupt.

enumerator kENET\_DmaRxBuffUnavail

Rx buffer unavailable.

enumerator kENET\_DmaRxStop

Rx stop.

enumerator kENET\_DmaRxWatchdogTimeout

Rx watchdog timeout.

enumerator kENET\_DmaEarlyTx

Early transmit.

enumerator kENET\_DmaEarlyRx

Early receive.

enumerator kENET\_DmaBusErr

Fatal bus error.

enum \_enet\_mac\_interrupt\_enable

List of mac interrupts supported by the ENET interrupt. This enumeration uses one-hot encoding to allow a logical OR of multiple members.

*Values:*

enumerator kENET\_MacPmt

enumerator kENET\_MacTimestamp

enum \_enet\_event

Defines the common interrupt event for callback use.

*Values:*

enumerator kENET\_RxIntEvent

Receive interrupt event.

enumerator kENET\_TxIntEvent

Transmit interrupt event.

enumerator kENET\_WakeUpIntEvent

Wake up interrupt event.

enumerator kENET\_TimeStampIntEvent

Time stamp interrupt event.

enum \_enet\_dma\_tx\_sche

Define the DMA transmit arbitration for multi-queue.

*Values:*

enumerator kENET\_FixPri

Fixed priority. channel 0 has lower priority than channel 1.

enumerator kENET\_WeightStrPri

Weighted(burst length) strict priority.

enumerator kENET\_WeightRoundRobin

Weighted (weight factor) round robin.

enum \_enet\_mtl\_multiqueue\_txsche

Define the MTL Tx scheduling algorithm for multiple queues/rings.

*Values:*

enumerator kENET\_txWeightRR

Tx weight round-robin.

enumerator kENET\_txStrPrio

Tx strict priority.

enum \_enet\_mtl\_multiqueue\_rxsche

Define the MTL Rx scheduling algorithm for multiple queues/rings.

*Values:*

enumerator kENET\_rxStrPrio

Tx weight round-robin, Rx strict priority.

enumerator kENET\_rxWeightStrPrio

Tx strict priority, Rx weight strict priority.

enum \_enet\_mtl\_rxqueuemap

Define the MTL Rx queue and DMA channel mapping.

*Values:*

enumerator kENET\_StaticDirectMap

The received frame in Rx Qn(n = 0,1) directly map to dma channel n.

enumerator kENET\_DynamicMap

The received frame in Rx Qn(n = 0,1) map to the dma channel m(m = 0,1) related with the same Mac.

enum \_enet\_ptp\_event\_type

Defines the ENET PTP message related constant.

*Values:*

enumerator kENET\_PtpEventMsgType

PTP event message type.

enumerator kENET\_PtpSrcPortIdLen

PTP message sequence id length.

enumerator kENET\_PtpEventPort

PTP event port number.

enumerator kENET\_PtpGnrlPort

PTP general port number.

enum `_enet_tx_offload`

Define the Tx checksum offload options.

*Values:*

enumerator `kENET_TxOffloadDisable`

Disable Tx checksum offload.

enumerator `kENET_TxOffloadIPHeader`

Enable IP header checksum calculation and insertion.

enumerator `kENET_TxOffloadIPHeaderPlusPayload`

Enable IP header and payload checksum calculation and insertion.

enumerator `kENET_TxOffloadAll`

Enable IP header, payload and pseudo header checksum calculation and insertion.

typedef enum `_enet_mii_mode` `enet_mii_mode_t`

Defines the MII/RMII mode for data interface between the MAC and the PHY.

typedef enum `_enet_mii_speed` `enet_mii_speed_t`

Defines the 10/100 Mbps speed for the MII data interface.

typedef enum `_enet_mii_duplex` `enet_mii_duplex_t`

Defines the half or full duplex for the MII data interface.

typedef enum `_enet_mii_normal_opcode` `enet_mii_normal_opcode_t`

Define the MII opcode for normal MDIO\_CLAUSES\_22 Frame.

typedef enum `_enet_dma_burstlen` `enet_dma_burstlen_t`

Define the DMA maximum transmit burst length.

typedef enum `_enet_desc_flag` `enet_desc_flag_t`

Define the flag for the descriptor.

typedef enum `_enet_systime_op` `enet_systime_op_t`

Define the system time adjust operation control.

typedef enum `_enet_ts_rollover_type` `enet_ts_rollover_type_t`

Define the system time rollover control.

typedef enum `_enet_special_config` `enet_special_config_t`

Defines some special configuration for ENET.

These control flags are provided for special user requirements. Normally, there is no need to set these control flags for ENET initialization. But if you have some special requirements, set the flags to `specialControl` in the `enet_config_t`.

---

**Note:** “`kENET_StoreAndForward`” is recommended to be set when the `ENET_PTP1588FEATURE_REQUIRED` is defined or else the timestamp will be messed up when the overflow happens.

---

typedef enum `_enet_dma_interrupt_enable` `enet_dma_interrupt_enable_t`

List of DMA interrupts supported by the ENET interrupt. This enumeration uses one-hot encoding to allow a logical OR of multiple members.

typedef enum `_enet_mac_interrupt_enable` `enet_mac_interrupt_enable_t`

List of mac interrupts supported by the ENET interrupt. This enumeration uses one-hot encoding to allow a logical OR of multiple members.

```
typedef enum _enet_event enet_event_t
```

Defines the common interrupt event for callback use.

```
typedef enum _enet_dma_tx_sche enet_dma_tx_sche_t
```

Define the DMA transmit arbitration for multi-queue.

```
typedef enum _enet_mtl_multiqueue_txsche enet_mtl_multiqueue_txsche_t
```

Define the MTL Tx scheduling algorithm for multiple queues/rings.

```
typedef enum _enet_mtl_multiqueue_rxsche enet_mtl_multiqueue_rxsche_t
```

Define the MTL Rx scheduling algorithm for multiple queues/rings.

```
typedef enum _enet_mtl_rxqueuemap enet_mtl_rxqueuemap_t
```

Define the MTL Rx queue and DMA channel mapping.

```
typedef enum _enet_ptp_event_type enet_ptp_event_type_t
```

Defines the ENET PTP message related constant.

```
typedef enum _enet_tx_offload enet_tx_offload_t
```

Define the Tx checksum offload options.

```
typedef struct _enet_rx_bd_struct enet_rx_bd_struct_t
```

Defines the receive descriptor structure. It has the read-format and write-back format structures. They both have the same size with different region definition. So we define common name as the receive descriptor structure. When initialize the buffer descriptors, read-format region mask bits should be used. When Rx frame has been in the buffer descriptors, write-back format region store the Rx result information.

```
typedef struct _enet_tx_bd_struct enet_tx_bd_struct_t
```

Defines the transmit descriptor structure. It has the read-format and write-back format structure. They both have the same size with different region definition. So we define common name as the transmit descriptor structure. When initialize the buffer descriptors for Tx, read-format region mask bits should be used. When frame has been transmitted, write-back format region store the Tx result information.

```
typedef struct _enet_tx_bd_config_struct enet_tx_bd_config_struct_t
```

Defines the Tx BD configuration structure.

```
typedef struct _enet_ptp_time enet_ptp_time_t
```

Defines the ENET PTP time stamp structure.

```
typedef struct _enet_tx_reclaim_info enet_tx_reclaim_info_t
```

Defines the Tx reclaim information structure.

```
typedef struct _enet_tx_dirty_ring enet_tx_dirty_ring_t
```

Defines the ENET transmit dirty addresses ring/queue structure.

```
typedef struct _enet_buffer_config enet_buffer_config_t
```

Defines the buffer descriptor configure structure.

Notes:

- a. The receive and transmit descriptor start address pointer and tail pointer must be word-aligned.
- b. The recommended minimum Tx/Rx ring length is 4.
- c. The Tx/Rx descriptor tail address shall be the address pointer to the address just after the end of the last descriptor. because only the descriptors between the start address and the tail address will be used by DMA.
- d. The descriptor address is the start address of all used contiguous memory. for example, the rxDescStartAddrAlign is the start address of rxRingLen contiguous descriptor memory for Rx descriptor ring 0.

- e. The “*\*rxBufferstartAddr*” is the first element of *rxRingLen* (*2\*rxRingLen* for double buffers) Rx buffers. It means the *\*rxBufferStartAddr* is the Rx buffer for the first descriptor the *\*rxBufferStartAddr + 1* is the Rx buffer for the second descriptor or the Rx buffer for the second buffer in the first descriptor. So please make sure the *rxBufferStartAddr* is the address of a *rxRingLen* or *2\*rxRingLen* array.

```
typedef struct enet_multiqueue_config enet_multiqueue_config_t
```

Defines the configuration when multi-queue is used.

```
typedef void (*enet_rx_alloc_callback_t)(ENET_Type *base, void *userData, uint8_t channel)
```

Defines the Rx memory buffer alloc function pointer.

```
typedef void (*enet_rx_free_callback_t)(ENET_Type *base, void *buffer, void *userData, uint8_t channel)
```

Defines the Rx memory buffer free function pointer.

```
typedef struct enet_config enet_config_t
```

Defines the basic configuration structure for the ENET device.

Note:

- a. Default the signal queue is used so the “*multiqueueCfg*” is set default with NULL. Set the pointer with a valid configuration pointer if the multiple queues are required. If multiple queue is enabled, please make sure the buffer configuration for all are prepared also.

```
typedef struct enet_handle enet_handle_t
```

```
typedef void (*enet_callback_t)(ENET_Type *base, enet_handle_t *handle, enet_event_t event, uint8_t channel, enet_tx_reclaim_info_t *txReclaimInfo, void *userData)
```

ENET callback function.

```
typedef struct enet_tx_bd_ring enet_tx_bd_ring_t
```

Defines the ENET transmit buffer descriptor ring/queue structure.

```
typedef struct enet_rx_bd_ring enet_rx_bd_ring_t
```

Defines the ENET receive buffer descriptor ring/queue structure.

```
typedef struct enet_buffer_struct enet_buffer_struct_t
```

```
typedef struct enet_rx_frame_attribute_struct enet_rx_frame_attribute_t
```

Rx frame attribute structure.

```
typedef struct enet_rx_frame_error enet_rx_frame_error_t
```

Defines the Rx frame error structure.

```
typedef struct enet_rx_frame_struct enet_rx_frame_struct_t
```

Defines the Rx frame data structure.

```
typedef struct enet_tx_config_struct enet_tx_config_struct_t
```

```
typedef struct enet_tx_frame_struct enet_tx_frame_struct_t
```

```
typedef void (*enet_isr_t)(ENET_Type *base, enet_handle_t *handle)
```

```
const clock_ip_name_t s_enetClock[]
```

Pointers to enet clocks for each instance.

```
struct enet_rx_bd_struct
```

*#include <fsl\_enet.h>* Defines the receive descriptor structure It has the read-format and write-back format structures. They both have the same size with different region definition. So we define common name as the receive descriptor structure. When initialize the buffer descriptors, read-format region mask bits should be used. When Rx frame has been in the buffer descriptors, write-back format region store the Rx result information.



### Public Members

\_\_IO uint32\_t rdes0  
Receive descriptor 0

\_\_IO uint32\_t rdes1  
Receive descriptor 1

\_\_IO uint32\_t rdes2  
Receive descriptor 2

\_\_IO uint32\_t rdes3  
Receive descriptor 3

struct \_enet\_tx\_bd\_struct

*#include <fsl\_enet.h>* Defines the transmit descriptor structure. It has the read-format and write-back format structure. They both have the same size with different region definition. So we define a common name as the transmit descriptor structure. When initialize the buffer descriptors for Tx, read-format region mask bits should be used. When a frame has been transmitted, write-back format region stores the Tx result information.

### Public Members

\_\_IO uint32\_t tdes0  
Transmit descriptor 0

\_\_IO uint32\_t tdes1  
Transmit descriptor 1

\_\_IO uint32\_t tdes2  
Transmit descriptor 2

\_\_IO uint32\_t tdes3  
Transmit descriptor 3

struct \_enet\_tx\_bd\_config\_struct

*#include <fsl\_enet.h>* Defines the Tx BD configuration structure.

### Public Members

void \*buffer1  
The first buffer address in the descriptor.

uint32\_t bytes1  
The bytes in the first buffer.

void \*buffer2  
The second buffer address in the descriptor.

uint32\_t bytes2  
The bytes in the second buffer.

uint32\_t framelen  
The length of the frame to be transmitted.

bool intEnable  
Interrupt enable flag.

bool tsEnable  
The timestamp enable.

*enet\_tx\_offload\_t* txOffloadOps

The Tx checksum offload option, only valid for Queue 0.

*enet\_desc\_flag\_t* flag

The flag of this tx descriptor, see “enet\_qos\_desc\_flag”.

uint8\_t slotNum

The slot number used for AV mode only.

struct \_enet\_ptp\_time

*#include <fsl\_enet.h>* Defines the ENET PTP time stamp structure.

### Public Members

uint64\_t second

Second.

uint32\_t nanosecond

Nanosecond.

struct enet\_tx\_reclaim\_info

*#include <fsl\_enet.h>* Defines the Tx reclaim information structure.

### Public Members

void \*context

User specified data, could be buffer address for free

bool isTsAvail

Flag indicates timestamp available status

*enet\_ptp\_time\_t* timeStamp

Timestamp of frame

struct \_enet\_tx\_dirty\_ring

*#include <fsl\_enet.h>* Defines the ENET transmit dirty addresses ring/queue structure.

### Public Members

*enet\_tx\_reclaim\_info\_t* \*txDirtyBase

Dirty buffer descriptor base address pointer.

uint16\_t txGenIdx

Tx generate index.

uint16\_t txConsumeIdx

Tx consume index.

uint16\_t txRingLen

Tx ring length.

bool isFull

Tx ring is full flag, add this parameter to avoid waste one element.

struct \_enet\_buffer\_config

*#include <fsl\_enet.h>* Defines the buffer descriptor configure structure.

Notes:

- a. The receive and transmit descriptor start address pointer and tail pointer must be word-aligned.
- b. The recommended minimum Tx/Rx ring length is 4.
- c. The Tx/Rx descriptor tail address shall be the address pointer to the address just after the end of the last last descriptor. because only the descriptors between the start address and the tail address will be used by DMA.
- d. The decriptor address is the start address of all used contiguous memory. for example, the rxDescStartAddrAlign is the start address of rxRingLen contiguous descriptor memorise for Rx descriptor ring 0.
- e. The “\*rxBufferstartAddr” is the first element of rxRingLen (2\*rxRingLen for double buffers) Rx buffers. It means the \*rxBufferStartAddr is the Rx buffer for the first descriptor the \*rxBufferStartAddr + 1 is the Rx buffer for the second descriptor or the Rx buffer for the second buffer in the first descriptor. So please make sure the rxBufferStartAddr is the address of a rxRingLen or 2\*rxRingLen array.

### Public Members

uint8\_t rxRingLen

The length of receive buffer descriptor ring.

uint8\_t txRingLen

The length of transmit buffer descriptor ring.

enet\_tx\_bd\_struct\_t \*txDescStartAddrAlign

Aligned transmit descriptor start address.

enet\_tx\_bd\_struct\_t \*txDescTailAddrAlign

Aligned transmit descriptor tail address.

enet\_tx\_reclaim\_info\_t \*txDirtyStartAddr

Start address of the dirty Tx frame information.

enet\_rx\_bd\_struct\_t \*rxDescStartAddrAlign

Aligned receive descriptor start address.

enet\_rx\_bd\_struct\_t \*rxDescTailAddrAlign

Aligned receive descriptor tail address.

uint32\_t \*rxBufferStartAddr

Start address of the Rx buffers.

uint32\_t rxBuffSizeAlign

Aligned receive data buffer size.

struct enet\_multiqueue\_config

#include <fsl\_enet.h> Defines the configuration when multi-queue is used.

### Public Members

enet\_dma\_tx\_sche\_t dmaTxSche

Transmit arbitration.

enet\_dma\_burstlen\_t burstLen

Burset len for the queue 1.

uint8\_t txdmaChnWeight[(2U)]

Transmit channel weight.

*enet\_mtl\_multiqueue\_txsche\_t* mtltxSche

Transmit schedule for multi-queue.

*enet\_mtl\_multiqueue\_rxsche\_t* mtlrxSche

Receive schedule for multi-queue.

*uint8\_t* rxqueuweight[(2U)]

Refer to the MTL RxQ Control register.

*uint32\_t* txqueuweight[(2U)]

Refer to the MTL TxQ Quantum Weight register.

*uint8\_t* rxqueuePrio[(2U)]

Receive queue priority.

*uint8\_t* txqueuePrio[(2U)]

Refer to Transmit Queue Priority Mapping register.

*enet\_mtl\_rxqueuemap\_t* mtlrxQuemap

Rx queue DMA Channel mapping.

**struct** *\_enet\_config*

*#include <fsl\_enet.h>* Defines the basic configuration structure for the ENET device.

Note:

- a. Default the signal queue is used so the “multiqueueCfg” is set default with NULL. Set the pointer with a valid configuration pointer if the multiple queues are required. If multiple queue is enabled, please make sure the buffer configuration for all are prepared also.

## Public Members

*uint16\_t* specialControl

The logical or of *enet\_special\_config\_t*

*enet\_multiqueue\_config\_t* \*multiqueueCfg

Use both Tx/Rx queue(dma channel) 0 and 1.

*uint32\_t* interrupt

MAC interrupt source. A logical OR of *enet\_dma\_interrupt\_enable\_t* and *enet\_mac\_interrupt\_enable\_t*.

*enet\_mii\_mode\_t* miiMode

MII mode.

*enet\_mii\_speed\_t* miiSpeed

MII Speed.

*enet\_mii\_duplex\_t* miiDuplex

MII duplex.

*uint16\_t* pauseDuration

Used in the Tx flow control frame, only valid when *kENET\_FlowControlEnable* is set.

*enet\_rx\_alloc\_callback\_t* rxBuffAlloc

Callback to alloc memory, must be provided for zero-copy Rx.

*enet\_rx\_free\_callback\_t* rxBuffFree

Callback to free memory, must be provided for zero-copy Rx.

**struct** *\_enet\_tx\_bd\_ring*

*#include <fsl\_enet.h>* Defines the ENET transmit buffer descriptor ring/queue structure.

**Public Members**

*enet\_tx\_bd\_struct\_t* \*txBdBase  
Buffer descriptor base address pointer.

uint16\_t txGenIdx  
Tx generate index.

uint16\_t txConsumIdx  
Tx consum index.

volatile uint16\_t txDescUsed  
Tx descriptor used number.

uint16\_t txRingLen  
Tx ring length.

struct \_\_enet\_rx\_bd\_ring  
*#include <fsl\_enet.h>* Defines the ENET receive buffer descriptor ring/queue structure.

**Public Members**

*enet\_rx\_bd\_struct\_t* \*rxBdBase  
Buffer descriptor base address pointer.

uint16\_t rxGenIdx  
The current available receive buffer descriptor pointer.

uint16\_t rxRingLen  
Receive ring length.

uint32\_t rxBuffSizeAlign  
Receive buffer size.

struct \_\_enet\_handle  
*#include <fsl\_enet.h>* Defines the ENET handler structure.

**Public Members**

bool multiQueEnable  
Multi-queue enable status.

bool doubleBuffEnable  
The double buffer enable status.

bool rxintEnable  
Rx interrupt enable status.

*enet\_rx\_bd\_ring\_t* rxBdRing[(2U)]  
Receive buffer descriptor.

*enet\_tx\_bd\_ring\_t* txBdRing[(2U)]  
Transmit buffer descriptor.

*enet\_tx\_dirty\_ring\_t* txDirtyRing[(2U)]  
Transmit dirty buffers addresses.

uint32\_t \*rxBufferStartAddr[(2U)]  
The Init-Rx buffers used for reinit corrupted BD due to write-back operation.

uint32\_t txLenLimitation[(2U)]

Tx frame length limitation.

enet\_callback\_t callback

Callback function.

void \*userData

Callback function parameter.

enet\_rx\_alloc\_callback\_t rxBuffAlloc

Callback to alloc memory, must be provided for zero-copy Rx.

enet\_rx\_free\_callback\_t rxBuffFree

Callback to free memory, must be provided for zero-copy Rx.

struct \_\_enet\_buffer\_struct

#include <fsl\_enet.h>

### Public Members

void \*buffer

The buffer stores the whole or partial frame.

uint16\_t length

The byte length of this buffer.

struct \_\_enet\_rx\_frame\_attribute\_struct

#include <fsl\_enet.h> Rx frame attribute structure.

### Public Members

bool isTsAvail

Rx frame timestamp is available or not.

enet\_ptp\_time\_t timestamp

The nanosecond part timestamp of this Rx frame.

struct \_\_enet\_rx\_frame\_error

#include <fsl\_enet.h> Defines the Rx frame error structure.

### Public Members

bool statsDribbleErr

The received packet has a non-integer multiple of bytes (odd nibbles).

bool statsRxErr

Receive error.

bool statsOverflowErr

Rx FIFO overflow error.

bool statsWatchdogTimeoutErr

Receive watchdog timeout.

bool statsGaintPacketErr

Receive error.

bool statsRxFcsErr

Receive CRC error.

```
struct __enet_rx_frame_struct
#include <fsl_enet.h> Defines the Rx frame data structure.
```

### Public Members

*enet\_buffer\_struct\_t* \*rxBuffArray  
Rx frame buffer structure.

uint16\_t totLen  
Rx frame total length.

*enet\_rx\_frame\_attribute\_t* rxAttribute  
Rx frame attribute structure.

*enet\_rx\_frame\_error\_t* rxFrameError  
Rx frame error.

```
struct __enet_tx_config_struct
#include <fsl_enet.h>
```

### Public Members

uint8\_t intEnable  
Enable interrupt every time one BD is completed.

uint8\_t tsEnable  
Transmit timestamp enable.

uint8\_t slotNum  
Slot number control bits in AV mode.

*enet\_tx\_offload\_t* txOffloadOps  
Tx checksum offload option.

```
struct __enet_tx_frame_struct
#include <fsl_enet.h>
```

### Public Members

*enet\_buffer\_struct\_t* \*txBuffArray  
Tx frame buffer structure.

uint32\_t txBuffNum  
Buffer number of this Tx frame.

*enet\_tx\_config\_struct\_t* txConfig  
Tx extra configuration.

void \*context  
Driver reclaims and gives it in Tx over callback.

## 2.27 GPIO: General Purpose I/O

```
void GPIO_PortInit(GPIO_Type *base, uint32_t port)
```

Initializes the GPIO peripheral.

This function ungates the GPIO clock.

#### Parameters

- base – GPIO peripheral base pointer.
- port – GPIO port number.

```
void GPIO_PinInit(GPIO_Type *base, uint32_t port, uint32_t pin, const gpio_pin_config_t *config)
```

Initializes a GPIO pin used by the board.

To initialize the GPIO, define a pin configuration, either input or output, in the user file. Then, call the GPIO\_PinInit() function.

This is an example to define an input pin or output pin configuration:

```
Define a digital input pin configuration,
gpio_pin_config_t config =
{
    kGPIO_DigitalInput,
    0,
}
Define a digital output pin configuration,
gpio_pin_config_t config =
{
    kGPIO_DigitalOutput,
    0,
}
```

#### Parameters

- base – GPIO peripheral base pointer(Typically GPIO)
- port – GPIO port number
- pin – GPIO pin number
- config – GPIO pin configuration pointer

```
static inline void GPIO_PinWrite(GPIO_Type *base, uint32_t port, uint32_t pin, uint8_t output)
```

Sets the output level of the one GPIO pin to the logic 1 or 0.

#### Parameters

- base – GPIO peripheral base pointer(Typically GPIO)
- port – GPIO port number
- pin – GPIO pin number
- output – GPIO pin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

```
static inline uint32_t GPIO_PinRead(GPIO_Type *base, uint32_t port, uint32_t pin)
```

Reads the current input value of the GPIO PIN.

#### Parameters

- base – GPIO peripheral base pointer(Typically GPIO)
- port – GPIO port number
- pin – GPIO pin number



**Return values**

GPIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

FSL\_GPIO\_DRIVER\_VERSION

LPC GPIO driver version.

enum \_gpio\_pin\_direction

LPC GPIO direction definition.

*Values:*

enumerator kGPIO\_DigitalInput

Set current pin as digital input

enumerator kGPIO\_DigitalOutput

Set current pin as digital output

typedef enum \_gpio\_pin\_direction gpio\_pin\_direction\_t

LPC GPIO direction definition.

typedef struct \_gpio\_pin\_config gpio\_pin\_config\_t

The GPIO pin configuration structure.

Every pin can only be configured as either output pin or input pin at a time. If configured as a input pin, then leave the outputConfig unused.

static inline void GPIO\_PortSet(GPIO\_Type \*base, uint32\_t port, uint32\_t mask)

Sets the output level of the multiple GPIO pins to the logic 1.

**Parameters**

- base – GPIO peripheral base pointer(Typically GPIO)
- port – GPIO port number
- mask – GPIO pin number macro

static inline void GPIO\_PortClear(GPIO\_Type \*base, uint32\_t port, uint32\_t mask)

Sets the output level of the multiple GPIO pins to the logic 0.

**Parameters**

- base – GPIO peripheral base pointer(Typically GPIO)
- port – GPIO port number
- mask – GPIO pin number macro

static inline void GPIO\_PortToggle(GPIO\_Type \*base, uint32\_t port, uint32\_t mask)

Reverses current output logic of the multiple GPIO pins.

**Parameters**

- base – GPIO peripheral base pointer(Typically GPIO)
- port – GPIO port number
- mask – GPIO pin number macro

struct \_gpio\_pin\_config

*#include <fsl\_gpio.h>* The GPIO pin configuration structure.

Every pin can only be configured as either output pin or input pin at a time. If configured as a input pin, then leave the outputConfig unused.

### Public Members

*gpio\_pin\_direction\_t* pinDirection

GPIO direction, input or output

uint8\_t outputLogic

Set default output logic, no use in input

## 2.28 IOCON: I/O pin configuration

FSL\_IOCON\_DRIVER\_VERSION

IOCON driver version.

typedef struct *iocon\_group* iocon\_group\_t

Array of IOCON pin definitions passed to IOCON\_SetPinMuxing() must be in this format.

\_\_\_STATIC\_\_\_INLINE void IOCON\_PinMuxSet (IOCON\_Type \*base, uint8\_t port, uint8\_t pin, uint32\_t modefunc)

Sets I/O Control pin mux.

### Parameters

- base – : The base of IOCON peripheral on the chip
- port – : GPIO port to mux
- pin – : GPIO pin to mux
- modefunc – : OR'ed values of type IOCON\_\*

### Returns

Nothing

\_\_\_STATIC\_\_\_INLINE void IOCON\_SetPinMuxing (IOCON\_Type \*base, const iocon\_group\_t \*pinArray, uint32\_t arrayLength)

Set all I/O Control pin muxing.

### Parameters

- base – : The base of IOCON peripheral on the chip
- pinArray – : Pointer to array of pin mux selections
- arrayLength – : Number of entries in pinArray

### Returns

Nothing

FSL\_COMPONENT\_ID

IOCON\_FUNC0

IOCON function and mode selection definitions.

---

**Note:** See the User Manual for specific modes and functions supported by the various pins.  
Selects pin function 0

---

IOCON\_FUNC1

Selects pin function 1

IOCON\_FUNC2

Selects pin function 2

IOCON\_FUNC3

Selects pin function 3

IOCON\_FUNC4

Selects pin function 4

IOCON\_FUNC5

Selects pin function 5

IOCON\_FUNC6

Selects pin function 6

IOCON\_FUNC7

Selects pin function 7

struct `_iocon_group`

`#include <fsl_iocon.h>` Array of IOCON pin definitions passed to `IOCON_SetPinMuxing()` must be in this format.

## 2.29 LCDC: LCD Controller Driver

`status_t` `LCDC_Init(LCD_Type *base, const lcdc_config_t *config, uint32_t srcClock_Hz)`

Initialize the LCD module.

### Parameters

- `base` – LCD peripheral base address.
- `config` – Pointer to configuration structure, see to `lcdc_config_t`.
- `srcClock_Hz` – The LCD input clock (LCDCLK) frequency in Hz.

### Return values

- `kStatus_Success` – LCD is initialized successfully.
- `kStatus_InvalidArgument` – Initialize failed because of invalid argument.

`void` `LCDC_Deinit(LCD_Type *base)`

Deinitialize the LCD module.

### Parameters

- `base` – LCD peripheral base address.

`void` `LCDC_GetDefaultConfig(lcdc_config_t *config)`

Gets default pre-defined settings for initial configuration.

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

```
config->panelClock_Hz = 0U;
config->ppl = 0U;
config->hsw = 0U;
config->hfp = 0U;
config->hbp = 0U;
config->lpp = 0U;
config->vsw = 0U;
config->vfp = 0U;
config->vbp = 0U;
config->acBiasFreq = 1U;
config->polarityFlags = 0U;
config->enableLineEnd = false;
config->lineEndDelay = 0U;
```

(continues on next page)

(continued from previous page)

```
config->upperPanelAddr = 0U;  
config->lowerPanelAddr = 0U;  
config->bpp = kLDCD_1BPP;  
config->dataFormat = kLDCD_LittleEndian;  
config->swapRedBlue = false;  
config->display = kLDCD_DisplayTFT;
```

### Parameters

- config – Pointer to configuration structure.

static inline void LDCD\_Start(LCD\_Type \*base)

Start to output LCD timing signal.

The LCD power up sequence should be:

- Apply power to LCD, here all output signals are held low.
- When LCD power stabilized, call LDCD\_Start to output the timing signals.
- Apply contrast voltage to LCD panel. Delay if the display requires.
- Call LDCD\_PowerUp.

### Parameters

- base – LCD peripheral base address.

static inline void LDCD\_Stop(LCD\_Type \*base)

Stop the LCD timing signal.

The LCD power down sequence should be:

- Call LDCD\_PowerDown.
- Delay if the display requires. Disable contrast voltage to LCD panel.
- Call LDCD\_Stop to disable the timing signals.
- Disable power to LCD.

### Parameters

- base – LCD peripheral base address.

static inline void LDCD\_PowerUp(LCD\_Type \*base)

Power up the LCD and output the pixel signal.

### Parameters

- base – LCD peripheral base address.

static inline void LDCD\_PowerDown(LCD\_Type \*base)

Power down the LCD and disable the output pixel signal.

### Parameters

- base – LCD peripheral base address.

void LDCD\_SetPanelAddr(LCD\_Type \*base, *ldcd\_panel\_t* panel, uint32\_t addr)

Sets panel frame base address.

### Parameters

- base – LCD peripheral base address.
- panel – Which panel to set.
- addr – Frame base address, must be doubleword(64-bit) aligned.

```
void LCDC_SetPalette(LCD_Type *base, const uint32_t *palette, uint8_t count_words)
```

Sets palette.

#### Parameters

- base – LCD peripheral base address.
- palette – Pointer to the palette array.
- count\_words – Length of the palette array to set (how many words), it should not be larger than LCDC\_PALETTE\_SIZE\_WORDS.

```
static inline void LCDC_SetVerticalInterruptMode(LCD_Type *base,
                                                lcdc_vertical_compare_interrupt_mode_t mode)
```

Sets the vertical compare interrupt mode.

#### Parameters

- base – LCD peripheral base address.
- mode – The vertical compare interrupt mode.

```
void LCDC_EnableInterrupts(LCD_Type *base, uint32_t mask)
```

Enable LCD interrupts.

Example to enable LCD base address update interrupt and vertical compare interrupt:

```
LCDC_EnableInterrupts(LCD, kLCDC_BaseAddrUpdateInterrupt | kLCDC_
↳ VerticalCompareInterrupt);
```

#### Parameters

- base – LCD peripheral base address.
- mask – Interrupts to enable, it is OR'ed value of \_ldc\_interrups.

```
void LCDC_DisableInterrupts(LCD_Type *base, uint32_t mask)
```

Disable LCD interrupts.

Example to disable LCD base address update interrupt and vertical compare interrupt:

```
LCDC_DisableInterrupts(LCD, kLCDC_BaseAddrUpdateInterrupt | kLCDC_
↳ VerticalCompareInterrupt);
```

#### Parameters

- base – LCD peripheral base address.
- mask – Interrupts to disable, it is OR'ed value of \_ldc\_interrups.

```
uint32_t LCDC_GetInterruptsPendingStatus(LCD_Type *base)
```

Get LCD interrupt pending status.

Example:

```
uint32_t status;

status = LCDC_GetInterruptsPendingStatus(LCD);

if (kLCDC_BaseAddrUpdateInterrupt & status)
{
    LCD base address update interrupt occurred.
}

if (kLCDC_VeriticalCompareInterrupt & status)
```

(continues on next page)

(continued from previous page)

```
{
    LCD vertical compare interrupt occurred.
}
```

**Parameters**

- base – LCD peripheral base address.

**Returns**

Interrupts pending status, it is OR'ed value of `_lcdc_interrupts`.

`uint32_t` `LCDC_GetEnabledInterruptsPendingStatus(LCD_Type *base)`

Get LCD enabled interrupt pending status.

This function is similar with `LCDC_GetInterruptsPendingStatus`, the only difference is, this function only returns the pending status of the interrupts that have been enabled using `LCDC_EnableInterrupts`.

**Parameters**

- base – LCD peripheral base address.

**Returns**

Interrupts pending status, it is OR'ed value of `_lcdc_interrupts`.

`void` `LCDC_ClearInterruptsStatus(LCD_Type *base, uint32_t mask)`

Clear LCD interrupts pending status.

Example to clear LCD base address update interrupt and vertical compare interrupt pending status:

```
LCDC_ClearInterruptsStatus(LCD, kLCDC_BaseAddrUpdateInterrupt | kLCDC_
↪ VerticalCompareInterrupt);
```

**Parameters**

- base – LCD peripheral base address.
- mask – Interrupts to disable, it is OR'ed value of `_lcdc_interrupts`.

`void` `LCDC_SetCursorConfig(LCD_Type *base, const lcdc_cursor_config_t *config)`

Set the hardware cursor configuration.

This function should be called before enabling the hardware cursor. It supports initializing multiple cursor images at a time when using 32x32 pixels cursor.

For example:

```
uint32_t cursor0Img[LCDC_CURSOR_IMG_32X32_WORDS] = {...};
uint32_t cursor2Img[LCDC_CURSOR_IMG_32X32_WORDS] = {...};

lcdc_cursor_config_t cursorConfig;

LCDC_CursorGetDefaultConfig(&cursorConfig);

cursorConfig.image[0] = cursor0Img;
cursorConfig.image[2] = cursor2Img;

LCDC_SetCursorConfig(LCD, &cursorConfig);

LCDC_ChooseCursor(LCD, 0);
LCDC_SetCursorPosition(LCD, 0, 0);

LCDC_EnableCursor(LCD);
```

In this example, cursor 0 and cursor 2 image data are initialized, but cursor 1 and cursor 3 image data are not initialized because image[1] and image[2] are all NULL. With this, application could initialize all cursor images it will use at the beginning and call `LCDC_SetCursorImage` directly to display the one which it needs.

#### Parameters

- base – LCD peripheral base address.
- config – Pointer to the hardware cursor configuration structure.

`void LCDC_CursorGetDefaultConfig(lcdc_cursor_config_t *config)`

Get the hardware cursor default configuration.

The default configuration values are:

```
config->size = kLCDC_CursorSize32;
config->syncMode = kLCDC_CursorAsync;
config->palette0.red = 0U;
config->palette0.green = 0U;
config->palette0.blue = 0U;
config->palette1.red = 255U;
config->palette1.green = 255U;
config->palette1.blue = 255U;
config->image[0] = (uint32_t *)0;
config->image[1] = (uint32_t *)0;
config->image[2] = (uint32_t *)0;
config->image[3] = (uint32_t *)0;
```

#### Parameters

- config – Pointer to the hardware cursor configuration structure.

`static inline void LCDC_EnableCursor(LCD_Type *base, bool enable)`

Enable or disable the cursor.

#### Parameters

- base – LCD peripheral base address.
- enable – True to enable, false to disable.

`static inline void LCDC_ChooseCursor(LCD_Type *base, uint8_t index)`

Choose which cursor to display.

When using 32x32 cursor, the number of cursors supports is `LCDC_CURSOR_COUNT`. When using 64x64 cursor, the LCD only supports one cursor. This function selects which cursor to display when using 32x32 cursor. When synchronization mode is `kLCDC_CursorSync`, the change effects in the next frame. When synchronization mode is `kLCDC_CursorAsync`, change effects immediately.

---

**Note:** The function `LCDC_SetCursorPosition` must be called after this function to show the new cursor.

---

#### Parameters

- base – LCD peripheral base address.
- index – Index of the cursor to display.

`void LCDC_SetCursorPosition(LCD_Type *base, int32_t positionX, int32_t positionY)`

Set the position of cursor.

When synchronization mode is `kLDCD_CursorSync`, position change effects in the next frame. When synchronization mode is `kLDCD_CursorAsync`, position change effects immediately.

#### Parameters

- `base` – LCD peripheral base address.
- `positionX` – X ordinate of the cursor top-left measured in pixels
- `positionY` – Y ordinate of the cursor top-left measured in pixels

```
void LDCD_SetCursorImage(LCD_Type *base, lcdc_cursor_size_t size, uint8_t index, const
                        uint32_t *image)
```

Set the cursor image.

The interrupt `kLDCD_CursorInterrupt` indicates that last cursor pixel is displayed. When the hardware cursor is enabled,

#### Parameters

- `base` – LCD peripheral base address.
- `size` – The cursor size.
- `index` – Index of the cursor to set when using 32x32 cursor.
- `image` – Pointer to the cursor image. When using 32x32 cursor, the image size should be `LDCD_CURSOR_IMG_32X32_WORDS`. When using 64x64 cursor, the image size should be `LDCD_CURSOR_IMG_64X64_WORDS`.

`FSL_LCDC_DRIVER_VERSION`

LDCD driver version.

`enum _lcdc_polarity_flags`

LCD signal polarity flags.

*Values:*

`enumerator kLDCD_InvertVsyncPolarity`

Invert the VSYNC polarity, set to active low.

`enumerator kLDCD_InvertHsyncPolarity`

Invert the HSYNC polarity, set to active low.

`enumerator kLDCD_InvertClkPolarity`

Invert the panel clock polarity, set to drive data on falling edge.

`enumerator kLDCD_InvertDePolarity`

Invert the data enable (DE) polarity, set to active low.

`enum _lcdc_bpp`

LCD bits per pixel.

*Values:*

`enumerator kLDCD_1BPP`

1 bpp.

`enumerator kLDCD_2BPP`

2 bpp.

`enumerator kLDCD_4BPP`

4 bpp.

`enumerator kLDCD_8BPP`

8 bpp.



enumerator kLCDC\_16BPP

16 bpp.

enumerator kLCDC\_24BPP

24 bpp, TFT panel only.

enumerator kLCDC\_16BPP565

16 bpp, 5:6:5 mode.

enumerator kLCDC\_12BPP

12 bpp, 4:4:4 mode.

enum \_lcdc\_display

The types of display panel.

*Values:*

enumerator kLCDC\_DisplayTFT

Active matrix TFT panels with up to 24-bit bus interface.

enumerator kLCDC\_DisplaySingleMonoSTN4Bit

Single-panel monochrome STN (4-bit bus interface).

enumerator kLCDC\_DisplaySingleMonoSTN8Bit

Single-panel monochrome STN (8-bit bus interface).

enumerator kLCDC\_DisplayDualMonoSTN4Bit

Dual-panel monochrome STN (4-bit bus interface).

enumerator kLCDC\_DisplayDualMonoSTN8Bit

Dual-panel monochrome STN (8-bit bus interface).

enumerator kLCDC\_DisplaySingleColorSTN8Bit

Single-panel color STN (8-bit bus interface).

enumerator kLCDC\_DisplayDualColorSTN8Bit

Dual-panel color STN (8-bit bus interface).

enum \_lcdc\_data\_format

LCD panel buffer data format.

*Values:*

enumerator kLCDC\_LittleEndian

Little endian byte, little endian pixel.

enumerator kLCDC\_BigEndian

Big endian byte, big endian pixel.

enumerator kLCDC\_WinCeMode

little-endian byte, big-endian pixel for Windows CE mode.

enum \_lcdc\_vertical\_compare\_interrupt\_mode

LCD vertical compare interrupt mode.

*Values:*

enumerator kLCDC\_StartOfVsync

Generate vertical compare interrupt at start of VSYNC.

enumerator kLCDC\_StartOfBackPorch

Generate vertical compare interrupt at start of back porch.

enumerator kLDCD\_StartOfActiveVideo

Generate vertical compare interrupt at start of active video.

enumerator kLDCD\_StartOfFrontPorch

Generate vertical compare interrupt at start of front porch.

enum \_lcdc\_interrupts

LCD interrupts.

*Values:*

enumerator kLDCD\_CursorInterrupt

Cursor image read finished interrupt.

enumerator kLDCD\_FifoUnderflowInterrupt

FIFO underflow interrupt.

enumerator kLDCD\_BaseAddrUpdateInterrupt

Panel frame base address update interrupt.

enumerator kLDCD\_VerticalCompareInterrupt

Vertical compare interrupt.

enumerator kLDCD\_AhbErrorInterrupt

AHB master error interrupt.

enum \_lcdc\_panel

LCD panel frame.

*Values:*

enumerator kLDCD\_UpperPanel

Upper panel frame.

enumerator kLDCD\_LowerPanel

Lower panel frame.

enum \_lcdc\_cursor\_size

LCD hardware cursor size.

*Values:*

enumerator kLDCD\_CursorSize32

32x32 pixel cursor.

enumerator kLDCD\_CursorSize64

64x64 pixel cursor.

enum \_lcdc\_cursor\_sync\_mode

LCD hardware cursor frame synchronization mode.

*Values:*

enumerator kLDCD\_CursorAsync

Cursor change will be displayed immediately.

enumerator kLDCD\_CursorSync

Cursor change will be displayed in next frame.

typedef enum \_lcdc\_bpp lcdc\_bpp\_t

LCD bits per pixel.

typedef enum \_lcdc\_display lcdc\_display\_t

The types of display panel.

typedef enum *\_lcdc\_data\_format* lcdc\_data\_format\_t

LCD panel buffer data format.

typedef struct *\_lcdc\_config* lcdc\_config\_t

LCD configuration structure.

typedef enum *\_lcdc\_vertical\_compare\_interrupt\_mode* lcdc\_vertical\_compare\_interrupt\_mode\_t

LCD vertical compare interrupt mode.

typedef enum *\_lcdc\_panel* lcdc\_panel\_t

LCD panel frame.

typedef enum *\_lcdc\_cursor\_size* lcdc\_cursor\_size\_t

LCD hardware cursor size.

typedef struct *\_lcdc\_cursor\_palette* lcdc\_cursor\_palette\_t

LCD hardware cursor palette.

typedef enum *\_lcdc\_cursor\_sync\_mode* lcdc\_cursor\_sync\_mode\_t

LCD hardware cursor frame synchronization mode.

typedef struct *\_lcdc\_cursor\_config* lcdc\_cursor\_config\_t

LCD hardware cursor configuration structure.

LCDC\_CURSOR\_COUNT

How many hardware cursors supports.

LCDC\_CURSOR\_IMG\_BPP

LCD cursor image bits per pixel.

LCDC\_CURSOR\_IMG\_32X32\_WORDS

LCD 32x32 cursor image size in word(32-bit).

LCDC\_CURSOR\_IMG\_64X64\_WORDS

LCD 64x64 cursor image size in word(32-bit).

LCDC\_PALETTE\_SIZE\_WORDS

LCD palette size in words(32-bit).

struct *\_lcdc\_config*

*#include <fsl\_lcdc.h>* LCD configuration structure.

### Public Members

uint32\_t panelClock\_Hz

Panel clock in Hz.

uint16\_t ppl

Pixels per line, it must could be divided by 16.

uint8\_t hsw

HSYNC pulse width.

uint8\_t hfp

Horizontal front porch.

uint8\_t hbp

Horizontal back porch.

uint16\_t lpp

Lines per panal.

uint8\_t vsw

VSYNC pulse width.

uint8\_t vfp

Vrtical front porch.

uint8\_t vbp

Vertical back porch.

uint8\_t acBiasFreq

The number of line clocks between AC bias pin toggling. Only used for STN display.

uint16\_t polarityFlags

OR'ed value of `_ldc_polarity_flags`, used to contol the signal polarity.

bool enableLineEnd

Enable line end or not, the line end is a positive pulse with 4 panel clock.

uint8\_t lineEndDelay

The panel clocks between the last pixel of line and the start of line end.

uint32\_t upperPanelAddr

LCD upper panel base address, must be double-word(64-bit) align.

uint32\_t lowerPanelAddr

LCD lower panel base address, must be double-word(64-bit) align.

*ldc\_bpp\_t* bpp

LCD bits per pixel.

*ldc\_data\_format\_t* dataFormat

Data format.

bool swapRedBlue

Set true to use BGR format, set false to choose RGB format.

*ldc\_display\_t* display

The display type.

struct `_ldc_cursor_palette`

*#include <fsl\_ldc.h>* LCD hardware cursor palette.

### Public Members

uint8\_t red

Red color component.

uint8\_t green

Red color component.

uint8\_t blue

Red color component.

struct `_ldc_cursor_config`

*#include <fsl\_ldc.h>* LCD hardware cursor configuration structure.

### Public Members

*ldc\_cursor\_size\_t* size

Cursor size.

*lcdc\_cursor\_sync\_mode\_t* syncMode

Cursor synchronization mode.

*lcdc\_cursor\_palette\_t* palette0

Cursor palette 0.

*lcdc\_cursor\_palette\_t* palette1

Cursor palette 1.

uint32\_t \*image[4U]

Pointer to cursor image data.

## 2.30 MCAN: Controller Area Network Driver

void MCAN\_Init(CAN\_Type \*base, const *mcan\_config\_t* \*config, uint32\_t sourceClock\_Hz)

Initializes an MCAN instance.

This function initializes the MCAN module with user-defined settings. This example shows how to set up the *mcan\_config\_t* parameters and how to call the MCAN\_Init function by passing in these parameters.

```
mcan_config_t config;
config->baudRateA = 500000U;
config->baudRateD = 1000000U;
config->enableCanfdNormal = false;
config->enableCanfdSwitch = false;
config->enableLoopBackInt = false;
config->enableLoopBackExt = false;
config->enableBusMon = false;
MCAN_Init(CANFD0, &config, 8000000UL);
```

### Parameters

- base – MCAN peripheral base address.
- config – Pointer to the user-defined configuration structure.
- sourceClock\_Hz – MCAN Protocol Engine clock source frequency in Hz.

void MCAN\_Deinit(CAN\_Type \*base)

Deinitializes an MCAN instance.

This function deinitializes the MCAN module.

### Parameters

- base – MCAN peripheral base address.

void MCAN\_GetDefaultConfig(*mcan\_config\_t* \*config)

Gets the default configuration structure.

This function initializes the MCAN configuration structure to default values. The default values are as follows. config->baudRateA = 500000U; config->baudRateD = 1000000U; config->enableCanfdNormal = false; config->enableCanfdSwitch = false; config->enableLoopBackInt = false; config->enableLoopBackExt = false; config->enableBusMon = false;

### Parameters

- config – Pointer to the MCAN configuration structure.

static inline void MCAN\_EnterInitialMode(CAN\_Type \*base)

MCAN enters initialization mode.

After enter initialization mode, users can write access to the protected configuration registers.

#### Parameters

- base – MCAN peripheral base address.

static inline void MCAN\_EnterNormalMode(CAN\_Type \*base)

MCAN enters normal mode.

After initialization, INIT bit in CCCR register must be cleared to enter normal mode thus synchronizes to the CAN bus and ready for communication.

#### Parameters

- base – MCAN peripheral base address.

static inline void MCAN\_SetMsgRAMBase(CAN\_Type \*base, uint32\_t value)

Sets the MCAN Message RAM base address.

This function sets the Message RAM base address.

#### Parameters

- base – MCAN peripheral base address.
- value – Desired Message RAM base.

static inline uint32\_t MCAN\_GetMsgRAMBase(CAN\_Type \*base)

Gets the MCAN Message RAM base address.

This function gets the Message RAM base address.

#### Parameters

- base – MCAN peripheral base address.

#### Returns

Message RAM base address.

bool MCAN\_CalculateImprovedTimingValues(uint32\_t baudRate, uint32\_t sourceClock\_Hz, *mcan\_timing\_config\_t* \*pconfig)

Calculates the improved timing values by specific baudrates for classical CAN.

#### Parameters

- baudRate – The classical CAN speed in bps defined by user
- sourceClock\_Hz – The Source clock data speed in bps. Zero to disable baudrate switching
- pconfig – Pointer to the MCAN timing configuration structure.

#### Returns

TRUE if timing configuration found, FALSE if failed to find configuration

bool MCAN\_CalculateSpecifiedTimingValues(uint32\_t sourceClock\_Hz, *mcan\_timing\_config\_t* \*pconfig, const *mcan\_timing\_param\_t* \*pParamConfig)

Calculates the specified timing values for classical CAN with user-defined settings.

User can specify baudrates, sample point position, bus length, and transceiver propagation delay. This example shows how to set up the *mcan\_timing\_param\_t* parameters and how to call the this function by passing in these parameters.

```

mcan_timing_config_t timing_config;
mcan_timing_param_t timing_param;
timing_param.busLength = 1U;
timing_param.propTxRx = 230U;
timing_param.nominalbaudRate = 500000U;
timing_param.nominalSP = 800U;
MCAN_CalculateSpecifiedTimingValues(MCAN_CLK_FREQ, &timing_config, &timing_param);

```

Note that due to integer division will sacrifice the precision, actual sample point may not equal to expected. If actual sample point is not in allowed 2% range, this function will return false. So it is better to select higher source clock when baudrate is relatively high. This will ensure more time quanta and higher precision of sample point. Parameter busLength and propTxRx are optional and intended to verify whether propagation delay is too long to corrupt sample point. User can set these parameter zero if you do not want to consider this factor.

### Parameters

- sourceClock\_Hz – The Source clock data speed in bps.
- pconfig – Pointer to the MCAN timing configuration structure.
- config – Pointer to the MCAN timing parameters structure.

### Returns

TRUE if timing configuration found, FALSE if failed to find configuration

**void MCAN\_SetArbitrationTimingConfig(CAN\_Type \*base, const *mcan\_timing\_config\_t* \*config)**  
Sets the MCAN protocol arbitration phase timing characteristic.

This function gives user settings to CAN bus timing characteristic. The function is for an experienced user. For less experienced users, call the MCAN\_Init() and fill the baud rate field with a desired value. This provides the default arbitration phase timing characteristics.

Note that calling MCAN\_SetArbitrationTimingConfig() overrides the baud rate set in MCAN\_Init().

### Parameters

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

**status\_t MCAN\_SetBaudRate(CAN\_Type \*base, uint32\_t sourceClock\_Hz, uint32\_t baudRate\_Bps)**

Set Baud Rate of MCAN classic mode.

This function set the baud rate of MCAN base on MCAN\_CalculateImprovedTimingValues() API calculated timing values.

### Parameters

- base – MCAN peripheral base address.
- sourceClock\_Hz – Source Clock in Hz.
- baudRate\_Bps – Baud Rate in Bps.

### Returns

kStatus\_Success - Set CAN baud rate (only has Nominal phase) successfully.

**bool MCAN\_FDCalculateImprovedTimingValues(uint32\_t baudRate, uint32\_t baudRateFD, uint32\_t sourceClock\_Hz, *mcan\_timing\_config\_t* \*pconfig)**

Calculates the improved timing values by specific baudrates for CANFD.

### Parameters

- baudRate – The CANFD bus control speed in bps defined by user
- baudRateFD – The CANFD bus data speed in bps defined by user
- sourceClock\_Hz – The Source clock data speed in bps.
- pconfig – Pointer to the MCAN timing configuration structure.

**Returns**

TRUE if timing configuration found, FALSE if failed to find configuration

```
bool MCAN_FDCalculateSpecifiedTimingValues(uint32_t sourceClock_Hz, mcan_timing_config_t
                                         *pconfig, const mcan_timing_param_t
                                         *pParamConfig)
```

Calculates the specified timing values for CANFD with user-defined settings.

User can specify baudrates, sample point position, bus length, and transceiver propagation delay. This example shows how to set up the `mcan_timing_param_t` parameters and how to call the this function by passing in these parameters.

```
mcan_timing_config_t timing_config;
mcan_timing_param_t timing_param;
timing_param.busLength = 1U;
timing_param.propTxRx = 230U;
timing_param.nominalbaudRate = 500000U;
timing_param.nominalSP = 800U;
timing_param.databaudRate = 4000000U;
timing_param.dataSP = 700U;
MCAN_FDCalculateSpecifiedTimingValues(MCAN_CLK_FREQ, &timing_config, &timing_param);
```

Note that due to integer division will sacrifice the precision, actual sample point may not equal to expected. So it is better to select higher source clock when baudrate is relatively high. Select higher nominal baudrate when source clock is relatively high because large clock predivider will lead to less time quanta in data phase. This function will set predivider in arbitration phase equal to data phase. These methods will ensure more time quanta and higher precision of sample point. Parameter `busLength` and `propTxRx` are optional and intended to verify whether propagation delay is too long to corrupt sample point. User can set these parameter zero if you do not want to consider this factor.

**Parameters**

- sourceClock\_Hz – The Source clock data speed in bps.
- pconfig – Pointer to the MCAN timing configuration structure.
- config – Pointer to the MCAN timing parameters structure.

**Returns**

TRUE if timing configuration found, FALSE if failed to find configuration

```
status_t MCAN_SetBaudRateFD(CAN_Type *base, uint32_t sourceClock_Hz, uint32_t
                           baudRateN_Bps, uint32_t baudRateD_Bps)
```

Set Baud Rate of MCAN FD mode.

This function set the baud rate of MCAN FD base on MCAN\_FDCalculateImprovedTimingValues API calculated timing values.

**Parameters**

- base – MCAN peripheral base address.
- sourceClock\_Hz – Source Clock in Hz.
- baudRateN\_Bps – Nominal Baud Rate in Bps.
- baudRateD\_Bps – Data Baud Rate in Bps.



**Returns**

kStatus\_Success - Set CAN FD baud rate (include Nominal and Data phase) successfully.

void MCAN\_SetDataTimingConfig(CAN\_Type \*base, const *mcan\_timing\_config\_t* \*config)

Sets the MCAN protocol data phase timing characteristic.

This function gives user settings to CAN bus timing characteristic. The function is for an experienced user. For less experienced users, call the MCAN\_Init() and fill the baud rate field with a desired value. This provides the default data phase timing characteristics.

Note that calling MCAN\_SetArbitrationTimingConfig() overrides the baud rate set in MCAN\_Init().

**Parameters**

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

void MCAN\_SetRxFifo0Config(CAN\_Type \*base, const *mcan\_rx\_fifo\_config\_t* \*config)

Configures an MCAN receive fifo 0 buffer.

This function sets start address, element size, watermark, operation mode and datafield size of the receive fifo 0.

**Parameters**

- base – MCAN peripheral base address.
- config – The receive fifo 0 configuration structure.

void MCAN\_SetRxFifo1Config(CAN\_Type \*base, const *mcan\_rx\_fifo\_config\_t* \*config)

Configures an MCAN receive fifo 1 buffer.

This function sets start address, element size, watermark, operation mode and datafield size of the receive fifo 1.

**Parameters**

- base – MCAN peripheral base address.
- config – The receive fifo 1 configuration structure.

void MCAN\_SetRxBufferConfig(CAN\_Type \*base, const *mcan\_rx\_buffer\_config\_t* \*config)

Configures an MCAN receive buffer.

This function sets start address and datafield size of the receive buffer.

**Parameters**

- base – MCAN peripheral base address.
- config – The receive buffer configuration structure.

void MCAN\_SetTxEventFifoConfig(CAN\_Type \*base, const *mcan\_tx\_fifo\_config\_t* \*config)

Configures an MCAN transmit event fifo.

This function sets start address, element size, watermark of the transmit event fifo.

**Parameters**

- base – MCAN peripheral base address.
- config – The transmit event fifo configuration structure.

void MCAN\_SetTxBufferConfig(CAN\_Type \*base, const *mcan\_tx\_buffer\_config\_t* \*config)

Configures an MCAN transmit buffer.

This function sets start address, element size, fifo/queue mode and datafield size of the transmit buffer.

**Parameters**

- base – MCAN peripheral base address.
- config – The transmit buffer configuration structure.

void MCAN\_SetFilterConfig(CAN\_Type \*base, const *mcan\_frame\_filter\_config\_t* \*config)

Set filter configuration.

This function sets remote and non masking frames in global filter configuration, also the start address, list size in standard/extended ID filter configuration.

**Parameters**

- base – MCAN peripheral base address.
- config – The MCAN filter configuration.

*status\_t* MCAN\_SetMessageRamConfig(CAN\_Type \*base, const *mcan\_memory\_config\_t* \*config)

Set Message RAM related configuration.

---

**Note:** This function include Standard/extended ID filter, Rx FIFO 0/1, Rx buffer, Tx event FIFO and Tx buffer configurations

---

**Parameters**

- base – MCAN peripheral base address.
- config – The MCAN filter configuration.

**Return values**

- kStatus\_Success – - Message RAM related configuration Successfully.
- kStatus\_Fail – - Message RAM related configure fail due to wrong address parameter.

void MCAN\_SetSTDFilterElement(CAN\_Type \*base, const *mcan\_frame\_filter\_config\_t* \*config, const *mcan\_std\_filter\_element\_config\_t* \*filter, uint8\_t idx)

Set standard message ID filter element configuration.

**Parameters**

- base – MCAN peripheral base address.
- config – The MCAN filter configuration.
- filter – The MCAN standard message ID filter element configuration.
- idx – The standard message ID filter element index.

void MCAN\_SetEXTFilterElement(CAN\_Type \*base, const *mcan\_frame\_filter\_config\_t* \*config, const *mcan\_ext\_filter\_element\_config\_t* \*filter, uint8\_t idx)

Set extended message ID filter element configuration.

**Parameters**

- base – MCAN peripheral base address.
- config – The MCAN filter configuration.
- filter – The MCAN extended message ID filter element configuration.
- idx – The extended message ID filter element index.

```
static inline uint32_t MCAN_GetStatusFlag(CAN_Type *base, uint32_t mask)
```

Gets the MCAN module interrupt flags.

This function gets all MCAN interrupt status flags.

**Parameters**

- base – MCAN peripheral base address.
- mask – The ORed MCAN interrupt mask.

**Returns**

MCAN status flags which are ORed.

```
static inline void MCAN_ClearStatusFlag(CAN_Type *base, uint32_t mask)
```

Clears the MCAN module interrupt flags.

This function clears MCAN interrupt status flags.

**Parameters**

- base – MCAN peripheral base address.
- mask – The ORed MCAN interrupt mask.

```
static inline bool MCAN_GetRxBufferStatusFlag(CAN_Type *base, uint8_t idx)
```

Gets the new data flag of specific Rx Buffer.

This function gets new data flag of specific Rx Buffer.

**Parameters**

- base – MCAN peripheral base address.
- idx – Rx Buffer index.

**Returns**

Rx Buffer new data status flag.

```
static inline void MCAN_ClearRxBufferStatusFlag(CAN_Type *base, uint8_t idx)
```

Clears the new data flag of specific Rx Buffer.

This function clears new data flag of specific Rx Buffer.

**Parameters**

- base – MCAN peripheral base address.
- idx – Rx Buffer index.

```
static inline void MCAN_EnableInterrupts(CAN_Type *base, uint32_t line, uint32_t mask)
```

Enables MCAN interrupts according to the provided interrupt line and mask.

This function enables the MCAN interrupts according to the provided interrupt line and mask. The mask is a logical OR of enumeration members.

**Parameters**

- base – MCAN peripheral base address.
- line – Interrupt line number, 0 or 1.
- mask – The interrupts to enable.

```
static inline void MCAN_EnableTransmitBufferInterrupts(CAN_Type *base, uint8_t idx)
```

Enables MCAN Tx Buffer interrupts according to the provided index.

This function enables the MCAN Tx Buffer interrupts.

**Parameters**

- base – MCAN peripheral base address.

- idx – Tx Buffer index.

static inline void MCAN\_DisableTransmitBufferInterrupts(CAN\_Type \*base, uint8\_t idx)

Disables MCAN Tx Buffer interrupts according to the provided index.

This function disables the MCAN Tx Buffer interrupts.

#### Parameters

- base – MCAN peripheral base address.
- idx – Tx Buffer index.

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

Disables MCAN interrupts according to the provided mask.

This function disables the MCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members.

#### Parameters

- base – MCAN peripheral base address.
- mask – The interrupts to disable.

uint32\_t MCAN\_IsTransmitRequestPending(CAN\_Type \*base, uint8\_t idx)

Gets the Tx buffer request pending status.

This function returns Tx Message Buffer transmission request pending status.

#### Parameters

- base – MCAN peripheral base address.
- idx – The MCAN Tx Buffer index.

uint32\_t MCAN\_IsTransmitOccurred(CAN\_Type \*base, uint8\_t idx)

Gets the Tx buffer transmission occurred status.

This function returns Tx Message Buffer transmission occurred status.

#### Parameters

- base – MCAN peripheral base address.
- idx – The MCAN Tx Buffer index.

status\_t MCAN\_WriteTxBuffer(CAN\_Type \*base, uint8\_t idx, const *mcan\_tx\_buffer\_frame\_t* \*pTxFrame)

Writes an MCAN Message to the Transmit Buffer.

This function writes a CAN Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN Message transmit. After that the function returns immediately.

#### Parameters

- base – MCAN peripheral base address.
- idx – The MCAN Tx Buffer index.
- pTxFrame – Pointer to CAN message frame to be sent.

status\_t MCAN\_ReadRxBuffer(CAN\_Type \*base, uint8\_t idx, *mcan\_rx\_buffer\_frame\_t* \*pRxFrame)

Reads an MCAN Message from Rx Buffer.

This function reads a CAN message from the Rx Buffer in the Message RAM.

#### Parameters

- base – MCAN peripheral base address.

- idx – The MCAN Rx Buffer index.
- pRxFrame – Pointer to CAN message frame structure for reception.

**Return values**

kStatus\_Success – - Read Message from Rx Buffer successfully.

```
status_t MCAN_ReadRxFifo(CAN_Type *base, uint8_t fifoBlock, mcan_rx_buffer_frame_t  
                        *pRxFrame)
```

Reads an MCAN Message from Rx FIFO.

This function reads a CAN message from the Rx FIFO in the Message RAM.

**Parameters**

- base – MCAN peripheral base address.
- fifoBlock – Rx FIFO block 0 or 1.
- pRxFrame – Pointer to CAN message frame structure for reception.

**Return values**

kStatus\_Success – - Read Message from Rx FIFO successfully.

```
static inline void MCAN_TransmitAddRequest(CAN_Type *base, uint8_t idx)
```

Tx Buffer add request to send message out.

This function add sending request to corresponding Tx Buffer.

**Parameters**

- base – MCAN peripheral base address.
- idx – Tx Buffer index.

```
static inline void MCAN_TransmitCancelRequest(CAN_Type *base, uint8_t idx)
```

Tx Buffer cancel sending request.

This function clears Tx buffer request pending bit.

**Parameters**

- base – MCAN peripheral base address.
- idx – Tx Buffer index.

```
status_t MCAN_TransferSendBlocking(CAN_Type *base, uint8_t idx, mcan_tx_buffer_frame_t  
                                *pTxFrame)
```

Performs a polling send transaction on the CAN bus.

Note that a transfer handle does not need to be created before calling this API.

**Parameters**

- base – MCAN peripheral base pointer.
- idx – The MCAN buffer index.
- pTxFrame – Pointer to CAN message frame to be sent.

**Return values**

- kStatus\_Success – - Write Tx Message Buffer Successfully.
- kStatus\_Fail – - Tx Message Buffer is currently in use.

```
status_t MCAN_TransferReceiveBlocking(CAN_Type *base, uint8_t idx, mcan_rx_buffer_frame_t  
                                *pRxFrame)
```

Performs a polling receive transaction on the CAN bus.

Note that a transfer handle does not need to be created before calling this API.

**Parameters**

- base – MCAN peripheral base pointer.
- idx – The MCAN buffer index.
- pRxFrame – Pointer to CAN message frame structure for reception.

**Return values**

- kStatus\_Success – Read Rx Message Buffer Successfully.
- kStatus\_Fail – No new message.

*status\_t* MCAN\_TransferReceiveFifoBlocking(CAN\_Type \*base, uint8\_t fifoBlock,  
mcan\_rx\_buffer\_frame\_t \*pRxFrame)

Performs a polling receive transaction from Rx FIFO on the CAN bus.

Note that a transfer handle does not need to be created before calling this API.

**Parameters**

- base – MCAN peripheral base pointer.
- fifoBlock – Rx FIFO block, 0 or 1.
- pRxFrame – Pointer to CAN message frame structure for reception.

**Return values**

- kStatus\_Success – Read Message from Rx FIFO successfully.
- kStatus\_Fail – No new message in Rx FIFO.

void MCAN\_TransferCreateHandle(CAN\_Type \*base, mcan\_handle\_t \*handle,  
mcan\_transfer\_callback\_t callback, void \*userData)

Initializes the MCAN handle.

This function initializes the MCAN handle, which can be used for other MCAN transactional APIs. Usually, for a specified MCAN instance, call this API once to get the initialized handle.

**Parameters**

- base – MCAN peripheral base address.
- handle – MCAN handle pointer.
- callback – The callback function.
- userData – The parameter of the callback function.

*status\_t* MCAN\_TransferSendNonBlocking(CAN\_Type \*base, mcan\_handle\_t \*handle,  
mcan\_buffer\_transfer\_t \*xfer)

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

**Parameters**

- base – MCAN peripheral base address.
- handle – MCAN handle pointer.
- xfer – MCAN Buffer transfer structure. See the mcan\_buffer\_transfer\_t.

**Return values**

- kStatus\_Success – Start Tx Buffer sending process successfully.
- kStatus\_Fail – Write Tx Buffer failed.
- kStatus\_MCAN\_TxBusy – Tx Buffer is in use.

```
status_t MCAN_TransferReceiveFifoNonBlocking(CAN_Type *base, uint8_t fifoBlock,  
                                              mcan_handle_t *handle, mcan_fifo_transfer_t  
                                              *xfer)
```

Receives a message from Rx FIFO using IRQ.

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

#### Parameters

- base – MCAN peripheral base address.
- handle – MCAN handle pointer.
- fifoBlock – Rx FIFO block, 0 or 1.
- xfer – MCAN Rx FIFO transfer structure. See the *mcan\_fifo\_transfer\_t*.

#### Return values

- *kStatus\_Success* – - Start Rx FIFO receiving process successfully.
- *kStatus\_MCAN\_RxFifo0Busy* – - Rx FIFO 0 is currently in use.
- *kStatus\_MCAN\_RxFifo1Busy* – - Rx FIFO 1 is currently in use.

```
void MCAN_TransferAbortSend(CAN_Type *base, mcan_handle_t *handle, uint8_t bufferIdx)
```

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

#### Parameters

- base – MCAN peripheral base address.
- handle – MCAN handle pointer.
- bufferIdx – The MCAN Buffer index.

```
void MCAN_TransferAbortReceiveFifo(CAN_Type *base, uint8_t fifoBlock, mcan_handle_t  
                                   *handle)
```

Aborts the interrupt driven message receive from Rx FIFO process.

This function aborts the interrupt driven message receive from Rx FIFO process.

#### Parameters

- base – MCAN peripheral base address.
- fifoBlock – MCAN Fifo block, 0 or 1.
- handle – MCAN handle pointer.

```
void MCAN_TransferHandleIRQ(CAN_Type *base, mcan_handle_t *handle)
```

MCAN IRQ handle function.

This function handles the MCAN Error, the Buffer, and the Rx FIFO IRQ request.

#### Parameters

- base – MCAN peripheral base address.
- handle – MCAN handle pointer.

```
FSL_MCAN_DRIVER_VERSION
```

MCAN driver version.

MCAN transfer status.

*Values:*

enumerator kStatus\_MCAN\_TxBusy  
Tx Buffer is Busy.

enumerator kStatus\_MCAN\_TxIdle  
Tx Buffer is Idle.

enumerator kStatus\_MCAN\_RxBusy  
Rx Buffer is Busy.

enumerator kStatus\_MCAN\_RxIdle  
Rx Buffer is Idle.

enumerator kStatus\_MCAN\_RxFifo0New  
New message written to Rx FIFO 0.

enumerator kStatus\_MCAN\_RxFifo0Idle  
Rx FIFO 0 is Idle.

enumerator kStatus\_MCAN\_RxFifo0Watermark  
Rx FIFO 0 fill level reached watermark.

enumerator kStatus\_MCAN\_RxFifo0Full  
Rx FIFO 0 full.

enumerator kStatus\_MCAN\_RxFifo0Lost  
Rx FIFO 0 message lost.

enumerator kStatus\_MCAN\_RxFifo1New  
New message written to Rx FIFO 1.

enumerator kStatus\_MCAN\_RxFifo1Idle  
Rx FIFO 1 is Idle.

enumerator kStatus\_MCAN\_RxFifo1Watermark  
Rx FIFO 1 fill level reached watermark.

enumerator kStatus\_MCAN\_RxFifo1Full  
Rx FIFO 1 full.

enumerator kStatus\_MCAN\_RxFifo1Lost  
Rx FIFO 1 message lost.

enumerator kStatus\_MCAN\_RxFifo0Busy  
Rx FIFO 0 is busy.

enumerator kStatus\_MCAN\_RxFifo1Busy  
Rx FIFO 1 is busy.

enumerator kStatus\_MCAN\_ErrorStatus  
MCAN Module Error and Status.

enumerator kStatus\_MCAN\_UnHandled  
UnHandled Interrupt asserted.

enum \_mcan\_flags  
MCAN status flags.

This provides constants for the MCAN status flags for use in the MCAN functions. Note: The CPU read action clears MCAN\_ErrorFlag, therefore user need to read MCAN\_ErrorFlag and distinguish which error is occur using \_mcan\_error\_flags enumerations.

*Values:*



enumerator kMCAN\_AccesstoRsvdFlag  
CAN Synchronization Status.

enumerator kMCAN\_ProtocolErrDIntFlag  
Tx Warning Interrupt Flag.

enumerator kMCAN\_ProtocolErrAIntFlag  
Rx Warning Interrupt Flag.

enumerator kMCAN\_BusOffIntFlag  
Tx Error Warning Status.

enumerator kMCAN\_ErrorWarningIntFlag  
Rx Error Warning Status.

enumerator kMCAN\_ErrorPassiveIntFlag  
Rx Error Warning Status.

enum \_mcan\_rx\_fifo\_flags

MCAN Rx FIFO status flags.

The MCAN Rx FIFO Status enumerations are used to determine the status of the Rx FIFO.

*Values:*

enumerator kMCAN\_RxFifo0NewFlag  
Rx FIFO 0 new message flag.

enumerator kMCAN\_RxFifo0WatermarkFlag  
Rx FIFO 0 watermark reached flag.

enumerator kMCAN\_RxFifo0FullFlag  
Rx FIFO 0 full flag.

enumerator kMCAN\_RxFifo0LostFlag  
Rx FIFO 0 message lost flag.

enumerator kMCAN\_RxFifo1NewFlag  
Rx FIFO 0 new message flag.

enumerator kMCAN\_RxFifo1WatermarkFlag  
Rx FIFO 0 watermark reached flag.

enumerator kMCAN\_RxFifo1FullFlag  
Rx FIFO 0 full flag.

enumerator kMCAN\_RxFifo1LostFlag  
Rx FIFO 0 message lost flag.

enum \_mcan\_tx\_flags

MCAN Tx status flags.

The MCAN Tx Status enumerations are used to determine the status of the Tx Buffer/Event FIFO.

*Values:*

enumerator kMCAN\_TxTransmitCompleteFlag  
Transmission completed flag.

enumerator kMCAN\_TxTransmitCancelFinishFlag  
Transmission cancellation finished flag.

enumerator kMCAN\_TxEventFifoLostFlag

Tx Event FIFO element lost.

enumerator kMCAN\_TxEventFifoFullFlag

Tx Event FIFO full.

enumerator kMCAN\_TxEventFifoWatermarkFlag

Tx Event FIFO fill level reached watermark.

enumerator kMCAN\_TxEventFifoNewFlag

Tx Handler wrote Tx Event FIFO element flag.

enumerator kMCAN\_TxEventFifoEmptyFlag

Tx FIFO empty flag.

enum \_mcan\_interrupt\_enable

MCAN interrupt configuration structure, default settings all disabled.

This structure contains the settings for all of the MCAN Module interrupt configurations.

*Values:*

enumerator kMCAN\_BusOffInterruptEnable

Bus Off interrupt.

enumerator kMCAN\_ErrorInterruptEnable

Error interrupt.

enumerator kMCAN\_WarningInterruptEnable

Rx Warning interrupt.

enum \_mcan\_frame\_idformat

MCAN frame format.

*Values:*

enumerator kMCAN\_FrameIDStandard

Standard frame format attribute.

enumerator kMCAN\_FrameIDExtend

Extend frame format attribute.

enum \_mcan\_frame\_type

MCAN frame type.

*Values:*

enumerator kMCAN\_FrameTypeData

Data frame type attribute.

enumerator kMCAN\_FrameTypeRemote

Remote frame type attribute.

enum \_mcan\_bytes\_in\_datafield

MCAN frame datafield size.

*Values:*

enumerator kMCAN\_8ByteDatafield

8 byte data field.

enumerator kMCAN\_12ByteDatafield

12 byte data field.

enumerator kMCAN\_16ByteDatafield  
16 byte data field.

enumerator kMCAN\_20ByteDatafield  
20 byte data field.

enumerator kMCAN\_24ByteDatafield  
24 byte data field.

enumerator kMCAN\_32ByteDatafield  
32 byte data field.

enumerator kMCAN\_48ByteDatafield  
48 byte data field.

enumerator kMCAN\_64ByteDatafield  
64 byte data field.

enum \_\_mcan\_fifo\_type  
MCAN Rx FIFO block number.

*Values:*

enumerator kMCAN\_Fifo0  
CAN Rx FIFO 0.

enumerator kMCAN\_Fifo1  
CAN Rx FIFO 1.

enum \_\_mcan\_fifo\_opmode\_config  
MCAN FIFO Operation Mode.

*Values:*

enumerator kMCAN\_FifoBlocking  
FIFO blocking mode.

enumerator kMCAN\_FifoOverwrite  
FIFO overwrite mode.

enum \_\_mcan\_txmode\_config  
MCAN Tx FIFO/Queue Mode.

*Values:*

enumerator kMCAN\_txFifo  
Tx FIFO operation.

enumerator kMCAN\_txQueue  
Tx Queue operation.

enum \_\_mcan\_remote\_frame\_config  
MCAN remote frames treatment.

*Values:*

enumerator kMCAN\_filterFrame  
Filter remote frames.

enumerator kMCAN\_rejectFrame  
Reject all remote frames.

enum `_mcan_nonmasking_frame_config`

MCAN non-masking frames treatment.

*Values:*

enumerator `kMCAN_acceptinFifo0`

Accept non-masking frames in Rx FIFO 0.

enumerator `kMCAN_acceptinFifo1`

Accept non-masking frames in Rx FIFO 1.

enumerator `kMCAN_reject0`

Reject non-masking frames.

enumerator `kMCAN_reject1`

Reject non-masking frames.

enum `_mcan_fec_config`

MCAN Filter Element Configuration.

*Values:*

enumerator `kMCAN_disable`

Disable filter element.

enumerator `kMCAN_storeinFifo0`

Store in Rx FIFO 0 if filter matches.

enumerator `kMCAN_storeinFifo1`

Store in Rx FIFO 1 if filter matches.

enumerator `kMCAN_reject`

Reject ID if filter matches.

enumerator `kMCAN_setprio`

Set priority if filter matches.

enumerator `kMCAN_setpriofifo0`

Set priority and store in FIFO 0 if filter matches.

enumerator `kMCAN_setpriofifo1`

Set priority and store in FIFO 1 if filter matches.

enumerator `kMCAN_storeinbuffer`

Store into Rx Buffer or as debug message.

enum `_mcan_std_filter_type`

MCAN Filter Type.

*Values:*

enumerator `kMCAN_range`

Range filter from SFID1 to SFID2.

enumerator `kMCAN_dual`

Dual ID filter for SFID1 or SFID2.

enumerator `kMCAN_classic`

Classic filter: SFID1 = filter; SFID2 = mask.

enumerator `kMCAN_disableORrange2`

Filter element disabled for standard filter or Range filter; XIDAM mask not applied for extended filter.

`typedef enum _mcan_frame_idformat mcan_frame_idformat_t`  
MCAN frame format.

`typedef enum _mcan_frame_type mcan_frame_type_t`  
MCAN frame type.

`typedef enum _mcan_bytes_in_datafield mcan_bytes_in_datafield_t`  
MCAN frame datafield size.

`typedef struct _mcan_tx_buffer_frame mcan_tx_buffer_frame_t`  
MCAN Tx Buffer structure.

`typedef struct _mcan_rx_buffer_frame mcan_rx_buffer_frame_t`  
MCAN Rx FIFO/Buffer structure.

`typedef enum _mcan_fifo_type mcan_fifo_type_t`  
MCAN Rx FIFO block number.

`typedef enum _mcan_fifo_opmode_config mcan_fifo_opmode_config_t`  
MCAN FIFO Operation Mode.

`typedef enum _mcan_txmode_config mcan_txmode_config_t`  
MCAN Tx FIFO/Queue Mode.

`typedef enum _mcan_remote_frame_config mcan_remote_frame_config_t`  
MCAN remote frames treatment.

`typedef enum _mcan_nonmasking_frame_config mcan_nonmasking_frame_config_t`  
MCAN non-masking frames treatment.

`typedef enum _mcan_fec_config mcan_fec_config_t`  
MCAN Filter Element Configuration.

`typedef struct _mcan_rx_fifo_config mcan_rx_fifo_config_t`  
MCAN Rx FIFO configuration.

`typedef struct _mcan_rx_buffer_config mcan_rx_buffer_config_t`  
MCAN Rx Buffer configuration.

`typedef struct _mcan_tx_fifo_config mcan_tx_fifo_config_t`  
MCAN Tx Event FIFO configuration.

`typedef struct _mcan_tx_buffer_config mcan_tx_buffer_config_t`  
MCAN Tx Buffer configuration.

`typedef enum _mcan_std_filter_type mcan_filter_type_t`  
MCAN Filter Type.

`typedef struct _mcan_std_filter_element_config mcan_std_filter_element_config_t`  
MCAN Standard Message ID Filter Element.

`typedef struct _mcan_ext_filter_element_config mcan_ext_filter_element_config_t`  
MCAN Extended Message ID Filter Element.

`typedef struct _mcan_frame_filter_config mcan_frame_filter_config_t`  
MCAN Rx filter configuration.

`typedef struct _mcan_timing_config mcan_timing_config_t`  
MCAN protocol timing characteristic configuration structure.

`typedef struct _mcan_timing_param mcan_timing_param_t`  
MCAN bit timing parameter configuration structure.

```
typedef struct _mcan_memory_config mcan_memory_config_t
```

MCAN Message RAM related configuration structure.

```
typedef struct _mcan_config mcan_config_t
```

MCAN module configuration structure.

```
typedef struct _mcan_buffer_transfer mcan_buffer_transfer_t
```

MCAN Buffer transfer.

```
typedef struct _mcan_fifo_transfer mcan_fifo_transfer_t
```

MCAN Rx FIFO transfer.

```
typedef struct _mcan_handle mcan_handle_t
```

MCAN handle structure definition.

```
typedef void (*mcan_transfer_callback_t)(CAN_Type *base, mcan_handle_t *handle, status_t  
status, uint32_t result, void *userData)
```

MCAN transfer callback function.

The MCAN transfer callback returns a value from the underlying layer. If the status equals to `kStatus_MCAN_ErrorStatus`, the result parameter is the Content of MCAN status register which can be used to get the working status(or error status) of MCAN module. If the status equals to other MCAN Message Buffer transfer status, the result is the index of Message Buffer that generate transfer event. If the status equals to other MCAN Message Buffer transfer status, the result is meaningless and should be Ignored.

MCAN\_RETRY\_TIMES

```
struct _mcan_tx_buffer_frame
```

*#include <fsl\_mcan.h>* MCAN Tx Buffer structure.

### Public Members

uint8\_t size

classical CAN is 8(bytes), FD is 12/64 such.

```
struct _mcan_rx_buffer_frame
```

*#include <fsl\_mcan.h>* MCAN Rx FIFO/Buffer structure.

### Public Members

uint8\_t size

classical CAN is 8(bytes), FD is 12/64 such.

```
struct _mcan_rx_fifo_config
```

*#include <fsl\_mcan.h>* MCAN Rx FIFO configuration.

### Public Members

uint32\_t address

FIFO start address.

uint32\_t elementSize

FIFO element number.

uint32\_t watermark

FIFO watermark level.

*mcan\_fifo\_opmode\_config\_t* opmode  
FIFO blocking/overwrite mode.

*mcan\_bytes\_in\_datafield\_t* datafieldSize  
Data field size per frame, size>8 is for CANFD.

struct *\_mcan\_rx\_buffer\_config*  
*#include <fsl\_mcan.h>* MCAN Rx Buffer configuration.

### Public Members

uint32\_t address  
Rx Buffer start address.

*mcan\_bytes\_in\_datafield\_t* datafieldSize  
Data field size per frame, size>8 is for CANFD.

struct *\_mcan\_tx\_fifo\_config*  
*#include <fsl\_mcan.h>* MCAN Tx Event FIFO configuration.

### Public Members

uint32\_t address  
Event fifo start address.

uint32\_t elementSize  
FIFO element number.

uint32\_t watermark  
FIFO watermark level.

struct *\_mcan\_tx\_buffer\_config*  
*#include <fsl\_mcan.h>* MCAN Tx Buffer configuration.

### Public Members

uint32\_t address  
Tx Buffers Start Address.

uint32\_t dedicatedSize  
Number of Dedicated Transmit Buffers.

uint32\_t fqSize  
Transmit FIFO/Queue Size.

*mcan\_txmode\_config\_t* mode  
Tx FIFO/Queue Mode.

*mcan\_bytes\_in\_datafield\_t* datafieldSize  
Data field size per frame, size>8 is for CANFD.

struct *\_mcan\_std\_filter\_element\_config*  
*#include <fsl\_mcan.h>* MCAN Standard Message ID Filter Element.

**Public Members**

uint32\_t sfid2

Standard Filter ID 2.

uint32\_t \_\_\_\_pad0\_\_\_\_

Reserved.

uint32\_t sfid1

Standard Filter ID 1.

uint32\_t sfec

Standard Filter Element Configuration.

uint32\_t sft

Standard Filter Type.

struct \_mcan\_ext\_filter\_element\_config

*#include <fsl\_mcan.h>* MCAN Extended Message ID Filter Element.

**Public Members**

uint32\_t efid1

Extended Filter ID 1.

uint32\_t efec

Extended Filter Element Configuration.

uint32\_t efid2

Extended Filter ID 2.

uint32\_t \_\_\_\_pad0\_\_\_\_

Reserved.

uint32\_t eft

Extended Filter Type.

struct \_mcan\_frame\_filter\_config

*#include <fsl\_mcan.h>* MCAN Rx filter configuration.

**Public Members**

uint32\_t address

Filter start address.

uint32\_t listSize

Filter list size.

mcan\_frame\_idformat\_t idFormat

Frame format.

mcan\_remote\_frame\_config\_t remFrame

Remote frame treatment.

mcan\_nonmasking\_frame\_config\_t nmFrame

Non-masking frame treatment.

struct \_mcan\_timing\_config

*#include <fsl\_mcan.h>* MCAN protocol timing characteristic configuration structure.



**Public Members**

uint16\_t preDivider  
Nominal Clock Pre-scaler Division Factor.

uint8\_t rJumpwidth  
Nominal Re-sync Jump Width.

uint8\_t seg1  
Nominal Time Segment 1.

uint8\_t seg2  
Nominal Time Segment 2.

uint16\_t datapreDivider  
Data Clock Pre-scaler Division Factor.

uint8\_t datarJumpwidth  
Data Re-sync Jump Width.

uint8\_t dataseg1  
Data Time Segment 1.

uint8\_t dataseg2  
Data Time Segment 2.

struct \_\_mcan\_timing\_param  
*#include <fsl\_mcan.h>* MCAN bit timing parameter configuration structure.

**Public Members**

uint32\_t busLength  
Maximum Bus length in meter.

uint32\_t propTxRx  
Transceiver propagation delay in nanosecond.

uint32\_t nominalbaudRate  
Baud rate of Arbitration phase in bps.

uint32\_t nominalSP  
Sample point of Arbitration phase, range in 10 ~ 990, 800 means 80%.

uint32\_t databaudRate  
Baud rate of Data phase in bps.

uint32\_t dataSP  
Sample point of Data phase, range in 0 ~ 1000, 800 means 80%.

struct \_\_mcan\_memory\_config  
*#include <fsl\_mcan.h>* MCAN Message RAM related configuration structure.

**Public Members**

uint32\_t baseAddr  
Message RAM base address, should be 4k alignment.

struct \_\_mcan\_config  
*#include <fsl\_mcan.h>* MCAN module configuration structure.

### Public Members

uint32\_t baudRateA

Baud rate of Arbitration phase in bps.

uint32\_t baudRateD

Baud rate of Data phase in bps.

bool enableCanfdNormal

Enable or Disable CANFD normal.

bool enableCanfdSwitch

Enable or Disable CANFD with baudrate switch.

bool enableLoopBackInt

Enable or Disable Internal Back.

bool enableLoopBackExt

Enable or Disable External Loop Back.

bool enableBusMon

Enable or Disable Bus Monitoring Mode.

*mcan\_timing\_config\_t* timingConfig

Protocol timing .

struct *\_mcan\_buffer\_transfer*

*#include <fsl\_mcan.h>* MCAN Buffer transfer.

### Public Members

*mcan\_tx\_buffer\_frame\_t* \*frame

The buffer of CAN Message to be transfer.

uint8\_t bufferIdx

The index of Message buffer used to transfer Message.

struct *\_mcan\_fifo\_transfer*

*#include <fsl\_mcan.h>* MCAN Rx FIFO transfer.

### Public Members

*mcan\_rx\_buffer\_frame\_t* \*frame

The buffer of CAN Message to be received from Rx FIFO.

struct *\_mcan\_handle*

*#include <fsl\_mcan.h>* MCAN handle structure.

### Public Members

*mcan\_transfer\_callback\_t* callback

Callback function.

void \*userData

MCAN callback function parameter.

*mcan\_tx\_buffer\_frame\_t* \*volatile bufferFrameBuf[64]

The buffer for received data from Buffers.

*mcan\_rx\_buffer\_frame\_t* \*volatile rxFifoFrameBuf

The buffer for received data from Rx FIFO.

volatile uint8\_t bufferState[64]

Message Buffer transfer state.

volatile uint8\_t rxFifoState

Rx FIFO transfer state.

struct \_\_unnamed11\_\_

### Public Members

uint32\_t id

CAN Frame Identifier.

uint32\_t rtr

CAN Frame Type(DATA or REMOTE).

uint32\_t xtd

CAN Frame Type(STD or EXT).

uint32\_t esi

CAN Frame Error State Indicator.

struct \_\_unnamed13\_\_

### Public Members

uint32\_t dlc

Data Length Code 9 10 11 12 13 14 15 Number of data bytes 12 16 20 24 32 48 64

uint32\_t brs

Bit Rate Switch.

uint32\_t fdf

CAN FD format.

uint32\_t \_\_pad1\_\_

Reserved.

uint32\_t efc

Event FIFO control.

uint32\_t mm

Message Marker.

struct \_\_unnamed15\_\_

### Public Members

uint32\_t id

CAN Frame Identifier.

uint32\_t rtr

CAN Frame Type(DATA or REMOTE).

uint32\_t xtd

CAN Frame Type(STD or EXT).

```
uint32_t esi
    CAN Frame Error State Indicator.
struct __unnamed17__
```

### Public Members

```
uint32_t rxts
    Rx Timestamp.
uint32_t dlc
    Data Length Code 9 10 11 12 13 14 15 Number of data bytes 12 16 20 24 32 48 64
uint32_t brs
    Bit Rate Switch.
uint32_t fdf
    CAN FD format.
uint32_t __pad0__
    Reserved.
uint32_t fidx
    Filter Index.
uint32_t anmf
    Accepted Non-matching Frame.
```

## 2.31 MRT: Multi-Rate Timer

```
void MRT_Init(MRT_Type *base, const mrt_config_t *config)
    Ungates the MRT clock and configures the peripheral for basic operation.
```

---

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

---

### Parameters

- *base* – Multi-Rate timer peripheral base address
- *config* – Pointer to user's MRT config structure. If MRT has MULTITASK bit field in MODCFG register, param config is useless.

```
void MRT_Deinit(MRT_Type *base)
    Gate the MRT clock.
```

### Parameters

- *base* – Multi-Rate timer peripheral base address

```
static inline void MRT_GetDefaultConfig(mrt_config_t *config)
    Fill in the MRT config struct with the default settings.
    The default values are:
```

```
config->enableMultiTask = false;
```

### Parameters

- `config` – Pointer to user's MRT config structure.

```
static inline void MRT_SetupChannelMode(MRT_Type *base, mrt_chnl_t channel, const  
                                       mrt_timer_mode_t mode)
```

Sets up an MRT channel mode.

#### Parameters

- `base` – Multi-Rate timer peripheral base address
- `channel` – Channel that is being configured.
- `mode` – Timer mode to use for the channel.

```
static inline void MRT_EnableInterrupts(MRT_Type *base, mrt_chnl_t channel, uint32_t mask)
```

Enables the MRT interrupt.

#### Parameters

- `base` – Multi-Rate timer peripheral base address
- `channel` – Timer channel number
- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `mrt_interrupt_enable_t`

```
static inline void MRT_DisableInterrupts(MRT_Type *base, mrt_chnl_t channel, uint32_t mask)
```

Disables the selected MRT interrupt.

#### Parameters

- `base` – Multi-Rate timer peripheral base address
- `channel` – Timer channel number
- `mask` – The interrupts to disable. This is a logical OR of members of the enumeration `mrt_interrupt_enable_t`

```
static inline uint32_t MRT_GetEnabledInterrupts(MRT_Type *base, mrt_chnl_t channel)
```

Gets the enabled MRT interrupts.

#### Parameters

- `base` – Multi-Rate timer peripheral base address
- `channel` – Timer channel number

#### Returns

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

```
static inline uint32_t MRT_GetStatusFlags(MRT_Type *base, mrt_chnl_t channel)
```

Gets the MRT status flags.

#### Parameters

- `base` – Multi-Rate timer peripheral base address
- `channel` – Timer channel number

#### Returns

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

```
static inline void MRT_ClearStatusFlags(MRT_Type *base, mrt_chnl_t channel, uint32_t mask)
```

Clears the MRT status flags.

#### Parameters

- `base` – Multi-Rate timer peripheral base address

- `channel` – Timer channel number
- `mask` – The status flags to clear. This is a logical OR of members of the enumeration `mrt_status_flags_t`

```
void MRT_UpdateTimerPeriod(MRT_Type *base, mrt_chnl_t channel, uint32_t count, bool  
                           immediateLoad)
```

Used to update the timer period in units of count.

The new value will be immediately loaded or will be loaded at the end of the current time interval. For one-shot interrupt mode the new value will be immediately loaded.

---

**Note:** User can call the utility macros provided in `fsl_common.h` to convert to ticks

---

#### Parameters

- `base` – Multi-Rate timer peripheral base address
- `channel` – Timer channel number
- `count` – Timer period in units of ticks
- `immediateLoad` – `true`: Load the new value immediately into the TIMER register; `false`: Load the new value at the end of current timer interval

```
static inline uint32_t MRT_GetCurrentTimerCount(MRT_Type *base, mrt_chnl_t channel)
```

Reads the current timer counting value.

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

---

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

---

#### Parameters

- `base` – Multi-Rate timer peripheral base address
- `channel` – Timer channel number

#### Returns

Current timer counting value in ticks

```
static inline void MRT_StartTimer(MRT_Type *base, mrt_chnl_t channel, uint32_t count)
```

Starts the timer counting.

After calling this function, timers load period value, counts down to 0 and depending on the timer mode it will either load the respective start value again or stop.

---

**Note:** User can call the utility macros provided in `fsl_common.h` to convert to ticks

---

#### Parameters

- `base` – Multi-Rate timer peripheral base address
- `channel` – Timer channel number.
- `count` – Timer period in units of ticks. Count can contain the LOAD bit, which control the force load feature.

```
static inline void MRT_StopTimer(MRT_Type *base, mrt_chnl_t channel)
```

Stops the timer counting.

This function stops the timer from counting.

#### Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number.

```
static inline uint32_t MRT_GetIdleChannel(MRT_Type *base)
```

Find the available channel.

This function returns the lowest available channel number.

#### Parameters

- base – Multi-Rate timer peripheral base address

```
static inline void MRT_ReleaseChannel(MRT_Type *base, mrt_chnl_t channel)
```

Release the channel when the timer is using the multi-task mode.

In multi-task mode, the INUSE flags allow more control over when MRT channels are released for further use. The user can hold on to a channel acquired by calling MRT\_GetIdleChannel() for as long as it is needed and release it by calling this function. This removes the need to ask for an available channel for every use.

#### Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number.

```
FSL_MRT_DRIVER_VERSION
```

```
enum _mrt_chnl
```

List of MRT channels.

*Values:*

```
enumerator kMRT_Channel_0
```

MRT channel number 0

```
enumerator kMRT_Channel_1
```

MRT channel number 1

```
enumerator kMRT_Channel_2
```

MRT channel number 2

```
enumerator kMRT_Channel_3
```

MRT channel number 3

```
enum _mrt_timer_mode
```

List of MRT timer modes.

*Values:*

```
enumerator kMRT_RepeatMode
```

Repeat Interrupt mode

```
enumerator kMRT_OneShotMode
```

One-shot Interrupt mode

```
enumerator kMRT_OneShotStallMode
```

One-shot stall mode

enum `_mrt_interrupt_enable`

List of MRT interrupts.

*Values:*

enumerator `kMRT_TimerInterruptEnable`

Timer interrupt enable

enum `_mrt_status_flags`

List of MRT status flags.

*Values:*

enumerator `kMRT_TimerInterruptFlag`

Timer interrupt flag

enumerator `kMRT_TimerRunFlag`

Indicates state of the timer

typedef enum `_mrt_chnl` `mrt_chnl_t`

List of MRT channels.

typedef enum `_mrt_timer_mode` `mrt_timer_mode_t`

List of MRT timer modes.

typedef enum `_mrt_interrupt_enable` `mrt_interrupt_enable_t`

List of MRT interrupts.

typedef enum `_mrt_status_flags` `mrt_status_flags_t`

List of MRT status flags.

typedef struct `_mrt_config` `mrt_config_t`

MRT configuration structure.

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

The config struct can be made const so it resides in flash

struct `_mrt_config`

`#include <fsl_mrt.h>` MRT configuration structure.

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

The config struct can be made const so it resides in flash

### Public Members

bool `enableMultiTask`

true: Timers run in multi-task mode; false: Timers run in hardware status mode

## 2.32 OTP: One-Time Programmable memory and API



FSL\_OTP\_DRIVER\_VERSION

OTP driver version 2.0.1.

Current version: 2.0.1

Change log:

- Version 2.0.1
  - Fixed MISRA-C 2012 violations.
- Version 2.0.0
  - Initial version.

enum \_otp\_bank

Bank bit flags.

*Values:*

enumerator kOTP\_Bank0

Bank 0.

enumerator kOTP\_Bank1

Bank 1.

enumerator kOTP\_Bank2

Bank 2.

enumerator kOTP\_Bank3

Bank 3.

enum \_otp\_word

Bank word bit flags.

*Values:*

enumerator kOTP\_Word0

Word 0.

enumerator kOTP\_Word1

Word 1.

enumerator kOTP\_Word2

Word 2.

enumerator kOTP\_Word3

Word 3.

enum \_otp\_lock

Lock modifications of a read or write access to a bank register.

*Values:*

enumerator kOTP\_LockDontLock

Do not lock.

enumerator kOTP\_LockLock

Lock till reset.

enum \_otp\_status

OTP error codes.

*Values:*

enumerator `kStatus_OTP_WrEnableInvalid`

Write enable invalid.

enumerator `kStatus_OTP_SomeBitsAlreadyProgrammed`

Some bits already programmed.

enumerator `kStatus_OTP_AllDataOrMaskZero`

All data or mask zero.

enumerator `kStatus_OTP_WriteAccessLocked`

Write access locked.

enumerator `kStatus_OTP_ReadDataMismatch`

Read data mismatch.

enumerator `kStatus_OTP_UsbIdEnabled`

USB ID enabled.

enumerator `kStatus_OTP_EthMacEnabled`

Ethernet MAC enabled.

enumerator `kStatus_OTP_AesKeysEnabled`

AES keys enabled.

enumerator `kStatus_OTP_IllegalBank`

Illegal bank.

enumerator `kStatus_OTP_ShufflerConfigNotValid`

Shuffler config not valid.

enumerator `kStatus_OTP_ShufflerNotEnabled`

Shuffler not enabled.

enumerator `kStatus_OTP_ShufflerCanOnlyProgSingleKey`

Shuffler can only program single key.

enumerator `kStatus_OTP_IllegalProgramData`

Illegal program data.

enumerator `kStatus_OTP_ReadAccessLocked`

Read access locked.

typedef enum *\_otp\_bank* otp\_bank\_t

Bank bit flags.

typedef enum *\_otp\_word* otp\_word\_t

Bank word bit flags.

typedef enum *\_otp\_lock* otp\_lock\_t

Lock modifications of a read or write access to a bank register.

static inline *status\_t* OTP\_Init(void)

Initializes OTP controller.

#### Returns

`kStatus_Success` upon successful execution, error status otherwise.

static inline *status\_t* OTP\_EnableBankWriteMask(*otp\_bank\_t* bankMask)

Unlock one or more OTP banks for write access.

#### Parameters

- `bankMask` – bit flag that specifies which banks to unlock.

**Returns**

kStatus\_Success upon successful execution, error status otherwise.

```
static inline status_t OTP__DisableBankWriteMask(otp_bank_t bankMask)
```

Lock one or more OTP banks for write access.

**Parameters**

- bankMask – bit flag that specifies which banks to lock.

**Returns**

kStatus\_Success upon successful execution, error status otherwise.

```
static inline status_t OTP__EnableBankWriteLock(uint32_t bankIndex, otp_word_t regEnableMask, otp_word_t regDisableMask, otp_lock_t lockWrite)
```

Locks or unlocks write access to a register of an OTP bank and possibly lock un/locking of it.

**Parameters**

- bankIndex – OTP bank index, 0 = bank 0, 1 = bank 1 etc.
- regEnableMask – bit flag that specifies for which words to enable writing.
- regDisableMask – bit flag that specifies for which words to disable writing.
- lockWrite – specifies if access set can be modified or is locked till reset.

**Returns**

kStatus\_Success upon successful execution, error status otherwise.

```
static inline status_t OTP__EnableBankReadLock(uint32_t bankIndex, otp_word_t regEnableMask, otp_word_t regDisableMask, otp_lock_t lockWrite)
```

Locks or unlocks read access to a register of an OTP bank and possibly lock un/locking of it.

**Parameters**

- bankIndex – OTP bank index, 0 = bank 0, 1 = bank 1 etc.
- regEnableMask – bit flag that specifies for which words to enable reading.
- regDisableMask – bit flag that specifies for which words to disable reading.
- lockWrite – specifies if access set can be modified or is locked till reset.

**Returns**

kStatus\_Success upon successful execution, error status otherwise.

```
static inline status_t OTP__ProgramRegister(uint32_t bankIndex, uint32_t regIndex, uint32_t value)
```

Program a single register in an OTP bank.

**Parameters**

- bankIndex – OTP bank index, 0 = bank 0, 1 = bank 1 etc.
- regIndex – OTP register index.
- value – value to write.

**Returns**

kStatus\_Success upon successful execution, error status otherwise.

```
static inline uint32_t OTP__GetDriverVersion(void)
```

Returns the version of the OTP driver in ROM.

**Returns**

version.

FSL\_COMPONENT\_ID

\_OTP\_ERR\_BASE

\_OTP\_MAKE\_STATUS(errorCode)

## 2.33 PINT: Pin Interrupt and Pattern Match Driver

FSL\_PINT\_DRIVER\_VERSION

enum \_pint\_pin\_enable

PINT Pin Interrupt enable type.

*Values:*

enumerator kPINT\_PinIntEnableNone

Do not generate Pin Interrupt

enumerator kPINT\_PinIntEnableRiseEdge

Generate Pin Interrupt on rising edge

enumerator kPINT\_PinIntEnableFallEdge

Generate Pin Interrupt on falling edge

enumerator kPINT\_PinIntEnableBothEdges

Generate Pin Interrupt on both edges

enumerator kPINT\_PinIntEnableLowLevel

Generate Pin Interrupt on low level

enumerator kPINT\_PinIntEnableHighLevel

Generate Pin Interrupt on high level

enum \_pint\_int

PINT Pin Interrupt type.

*Values:*

enumerator kPINT\_PinInt0

Pin Interrupt 0

enum \_pint\_pmatch\_input\_src

PINT Pattern Match bit slice input source type.

*Values:*

enumerator kPINT\_PatternMatchInp0Src

Input source 0

enumerator kPINT\_PatternMatchInp1Src

Input source 1

enumerator kPINT\_PatternMatchInp2Src

Input source 2

enumerator kPINT\_PatternMatchInp3Src

Input source 3

enumerator kPINT\_PatternMatchInp4Src

Input source 4

enumerator kPINT\_PatternMatchInp5Src  
Input source 5

enumerator kPINT\_PatternMatchInp6Src  
Input source 6

enumerator kPINT\_PatternMatchInp7Src  
Input source 7

enumerator kPINT\_SecPatternMatchInp0Src  
Input source 0

enumerator kPINT\_SecPatternMatchInp1Src  
Input source 1

enum \_pint\_pmatch\_bslice  
PINT Pattern Match bit slice type.  
*Values:*  
enumerator kPINT\_PatternMatchBSlice0  
Bit slice 0

enum \_pint\_pmatch\_bslice\_cfg  
PINT Pattern Match configuration type.  
*Values:*  
enumerator kPINT\_PatternMatchAlways  
Always Contributes to product term match

enumerator kPINT\_PatternMatchStickyRise  
Sticky Rising edge

enumerator kPINT\_PatternMatchStickyFall  
Sticky Falling edge

enumerator kPINT\_PatternMatchStickyBothEdges  
Sticky Rising or Falling edge

enumerator kPINT\_PatternMatchHigh  
High level

enumerator kPINT\_PatternMatchLow  
Low level

enumerator kPINT\_PatternMatchNever  
Never contributes to product term match

enumerator kPINT\_PatternMatchBothEdges  
Either rising or falling edge

typedef enum \_pint\_pin\_enable pint\_pin\_enable\_t  
PINT Pin Interrupt enable type.

typedef enum \_pint\_int pint\_pin\_int\_t  
PINT Pin Interrupt type.

typedef enum \_pint\_pmatch\_input\_src pint\_pmatch\_input\_src\_t  
PINT Pattern Match bit slice input source type.

typedef enum \_pint\_pmatch\_bslice pint\_pmatch\_bslice\_t  
PINT Pattern Match bit slice type.

```
typedef enum _pint_pmatch_bslice_cfg pint_pmatch_bslice_cfg_t
```

PINT Pattern Match configuration type.

```
typedef struct _pint_status pint_status_t
```

PINT event status.

```
typedef void (*pint_cb_t)(pint_pin_int_t pintr, pint_status_t *status)
```

PINT Callback function.

```
typedef struct _pint_pmatch_cfg pint_pmatch_cfg_t
```

```
void PINT_Init(PINT_Type *base)
```

Initialize PINT peripheral.

This function initializes the PINT peripheral and enables the clock.

#### Parameters

- base – Base address of the PINT peripheral.

#### Return values

None. –

```
void PINT_SetCallback(PINT_Type *base, pint_cb_t callback)
```

Set PINT callback.

This function set the callback for PINT interrupt handler.

#### Parameters

- base – Base address of the PINT peripheral.
- callback – Callback.

#### Return values

None. –

```
void PINT_PinInterruptConfig(PINT_Type *base, pint_pin_int_t intr, pint_pin_enable_t enable)
```

Configure PINT peripheral pin interrupt.

This function configures a given pin interrupt.

#### Parameters

- base – Base address of the PINT peripheral.
- intr – Pin interrupt.
- enable – Selects detection logic.

#### Return values

None. –

```
void PINT_PinInterruptGetConfig(PINT_Type *base, pint_pin_int_t pintr, pint_pin_enable_t *enable)
```

Get PINT peripheral pin interrupt configuration.

This function returns the configuration of a given pin interrupt.

#### Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.
- enable – Pointer to store the detection logic.

#### Return values

None. –

```
void PINT_PinInterruptClrStatus(PINT_Type *base, pint_pin_int_t pintr)
```

Clear Selected pin interrupt status only when the pin was triggered by edge-sensitive.

This function clears the selected pin interrupt status.

**Parameters**

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

**Return values**

None. –

```
static inline uint32_t PINT_PinInterruptGetStatus(PINT_Type *base, pint_pin_int_t pintr)
```

Get Selected pin interrupt status.

This function returns the selected pin interrupt status.

**Parameters**

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

**Return values**

status – = 0 No pin interrupt request. = 1 Selected Pin interrupt request active.

```
void PINT_PinInterruptClrStatusAll(PINT_Type *base)
```

Clear all pin interrupts status only when pins were triggered by edge-sensitive.

This function clears the status of all pin interrupts.

**Parameters**

- base – Base address of the PINT peripheral.

**Return values**

None. –

```
static inline uint32_t PINT_PinInterruptGetStatusAll(PINT_Type *base)
```

Get all pin interrupts status.

This function returns the status of all pin interrupts.

**Parameters**

- base – Base address of the PINT peripheral.

**Return values**

status – Each bit position indicates the status of corresponding pin interrupt.  
= 0 No pin interrupt request. = 1 Pin interrupt request active.

```
static inline void PINT_PinInterruptClrFallFlag(PINT_Type *base, pint_pin_int_t pintr)
```

Clear Selected pin interrupt fall flag.

This function clears the selected pin interrupt fall flag.

**Parameters**

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

**Return values**

None. –

```
static inline uint32_t PINT_PinInterruptGetFallFlag(PINT_Type *base, pint_pin_int_t pintr)
```

Get selected pin interrupt fall flag.

This function returns the selected pin interrupt fall flag.

**Parameters**

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

**Return values**

flag – = 0 Falling edge has not been detected. = 1 Falling edge has been detected.

```
static inline void PINT_PinInterruptClrFallFlagAll(PINT_Type *base)
```

Clear all pin interrupt fall flags.

This function clears the fall flag for all pin interrupts.

**Parameters**

- base – Base address of the PINT peripheral.

**Return values**

None. –

```
static inline uint32_t PINT_PinInterruptGetFallFlagAll(PINT_Type *base)
```

Get all pin interrupt fall flags.

This function returns the fall flag of all pin interrupts.

**Parameters**

- base – Base address of the PINT peripheral.

**Return values**

flags – Each bit position indicates the falling edge detection of the corresponding pin interrupt. 0 Falling edge has not been detected. = 1 Falling edge has been detected.

```
static inline void PINT_PinInterruptClrRiseFlag(PINT_Type *base, pint_pin_int_t pintr)
```

Clear Selected pin interrupt rise flag.

This function clears the selected pin interrupt rise flag.

**Parameters**

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

**Return values**

None. –

```
static inline uint32_t PINT_PinInterruptGetRiseFlag(PINT_Type *base, pint_pin_int_t pintr)
```

Get selected pin interrupt rise flag.

This function returns the selected pin interrupt rise flag.

**Parameters**

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

**Return values**

flag – = 0 Rising edge has not been detected. = 1 Rising edge has been detected.

```
static inline void PINT_PinInterruptClrRiseFlagAll(PINT_Type *base)
```

Clear all pin interrupt rise flags.

This function clears the rise flag for all pin interrupts.

**Parameters**

- base – Base address of the PINT peripheral.



**Return values**

None. –

```
static inline uint32_t PINT_PinInterruptGetRiseFlagAll(PINT_Type *base)
```

Get all pin interrupt rise flags.

This function returns the rise flag of all pin interrupts.

**Parameters**

- *base* – Base address of the PINT peripheral.

**Return values**

*flags* – Each bit position indicates the rising edge detection of the corresponding pin interrupt. 0 Rising edge has not been detected. = 1 Rising edge has been detected.

```
void PINT_PatternMatchConfig(PINT_Type *base, pint_pmatch_bslice_t bslice, pint_pmatch_cfg_t *cfg)
```

Configure PINT pattern match.

This function configures a given pattern match bit slice.

**Parameters**

- *base* – Base address of the PINT peripheral.
- *bslice* – Pattern match bit slice number.
- *cfg* – Pointer to bit slice configuration.

**Return values**

None. –

```
void PINT_PatternMatchGetConfig(PINT_Type *base, pint_pmatch_bslice_t bslice, pint_pmatch_cfg_t *cfg)
```

Get PINT pattern match configuration.

This function returns the configuration of a given pattern match bit slice.

**Parameters**

- *base* – Base address of the PINT peripheral.
- *bslice* – Pattern match bit slice number.
- *cfg* – Pointer to bit slice configuration.

**Return values**

None. –

```
static inline uint32_t PINT_PatternMatchGetStatus(PINT_Type *base, pint_pmatch_bslice_t bslice)
```

Get pattern match bit slice status.

This function returns the status of selected bit slice.

**Parameters**

- *base* – Base address of the PINT peripheral.
- *bslice* – Pattern match bit slice number.

**Return values**

*status* – = 0 Match has not been detected. = 1 Match has been detected.

```
static inline uint32_t PINT_PatternMatchGetStatusAll(PINT_Type *base)
```

Get status of all pattern match bit slices.

This function returns the status of all bit slices.

**Parameters**

- base – Base address of the PINT peripheral.

**Return values**

status – Each bit position indicates the match status of corresponding bit slice.  
= 0 Match has not been detected. = 1 Match has been detected.

uint32\_t PINT\_PatternMatchResetDetectLogic(PINT\_Type \*base)

Reset pattern match detection logic.

This function resets the pattern match detection logic if any of the product term is matching.

**Parameters**

- base – Base address of the PINT peripheral.

**Return values**

pmstatus – Each bit position indicates the match status of corresponding bit slice. = 0 Match was detected. = 1 Match was not detected.

static inline void PINT\_PatternMatchEnable(PINT\_Type \*base)

Enable pattern match function.

This function enables the pattern match function.

**Parameters**

- base – Base address of the PINT peripheral.

**Return values**

None. –

static inline void PINT\_PatternMatchDisable(PINT\_Type \*base)

Disable pattern match function.

This function disables the pattern match function.

**Parameters**

- base – Base address of the PINT peripheral.

**Return values**

None. –

static inline void PINT\_PatternMatchEnableRXEV(PINT\_Type \*base)

Enable RXEV output.

This function enables the pattern match RXEV output.

**Parameters**

- base – Base address of the PINT peripheral.

**Return values**

None. –

static inline void PINT\_PatternMatchDisableRXEV(PINT\_Type \*base)

Disable RXEV output.

This function disables the pattern match RXEV output.

**Parameters**

- base – Base address of the PINT peripheral.

**Return values**

None. –

`void PINT_EnableCallback(PINT_Type *base)`

Enable callback.

This function enables the interrupt for the selected PINT peripheral. Although the pin(s) are monitored as soon as they are enabled, the callback function is not enabled until this function is called.

**Parameters**

- `base` – Base address of the PINT peripheral.

**Return values**

None. –

`void PINT_DisableCallback(PINT_Type *base)`

Disable callback.

This function disables the interrupt for the selected PINT peripheral. Although the pins are still being monitored but the callback function is not called.

**Parameters**

- `base` – Base address of the peripheral.

**Return values**

None. –

`void PINT_Deinit(PINT_Type *base)`

Deinitialize PINT peripheral.

This function disables the PINT clock.

**Parameters**

- `base` – Base address of the PINT peripheral.

**Return values**

None. –

`void PINT_EnableCallbackByIndex(PINT_Type *base, pint_pin_int_t pintIdx)`

enable callback by pin index.

This function enables callback by pin index instead of enabling all pins.

**Parameters**

- `base` – Base address of the peripheral.
- `pintIdx` – pin index.

**Return values**

None. –

`void PINT_DisableCallbackByIndex(PINT_Type *base, pint_pin_int_t pintIdx)`

disable callback by pin index.

This function disables callback by pin index instead of disabling all pins.

**Parameters**

- `base` – Base address of the peripheral.
- `pintIdx` – pin index.

**Return values**

None. –

`PINT_USE_LEGACY_CALLBACK`

`PININT_BITSLICE_SRC_START`

```
PININT_BITSLICE_SRC_MASK
PININT_BITSLICE_CFG_START
PININT_BITSLICE_CFG_MASK
PININT_BITSLICE_ENDP_MASK
PINT_PIN_INT_LEVEL
PINT_PIN_INT_EDGE
PINT_PIN_INT_FALL_OR_HIGH_LEVEL
PINT_PIN_INT_RISE
PINT_PIN_RISE_EDGE
PINT_PIN_FALL_EDGE
PINT_PIN_BOTH_EDGE
PINT_PIN_LOW_LEVEL
PINT_PIN_HIGH_LEVEL

struct _pint_status
    #include <fsl_pint.h> PINT event status.

struct _pint_pmatch_cfg
    #include <fsl_pint.h>
```

## 2.34 Power Driver

```
enum pd_bits
    Values:

    enumerator kPDRUNCFG_LP_REG
    enumerator kPDRUNCFG_PD_FRO_EN
    enumerator kPDRUNCFG_PD_TS
    enumerator kPDRUNCFG_PD_BOD_RESET
    enumerator kPDRUNCFG_PD_BOD_INTR
    enumerator kPDRUNCFG_PD_VD2_ANA
    enumerator kPDRUNCFG_PD_ADC0
    enumerator kPDRUNCFG_PD_RAM0
    enumerator kPDRUNCFG_PD_RAM1
    enumerator kPDRUNCFG_PD_RAM2
    enumerator kPDRUNCFG_PD_RAM3
    enumerator kPDRUNCFG_PD_ROM
    enumerator kPDRUNCFG_PD_VDDA
```

enumerator kPDRUNCFG\_PD\_WDT\_OSC  
enumerator kPDRUNCFG\_PD\_USB0\_PHY  
enumerator kPDRUNCFG\_PD\_SYS\_PLL0  
enumerator kPDRUNCFG\_PD\_VREFP  
enumerator kPDRUNCFG\_PD\_FLASH\_BG  
enumerator kPDRUNCFG\_PD\_VD3  
enumerator kPDRUNCFG\_PD\_VD4  
enumerator kPDRUNCFG\_PD\_VD5  
enumerator kPDRUNCFG\_PD\_VD6  
enumerator kPDRUNCFG\_REQ\_DELAY  
enumerator kPDRUNCFG\_FORCE\_RBB  
enumerator kPDRUNCFG\_PD\_USB1\_PHY  
enumerator kPDRUNCFG\_PD\_USB\_PLL  
enumerator kPDRUNCFG\_PD\_AUDIO\_PLL  
enumerator kPDRUNCFG\_PD\_SYS\_OSC  
enumerator kPDRUNCFG\_PD\_EEPROM  
enumerator kPDRUNCFG\_PD\_rng  
enumerator kPDRUNCFG\_ForceUnsigned

enum \_\_power\_mode\_config

*Values:*

enumerator kPmu\_Sleep  
enumerator kPmu\_Deep\_Sleep  
enumerator kPmu\_Deep\_PowerDown

enum \_\_power\_bod\_status

The enumeration of BOD status flags.

*Values:*

enumerator kBod\_ResetStatusFlag  
BOD reset has occurred.  
enumerator kBod\_InterruptStatusFlag  
BOD interrupt has occurred

enum \_\_power\_bod\_reset\_level

The enumeration of BOD reset level.

*Values:*

enumerator kBod\_ResetLevel0  
Reset Level0: 1.62V.  
enumerator kBod\_ResetLevel1  
Reset Level0: 1.68V.

enumerator kBod\_ResetLevel2

Reset Level0: 2.21V.

enumerator kBod\_ResetLevel3

Reset Level0: 2.85V.

enum \_\_power\_bod\_interrupt\_level

The enumeration of BOD interrupt level.

*Values:*

enumerator kBod\_InterruptLevel0

Interrupt level: 1.63V.

enumerator kBod\_InterruptLevel1

Interrupt level: 1.68V.

enumerator kBod\_InterruptLevel2

Interrupt level: 1.95V.

enumerator kBod\_InterruptLevel3

Interrupt level: 2.86V.

typedef enum *pd\_bits* pd\_bit\_t

typedef enum *\_power\_mode\_config* power\_mode\_cfg\_t

typedef enum *\_power\_bod\_status* power\_bod\_status\_t

The enumeration of BOD status flags.

typedef enum *\_power\_bod\_reset\_level* power\_bod\_reset\_level\_t

The enumeration of BOD reset level.

typedef enum *\_power\_bod\_interrupt\_level* power\_bod\_interrupt\_level\_t

The enumeration of BOD interrupt level.

typedef struct *\_power\_bod\_config* power\_bod\_config\_t

The configuration of power bod, including reset level, interrupt level, and so on.

FSL\_POWER\_DRIVER\_VERSION

power driver version 2.2.0.

MAKE\_PD\_BITS(reg, slot)

PDRCFG0

PDRCFG1

static inline void POWER\_EnablePD(*pd\_bit\_t* en)

API to enable PDRUNCFG bit in the Syscon. Note that enabling the bit powers down the peripheral.

#### Parameters

- en – peripheral for which to enable the PDRUNCFG bit

#### Returns

none

static inline void POWER\_DisablePD(*pd\_bit\_t* en)

API to disable PDRUNCFG bit in the Syscon. Note that disabling the bit powers up the peripheral.

#### Parameters

- en – peripheral for which to disable the PDRUNCFG bit

**Returns**

none

static inline void POWER\_EnableDeepSleep(void)

API to enable deep sleep bit in the ARM Core.

**Returns**

none

static inline void POWER\_DisableDeepSleep(void)

API to disable deep sleep bit in the ARM Core.

**Returns**

none

void POWER\_OtpReload(void)

Power Library API to reload OTP. This API must be called if VD6 is power down and power back again since FROHF TRIM value is store in OTP. If not, when calling FROHF settng API in clock driver then the FROHF clock out put will be inaccurate.

**Returns**

none

void POWER\_SetPLL(void)

Power Library API to power the PLLs.

**Returns**

none

void POWER\_SetUsbPhy(void)

Power Library API to power the USB PHY.

**Returns**

none

void POWER\_EnterPowerMode(*power\_mode\_cfg\_t* mode, uint64\_t exclude\_from\_pd)

Power Library API to enter different power modes.

**Parameters**

- mode – power mode
- exclude\_from\_pd – Bit mask of the PDRUNCFG0(low 32bits) and PDRUNCFG1(high 32bits) that needs to be powered on during power mode selected.

**Returns**

none

void POWER\_EnterSleep(void)

Power Library API to enter sleep mode.

**Returns**

none

void POWER\_EnterDeepSleep(uint64\_t exclude\_from\_pd)

Power Library API to enter deep sleep mode.

**Parameters**

- exclude\_from\_pd – Bit mask of the PDRUNCFG0(low 32bits) and PDRUNCFG1(high 32bits) bits that needs to be powered on during deep sleep

**Returns**

none

void POWER\_EnterDeepPowerDown(uint64\_t exclude\_from\_pd)

Power Library API to enter deep power down mode.

**Parameters**

- `exclude_from_pd` – Bit mask of the PDRUNCFG0(low 32bits) and PDRUNCFG1(high 32bits) that needs to be powered on during deep power down mode, but this is has no effect as the voltages are cut off.

**Returns**

none

void POWER\_SetVoltageForFreq(uint32\_t freq)

Power Library API to choose normal regulation and set the voltage for the desired operating frequency.

**Parameters**

- `freq` – - The desired frequency at which the part would like to operate, note that the voltage and flash wait states should be set before changing frequency

**Returns**

none

uint32\_t POWER\_GetLibVersion(void)

Power Library API to return the library version.

**Returns**

version number of the power library

void POWER\_InitBod(const *power\_bod\_config\_t* \*bodConfig)

Initialize BOD, including enabling/disabling BOD interrupt, enabling/disabling BOD reset, setting BOD interrupt level, and reset level.

**Parameters**

- `bodConfig` – Pointer the the structure *power\_bod\_config\_t*.

void POWER\_GetDefaultBodConfig(*power\_bod\_config\_t* \*bodConfig)

Get default BOD configuration.

```
bodConfig->enableReset = true;
bodConfig->resetLevel = kBod_ResetLevel0;
bodConfig->enableInterrupt = false;
bodConfig->interruptLevel = kBod_InterruptLevel0;
```

**Parameters**

- `bodConfig` – Pointer the the structure *power\_bod\_config\_t*.

static inline void POWER\_DeinitBod(void)

De-initialize BOD.

static inline uint32\_t POWER\_GetBodStatusFlags(void)

Get Bod status flags.

**Returns**

uint32\_t

static inline void POWER\_ClearBodStatusFlags(uint32\_t mask)

Clear Bod status flags.

**Parameters**

- `mask` – The mask of status flags to clear, should be the OR'ed value of *power\_bod\_status\_t*.



bool enableReset

Enable/disable BOD reset function.

*power\_bod\_reset\_level\_t* resetLevel

BOD reset level, please refer to *power\_bod\_reset\_level\_t*.

bool enableInterrupt

Enable/disable BOD interrupt function.

*power\_bod\_interrupt\_level\_t* interruptLevel

BOD interrupt level, please refer to *power\_bod\_interrupt\_level\_t*.

struct \_\_power\_bod\_config

*#include <fsl\_power.h>* The configuration of power bod, including reset level, interrupt level, and so on.

## 2.35 PUF: Physical Unclonable Function

FSL\_PUF\_DRIVER\_VERSION

PUF driver version. Version 2.2.0.

Current version: 2.2.0

Change log:

- 2.0.0
  - Initial version.
- 2.0.1
  - Fixed puf\_wait\_usec function optimization issue.
- 2.0.2
  - Add PUF configuration structure and support for PUF SRAM controller. Remove magic constants.
- 2.0.3
  - Fix MISRA C-2012 issue.
- 2.1.0
  - Align driver with PUF SRAM controller registers on LPCXpresso55s16.
  - Update initialization logic .
- 2.1.1
  - Fix ARMGCC build warning .
- 2.1.2
  - Update: Add automatic big to little endian swap for user (pre-shared) keys destined to secret hardware bus (PUF key index 0).
- 2.1.3
  - Fix MISRA C-2012 issue.
- 2.1.4
  - Replace register uint32\_t ticksCount with volatile uint32\_t ticksCount in puf\_wait\_usec() to prevent optimization out delay loop.
- 2.1.5

- Use common SDK delay in `puf_wait_usec()`
- 2.1.6
  - Changed wait time in `PUF_Init()`, when initialization fails it will try `PUF_Powercycle()` with shorter time. If this shorter time will also fail, initialization will be tried with worst case time as before.
- 2.2.0
  - Add support for `kPUF_KeySlot4`.
  - Add new `PUF_ClearKey()` function, that clears a desired PUF internal HW key register.

`enum __puf_key_index_register`

*Values:*

`enumerator kPUF_KeyIndex_00`  
`enumerator kPUF_KeyIndex_01`  
`enumerator kPUF_KeyIndex_02`  
`enumerator kPUF_KeyIndex_03`  
`enumerator kPUF_KeyIndex_04`  
`enumerator kPUF_KeyIndex_05`  
`enumerator kPUF_KeyIndex_06`  
`enumerator kPUF_KeyIndex_07`  
`enumerator kPUF_KeyIndex_08`  
`enumerator kPUF_KeyIndex_09`  
`enumerator kPUF_KeyIndex_10`  
`enumerator kPUF_KeyIndex_11`  
`enumerator kPUF_KeyIndex_12`  
`enumerator kPUF_KeyIndex_13`  
`enumerator kPUF_KeyIndex_14`  
`enumerator kPUF_KeyIndex_15`

`enum __puf_min_max`

*Values:*

`enumerator kPUF_KeySizeMin`  
`enumerator kPUF_KeySizeMax`  
`enumerator kPUF_KeyIndexMax`

`enum __puf_key_slot`

PUF key slot.

*Values:*

`enumerator kPUF_KeySlot0`  
PUF key slot 0

enumerator kPUF\_KeySlot1  
PUF key slot 1

PUF status return codes.

*Values:*

enumerator kStatus\_EnrollNotAllowed

enumerator kStatus\_StartNotAllowed

typedef enum *puf\_key\_index\_register* puf\_key\_index\_register\_t

typedef enum *puf\_min\_max* puf\_min\_max\_t

typedef enum *puf\_key\_slot* puf\_key\_slot\_t  
PUF key slot.

PUF\_GET\_KEY\_CODE\_SIZE\_FOR\_KEY\_SIZE(x)

Get Key Code size in bytes from key size in bytes at compile time.

PUF\_MIN\_KEY\_CODE\_SIZE

PUF\_ACTIVATION\_CODE\_SIZE

KEYSTORE\_PUF\_DISCHARGE\_TIME\_FIRST\_TRY\_MS

KEYSTORE\_PUF\_DISCHARGE\_TIME\_MAX\_MS

struct puf\_config\_t

*#include <fsl\_puf.h>*

## 2.36 Reset Driver

enum \_SYSCON\_RSTn

Enumeration for peripheral reset control bits.

Defines the enumeration for peripheral reset control bits in PRESETCTRL/ASYNCPRESETCTRL registers

*Values:*

enumerator kSPIFI\_RST\_SHIFT\_RSTn  
SPIFI reset control

enumerator kMUX\_RST\_SHIFT\_RSTn  
Input mux reset control

enumerator kIOCON\_RST\_SHIFT\_RSTn  
IOCON reset control

enumerator kGPIO0\_RST\_SHIFT\_RSTn  
GPIO0 reset control

enumerator kGPIO1\_RST\_SHIFT\_RSTn  
GPIO1 reset control

enumerator kGPIO2\_RST\_SHIFT\_RSTn  
GPIO2 reset control

enumerator kGPIO3\_RST\_SHIFT\_RSTn  
GPIO3 reset control

enumerator kPINT\_RST\_SHIFT\_RSTn  
Pin interrupt (PINT) reset control

enumerator kGINT\_RST\_SHIFT\_RSTn  
Grouped interrupt (PINT) reset control.

enumerator kDMA\_RST\_SHIFT\_RSTn  
DMA reset control

enumerator kCRC\_RST\_SHIFT\_RSTn  
CRC reset control

enumerator kWWDT\_RST\_SHIFT\_RSTn  
Watchdog timer reset control

enumerator kADC0\_RST\_SHIFT\_RSTn  
ADC0 reset control

enumerator kMRT\_RST\_SHIFT\_RSTn  
Multi-rate timer (MRT) reset control

enumerator kSCT0\_RST\_SHIFT\_RSTn  
SCTimer/PWM 0 (SCT0) reset control

enumerator kMCAN0\_RST\_SHIFT\_RSTn  
MCAN0 reset control

enumerator kMCAN1\_RST\_SHIFT\_RSTn  
MCAN1 reset control

enumerator kUTICK\_RST\_SHIFT\_RSTn  
Micro-tick timer reset control

enumerator kFC0\_RST\_SHIFT\_RSTn  
Flexcomm Interface 0 reset control

enumerator kFC1\_RST\_SHIFT\_RSTn  
Flexcomm Interface 1 reset control

enumerator kFC2\_RST\_SHIFT\_RSTn  
Flexcomm Interface 2 reset control

enumerator kFC3\_RST\_SHIFT\_RSTn  
Flexcomm Interface 3 reset control

enumerator kFC4\_RST\_SHIFT\_RSTn  
Flexcomm Interface 4 reset control

enumerator kFC5\_RST\_SHIFT\_RSTn  
Flexcomm Interface 5 reset control

enumerator kFC6\_RST\_SHIFT\_RSTn  
Flexcomm Interface 6 reset control

enumerator kFC7\_RST\_SHIFT\_RSTn  
Flexcomm Interface 7 reset control

enumerator kDMIC\_RST\_SHIFT\_RSTn  
Digital microphone interface reset control

enumerator kCT32B2\_RST\_SHIFT\_RSTn  
CT32B2 reset control

enumerator kUSB0D\_RST\_SHIFT\_RSTn  
USB0D reset control

enumerator kCT32B0\_RST\_SHIFT\_RSTn  
CT32B0 reset control

enumerator kCT32B1\_RST\_SHIFT\_RSTn  
CT32B1 reset control

enumerator kLCD\_RST\_SHIFT\_RSTn  
LCD reset control

enumerator kSDIO\_RST\_SHIFT\_RSTn  
SDIO reset control

enumerator kUSB1H\_RST\_SHIFT\_RSTn  
USB1H reset control

enumerator kUSB1D\_RST\_SHIFT\_RSTn  
USB1D reset control

enumerator kUSB1RAM\_RST\_SHIFT\_RSTn  
USB1RAM reset control

enumerator kEMC\_RST\_SHIFT\_RSTn  
EMC reset control

enumerator kETH\_RST\_SHIFT\_RSTn  
ETH reset control

enumerator kGPIO4\_RST\_SHIFT\_RSTn  
GPIO4 reset control

enumerator kGPIO5\_RST\_SHIFT\_RSTn  
GPIO5 reset control

enumerator kAES\_RST\_SHIFT\_RSTn  
AES reset control

enumerator kOTP\_RST\_SHIFT\_RSTn  
OTP reset control

enumerator kRNG\_RST\_SHIFT\_RSTn  
RNG reset control

enumerator kFC8\_RST\_SHIFT\_RSTn  
Flexcomm Interface 8 reset control

enumerator kFC9\_RST\_SHIFT\_RSTn  
Flexcomm Interface 9 reset control

enumerator kUSB0HMR\_RST\_SHIFT\_RSTn  
USB0HMR reset control

enumerator kUSB0HSL\_RST\_SHIFT\_RSTn  
USB0HSL reset control

enumerator kSHA\_RST\_SHIFT\_RSTn  
SHA reset control

enumerator kSC0\_RST\_SHIFT\_RSTn  
SC0 reset control

enumerator kSC1\_RST\_SHIFT\_RSTn  
SC1 reset control

enumerator kFC10\_RST\_SHIFT\_RSTn  
Flexcomm Interface 10 reset control

enumerator kPUF\_RST\_SHIFT\_RSTn  
PUF reset control

enumerator kCT32B3\_RST\_SHIFT\_RSTn  
CT32B3 reset control

enumerator kCT32B4\_RST\_SHIFT\_RSTn  
CT32B4 reset control

typedef enum \_SYSCON\_RSTn SYSCON\_RSTn\_t

Enumeration for peripheral reset control bits.

Defines the enumeration for peripheral reset control bits in PRESETCTRL/ASYNCPRESETCTRL registers

typedef SYSCON\_RSTn\_t reset\_ip\_name\_t

void RESET\_SetPeripheralReset(*reset\_ip\_name\_t* peripheral)

Assert reset to peripheral.

Asserts reset signal to specified peripheral module.

#### Parameters

- *peripheral* – Assert reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

void RESET\_ClearPeripheralReset(*reset\_ip\_name\_t* peripheral)

Clear reset to peripheral.

Clears reset signal to specified peripheral module, allows it to operate.

#### Parameters

- *peripheral* – Clear reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

void RESET\_PeripheralReset(*reset\_ip\_name\_t* peripheral)

Reset peripheral module.

Reset peripheral module.

#### Parameters

- *peripheral* – Peripheral to reset. The enum argument contains encoding of reset register and reset bit position in the reset register.

static inline void RESET\_ReleasePeripheralReset(*reset\_ip\_name\_t* peripheral)

Release peripheral module.

Release peripheral module.

#### Parameters

- *peripheral* – Peripheral to release. The enum argument contains encoding of reset register and reset bit position in the reset register.

FSL\_RESET\_DRIVER\_VERSION

reset driver version 2.4.0

ADC\_RSTS

Array initializers with peripheral reset bits

AES\_RSTS

CRC\_RSTS

CTIMER\_RSTS

DMA\_RSTS\_N

DMIC\_RSTS

EMC\_RSTS

ETH\_RST

FLEXCOMM\_RSTS

GINT\_RSTS

GPIO\_RSTS\_N

INPUTMUX\_RSTS

IOCON\_RSTS

FLASH\_RSTS

LCD\_RSTS

MRT\_RSTS

MCAN\_RSTS

OTP\_RSTS

PINT\_RSTS

RNG\_RSTS

SDIO\_RST

SCT\_RSTS

SHA\_RST

SPIFI\_RSTS

USB0D\_RST

USB0HMR\_RST

USB0HSL\_RST

USB1H\_RST

USB1D\_RST

USB1RAM\_RST

UTICK\_RSTS

WWDT\_RSTS

USB1RAM\_RSTS

USB1H\_RSTS

USB1D\_RSTS

USB0HSL\_RSTS

USB0HMR\_RSTS

USB0D\_RSTS

SHA\_RSTS

SDIO\_RSTS

ETH\_RSTS

## 2.37 RIT: Repetitive Interrupt Timer

`void RIT_Init(RIT_Type *base, const rit_config_t *config)`

Ungates the RIT clock, enables the RIT module, and configures the peripheral for basic operations.

---

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

---

### Parameters

- `base` – RIT peripheral base address
- `config` – Pointer to the user's RIT config structure

`void RIT_Deinit(RIT_Type *base)`

Gates the RIT clock and disables the RIT module.

### Parameters

- `base` – RIT peripheral base address

`void RIT_GetDefaultConfig(rit_config_t *config)`

Fills in the RIT configuration structure with the default settings.

The default values are as follows.

```
config->enableRunInDebug = false;
```

### Parameters

- `config` – Pointer to the onfiguration structure.

`static inline uint32_t RIT_GetStatusFlags(RIT_Type *base)`

Gets the RIT status flags.

### Parameters

- `base` – RIT peripheral base address

### Returns

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



```
static inline void RIT_ClearStatusFlags(RIT_Type *base, uint32_t mask)
```

Clears the RIT status flags.

#### Parameters

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

```
void RIT_SetTimerCompare(RIT_Type *base, uint64_t count)
```

Sets the timer period in units of count.

This function sets the RI compare value. If the counter value equals to the compare value, it will generate an interrupt.

---

**Note:** Users can call the utility macros provided in `fsl_common.h` to convert to ticks

---

#### Parameters

- base – RIT peripheral base address
- count – Timer period in units of ticks

```
void RIT_SetMaskBit(RIT_Type *base, uint64_t count)
```

Sets the mask bit of count compare.

This function sets the RI mask value. A 1 written to any bit will force the compare to be true for the corresponding bit of the counter and compare register (causes the comparison of the register bits to be always true).

---

**Note:** Users can call the utility macros provided in `fsl_common.h` to convert to ticks

---

#### Parameters

- base – RIT peripheral base address
- count – Timer period in units of ticks

```
uint64_t RIT_GetCompareTimerCount(RIT_Type *base)
```

Reads the current value of compare register.

---

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

---

#### Parameters

- base – RIT peripheral base address

#### Returns

Current RI compare value

```
uint64_t RIT_GetCounterTimerCount(RIT_Type *base)
```

Reads the current timer counting value of counter register.

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

---

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

---

**Parameters**

- base – RIT peripheral base address

**Returns**

Current timer counting value in ticks

uint64\_t RIT\_GetMaskTimerCount(RIT\_Type \*base)

Reads the current value of mask register.

---

**Note:** Users can call the utility macros provided in fsl\_common.h to convert ticks to usec or msec

---

**Parameters**

- base – RIT peripheral base address

**Returns**

Current RI mask value

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

Starts the timer counting.

After calling this function, timers load initial value(0U), count up to desired value or overflow then the counter will count up again.

**Parameters**

- base – RIT peripheral base address

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

Stops the timer counting.

This function stop timer counting. Timer reload their new value after the next time they call the RIT\_StartTimer.

**Parameters**

- base – RIT peripheral base address

FSL\_RIT\_DRIVER\_VERSION

Version 2.1.2

enum \_rit\_status\_flags

List of RIT status flags.

*Values:*

enumerator kRIT\_TimerFlag

Timer flag

typedef enum \_rit\_status\_flags rit\_status\_flags\_t

List of RIT status flags.

typedef struct \_rit\_config rit\_config\_t

RIT config structure.

This structure holds the configuration settings for the RIT peripheral. To initialize this structure to reasonable defaults, call the RIT\_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 void RIT_ClearCounter(RIT_Type *base, bool enable)
```

Sets the Timer Counter auto clear or not.

This function set the counter auto clear or not whenever the counter value equals the masked compare value specified by the contents of COMPVAL/COMPVAL\_H and MASK/MASK\_H registers..

*Deprecated:*

Do not use this function. It has been superceded by RIT\_SetCountAutoClear.

```
static inline void RIT_SetCountAutoClear(RIT_Type *base, bool enable)
```

Sets the Timer Counter auto clear or not.

This function set the counter auto clear or not whenever the counter value equals the masked compare value specified by the contents of COMPVAL/COMPVAL\_H and MASK/MASK\_H registers..

#### Parameters

- base – RIT peripheral base address
- enable – Enable/disable Counter auto clear when value equals the compare value.
  - true: Enable Counter auto clear.
  - false: Disable Counter auto clear.

```
struct _rit_config
```

*#include <fsl\_rit.h>* RIT config structure.

This structure holds the configuration settings for the RIT peripheral. To initialize this structure to reasonable defaults, call the RIT\_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 enableRunInDebug
```

true: The timer is halted when the processor is halted for debugging.; false: Debug has no effect on the timer operation.

## 2.38 RNG: Random Number Generator

```
FSL_RNG_DRIVER_VERSION
```

RNG driver version 2.1.0.

Current version: 2.1.0

Change log:

- Version 2.0.0
  - Initial version.
- Version 2.1.0
  - Renamed function RNG\_GetRandomData() to RNG\_GetRandomDataWord(). Added function RNG\_GetRandomData() which discarding next 32 words after reading RNG register which results into better entropy, as is recommended in UM.

- API is aligned with other RNG driver, having similar functionality as other RNG/TRNG drivers.

*status\_t* RNG\_GetRandomData(void \*data, size\_t dataSize)

Gets random data.

This function gets random data from the RNG.

#### Parameters

- data – Pointer address used to store random data.
- dataSize – Size of the buffer pointed by the data parameter.

#### Returns

Status from operation

static inline uint32\_t RNG\_GetRandomDataWord(void)

Gets random data.

This function returns single 32 bit random number. For each read word next 32 words should be discarded to achieve better entropy.

#### Returns

random data

FSL\_COMPONENT\_ID

## 2.39 RTC: Real Time Clock

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

Un-gate the RTC clock and enable the RTC oscillator.

---

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

---

#### Parameters

- base – RTC peripheral base address

static inline void RTC\_Deinit(RTC\_Type \*base)

Stop the timer and gate the RTC clock.

#### Parameters

- base – RTC peripheral base address

*status\_t* RTC\_SetDatetime(RTC\_Type \*base, const *rtc\_datetime\_t* \*datetime)

Set the RTC date and time according to the given time structure.

The RTC counter must be stopped prior to calling this function as writes to the RTC seconds register will fail if the RTC counter is running.

#### Parameters

- base – RTC peripheral base address
- datetime – Pointer to structure where the date and time details to set are stored

#### Returns

kStatus\_Success: Success in setting the time and starting the RTC  
kStatus\_InvalidArgument: Error because the datetime format is incorrect

```
void RTC_GetDatetime(RTC_Type *base, rtc_datetime_t *datetime)
```

Get the RTC time and stores it in the given time structure.

#### Parameters

- base – RTC peripheral base address
- datetime – Pointer to structure where the date and time details are stored.

```
status_t RTC_SetAlarm(RTC_Type *base, const rtc_datetime_t *alarmTime)
```

Set the RTC alarm time.

The function checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

#### Parameters

- base – RTC peripheral base address
- alarmTime – Pointer to structure where the alarm time is stored.

#### Returns

kStatus\_Success: success in setting the RTC alarm  
kStatus\_InvalidArgument: Error because the alarm datetime format is incorrect  
kStatus\_Fail: Error because the alarm time has already passed

```
void RTC_GetAlarm(RTC_Type *base, rtc_datetime_t *datetime)
```

Return the RTC alarm time.

#### Parameters

- base – RTC peripheral base address
- datetime – Pointer to structure where the alarm date and time details are stored.

```
static inline void RTC_EnableWakeupTimer(RTC_Type *base, bool enable)
```

Enable the RTC wake-up timer (1KHZ).

After calling this function, the RTC driver will use/un-use the RTC wake-up (1KHZ) at the same time.

#### Parameters

- base – RTC peripheral base address
- enable – Use/Un-use the RTC wake-up timer.
  - true: Use RTC wake-up timer at the same time.
  - false: Un-use RTC wake-up timer, RTC only use the normal seconds timer by default.

```
static inline uint32_t RTC_GetEnabledWakeupTimer(RTC_Type *base)
```

Get the enabled status of the RTC wake-up timer (1KHZ).

#### Parameters

- base – RTC peripheral base address

#### Returns

The enabled status of RTC wake-up timer (1KHZ).

```
static inline void RTC_EnableWakeUpTimerInterruptFromDPD(RTC_Type *base, bool enable)
```

Enable the wake-up timer interrupt from deep power down mode.

#### Parameters

- base – RTC peripheral base address

- `enable` – Enable/Disable wake-up timer interrupt from deep power down mode.
  - `true`: Enable wake-up timer interrupt from deep power down mode.
  - `false`: Disable wake-up timer interrupt from deep power down mode.

`static inline void RTC_EnableAlarmTimerInterruptFromDPD(RTC_Type *base, bool enable)`  
Enable the alarm timer interrupt from deep power down mode.

#### Parameters

- `base` – RTC peripheral base address
- `enable` – Enable/Disable alarm timer interrupt from deep power down mode.
  - `true`: Enable alarm timer interrupt from deep power down mode.
  - `false`: Disable alarm timer interrupt from deep power down mode.

`static inline void RTC_EnableInterrupts(RTC_Type *base, uint32_t mask)`  
Enables the selected RTC interrupts.

#### *Deprecated:*

Do not use this function. It has been superseded by `RTC_EnableAlarmTimerInterruptFromDPD` and `RTC_EnableWakeUpTimerInterruptFromDPD`

#### Parameters

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

`static inline void RTC_DisableInterrupts(RTC_Type *base, uint32_t mask)`  
Disables the selected RTC interrupts.

#### *Deprecated:*

Do not use this function. It has been superseded by `RTC_EnableAlarmTimerInterruptFromDPD` and `RTC_EnableWakeUpTimerInterruptFromDPD`

#### Parameters

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

`static inline uint32_t RTC_GetEnabledInterrupts(RTC_Type *base)`  
Get the enabled RTC interrupts.

#### *Deprecated:*

Do not use this function. It will be deleted in next release version.

#### Parameters

- `base` – RTC peripheral base address

#### Returns

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

```
static inline uint32_t RTC_GetStatusFlags(RTC_Type *base)
```

Get the RTC status flags.

**Parameters**

- base – RTC peripheral base address

**Returns**

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

```
static inline void RTC_ClearStatusFlags(RTC_Type *base, uint32_t mask)
```

Clear the RTC status flags.

**Parameters**

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

```
static inline void RTC_EnableTimer(RTC_Type *base, bool enable)
```

Enable the RTC timer counter.

After calling this function, the RTC inner counter increments once a second when only using the RTC seconds timer (1hz), while the RTC inner wake-up timer countdown once a millisecond when using RTC wake-up timer (1KHZ) at the same time. RTC timer contain two timers, one is the RTC normal seconds timer, the other one is the RTC wake-up timer, the RTC enable bit is the master switch for the whole RTC timer, so user can use the RTC seconds (1HZ) timer independently, but they can't use the RTC wake-up timer (1KHZ) independently.

**Parameters**

- base – RTC peripheral base address
- enable – Enable/Disable RTC Timer counter.
  - true: Enable RTC Timer counter.
  - false: Disable RTC Timer counter.

```
static inline void RTC_StartTimer(RTC_Type *base)
```

Starts the RTC time counter.

*Deprecated:*

Do not use this function. It has been superceded by `RTC_EnableTimer`

After calling this function, the timer counter increments once a second provided `SR[TOF]` or `SR[TIF]` are not set.

**Parameters**

- base – RTC peripheral base address

```
static inline void RTC_StopTimer(RTC_Type *base)
```

Stops the RTC time counter.

*Deprecated:*

Do not use this function. It has been superceded by `RTC_EnableTimer`

RTC's seconds register can be written to only when the timer is stopped.

**Parameters**

- base – RTC peripheral base address

FSL\_RTC\_DRIVER\_VERSION

Version 2.2.0

enum `_rtc_interrupt_enable`

List of RTC interrupts.

*Values:*

enumerator `kRTC_AlarmInterruptEnable`

Alarm interrupt.

enumerator `kRTC_WakeupInterruptEnable`

Wake-up interrupt.

enum `_rtc_status_flags`

List of RTC flags.

*Values:*

enumerator `kRTC_AlarmFlag`

Alarm flag

enumerator `kRTC_WakeupFlag`

1kHz wake-up timer flag

typedef enum `_rtc_interrupt_enable` `rtc_interrupt_enable_t`

List of RTC interrupts.

typedef enum `_rtc_status_flags` `rtc_status_flags_t`

List of RTC flags.

typedef struct `_rtc_datetime` `rtc_datetime_t`

Structure is used to hold the date and time.

static inline void `RTC_SetSecondsTimerMatch(RTC_Type *base, uint32_t matchValue)`

Set the RTC seconds timer (1HZ) MATCH value.

#### Parameters

- `base` – RTC peripheral base address
- `matchValue` – The value to be set into the RTC MATCH register

static inline uint32\_t `RTC_GetSecondsTimerMatch(RTC_Type *base)`

Read actual RTC seconds timer (1HZ) MATCH value.

#### Parameters

- `base` – RTC peripheral base address

#### Returns

The actual RTC seconds timer (1HZ) MATCH value.

static inline void `RTC_SetSecondsTimerCount(RTC_Type *base, uint32_t countValue)`

Set the RTC seconds timer (1HZ) COUNT value.

#### Parameters

- `base` – RTC peripheral base address
- `countValue` – The value to be loaded into the RTC COUNT register

static inline uint32\_t `RTC_GetSecondsTimerCount(RTC_Type *base)`

Read the actual RTC seconds timer (1HZ) COUNT value.

#### Parameters

- `base` – RTC peripheral base address



**Returns**

The actual RTC seconds timer (1HZ) COUNT value.

static inline void RTC\_SetWakeupCount(RTC\_Type \*base, uint16\_t wakeupValue)

Enable the RTC wake-up timer (1KHZ) and set countdown value to the RTC WAKE register.

**Parameters**

- base – RTC peripheral base address
- wakeupValue – The value to be loaded into the WAKE register in RTC wake-up timer (1KHZ).

static inline uint16\_t RTC\_GetWakeupCount(RTC\_Type \*base)

Read the actual value from the WAKE register value in RTC wake-up timer (1KHZ)

Read the WAKE register twice and compare the result, if the value match, the time can be used.

**Parameters**

- base – RTC peripheral base address

**Returns**

The actual value of the WAKE register value in RTC wake-up timer (1KHZ).

static inline void RTC\_Reset(RTC\_Type \*base)

Perform a software reset on the RTC module.

This resets all RTC registers to their reset value. The bit is cleared by software explicitly clearing it.

**Parameters**

- base – RTC peripheral base address

struct \_rtc\_datetime

*#include <fsl\_rtc.h>* Structure is used to hold the date and time.

**Public Members**

uint16\_t year

Range from 1970 to 2099.

uint8\_t month

Range from 1 to 12.

uint8\_t day

Range from 1 to 31 (depending on month).

uint8\_t hour

Range from 0 to 23.

uint8\_t minute

Range from 0 to 59.

uint8\_t second

Range from 0 to 59.

## 2.40 SCTimer: SCTimer/PWM (SCT)

*status\_t* SCTIMER\_Init(SCT\_Type \*base, const *sctimer\_config\_t* \*config)

Ungates the SCTimer clock and configures the peripheral for basic operation.

---

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

---

#### Parameters

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

#### Returns

kStatus\_Success indicates success; Else indicates failure.

void SCTIMER\_Deinit(SCT\_Type \*base)

Gates the SCTimer clock.

#### Parameters

- base – SCTimer peripheral base address

void SCTIMER\_GetDefaultConfig(*sctimer\_config\_t* \*config)

Fills in the SCTimer configuration structure with the default settings.

The default values are:

```
config->enableCounterUnify = true;
config->clockMode = kSCTIMER_System_ClockMode;
config->clockSelect = kSCTIMER_Clock_On_Rise_Input_0;
config->enableBidirection_l = false;
config->enableBidirection_h = false;
config->prescale_l = 0U;
config->prescale_h = 0U;
config->outInitState = 0U;
config->inputsync = 0xFU;
```

#### Parameters

- config – Pointer to the user configuration structure.

*status\_t* SCTIMER\_SetupPwm(SCT\_Type \*base, const *sctimer\_pwm\_signal\_param\_t* \*pwmParams, *sctimer\_pwm\_mode\_t* mode, uint32\_t pwmFreq\_Hz, uint32\_t srcClock\_Hz, uint32\_t \*event)

Configures the PWM signal parameters.

Call this function to configure the PWM signal period, mode, duty cycle, and edge. This function will create 2 events; one of the events will trigger on match with the pulse value and the other will trigger when the counter matches the PWM period. The PWM period event is also used as a limit event to reset the counter or change direction. Both events are enabled for the same state. The state number can be retrieved by calling the function SCTIMER\_GetCurrentStateNumber(). The counter is set to operate as one 32-bit counter (unify bit is set to 1). The counter operates in bi-directional mode when generating a center-aligned PWM.

---

**Note:** When setting PWM output from multiple output pins, they all should use the same PWM mode i.e all PWM's should be either edge-aligned or center-aligned. When using this API, the PWM signal frequency of all the initialized channels must be the same. Otherwise all the initialized channels' PWM signal frequency is equal to the last call to the API's pwmFreq\_Hz.

---

**Parameters**

- base – SCTimer peripheral base address
- pwmParams – PWM parameters to configure the output
- mode – PWM operation mode, options available in enumeration `sctimer_pwm_mode_t`
- pwmFreq\_Hz – PWM signal frequency in Hz
- srcClock\_Hz – SCTimer counter clock in Hz
- event – Pointer to a variable where the PWM period event number is stored

**Returns**

`kStatus_Success` on success `kStatus_Fail` If we have hit the limit in terms of number of events created or if an incorrect PWM duty cycle is passed in.

```
void SCTIMER_UpdatePwmDutycycle(SCT_Type *base, sctimer_out_t output, uint8_t  
                                dutyCyclePercent, uint32_t event)
```

Updates the duty cycle of an active PWM signal.

Before calling this function, the counter is set to operate as one 32-bit counter (unify bit is set to 1).

**Parameters**

- base – SCTimer peripheral base address
- output – The output to configure
- dutyCyclePercent – New PWM pulse width; the value should be between 1 to 100
- event – Event number associated with this PWM signal. This was returned to the user by the function `SCTIMER_SetupPwm()`.

```
static inline void SCTIMER_EnableInterrupts(SCT_Type *base, uint32_t mask)
```

Enables the selected SCTimer interrupts.

**Parameters**

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

```
static inline void SCTIMER_DisableInterrupts(SCT_Type *base, uint32_t mask)
```

Disables the selected SCTimer interrupts.

**Parameters**

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

```
static inline uint32_t SCTIMER_GetEnabledInterrupts(SCT_Type *base)
```

Gets the enabled SCTimer interrupts.

**Parameters**

- base – SCTimer peripheral base address

**Returns**

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

```
static inline uint32_t SCTIMER_GetStatusFlags(SCT_Type *base)
```

Gets the SCTimer status flags.

#### Parameters

- base – SCTimer peripheral base address

#### Returns

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

```
static inline void SCTIMER_ClearStatusFlags(SCT_Type *base, uint32_t mask)
```

Clears the SCTimer status flags.

#### Parameters

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

```
static inline void SCTIMER_StartTimer(SCT_Type *base, uint32_t countertoStart)
```

Starts the SCTimer counter.

---

**Note:** In 16-bit mode, we can enable both Counter\_L and Counter\_H, In 32-bit mode, we only can select Counter\_U.

---

#### Parameters

- base – SCTimer peripheral base address
- countertoStart – The SCTimer counters to enable. This is a logical OR of members of the enumeration `sctimer_counter_t`.

```
static inline void SCTIMER_StopTimer(SCT_Type *base, uint32_t countertoStop)
```

Halts the SCTimer counter.

#### Parameters

- base – SCTimer peripheral base address
- countertoStop – The SCTimer counters to stop. This is a logical OR of members of the enumeration `sctimer_counter_t`.

```
status_t SCTIMER_CreateAndScheduleEvent(SCT_Type *base, sctimer_event_t howToMonitor,  
                                         uint32_t matchValue, uint32_t whichIO,  
                                         sctimer_counter_t whichCounter, uint32_t *event)
```

Create an event that is triggered on a match or IO and schedule in current state.

This function will configure an event using the options provided by the user. If the event type uses the counter match, then the function will set the user provided match value into a match register and put this match register number into the event control register. The event is enabled for the current state and the event number is increased by one at the end. The function returns the event number; this event number can be used to configure actions to be done when this event is triggered.

#### Parameters

- base – SCTimer peripheral base address
- howToMonitor – Event type; options are available in the enumeration `sctimer_interrupt_enable_t`
- matchValue – The match value that will be programmed to a match register

- `whichIO` – The input or output that will be involved in event triggering. This field is ignored if the event type is “match only”
- `whichCounter` – SCTimer counter to use. In 16-bit mode, we can select `Counter_L` and `Counter_H`, In 32-bit mode, we can select `Counter_U`.
- `event` – Pointer to a variable where the new event number is stored

**Returns**

`kStatus_Success` on success `kStatus_Error` if we have hit the limit in terms of number of events created or if we have reached the limit in terms of number of match registers

`void SCTIMER_ScheduleEvent(SCT_Type *base, uint32_t event)`

Enable an event in the current state.

This function will allow the event passed in to trigger in the current state. The event must be created earlier by either calling the function `SCTIMER_SetupPwm()` or function `SCTIMER_CreateAndScheduleEvent()`.

**Parameters**

- `base` – SCTimer peripheral base address
- `event` – Event number to enable in the current state

`status_t SCTIMER_IncreaseState(SCT_Type *base)`

Increase the state by 1.

All future events created by calling the function `SCTIMER_ScheduleEvent()` will be enabled in this new state.

**Parameters**

- `base` – SCTimer peripheral base address

**Returns**

`kStatus_Success` on success `kStatus_Error` if we have hit the limit in terms of states used

`uint32_t SCTIMER_GetCurrentState(SCT_Type *base)`

Provides the current state.

User can use this to set the next state by calling the function `SCTIMER_SetupNextStateAction()`.

**Parameters**

- `base` – SCTimer peripheral base address

**Returns**

The current state

`static inline void SCTIMER_SetCounterState(SCT_Type *base, sctimer_counter_t whichCounter, uint32_t state)`

Set the counter current state.

The function is to set the state variable bit field of STATE register. Writing to the `STATE_L`, `STATE_H`, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

**Parameters**

- `base` – SCTimer peripheral base address
- `whichCounter` – SCTimer counter to use. In 16-bit mode, we can select `Counter_L` and `Counter_H`, In 32-bit mode, we can select `Counter_U`.
- `state` – The counter current state number (only support range from 0~31).

```
static inline uint16_t SCTIMER_GetCounterState(SCT_Type *base, sctimer_counter_t
                                              whichCounter)
```

Get the counter current state value.

The function is to get the state variable bit field of STATE register.

#### Parameters

- *base* – SCTimer peripheral base address
- *whichCounter* – SCTimer counter to use. In 16-bit mode, we can select Counter\_L and Counter\_H, In 32-bit mode, we can select Counter\_U.

#### Returns

The the counter current state value.

```
status_t SCTIMER_SetupCaptureAction(SCT_Type *base, sctimer_counter_t whichCounter,
                                     uint32_t *captureRegister, uint32_t event)
```

Setup capture of the counter value on trigger of a selected event.

#### Parameters

- *base* – SCTimer peripheral base address
- *whichCounter* – SCTimer counter to use. In 16-bit mode, we can select Counter\_L and Counter\_H, In 32-bit mode, we can select Counter\_U.
- *captureRegister* – Pointer to a variable where the capture register number will be returned. User can read the captured value from this register when the specified event is triggered.
- *event* – Event number that will trigger the capture

#### Returns

kStatus\_Success on success kStatus\_Error if we have hit the limit in terms of number of match/capture registers available

```
void SCTIMER_SetCallback(SCT_Type *base, sctimer_event_callback_t callback, uint32_t event)
```

Receive notification when the event trigger an interrupt.

If the interrupt for the event is enabled by the user, then a callback can be registered which will be invoked when the event is triggered

#### Parameters

- *base* – SCTimer peripheral base address
- *event* – Event number that will trigger the interrupt
- *callback* – Function to invoke when the event is triggered

```
static inline void SCTIMER_SetupStateLdMethodAction(SCT_Type *base, uint32_t event, bool
                                                    fgLoad)
```

Change the load method of transition to the specified state.

Change the load method of transition, it will be triggered by the event number that is passed in by the user.

#### Parameters

- *base* – SCTimer peripheral base address
- *event* – Event number that will change the method to trigger the state transition
- *fgLoad* – The method to load highest-numbered event occurring for that state to the STATE register.

- true: Load the STATEV value to STATE when the event occurs to be the next state.
- false: Add the STATEV value to STATE when the event occurs to be the next state.

```
static inline void SCTIMER_SetupNextStateActionwithLdMethod(SCT_Type *base, uint32_t
                                                         nextState, uint32_t event, bool
                                                         fgLoad)
```

Transition to the specified state with Load method.

This transition will be triggered by the event number that is passed in by the user; the method decide how to load the highest-numbered event occurring for that state to the STATE register.

#### Parameters

- base – SCTimer peripheral base address
- nextState – The next state SCTimer will transition to
- event – Event number that will trigger the state transition
- fgLoad – The method to load the highest-numbered event occurring for that state to the STATE register.
  - true: Load the STATEV value to STATE when the event occurs to be the next state.
  - false: Add the STATEV value to STATE when the event occurs to be the next state.

```
static inline void SCTIMER_SetupNextStateAction(SCT_Type *base, uint32_t nextState, uint32_t
                                              event)
```

Transition to the specified state.

#### *Deprecated:*

Do not use this function. It has been superceded by SCTIMER\_SetupNextStateActionwithLdMethod

This transition will be triggered by the event number that is passed in by the user.

#### Parameters

- base – SCTimer peripheral base address
- nextState – The next state SCTimer will transition to
- event – Event number that will trigger the state transition

```
static inline void SCTIMER_SetupEventActiveDirection(SCT_Type *base,
                                                    sctimer_event_active_direction_t
                                                    activeDirection, uint32_t event)
```

Setup event active direction when the counters are operating in BIDIR mode.

#### Parameters

- base – SCTimer peripheral base address
- activeDirection – Event generation active direction, see sctimer\_event\_active\_direction\_t.
- event – Event number that need setup the active direction.

```
static inline void SCTIMER_SetupOutputSetAction(SCT_Type *base, uint32_t whichIO, uint32_t event)
```

Set the Output.

This output will be set when the event number that is passed in by the user is triggered.

**Parameters**

- base – SCTimer peripheral base address
- whichIO – The output to set
- event – Event number that will trigger the output change

```
static inline void SCTIMER_SetupOutputClearAction(SCT_Type *base, uint32_t whichIO, uint32_t event)
```

Clear the Output.

This output will be cleared when the event number that is passed in by the user is triggered.

**Parameters**

- base – SCTimer peripheral base address
- whichIO – The output to clear
- event – Event number that will trigger the output change

```
void SCTIMER_SetupOutputToggleAction(SCT_Type *base, uint32_t whichIO, uint32_t event)
```

Toggle the output level.

This change in the output level is triggered by the event number that is passed in by the user.

**Parameters**

- base – SCTimer peripheral base address
- whichIO – The output to toggle
- event – Event number that will trigger the output change

```
static inline void SCTIMER_SetupCounterLimitAction(SCT_Type *base, sctimer_counter_t whichCounter, uint32_t event)
```

Limit the running counter.

The counter is limited when the event number that is passed in by the user is triggered.

**Parameters**

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter\_L and Counter\_H, In 32-bit mode, we can select Counter\_U.
- event – Event number that will trigger the counter to be limited

```
static inline void SCTIMER_SetupCounterStopAction(SCT_Type *base, sctimer_counter_t whichCounter, uint32_t event)
```

Stop the running counter.

The counter is stopped when the event number that is passed in by the user is triggered.

**Parameters**

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter\_L and Counter\_H, In 32-bit mode, we can select Counter\_U.
- event – Event number that will trigger the counter to be stopped



```
static inline void SCTIMER_SetupCounterStartAction(SCT_Type *base, sctimer_counter_t
                                                    whichCounter, uint32_t event)
```

Re-start the stopped counter.

The counter will re-start when the event number that is passed in by the user is triggered.

#### Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter\_L and Counter\_H, In 32-bit mode, we can select Counter\_U.
- event – Event number that will trigger the counter to re-start

```
static inline void SCTIMER_SetupCounterHaltAction(SCT_Type *base, sctimer_counter_t
                                                    whichCounter, uint32_t event)
```

Halt the running counter.

The counter is disabled (halted) when the event number that is passed in by the user is triggered. When the counter is halted, all further events are disabled. The HALT condition can only be removed by calling the SCTIMER\_StartTimer() function.

#### Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter\_L and Counter\_H, In 32-bit mode, we can select Counter\_U.
- event – Event number that will trigger the counter to be halted

```
static inline void SCTIMER_SetupDmaTriggerAction(SCT_Type *base, uint32_t dmaNumber,
                                                  uint32_t event)
```

Generate a DMA request.

DMA request will be triggered by the event number that is passed in by the user.

#### Parameters

- base – SCTimer peripheral base address
- dmaNumber – The DMA request to generate
- event – Event number that will trigger the DMA request

```
static inline void SCTIMER_SetCOUNTValue(SCT_Type *base, sctimer_counter_t whichCounter,
                                           uint32_t value)
```

Set the value of counter.

The function is to set the value of Count register, Writing to the COUNT\_L, COUNT\_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

#### Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter\_L and Counter\_H, In 32-bit mode, we can select Counter\_U.
- value – the counter value update to the COUNT register.

```
static inline uint32_t SCTIMER_GetCOUNTValue(SCT_Type *base, sctimer_counter_t
                                                whichCounter)
```

Get the value of counter.

The function is to read the value of Count register, software can read the counter registers at any time..

**Parameters**

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter\_L and Counter\_H, In 32-bit mode, we can select Counter\_U.

**Returns**

The value of counter selected.

```
static inline void SCTIMER_SetEventInState(SCT_Type *base, uint32_t event, uint32_t state)
```

Set the state mask bit field of EV\_STATE register.

**Parameters**

- base – SCTimer peripheral base address
- event – The EV\_STATE register be set.
- state – The state value in which the event is enabled to occur.

```
static inline void SCTIMER_ClearEventInState(SCT_Type *base, uint32_t event, uint32_t state)
```

Clear the state mask bit field of EV\_STATE register.

**Parameters**

- base – SCTimer peripheral base address
- event – The EV\_STATE register be clear.
- state – The state value in which the event is disabled to occur.

```
static inline bool SCTIMER_GetEventInState(SCT_Type *base, uint32_t event, uint32_t state)
```

Get the state mask bit field of EV\_STATE register.

---

**Note:** This function is to check whether the event is enabled in a specific state.

---

**Parameters**

- base – SCTimer peripheral base address
- event – The EV\_STATE register be read.
- state – The state value.

**Returns**

The the state mask bit field of EV\_STATE register.

- true: The event is enable in state.
- false: The event is disable in state.

```
static inline uint32_t SCTIMER_GetCaptureValue(SCT_Type *base, sctimer_counter_t  
whichCounter, uint8_t capChannel)
```

Get the value of capture register.

This function returns the captured value upon occurrence of the events selected by the corresponding Capture Control registers occurred.

**Parameters**

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter\_L and Counter\_H, In 32-bit mode, we can select Counter\_U.
- capChannel – SCTimer capture register of capture channel.

**Returns**

The SCTimer counter value at which this register was last captured.

void SCTIMER\_EventHandleIRQ(SCT\_Type \*base)

SCTimer interrupt handler.

**Parameters**

- base – SCTimer peripheral base address.

FSL\_SCTIMER\_DRIVER\_VERSION

Version

enum \_\_sctimer\_pwm\_mode

SCTimer PWM operation modes.

*Values:*

enumerator kSCTIMER\_EdgeAlignedPwm

Edge-aligned PWM

enumerator kSCTIMER\_CenterAlignedPwm

Center-aligned PWM

enum \_\_sctimer\_counter

SCTimer counters type.

*Values:*

enumerator kSCTIMER\_Counter\_L

16-bit Low counter.

enumerator kSCTIMER\_Counter\_H

16-bit High counter.

enumerator kSCTIMER\_Counter\_U

32-bit Unified counter.

enum \_\_sctimer\_input

List of SCTimer input pins.

*Values:*

enumerator kSCTIMER\_Input\_0

SCTIMER input 0

enumerator kSCTIMER\_Input\_1

SCTIMER input 1

enumerator kSCTIMER\_Input\_2

SCTIMER input 2

enumerator kSCTIMER\_Input\_3

SCTIMER input 3

enumerator kSCTIMER\_Input\_4

SCTIMER input 4

enumerator kSCTIMER\_Input\_5

SCTIMER input 5

enumerator kSCTIMER\_Input\_6

SCTIMER input 6

enumerator kSCTIMER\_Input\_7  
SCTIMER input 7

enum \_sctimer\_out

List of SCTimer output pins.

*Values:*

enumerator kSCTIMER\_Out\_0  
SCTIMER output 0

enumerator kSCTIMER\_Out\_1  
SCTIMER output 1

enumerator kSCTIMER\_Out\_2  
SCTIMER output 2

enumerator kSCTIMER\_Out\_3  
SCTIMER output 3

enumerator kSCTIMER\_Out\_4  
SCTIMER output 4

enumerator kSCTIMER\_Out\_5  
SCTIMER output 5

enumerator kSCTIMER\_Out\_6  
SCTIMER output 6

enumerator kSCTIMER\_Out\_7  
SCTIMER output 7

enumerator kSCTIMER\_Out\_8  
SCTIMER output 8

enumerator kSCTIMER\_Out\_9  
SCTIMER output 9

enum \_sctimer\_pwm\_level\_select

SCTimer PWM output pulse mode: high-true, low-true or no output.

*Values:*

enumerator kSCTIMER\_LowTrue  
Low true pulses

enumerator kSCTIMER\_HighTrue  
High true pulses

enum \_sctimer\_clock\_mode

SCTimer clock mode options.

*Values:*

enumerator kSCTIMER\_System\_ClockMode  
System Clock Mode

enumerator kSCTIMER\_Sampled\_ClockMode  
Sampled System Clock Mode

enumerator kSCTIMER\_Input\_ClockMode  
SCT Input Clock Mode

enumerator kSCTIMER\_Asynchronous\_ClockMode  
Asynchronous Mode

enum \_sctimer\_clock\_select  
SCTimer clock select options.

*Values:*

enumerator kSCTIMER\_Clock\_On\_Rise\_Input\_0  
Rising edges on input 0

enumerator kSCTIMER\_Clock\_On\_Fall\_Input\_0  
Falling edges on input 0

enumerator kSCTIMER\_Clock\_On\_Rise\_Input\_1  
Rising edges on input 1

enumerator kSCTIMER\_Clock\_On\_Fall\_Input\_1  
Falling edges on input 1

enumerator kSCTIMER\_Clock\_On\_Rise\_Input\_2  
Rising edges on input 2

enumerator kSCTIMER\_Clock\_On\_Fall\_Input\_2  
Falling edges on input 2

enumerator kSCTIMER\_Clock\_On\_Rise\_Input\_3  
Rising edges on input 3

enumerator kSCTIMER\_Clock\_On\_Fall\_Input\_3  
Falling edges on input 3

enumerator kSCTIMER\_Clock\_On\_Rise\_Input\_4  
Rising edges on input 4

enumerator kSCTIMER\_Clock\_On\_Fall\_Input\_4  
Falling edges on input 4

enumerator kSCTIMER\_Clock\_On\_Rise\_Input\_5  
Rising edges on input 5

enumerator kSCTIMER\_Clock\_On\_Fall\_Input\_5  
Falling edges on input 5

enumerator kSCTIMER\_Clock\_On\_Rise\_Input\_6  
Rising edges on input 6

enumerator kSCTIMER\_Clock\_On\_Fall\_Input\_6  
Falling edges on input 6

enumerator kSCTIMER\_Clock\_On\_Rise\_Input\_7  
Rising edges on input 7

enumerator kSCTIMER\_Clock\_On\_Fall\_Input\_7  
Falling edges on input 7

enum \_sctimer\_conflict\_resolution  
SCTimer output conflict resolution options.

Specifies what action should be taken if multiple events dictate that a given output should be both set and cleared at the same time

*Values:*

enumerator kSCTIMER\_ResolveNone

No change

enumerator kSCTIMER\_ResolveSet

Set output

enumerator kSCTIMER\_ResolveClear

Clear output

enumerator kSCTIMER\_ResolveToggle

Toggle output

enum \_sctimer\_event\_active\_direction

List of SCTimer event generation active direction when the counters are operating in BIDIR mode.

*Values:*

enumerator kSCTIMER\_ActiveIndependent

This event is triggered regardless of the count direction.

enumerator kSCTIMER\_ActiveInCountUp

This event is triggered only during up-counting when BIDIR = 1.

enumerator kSCTIMER\_ActiveInCountDown

This event is triggered only during down-counting when BIDIR = 1.

enum \_sctimer\_event

List of SCTimer event types.

*Values:*

enumerator kSCTIMER\_InputLowOrMatchEvent

enumerator kSCTIMER\_InputRiseOrMatchEvent

enumerator kSCTIMER\_InputFallOrMatchEvent

enumerator kSCTIMER\_InputHighOrMatchEvent

enumerator kSCTIMER\_MatchEventOnly

enumerator kSCTIMER\_InputLowEvent

enumerator kSCTIMER\_InputRiseEvent

enumerator kSCTIMER\_InputFallEvent

enumerator kSCTIMER\_InputHighEvent

enumerator kSCTIMER\_InputLowAndMatchEvent

enumerator kSCTIMER\_InputRiseAndMatchEvent

enumerator kSCTIMER\_InputFallAndMatchEvent

enumerator kSCTIMER\_InputHighAndMatchEvent

enumerator kSCTIMER\_OutputLowOrMatchEvent

enumerator kSCTIMER\_OutputRiseOrMatchEvent

enumerator kSCTIMER\_OutputFallOrMatchEvent

enumerator kSCTIMER\_OutputHighOrMatchEvent

enumerator kSCTIMER\_OutputLowEvent  
enumerator kSCTIMER\_OutputRiseEvent  
enumerator kSCTIMER\_OutputFallEvent  
enumerator kSCTIMER\_OutputHighEvent  
enumerator kSCTIMER\_OutputLowAndMatchEvent  
enumerator kSCTIMER\_OutputRiseAndMatchEvent  
enumerator kSCTIMER\_OutputFallAndMatchEvent  
enumerator kSCTIMER\_OutputHighAndMatchEvent

enum \_sctimer\_interrupt\_enable

List of SCTimer interrupts.

*Values:*

enumerator kSCTIMER\_Event0InterruptEnable  
Event 0 interrupt  
enumerator kSCTIMER\_Event1InterruptEnable  
Event 1 interrupt  
enumerator kSCTIMER\_Event2InterruptEnable  
Event 2 interrupt  
enumerator kSCTIMER\_Event3InterruptEnable  
Event 3 interrupt  
enumerator kSCTIMER\_Event4InterruptEnable  
Event 4 interrupt  
enumerator kSCTIMER\_Event5InterruptEnable  
Event 5 interrupt  
enumerator kSCTIMER\_Event6InterruptEnable  
Event 6 interrupt  
enumerator kSCTIMER\_Event7InterruptEnable  
Event 7 interrupt  
enumerator kSCTIMER\_Event8InterruptEnable  
Event 8 interrupt  
enumerator kSCTIMER\_Event9InterruptEnable  
Event 9 interrupt  
enumerator kSCTIMER\_Event10InterruptEnable  
Event 10 interrupt  
enumerator kSCTIMER\_Event11InterruptEnable  
Event 11 interrupt  
enumerator kSCTIMER\_Event12InterruptEnable  
Event 12 interrupt

enum \_sctimer\_status\_flags

List of SCTimer flags.

*Values:*

enumerator kSCTIMER\_Event0Flag

Event 0 Flag

enumerator kSCTIMER\_Event1Flag

Event 1 Flag

enumerator kSCTIMER\_Event2Flag

Event 2 Flag

enumerator kSCTIMER\_Event3Flag

Event 3 Flag

enumerator kSCTIMER\_Event4Flag

Event 4 Flag

enumerator kSCTIMER\_Event5Flag

Event 5 Flag

enumerator kSCTIMER\_Event6Flag

Event 6 Flag

enumerator kSCTIMER\_Event7Flag

Event 7 Flag

enumerator kSCTIMER\_Event8Flag

Event 8 Flag

enumerator kSCTIMER\_Event9Flag

Event 9 Flag

enumerator kSCTIMER\_Event10Flag

Event 10 Flag

enumerator kSCTIMER\_Event11Flag

Event 11 Flag

enumerator kSCTIMER\_Event12Flag

Event 12 Flag

enumerator kSCTIMER\_BusErrorLFlag

Bus error due to write when L counter was not halted

enumerator kSCTIMER\_BusErrorHFlag

Bus error due to write when H counter was not halted

typedef enum *\_sctimer\_pwm\_mode* sctimer\_pwm\_mode\_t

SCTimer PWM operation modes.

typedef enum *\_sctimer\_counter* sctimer\_counter\_t

SCTimer counters type.

typedef enum *\_sctimer\_input* sctimer\_input\_t

List of SCTimer input pins.

typedef enum *\_sctimer\_out* sctimer\_out\_t

List of SCTimer output pins.

typedef enum *\_sctimer\_pwm\_level\_select* sctimer\_pwm\_level\_select\_t

SCTimer PWM output pulse mode: high-true, low-true or no output.

typedef struct *\_sctimer\_pwm\_signal\_param* sctimer\_pwm\_signal\_param\_t

Options to configure a SCTimer PWM signal.



```
typedef enum _sctimer_clock_mode sctimer__clock_mode_t
```

SCTimer clock mode options.

```
typedef enum _sctimer_clock_select sctimer__clock_select_t
```

SCTimer clock select options.

```
typedef enum _sctimer_conflict_resolution sctimer__conflict_resolution_t
```

SCTimer output conflict resolution options.

Specifies what action should be taken if multiple events dictate that a given output should be both set and cleared at the same time

```
typedef enum _sctimer_event_active_direction sctimer__event_active_direction_t
```

List of SCTimer event generation active direction when the counters are operating in BIDIR mode.

```
typedef enum _sctimer_event sctimer__event_t
```

List of SCTimer event types.

```
typedef void (*sctimer__event_callback_t)(void)
```

SCTimer callback typedef.

```
typedef enum _sctimer_interrupt_enable sctimer__interrupt_enable_t
```

List of SCTimer interrupts.

```
typedef enum _sctimer_status_flags sctimer__status_flags_t
```

List of SCTimer flags.

```
typedef struct _sctimer_config sctimer__config_t
```

SCTimer configuration structure.

This structure holds the configuration settings for the SCTimer peripheral. To initialize this structure to reasonable defaults, call the SCTMR\_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

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

```
SCT_EV_STATE_STATEMSK(x)
```

```
struct _sctimer_pwm_signal_param
```

*#include <fsl\_sctimer.h>* Options to configure a SCTimer PWM signal.

## Public Members

```
sctimer_out_t output
```

The output pin to use to generate the PWM signal

```
sctimer_pwm_level_select_t level
```

PWM output active level select.

```
uint8_t dutyCyclePercent
```

PWM pulse width, value should be between 0 to 100 0 = always inactive signal (0% duty cycle) 100 = always active signal (100% duty cycle).

```
struct _sctimer_config
```

*#include <fsl\_sctimer.h>* SCTimer configuration structure.

This structure holds the configuration settings for the SCTimer peripheral. To initialize this structure to reasonable defaults, call the SCTMR\_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

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

## Public Members

bool enableCounterUnify

true: SCT operates as a unified 32-bit counter; false: SCT operates as two 16-bit counters. User can use the 16-bit low counter and the 16-bit high counters at the same time; for Hardware limit, user can not use unified 32-bit counter and any 16-bit low/high counter at the same time.

sctimer\_clock\_mode\_t clockMode

SCT clock mode value

sctimer\_clock\_select\_t clockSelect

SCT clock select value

bool enableBidirection\_l

true: Up-down count mode for the L or unified counter false: Up count mode only for the L or unified counter

bool enableBidirection\_h

true: Up-down count mode for the H or unified counter false: Up count mode only for the H or unified counter. This field is used only if the enableCounterUnify is set to false

uint8\_t prescale\_l

Prescale value to produce the L or unified counter clock

uint8\_t prescale\_h

Prescale value to produce the H counter clock. This field is used only if the enableCounterUnify is set to false

uint8\_t outInitState

Defines the initial output value

uint8\_t inputsync

SCT INSYNC value, INSYNC field in the CONFIG register, from bit9 to bit 16. it is used to define synchronization for input N: bit 9 = input 0 bit 10 = input 1 bit 11 = input 2 bit 12 = input 3 All other bits are reserved (bit13 ~bit 16). How User to set the the value for the member inputsync. IE: delay for input0, and input 1, bypasses for input 2 and input 3 MACRO definition in user level. #define INPUTSYNC0 (0U) #define INPUTSYNC1 (1U) #define INPUTSYNC2 (2U) #define INPUTSYNC3 (3U) User Code. sctimerInfo.inputsync = (1 « INPUTSYNC2) | (1 « INPUTSYNC3);

## 2.41 SDIF: SD/MMC/SDIO card interface

FSL\_SDIF\_DRIVER\_VERSION

Driver version 2.0.15.

\_sdif\_status SDIF status

*Values:*

enumerator kStatus\_SDIF\_DescriptorBufferLenError

Set DMA descriptor failed

enumerator kStatus\_SDIF\_InvalidArgument

invalid argument status

enumerator kStatus\_SDIF\_SyncCmdTimeout

sync command to CIU timeout status

enumerator kStatus\_SDIF\_SendCmdFail  
send command to card fail

enumerator kStatus\_SDIF\_SendCmdErrorBufferFull  
send command to card fail, due to command buffer full user need to resend this command

enumerator kStatus\_SDIF\_DMATransferFailWithFBE  
DMA transfer data fail with fatal bus error , to do with this error :issue a hard reset/controller reset

enumerator kStatus\_SDIF\_DMATransferDescriptorUnavailable  
DMA descriptor unavailable

enumerator kStatus\_SDIF\_DataTransferFail  
transfer data fail

enumerator kStatus\_SDIF\_ResponseError  
response error

enumerator kStatus\_SDIF\_DMAAddrNotAlign  
DMA address not align

enumerator kStatus\_SDIF\_BusyTransferring  
SDIF transfer busy status

enumerator kStatus\_SDIF\_DataTransferSuccess  
transfer data success

enumerator kStatus\_SDIF\_SendCmdSuccess  
transfer command success

\_sdif\_capability\_flag Host controller capabilities flag mask

*Values:*

enumerator kSDIF\_SupportHighSpeedFlag  
Support high-speed

enumerator kSDIF\_SupportDmaFlag  
Support DMA

enumerator kSDIF\_SupportSuspendResumeFlag  
Support suspend/resume

enumerator kSDIF\_SupportV330Flag  
Support voltage 3.3V

enumerator kSDIF\_Support4BitFlag  
Support 4 bit mode

enumerator kSDIF\_Support8BitFlag  
Support 8 bit mode

\_sdif\_reset\_type define the reset type

*Values:*

enumerator kSDIF\_ResetController  
reset controller,will reset: BIU/CIU interface CIU and state machine,ABORT\_READ\_DATA,SEND\_IRQ\_RESPONSE and READ\_WAIT bits of control register,START\_CMD bit of the command register

enumerator kSDIF\_\_ResetFIFO  
reset data FIFO

enumerator kSDIF\_\_ResetDMAInterface  
reset DMA interface

enumerator kSDIF\_\_ResetAll  
reset all

enum \_sdif\_\_bus\_\_width  
define the card bus width type

*Values:*

enumerator kSDIF\_\_Bus1BitWidth  
1bit bus width, 1bit mode and 4bit mode share one register bit

enumerator kSDIF\_\_Bus4BitWidth  
4bit mode mask

enumerator kSDIF\_\_Bus8BitWidth  
support 8 bit mode

\_sdif\_\_command\_\_flags define the command flags

*Values:*

enumerator kSDIF\_\_CmdResponseExpect  
command request response

enumerator kSDIF\_\_CmdResponseLengthLong  
command response length long

enumerator kSDIF\_\_CmdCheckResponseCRC  
request check command response CRC

enumerator kSDIF\_\_DataExpect  
request data transfer,either read/write

enumerator kSDIF\_\_DataWriteToCard  
data transfer direction

enumerator kSDIF\_\_DataStreamTransfer  
data transfer mode :stream/block transfer command

enumerator kSDIF\_\_DataTransferAutoStop  
data transfer with auto stop at the end of

enumerator kSDIF\_\_WaitPreTransferComplete  
wait pre transfer complete before sending this cmd

enumerator kSDIF\_\_TransferStopAbort  
when host issue stop or abort cmd to stop data transfer ,this bit should set so that  
cmd/data state-machines of CIU can return to idle correctly

enumerator kSDIF\_\_SendInitialization  
send initialization 80 clocks for SD card after power on

enumerator kSDIF\_\_CmdUpdateClockRegisterOnly  
send cmd update the CIU clock register only

enumerator kSDIF\_\_CmdtoReadCEATADevice  
host is perform read access to CE-ATA device

enumerator kSDIF\_CmdExpectCCS  
command expect command completion signal signal

enumerator kSDIF\_BootModeEnable  
this bit should only be set for mandatory boot mode

enumerator kSDIF\_BootModeExpectAck  
boot mode expect ack

enumerator kSDIF\_BootModeDisable  
when software set this bit along with START\_CMD, CIU terminates the boot operation

enumerator kSDIF\_BootModeAlternate  
select boot mode ,alternate or mandatory

enumerator kSDIF\_CmdVoltageSwitch  
this bit set for CMD11 only

enumerator kSDIF\_CmdDataUseHoldReg  
cmd and data send to card through the HOLD register

\_sdif\_command\_type The command type

*Values:*

enumerator kCARD\_CommandTypeNormal  
Normal command

enumerator kCARD\_CommandTypeSuspend  
Suspend command

enumerator kCARD\_CommandTypeResume  
Resume command

enumerator kCARD\_CommandTypeAbort  
Abort command

\_sdif\_response\_type The command response type.

Define the command response type from card to host controller.

*Values:*

enumerator kCARD\_ResponseTypeName  
Response type: none

enumerator kCARD\_ResponseTypeR1  
Response type: R1

enumerator kCARD\_ResponseTypeR1b  
Response type: R1b

enumerator kCARD\_ResponseTypeR2  
Response type: R2

enumerator kCARD\_ResponseTypeR3  
Response type: R3

enumerator kCARD\_ResponseTypeR4  
Response type: R4

enumerator kCARD\_ResponseTypeR5

Response type: R5

enumerator kCARD\_ResponseTypeR5b

Response type: R5b

enumerator kCARD\_ResponseTypeR6

Response type: R6

enumerator kCARD\_ResponseTypeR7

Response type: R7

`_sdif_interrupt_mask` define the interrupt mask flags

*Values:*

enumerator kSDIF\_CardDetect

mask for card detect

enumerator kSDIF\_ResponseError

command response error

enumerator kSDIF\_CommandDone

command transfer over

enumerator kSDIF\_DataTransferOver

data transfer over flag

enumerator kSDIF\_WriteFIFORequest

write FIFO request

enumerator kSDIF\_ReadFIFORequest

read FIFO request

enumerator kSDIF\_ResponseCRCError

response CRC error

enumerator kSDIF\_DataCRCError

data CRC error

enumerator kSDIF\_ResponseTimeout

response timeout

enumerator kSDIF\_DataReadTimeout

read data timeout

enumerator kSDIF\_DataStarvationByHostTimeout

data starvation by host time out

enumerator kSDIF\_FIFOError

indicate the FIFO under run or overrun error

enumerator kSDIF\_HardwareLockError

hardware lock write error

enumerator kSDIF\_DataStartBitError

start bit error

enumerator kSDIF\_AutoCmdDone

indicate the auto command done

enumerator kSDIF\_\_DataEndBitError  
end bit error

enumerator kSDIF\_\_SDIOInterrupt  
interrupt from the SDIO card

enumerator kSDIF\_\_CommandTransferStatus  
command transfer status collection

enumerator kSDIF\_\_DataTransferStatus  
data transfer status collection

enumerator kSDIF\_\_DataTransferError

enumerator kSDIF\_\_AllInterruptStatus  
all interrupt mask

`_sdif_dma_status` define the internal DMA status flags

*Values:*

enumerator kSDIF\_\_DMATransFinishOneDescriptor  
DMA transfer finished for one DMA descriptor

enumerator kSDIF\_\_DMARecvFinishOneDescriptor  
DMA receive finished for one DMA descriptor

enumerator kSDIF\_\_DMAFatalBusError  
DMA fatal bus error

enumerator kSDIF\_\_DMADescriptorUnavailable  
DMA descriptor unavailable

enumerator kSDIF\_\_DMACardErrorSummary  
card error summary

enumerator kSDIF\_\_NormalInterruptSummary  
normal interrupt summary

enumerator kSDIF\_\_AbnormalInterruptSummary  
abnormal interrupt summary

enumerator kSDIF\_\_DMAAllStatus

`_sdif_dma_descriptor_flag` define the internal DMA descriptor flag

*Deprecated:*

Do not use this enum anymore, please use `SDIF_DMA_DESCRIPTOR_XXX_FLAG` instead.

*Values:*

enumerator kSDIF\_\_DisableCompleteInterrupt  
disable the complete interrupt flag for the ends in the buffer pointed to by this descriptor

enumerator kSDIF\_\_DMADescriptorDataBufferEnd  
indicate this descriptor contain the last data buffer of data

enumerator `kSDIF_DMADescriptorDataBufferStart`

indicate this descriptor contain the first data buffer of data,if first buffer size is 0,next descriptor contain the begin of the data

enumerator `kSDIF_DMASecondAddrChained`

indicate that the second addr in the descriptor is the next descriptor addr not the data buffer

enumerator `kSDIF_DMADescriptorEnd`

indicate that the descriptor list reached its final descriptor

enumerator `kSDIF_DMADescriptorOwnByDMA`

indicate the descriptor is own by SD/MMC DMA

enum `_sdif_dma_mode`

define the internal DMA mode

*Values:*

enumerator `kSDIF_ChainDMAMode`

enumerator `kSDIF_DualDMAMode`

typedef enum `_sdif_bus_width` `sdif_bus_width_t`

define the card bus width type

typedef enum `_sdif_dma_mode` `sdif_dma_mode_t`

define the internal DMA mode

typedef struct `_sdif_dma_descriptor` `sdif_dma_descriptor_t`

define the internal DMA descriptor

typedef struct `_sdif_dma_config` `sdif_dma_config_t`

Defines the internal DMA configure structure.

typedef struct `_sdif_data` `sdif_data_t`

Card data descriptor.

typedef struct `_sdif_command` `sdif_command_t`

Card command descriptor.

Define card command-related attribute.

typedef struct `_sdif_transfer` `sdif_transfer_t`

Transfer state.

typedef struct `_sdif_config` `sdif_config_t`

Data structure to initialize the sdif.

typedef struct `_sdif_capability` `sdif_capability_t`

SDIF capability information. Defines a structure to get the capability information of SDIF.

typedef struct `_sdif_transfer_callback` `sdif_transfer_callback_t`

sdif callback functions.

typedef struct `_sdif_handle` `sdif_handle_t`

sdif handle

Defines the structure to save the sdif state information and callback function. The detail interrupt status when send command or transfer data can be obtained from `interruptFlags` field by using mask defined in `sdif_interrupt_flag_t`;



---

**Note:** All the fields except interruptFlags and transferredWords must be allocated by the user.

---

typedef *status\_t* (\*sdif\_transfer\_function\_t)(SDIF\_Type \*base, *sdif\_transfer\_t* \*content)  
sdif transfer function.

typedef struct *\_sdif\_host* sdif\_host\_t  
sdif host descriptor

void SDIF\_Init(SDIF\_Type \*base, *sdif\_config\_t* \*config)  
SDIF module initialization function.  
Configures the SDIF according to the user configuration.

**Parameters**

- base – SDIF peripheral base address.
- config – SDIF configuration information.

void SDIF\_Deinit(SDIF\_Type \*base)  
SDIF module deinit function. user should call this function follow with IP reset.

**Parameters**

- base – SDIF peripheral base address.

bool SDIF\_SendCardActive(SDIF\_Type \*base, uint32\_t timeout)  
SDIF send initialize 80 clocks for SD card after initial.

**Parameters**

- base – SDIF peripheral base address.
- timeout – timeout value

static inline void SDIF\_EnableCardClock(SDIF\_Type \*base, bool enable)  
SDIF module enable/disable card clock.

**Parameters**

- base – SDIF peripheral base address.
- enable – enable/disable flag

static inline void SDIF\_EnableLowPowerMode(SDIF\_Type \*base, bool enable)  
SDIF module enable/disable module disable the card clock to enter low power mode when card is idle,for SDIF cards, if interrupts must be detected, clock should not be stopped.

**Parameters**

- base – SDIF peripheral base address.
- enable – enable/disable flag

static inline void SDIF\_EnableCardPower(SDIF\_Type \*base, bool enable)  
enable/disable the card power. once turn power on, software should wait for regulator/switch ramp-up time before trying to initialize card.

**Parameters**

- base – SDIF peripheral base address.
- enable – enable/disable flag.

`void SDIF_SetCardBusWidth(SDIF_Type *base, sdif_bus_width_t type)`  
set card data bus width

**Parameters**

- base – SDIF peripheral base address.
- type – bus width type

`static inline uint32_t SDIF_DetectCardInsert(SDIF_Type *base, bool data3)`  
SDIF module detect card insert status function.

**Parameters**

- base – SDIF peripheral base address.
- data3 – indicate use data3 as card insert detect pin

**Return values**

1 – card is inserted 0 card is removed

`uint32_t SDIF_SetCardClock(SDIF_Type *base, uint32_t srcClock_Hz, uint32_t target_HZ)`  
Sets the card bus clock frequency.

**Parameters**

- base – SDIF peripheral base address.
- srcClock\_Hz – SDIF source clock frequency united in Hz.
- target\_HZ – card bus clock frequency united in Hz.

**Returns**

The nearest frequency of busClock\_Hz configured to SD bus.

`bool SDIF_Reset(SDIF_Type *base, uint32_t mask, uint32_t timeout)`  
reset the different block of the interface.

**Parameters**

- base – SDIF peripheral base address.
- mask – indicate which block to reset.
- timeout – timeout value, set to wait the bit self clear

**Returns**

reset result.

`static inline uint32_t SDIF_GetCardWriteProtect(SDIF_Type *base)`  
get the card write protect status

**Parameters**

- base – SDIF peripheral base address.

`static inline void SDIF_AssertHardwareReset(SDIF_Type *base)`  
toggle state on hardware reset PIN This is used which card has a reset PIN typically.

**Parameters**

- base – SDIF peripheral base address.

`status_t SDIF_SendCommand(SDIF_Type *base, sdif_command_t *cmd, uint32_t timeout)`  
send command to the card

This api include polling the status of the bit START\_COMMAND, if 0 used as timeout value, then this function will return directly without polling the START\_CMD status.

**Parameters**

- `base` – SDIF peripheral base address.
- `cmd` – configuration collection
- `timeout` – the timeout value of polling `START_CMD` auto clear status.

**Returns**

command excute status

static inline void SDIF\_\_EnableGlobalInterrupt(SDIF\_Type \*base, bool enable)  
SDIF enable/disable global interrupt.

**Parameters**

- `base` – SDIF peripheral base address.
- `enable` – enable/disable flag

static inline void SDIF\_\_EnableInterrupt(SDIF\_Type \*base, uint32\_t mask)  
SDIF enable interrupt.

**Parameters**

- `base` – SDIF peripheral base address.
- `mask` – mask

static inline void SDIF\_\_DisableInterrupt(SDIF\_Type \*base, uint32\_t mask)  
SDIF disable interrupt.

**Parameters**

- `base` – SDIF peripheral base address.
- `mask` – mask

static inline uint32\_t SDIF\_\_GetInterruptStatus(SDIF\_Type \*base)  
SDIF get interrupt status.

**Parameters**

- `base` – SDIF peripheral base address.

static inline uint32\_t SDIF\_\_GetEnabledInterruptStatus(SDIF\_Type \*base)  
SDIF get enabled interrupt status.

**Parameters**

- `base` – SDIF peripheral base address.

static inline void SDIF\_\_ClearInterruptStatus(SDIF\_Type \*base, uint32\_t mask)  
SDIF clear interrupt status.

**Parameters**

- `base` – SDIF peripheral base address.
- `mask` – mask to clear

void SDIF\_\_TransferCreateHandle(SDIF\_Type \*base, *sdif\_handle\_t* \*handle,  
*sdif\_transfer\_callback\_t* \*callback, void \*userData)

Creates the SDIF handle. register call back function for interrupt and enable the interrupt.

**Parameters**

- `base` – SDIF peripheral base address.
- `handle` – SDIF handle pointer.
- `callback` – Structure pointer to contain all callback functions.
- `userData` – Callback function parameter.

static inline void SDIF\_EnableDmaInterrupt(SDIF\_Type \*base, uint32\_t mask)  
SDIF enable DMA interrupt.

**Parameters**

- base – SDIF peripheral base address.
- mask – mask to set

static inline void SDIF\_DisableDmaInterrupt(SDIF\_Type \*base, uint32\_t mask)  
SDIF disable DMA interrupt.

**Parameters**

- base – SDIF peripheral base address.
- mask – mask to clear

static inline uint32\_t SDIF\_GetInternalDMAStatus(SDIF\_Type \*base)  
SDIF get internal DMA status.

**Parameters**

- base – SDIF peripheral base address.

**Returns**

the internal DMA status register

static inline uint32\_t SDIF\_GetEnabledDMAInterruptStatus(SDIF\_Type \*base)  
SDIF get enabled internal DMA interrupt status.

**Parameters**

- base – SDIF peripheral base address.

**Returns**

the internal DMA status register

static inline void SDIF\_ClearInternalDMAStatus(SDIF\_Type \*base, uint32\_t mask)  
SDIF clear internal DMA status.

**Parameters**

- base – SDIF peripheral base address.
- mask – mask to clear

*status\_t* SDIF\_InternalDMAConfig(SDIF\_Type \*base, *sdif\_dma\_config\_t* \*config, const uint32\_t  
\*data, uint32\_t dataSize)

SDIF internal DMA config function.

**Parameters**

- base – SDIF peripheral base address.
- config – DMA configuration collection
- data – buffer pointer
- dataSize – buffer size

static inline void SDIF\_EnableInternalDMA(SDIF\_Type \*base, bool enable)  
SDIF internal DMA enable.

**Parameters**

- base – SDIF peripheral base address.
- enable – internal DMA enable or disable flag.

static inline void SDIF\_SendReadWait(SDIF\_Type \*base)

SDIF send read wait to SDIF card function.

**Parameters**

- base – SDIF peripheral base address.

bool SDIF\_AbortReadData(SDIF\_Type \*base, uint32\_t timeout)

SDIF abort the read data when SDIF card is in suspend state Once assert this bit, data state machine will be reset which is waiting for the next blocking data, used in SDIO card suspend sequence, should call after suspend cmd send.

**Parameters**

- base – SDIF peripheral base address.
- timeout – timeout value to wait this bit self clear which indicate the data machine reset to idle

static inline void SDIF\_EnableCEATAInterrupt(SDIF\_Type \*base, bool enable)

SDIF enable/disable CE-ATA card interrupt this bit should set together with the card register.

**Parameters**

- base – SDIF peripheral base address.
- enable – enable/disable flag

status\_t SDIF\_TransferNonBlocking(SDIF\_Type \*base, sdif\_handle\_t \*handle, sdif\_dma\_config\_t \*dmaConfig, sdif\_transfer\_t \*transfer)

SDIF transfer function data/cmd in a non-blocking way this API should be use in interrupt mode, when use this API user must call SDIF\_TransferCreateHandle first, all status check through interrupt.

**Parameters**

- base – SDIF peripheral base address.
- handle – handle
- dmaConfig – config structure This parameter can be config as:
  - a. NULL In this condition, polling transfer mode is selected
  - b. available DMA config In this condition, DMA transfer mode is selected
- transfer – transfer configuration collection

status\_t SDIF\_TransferBlocking(SDIF\_Type \*base, sdif\_dma\_config\_t \*dmaConfig, sdif\_transfer\_t \*transfer)

SDIF transfer function data/cmd in a blocking way.

**Parameters**

- base – SDIF peripheral base address.
- dmaConfig – config structure
  - a. NULL In this condition, polling transfer mode is selected
  - b. available DMA config In this condition, DMA transfer mode is selected
- transfer – transfer configuration collection

status\_t SDIF\_ReleaseDMADescriptor(SDIF\_Type \*base, sdif\_dma\_config\_t \*dmaConfig)

SDIF release the DMA descriptor to DMA engine this function should be called when DMA descriptor unavailable status occurs.

**Parameters**

- base – SDIF peripheral base address.

- dmaConfig – DMA config pointer

void SDIF\_GetCapability(SDIF\_Type \*base, *sdif\_capability\_t* \*capability)

SDIF return the controller capability.

#### Parameters

- base – SDIF peripheral base address.
- capability – capability pointer

static inline uint32\_t SDIF\_GetControllerStatus(SDIF\_Type \*base)

SDIF return the controller status.

#### Parameters

- base – SDIF peripheral base address.

static inline void SDIF\_SendCCSD(SDIF\_Type \*base, bool withAutoStop)

SDIF send command complete signal disable to CE-ATA card.

#### Parameters

- base – SDIF peripheral base address.
- withAutoStop – auto stop flag

void SDIF\_ConfigClockDelay(uint32\_t target\_HZ, uint32\_t divider)

SDIF config the clock delay This function is used to config the cclk\_in delay to sample and driver the data ,should meet the min setup time and hold time, and user need to config this parameter according to your board setting.

#### Parameters

- target\_HZ – freq work mode
- divider – not used in this function anymore, use DELAY value instead of phase directly.

SDIF\_CLOCK\_RANGE\_NEED\_DELAY

SDIOCLKCTRL setting Below clock delay setting should depend on specific platform, so it can be redefined when timing mismatch issue occur. Such as: response error/CRC error and so on.

clock range value which need to add delay to avoid timing issue

SDIF\_HIGHSPEED\_SAMPLE\_DELAY

High speed mode clk\_sample fixed delay.

12 \* 250ps = 3ns

SDIF\_HIGHSPEED\_DRV\_DELAY

High speed mode clk\_drv fixed delay.

31 \* 250ps = 7.75ns

SDIF\_HIGHSPEED\_SAMPLE\_PHASE\_SHIFT

High speed mode clk\_sample phase shift.

SDIF\_HIGHSPEED\_DRV\_PHASE\_SHIFT

High speed mode clk\_drv phase shift.

SDIF\_DEFAULT\_MODE\_SAMPLE\_DELAY

default mode sample fixed delay

12 \* 250ps = 3ns

SDIF\_DEFAULT\_MODE\_DRV\_DELAY

31 \* 250ps = 7.75ns

SDIF\_INTERNAL\_DMA\_ADDR\_ALIGN

SDIF internal DMA descriptor address and the data buffer address align.

SDIF\_DMA\_DESCRIPTOR\_DISABLE\_COMPLETE\_INT\_FLAG

SDIF DMA descriptor flag.

SDIF\_DMA\_DESCRIPTOR\_DATA\_BUFFER\_END\_FLAG

SDIF\_DMA\_DESCRIPTOR\_DATA\_BUFFER\_START\_FLAG

SDIF\_DMA\_DESCRIPTOR\_SECOND\_ADDR\_CHAIN\_FLAG

SDIF\_DMA\_DESCRIPTOR\_DESCRIPTOR\_END\_FLAG

SDIF\_DMA\_DESCRIPTOR\_OWN\_BY\_DMA\_FLAG

struct \_sdif\_dma\_descriptor

#include <fsl\_sdif.h> define the internal DMA descriptor

### Public Members

uint32\_t dmaDesAttribute

internal DMA attribute control and status

uint32\_t dmaDataBufferSize

internal DMA transfer buffer size control

const uint32\_t \*dmaDataBufferAddr0

internal DMA buffer 0 addr ,the buffer size must be 32bit aligned

const uint32\_t \*dmaDataBufferAddr1

internal DMA buffer 1 addr ,the buffer size must be 32bit aligned

struct \_sdif\_dma\_config

#include <fsl\_sdif.h> Defines the internal DMA configure structure.

### Public Members

bool enableFixBurstLen

fix burst len enable/disable flag,When set, the AHB will use only SINGLE, INCR4, INCR8 or INCR16 during start of normal burst transfers. When reset, the AHB will use SINGLE and INCR burst transfer operations

sdif\_dma\_mode\_t mode

define the DMA mode

uint8\_t dmaDesSkipLen

define the descriptor skip length ,the length between two descriptor this field is special for dual DMA mode

uint32\_t \*dmaDesBufferStartAddr

internal DMA descriptor start address

uint32\_t dmaDesBufferLen

internal DMA buffer descriptor buffer len ,user need to pay attention to the dma descriptor buffer length if it is bigger enough for your transfer

struct \_sdif\_data

#include <fsl\_sdif.h> Card data descriptor.

### Public Members

`bool streamTransfer`  
    indicate this is a stream data transfer command

`bool enableAutoCommand12`  
    indicate if auto stop will send when data transfer over

`bool enableIgnoreError`  
    indicate if enable ignore error when transfer data

`size_t blockSize`  
    Block size, take care when configure this parameter

`uint32_t blockCount`  
    Block count

`uint32_t *rxData`  
    data buffer to receive

`const uint32_t *txData`  
    data buffer to transfer

`struct __sdif_command`  
    *#include <fsl\_sdif.h>* Card command descriptor.  
    Define card command-related attribute.

### Public Members

`uint32_t index`  
    Command index

`uint32_t argument`  
    Command argument

`uint32_t response[4U]`  
    Response for this command

`uint32_t type`  
    define the command type

`uint32_t responseType`  
    Command response type

`uint32_t flags`  
    Cmd flags

`uint32_t responseErrorFlags`  
    response error flags, need to check the flags when receive the cmd response

`struct __sdif_transfer`  
    *#include <fsl\_sdif.h>* Transfer state.

### Public Members

`sdif_data_t *data`  
    Data to transfer

`sdif_command_t *command`  
    Command to send



struct \_\_sdif\_config  
#include <fsl\_sdif.h> Data structure to initialize the sdif.

#### Public Members

uint8\_t responseTimeout  
command response timeout value

uint32\_t cardDetDebounce\_Clock  
define the debounce clock count which will used in card detect logic, typical value is 5-25ms

uint32\_t dataTimeout  
data timeout value

struct \_\_sdif\_capability  
#include <fsl\_sdif.h> SDIF capability information. Defines a structure to get the capability information of SDIF.

#### Public Members

uint32\_t sdVersion  
support SD card/sdio version

uint32\_t mmcVersion  
support emmc card version

uint32\_t maxBlockLength  
Maximum block length united as byte

uint32\_t maxBlockCount  
Maximum byte count can be transfered

uint32\_t flags  
Capability flags to indicate the support information

struct \_\_sdif\_transfer\_callback  
#include <fsl\_sdif.h> sdif callback functions.

#### Public Members

void (\*cardInserted)(SDIF\_Type \*base, void \*userData)  
card insert call back

void (\*cardRemoved)(SDIF\_Type \*base, void \*userData)  
card remove call back

void (\*SDIOInterrupt)(SDIF\_Type \*base, void \*userData)  
SDIO card interrupt occurs

void (\*DMADesUnavailable)(SDIF\_Type \*base, void \*userData)  
DMA descriptor unavailable

void (\*CommandReload)(SDIF\_Type \*base, void \*userData)  
command buffer full, need re-load

void (\*TransferComplete)(SDIF\_Type \*base, void \*handle, status\_t status, void \*userData)  
Transfer complete callback

struct `_sdif_handle`

*#include <fsl\_sdif.h>* sdif handle

Defines the structure to save the sdif state information and callback function. The detail interrupt status when send command or transfer data can be obtained from `interruptFlags` field by using mask defined in `sdif_interrupt_flag_t`;

---

**Note:** All the fields except `interruptFlags` and `transferredWords` must be allocated by the user.

---

### Public Members

*sdif\_data\_t* \*volatile data

Data to transfer

*sdif\_command\_t* \*volatile command

Command to send

volatile uint32\_t transferredWords

Words transferred by polling way

*sdif\_transfer\_callback\_t* callback

Callback function

void \*userData

Parameter for transfer complete callback

struct `_sdif_host`

*#include <fsl\_sdif.h>* sdif host descriptor

### Public Members

SDIF\_Type \*base

sdif peripheral base address

uint32\_t sourceClock\_Hz

sdif source clock frequency united in Hz

*sdif\_config\_t* config

sdif configuration

*sdif\_transfer\_function\_t* transfer

sdif transfer function

*sdif\_capability\_t* capability

sdif capability information

## 2.42 SHA: SHA encryption decryption driver

FSL\_SHA\_DRIVER\_VERSION

Defines LPC SHA driver version 2.3.2.

Current version: 2.3.2

Change log:

- Version 2.0.0

- Initial version
- Version 2.1.0
  - Updated “sha\_ldm\_stm\_16\_words” “sha\_one\_block” API to match QN9090. QN9090 has no ALIAS register.
  - Added “SHA\_ClkInit” “SHA\_ClkInit”
- Version 2.1.1
  - MISRA C-2012 issue fixed: rule 10.3, 10.4, 11.9, 14.4, 16.4 and 17.7.
- Version 2.2.0
  - Support MEMADDR pseudo DMA for loading input data in SHA\_Update function (LPCXpresso54018 and LPCXpresso54628).
- Version 2.2.1
  - MISRA C-2012 issue fix.
- Version 2.2.2 Modified SHA\_Finish function. While using pseudo DMA with maximum optimization, compiler optimize out condition. Which caused block in this function and did not check state, which has been set in interrupt.
- Version 2.3.0 Modified SHA\_Update to use blocking version of AHB Master mode when its available on chip. Added SHA\_UpdateNonBlocking() function which uses nonblocking AHB Master mode. Fixed incorrect calculation of SHA when calling SHA\_Update multiple times when is CPU used to load data. Added Reset into SHA\_ClkInit and SHA\_ClkDeinit function.
- Version 2.3.1 Modified sha\_process\_message\_data\_master() to ensure that MEMCTRL will be written within 64 cycles of writing last word to INDATA as is mentioned in errata, even with different optimization levels.
- Version 2.3.2 Add -O2 optimization for GCC to sha\_process\_message\_data\_master(), because without it the function hangs under some conditions.

enum \_sha\_algo\_t

Supported cryptographic block cipher functions for HASH creation

*Values:*

enumerator kSHA\_Sha1

SHA\_1

enumerator kSHA\_Sha256

SHA\_256

typedef enum \_sha\_algo\_t sha\_algo\_t

Supported cryptographic block cipher functions for HASH creation

typedef struct \_sha\_ctx\_t sha\_ctx\_t

Storage type used to save hash context.

typedef void (\*sha\_callback\_t)(SHA\_Type \*base, sha\_ctx\_t \*ctx, status\_t status, void \*userData)

background hash callback function.

SHA\_CTX\_SIZE

SHA Context size.

struct \_sha\_ctx\_t

#include <fsl\_sha.h> Storage type used to save hash context.

## 2.43 Sha\_algorithm\_level\_api

*status\_t* SHA\_Init(*SHA\_Type* \*base, *sha\_ctx\_t* \*ctx, *sha\_algo\_t* algo)

Initialize HASH context.

This function initializes new hash context.

### Parameters

- base – SHA peripheral base address
- ctx – **[out]** Output hash context
- algo – Underlying algorithm to use for hash computation. Either SHA-1 or SHA-256.

### Returns

Status of initialization

*status\_t* SHA\_Update(*SHA\_Type* \*base, *sha\_ctx\_t* \*ctx, const uint8\_t \*message, size\_t messageSize)

Add data to current HASH.

Add data to current HASH. This can be called repeatedly with an arbitrary amount of data to be hashed.

### Parameters

- base – SHA peripheral base address
- ctx – **[inout]** HASH context
- message – Input message
- messageSize – Size of input message in bytes

### Returns

Status of the hash update operation

*status\_t* SHA\_Finish(*SHA\_Type* \*base, *sha\_ctx\_t* \*ctx, uint8\_t \*output, size\_t \*outputSize)

Finalize hashing.

Outputs the final hash and erases the context. SHA-1 or SHA-256 padding bits are automatically added by this function.

### Parameters

- base – SHA peripheral base address
- ctx – **[inout]** HASH context
- output – **[out]** Output hash data
- outputSize – **[inout]** On input, determines the size of bytes of the output array. On output, tells how many bytes have been written to output.

### Returns

Status of the hash finish operation

void SHA\_SetCallback(*SHA\_Type* \*base, *sha\_ctx\_t* \*ctx, *sha\_callback\_t* callback, void \*userData)

Initializes the SHA handle for background hashing.

This function initializes the hash context for background hashing (Non-blocking) APIs. This is less typical interface to hash function, but can be used for parallel processing, when main CPU has something else to do. Example is digital signature RSASSA-PKCS1-V1\_5-VERIFY((n,e),M,S) algorithm, where background hashing of M can be started, then CPU can compute  $S^e \bmod n$  (in parallel with background hashing) and once the digest becomes available, CPU can proceed to comparison of EM with EM'.

**Parameters**

- base – SHA peripheral base address.
- ctx – **[out]** Hash context.
- callback – Callback function.
- userData – User data (to be passed as an argument to callback function, once callback is invoked from isr).

*status\_t* SHA\_UpdateNonBlocking(*SHA\_Type* \*base, *sha\_ctx\_t* \*ctx, const *uint8\_t* \*input, *size\_t* inputSize)

Create running hash on given data.

Configures the SHA to compute new running hash as AHB master and returns immediately. SHA AHB Master mode supports only aligned input address and can be called only once per continuous block of data. Every call to this function must be preceded with SHA\_Init() and finished with SHA\_Finish(). Once callback function is invoked by SHA isr, it should set a flag for the main application to finalize the hashing (padding) and to read out the final digest by calling SHA\_Finish().

**Parameters**

- base – SHA peripheral base address
- ctx – Specifies callback. Last incomplete 512-bit block of the input is copied into clear buffer for padding.
- input – 32-bit word aligned pointer to Input data.
- inputSize – Size of input data in bytes (must be word aligned)

**Returns**

Status of the hash update operation.

*void* SHA\_ClkInit(*SHA\_Type* \*base)

Start SHA clock.

Start SHA clock

**Parameters**

- base – SHA peripheral base address

*void* SHA\_ClkDeinit(*SHA\_Type* \*base)

Stop SHA clock.

Stop SHA clock

**Parameters**

- base – SHA peripheral base address

## 2.44 SPI: Serial Peripheral Interface Driver

### 2.45 SPI DMA Driver

*status\_t* SPI\_MasterTransferCreateHandleDMA(*SPI\_Type* \*base, *spi\_dma\_handle\_t* \*handle, *spi\_dma\_callback\_t* callback, *void* \*userData, *dma\_handle\_t* \*txHandle, *dma\_handle\_t* \*rxHandle)

Initialize the SPI master DMA handle.

This function initializes the SPI master DMA handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, user need only call this API once to get the initialized handle.

#### Parameters

- base – SPI peripheral base address.
- handle – SPI handle pointer.
- callback – User callback function called at the end of a transfer.
- userData – User data for callback.
- txHandle – DMA handle pointer for SPI Tx, the handle shall be static allocated by users.
- rxHandle – DMA handle pointer for SPI Rx, the handle shall be static allocated by users.

```
status_t SPI_MasterTransferDMA(SPI_Type *base, spi_dma_handle_t *handle, spi_transfer_t *xfer)
```

Perform a non-blocking SPI transfer using DMA.

---

**Note:** This interface returned immediately after transfer initiates, users should call SPI\_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

---

#### Parameters

- base – SPI peripheral base address.
- handle – SPI DMA handle pointer.
- xfer – Pointer to dma transfer structure.

#### Return values

- kStatus\_Success – Successfully start a transfer.
- kStatus\_InvalidArgument – Input argument is invalid.
- kStatus\_SPI\_Busy – SPI is not idle, is running another transfer.

```
status_t SPI_MasterHalfDuplexTransferDMA(SPI_Type *base, spi_dma_handle_t *handle, spi_half_duplex_transfer_t *xfer)
```

Transfers a block of data using a DMA method.

This function using polling way to do the first half transimission and using DMA way to do the srcond half transimission, the transfer mechanism is half-duplex. When do the second half transimission, code will return right away. When all data is transferred, the callback function is called.

#### Parameters

- base – SPI base pointer
- handle – A pointer to the spi\_master\_dma\_handle\_t structure which stores the transfer state.
- xfer – A pointer to the spi\_half\_duplex\_transfer\_t structure.

#### Returns

status of status\_t.

```
static inline status_t SPI_SlaveTransferCreateHandleDMA(SPI_Type *base, spi_dma_handle_t
                                                         *handle, spi_dma_callback_t callback,
                                                         void *userData, dma_handle_t
                                                         *txHandle, dma_handle_t *rxHandle)
```

Initialize the SPI slave DMA handle.

This function initializes the SPI slave DMA handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, user need only call this API once to get the initialized handle.

#### Parameters

- base – SPI peripheral base address.
- handle – SPI handle pointer.
- callback – User callback function called at the end of a transfer.
- userData – User data for callback.
- txHandle – DMA handle pointer for SPI Tx, the handle shall be static allocated by users.
- rxHandle – DMA handle pointer for SPI Rx, the handle shall be static allocated by users.

```
static inline status_t SPI_SlaveTransferDMA(SPI_Type *base, spi_dma_handle_t *handle,
                                              spi_transfer_t *xfer)
```

Perform a non-blocking SPI transfer using DMA.

---

**Note:** This interface returned immediately after transfer initiates, users should call SPI\_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

---

#### Parameters

- base – SPI peripheral base address.
- handle – SPI DMA handle pointer.
- xfer – Pointer to dma transfer structure.

#### Return values

- kStatus\_Success – Successfully start a transfer.
- kStatus\_InvalidArgument – Input argument is invalid.
- kStatus\_SPI\_Busy – SPI is not idle, is running another transfer.

```
void SPI_MasterTransferAbortDMA(SPI_Type *base, spi_dma_handle_t *handle)
```

Abort a SPI transfer using DMA.

#### Parameters

- base – SPI peripheral base address.
- handle – SPI DMA handle pointer.

```
status_t SPI_MasterTransferGetCountDMA(SPI_Type *base, spi_dma_handle_t *handle, size_t
                                         *count)
```

Gets the master DMA transfered bytes.

This function gets the master DMA transfered bytes.

#### Parameters

- base – SPI peripheral base address.

- `handle` – A pointer to the `spi_dma_handle_t` structure which stores the transfer state.
- `count` – A number of bytes transferred by the non-blocking transaction.

**Returns**

status of `status_t`.

```
static inline void SPI_SlaveTransferAbortDMA(SPI_Type *base, spi_dma_handle_t *handle)
```

Abort a SPI transfer using DMA.

**Parameters**

- `base` – SPI peripheral base address.
- `handle` – SPI DMA handle pointer.

```
static inline status_t SPI_SlaveTransferGetCountDMA(SPI_Type *base, spi_dma_handle_t  
*handle, size_t *count)
```

Gets the slave DMA transferred bytes.

This function gets the slave DMA transferred bytes.

**Parameters**

- `base` – SPI peripheral base address.
- `handle` – A pointer to the `spi_dma_handle_t` structure which stores the transfer state.
- `count` – A number of bytes transferred by the non-blocking transaction.

**Returns**

status of `status_t`.

```
FSL_SPI_DMA_DRIVER_VERSION
```

SPI DMA driver version.

```
typedef struct _spi_dma_handle spi_dma_handle_t
```

```
typedef void (*spi_dma_callback_t)(SPI_Type *base, spi_dma_handle_t *handle, status_t status,  
void *userData)
```

SPI DMA callback called at the end of transfer.

```
struct _spi_dma_handle
```

`#include <fsl_spi_dma.h>` SPI DMA transfer handle, users should not touch the content of the handle.

**Public Members**

```
SPI_Type *base
```

SPI base address

```
volatile bool txInProgress
```

Send transfer finished

```
volatile bool rxInProgress
```

Receive transfer finished

```
uint8_t bytesPerFrame
```

Bytes in a frame for SPI transfer

```
uint8_t lastwordBytes
```

The Bytes of lastword for master



`uint16_t txDummy`  
The dummy data for TX.

`uint32_t lastword`  
The last word for master TX.

`dma_handle_t *txHandle`  
DMA handler for SPI send

`dma_handle_t *rxHandle`  
DMA handler for SPI receive

`spi_dma_callback_t callback`  
Callback for SPI DMA transfer

`void *userData`  
User Data for SPI DMA callback

`uint32_t state`  
Internal state of SPI DMA transfer

`size_t transferSize`  
Bytes need to be transfer

`uint32_t instance`  
Index of SPI instance

`const uint8_t *txNextData`  
The pointer of next time tx data

`size_t txRemainingBytes`  
lastwordBytes + txRemainingBytes is number of data to be send [in bytes]

`uint8_t *rxNextData`  
The pointer of next time rx data

`size_t rxRemainingBytes`  
Number of data to be received [in bytes]

`bool isSlave`  
SPI work in slave mode.

## 2.46 SPI Driver

`FSL_SPI_DRIVER_VERSION`  
SPI driver version.

`enum _spi_xfer_option`  
SPI transfer option.

*Values:*

`enumerator kSPI_FrameDelay`  
A delay may be inserted, defined in the DLY register.

`enumerator kSPI_FrameAssert`  
SSEL will be deasserted at the end of a transfer

enum `_spi_shift_direction`

SPI data shifter direction options.

*Values:*

enumerator `kSPI_MsbFirst`

Data transfers start with most significant bit.

enumerator `kSPI_LsbFirst`

Data transfers start with least significant bit.

enum `_spi_clock_polarity`

SPI clock polarity configuration.

*Values:*

enumerator `kSPI_ClockPolarityActiveHigh`

Active-high SPI clock (idles low).

enumerator `kSPI_ClockPolarityActiveLow`

Active-low SPI clock (idles high).

enum `_spi_clock_phase`

SPI clock phase configuration.

*Values:*

enumerator `kSPI_ClockPhaseFirstEdge`

First edge on SCK occurs at the middle of the first cycle of a data transfer.

enumerator `kSPI_ClockPhaseSecondEdge`

First edge on SCK occurs at the start of the first cycle of a data transfer.

enum `_spi_txfifo_watermark`

txFIFO watermark values

*Values:*

enumerator `kSPI_TxFifo0`

SPI tx watermark is empty

enumerator `kSPI_TxFifo1`

SPI tx watermark at 1 item

enumerator `kSPI_TxFifo2`

SPI tx watermark at 2 items

enumerator `kSPI_TxFifo3`

SPI tx watermark at 3 items

enumerator `kSPI_TxFifo4`

SPI tx watermark at 4 items

enumerator `kSPI_TxFifo5`

SPI tx watermark at 5 items

enumerator `kSPI_TxFifo6`

SPI tx watermark at 6 items

enumerator `kSPI_TxFifo7`

SPI tx watermark at 7 items

enum \_spi\_rxfifo\_watermark  
rxFIFO watermark values

*Values:*

enumerator kSPI\_RxFifo1  
SPI rx watermark at 1 item

enumerator kSPI\_RxFifo2  
SPI rx watermark at 2 items

enumerator kSPI\_RxFifo3  
SPI rx watermark at 3 items

enumerator kSPI\_RxFifo4  
SPI rx watermark at 4 items

enumerator kSPI\_RxFifo5  
SPI rx watermark at 5 items

enumerator kSPI\_RxFifo6  
SPI rx watermark at 6 items

enumerator kSPI\_RxFifo7  
SPI rx watermark at 7 items

enumerator kSPI\_RxFifo8  
SPI rx watermark at 8 items

enum \_spi\_data\_width  
Transfer data width.

*Values:*

enumerator kSPI\_Data4Bits  
4 bits data width

enumerator kSPI\_Data5Bits  
5 bits data width

enumerator kSPI\_Data6Bits  
6 bits data width

enumerator kSPI\_Data7Bits  
7 bits data width

enumerator kSPI\_Data8Bits  
8 bits data width

enumerator kSPI\_Data9Bits  
9 bits data width

enumerator kSPI\_Data10Bits  
10 bits data width

enumerator kSPI\_Data11Bits  
11 bits data width

enumerator kSPI\_Data12Bits  
12 bits data width

enumerator kSPI\_Data13Bits  
13 bits data width

enumerator kSPI\_Data14Bits

14 bits data width

enumerator kSPI\_Data15Bits

15 bits data width

enumerator kSPI\_Data16Bits

16 bits data width

enum \_spi\_ssel

Slave select.

*Values:*

enumerator kSPI\_Ssel0

Slave select 0

enumerator kSPI\_Ssel1

Slave select 1

enumerator kSPI\_Ssel2

Slave select 2

enumerator kSPI\_Ssel3

Slave select 3

enum \_spi\_spol

ssel polarity

*Values:*

enumerator kSPI\_Spol0ActiveHigh

enumerator kSPI\_Spol1ActiveHigh

enumerator kSPI\_Spol3ActiveHigh

enumerator kSPI\_SpolActiveAllHigh

enumerator kSPI\_SpolActiveAllLow

SPI transfer status.

*Values:*

enumerator kStatus\_SPI\_Busy

SPI bus is busy

enumerator kStatus\_SPI\_Idle

SPI is idle

enumerator kStatus\_SPI\_Error

SPI error

enumerator kStatus\_SPI\_BaudrateNotSupport

Baudrate is not support in current clock source

enumerator kStatus\_SPI\_Timeout

SPI timeout polling status flags.

enum \_spi\_interrupt\_enable

SPI interrupt sources.

*Values:*

enumerator kSPI\_RxLvlIrq  
Rx level interrupt

enumerator kSPI\_TxLvlIrq  
Tx level interrupt

enum *\_spi\_statusflags*  
SPI status flags.

*Values:*

enumerator kSPI\_TxEmptyFlag  
txFifo is empty

enumerator kSPI\_TxNotFullFlag  
txFifo is not full

enumerator kSPI\_RxNotEmptyFlag  
rxFIFO is not empty

enumerator kSPI\_RxFullFlag  
rxFIFO is full

typedef enum *\_spi\_xfer\_option* spi\_xfer\_option\_t  
SPI transfer option.

typedef enum *\_spi\_shift\_direction* spi\_shift\_direction\_t  
SPI data shifter direction options.

typedef enum *\_spi\_clock\_polarity* spi\_clock\_polarity\_t  
SPI clock polarity configuration.

typedef enum *\_spi\_clock\_phase* spi\_clock\_phase\_t  
SPI clock phase configuration.

typedef enum *\_spi\_txfifo\_watermark* spi\_txfifo\_watermark\_t  
txFIFO watermark values

typedef enum *\_spi\_rxfifo\_watermark* spi\_rxfifo\_watermark\_t  
rxFIFO watermark values

typedef enum *\_spi\_data\_width* spi\_data\_width\_t  
Transfer data width.

typedef enum *\_spi\_ssel* spi\_ssel\_t  
Slave select.

typedef enum *\_spi\_spol* spi\_spol\_t  
ssel polarity

typedef struct *\_spi\_delay\_config* spi\_delay\_config\_t  
SPI delay time configure structure. Note: The DLY register controls several programmable delays related to SPI signalling, it stands for how many SPI clock time will be inserted. The maximum value of these delay time is 15.

typedef struct *\_spi\_master\_config* spi\_master\_config\_t  
SPI master user configure structure.

typedef struct *\_spi\_slave\_config* spi\_slave\_config\_t  
SPI slave user configure structure.

typedef struct *\_spi\_transfer* spi\_transfer\_t  
SPI transfer structure.

```
typedef struct _spi_half_duplex_transfer spi_half_duplex_transfer_t
```

SPI half-duplex(master only) transfer structure.

```
typedef struct _spi_config spi_config_t
```

Internal configuration structure used in 'spi' and 'spi\_dma' driver.

```
typedef struct _spi_master_handle spi_master_handle_t
```

Master handle type.

```
typedef spi_master_handle_t spi_slave_handle_t
```

Slave handle type.

```
typedef void (*spi_master_callback_t)(SPI_Type *base, spi_master_handle_t *handle, status_t status, void *userData)
```

SPI master callback for finished transmit.

```
typedef void (*spi_slave_callback_t)(SPI_Type *base, spi_slave_handle_t *handle, status_t status, void *userData)
```

SPI slave callback for finished transmit.

```
typedef void (*flexcomm_spi_master_irq_handler_t)(SPI_Type *base, spi_master_handle_t *handle)
```

Typedef for master interrupt handler.

```
typedef void (*flexcomm_spi_slave_irq_handler_t)(SPI_Type *base, spi_slave_handle_t *handle)
```

Typedef for slave interrupt handler.

```
volatile uint8_t s_dummyData[]
```

SPI default SSEL COUNT.

Global variable for dummy data value setting.

```
SPI_DUMMYDATA
```

SPI dummy transfer data, the data is sent while txBuff is NULL.

```
SPI_RETRY_TIMES
```

Retry times for waiting flag.

```
SPI_DATA(n)
```

```
SPI_CTRLMASK
```

```
SPI_ASSERTNUM_SSEL(n)
```

```
SPI_DEASSERTNUM_SSEL(n)
```

```
SPI_DEASSERT_ALL
```

```
SPI_FIFOWR_FLAGS_MASK
```

```
SPI_FIFOTRIG_TXLVL_GET(base)
```

```
SPI_FIFOTRIG_RXLVL_GET(base)
```

```
struct _spi_delay_config
```

*#include <fsl\_spi.h>* SPI delay time configure structure. Note: The DLY register controls several programmable delays related to SPI signalling, it stands for how many SPI clock time will be inserted. The maxinum value of these delay time is 15.

**Public Members**

uint8\_t preDelay

Delay between SSEL assertion and the beginning of transfer.

uint8\_t postDelay

Delay between the end of transfer and SSEL deassertion.

uint8\_t frameDelay

Delay between frame to frame.

uint8\_t transferDelay

Delay between transfer to transfer.

struct \_\_spi\_master\_config

*#include <fsl\_spi.h>* SPI master user configure structure.

**Public Members**

bool enableLoopback

Enable loopback for test purpose

bool enableMaster

Enable SPI at initialization time

*spi\_clock\_polarity\_t* polarity

Clock polarity

*spi\_clock\_phase\_t* phase

Clock phase

*spi\_shift\_direction\_t* direction

MSB or LSB

uint32\_t baudRate\_Bps

Baud Rate for SPI in Hz

*spi\_data\_width\_t* dataWidth

Width of the data

*spi\_ssel\_t* sselNum

Slave select number

*spi\_spol\_t* sselPol

Configure active CS polarity

uint8\_t txWatermark

txFIFO watermark

uint8\_t rxWatermark

rxFIFO watermark

*spi\_delay\_config\_t* delayConfig

Delay configuration.

struct \_\_spi\_slave\_config

*#include <fsl\_spi.h>* SPI slave user configure structure.

**Public Members**

bool enableSlave  
    Enable SPI at initialization time

*spi\_clock\_polarity\_t* polarity  
    Clock polarity

*spi\_clock\_phase\_t* phase  
    Clock phase

*spi\_shift\_direction\_t* direction  
    MSB or LSB

*spi\_data\_width\_t* dataWidth  
    Width of the data

*spi\_spol\_t* sselPol  
    Configure active CS polarity

uint8\_t txWatermark  
    txFIFO watermark

uint8\_t rxWatermark  
    rxFIFO watermark

struct *\_spi\_transfer*  
    #include <fsl\_spi.h> SPI transfer structure.

**Public Members**

const uint8\_t \*txData  
    Send buffer

uint8\_t \*rxData  
    Receive buffer

uint32\_t configFlags  
    Additional option to control transfer, *spi\_xfer\_option\_t*.

size\_t dataSize  
    Transfer bytes

struct *\_spi\_half\_duplex\_transfer*  
    #include <fsl\_spi.h> SPI half-duplex(master only) transfer structure.

**Public Members**

const uint8\_t \*txData  
    Send buffer

uint8\_t \*rxData  
    Receive buffer

size\_t txDataSize  
    Transfer bytes for transmit

size\_t rxDataSize  
    Transfer bytes



uint32\_t configFlags

Transfer configuration flags, spi\_xfer\_option\_t.

bool isPcsAssertInTransfer

If PCS pin keep assert between transmit and receive. true for assert and false for de-assert.

bool isTransmitFirst

True for transmit first and false for receive first.

struct \_\_spi\_config

*#include <fsl\_spi.h>* Internal configuration structure used in 'spi' and 'spi\_dma' driver.

struct \_\_spi\_master\_handle

*#include <fsl\_spi.h>* SPI transfer handle structure.

### Public Members

const uint8\_t \*volatile txData

Transfer buffer

uint8\_t \*volatile rxData

Receive buffer

volatile size\_t txRemainingBytes

Number of data to be transmitted [in bytes]

volatile size\_t rxRemainingBytes

Number of data to be received [in bytes]

volatile int8\_t toReceiveCount

The number of data expected to receive in data width. Since the received count and sent count should be the same to complete the transfer, if the sent count is x and the received count is y, toReceiveCount is x-y.

size\_t totalByteCount

A number of transfer bytes

volatile uint32\_t state

SPI internal state

spi\_master\_callback\_t callback

SPI callback

void \*userData

Callback parameter

uint8\_t dataWidth

Width of the data [Valid values: 1 to 16]

uint8\_t sselNum

Slave select number to be asserted when transferring data [Valid values: 0 to 3]

uint32\_t configFlags

Additional option to control transfer

uint8\_t txWatermark

txFIFO watermark

uint8\_t rxWatermark

rxFIFO watermark

## 2.47 SPIFI: SPIFI flash interface driver

```
void SPIFI_TransferTxCreateHandleDMA(SPIFI_Type *base, spifi_dma_handle_t *handle,  
                                     spifi_dma_callback_t callback, void *userData,  
                                     dma_handle_t *dmaHandle)
```

Initializes the SPIFI handle for send which is used in transactional functions and set the callback.

### Parameters

- base – SPIFI peripheral base address
- handle – Pointer to `spifi_dma_handle_t` structure
- callback – SPIFI callback, NULL means no callback.
- userData – User callback function data.
- dmaHandle – User requested DMA handle for DMA transfer

```
void SPIFI_TransferRxCreateHandleDMA(SPIFI_Type *base, spifi_dma_handle_t *handle,  
                                     spifi_dma_callback_t callback, void *userData,  
                                     dma_handle_t *dmaHandle)
```

Initializes the SPIFI handle for receive which is used in transactional functions and set the callback.

### Parameters

- base – SPIFI peripheral base address
- handle – Pointer to `spifi_dma_handle_t` structure
- callback – SPIFI callback, NULL means no callback.
- userData – User callback function data.
- dmaHandle – User requested DMA handle for DMA transfer

```
status_t SPIFI_TransferSendDMA(SPIFI_Type *base, spifi_dma_handle_t *handle, spifi_transfer_t  
                               *xfer)
```

Transfers SPIFI data using an DMA non-blocking method.

This function writes data to the SPIFI transmit FIFO. This function is non-blocking.

### Parameters

- base – Pointer to QuadSPI Type.
- handle – Pointer to `spifi_dma_handle_t` structure
- xfer – SPIFI transfer structure.

```
status_t SPIFI_TransferReceiveDMA(SPIFI_Type *base, spifi_dma_handle_t *handle,  
                                   spifi_transfer_t *xfer)
```

Receives data using an DMA non-blocking method.

This function receive data from the SPIFI receive buffer/FIFO. This function is non-blocking.

### Parameters

- base – Pointer to QuadSPI Type.
- handle – Pointer to `spifi_dma_handle_t` structure
- xfer – SPIFI transfer structure.

`void SPIFI_TransferAbortSendDMA(SPIFI_Type *base, spifi_dma_handle_t *handle)`

Aborts the sent data using DMA.

This function aborts the sent data using DMA.

**Parameters**

- base – SPIFI peripheral base address.
- handle – Pointer to `spifi_dma_handle_t` structure

`void SPIFI_TransferAbortReceiveDMA(SPIFI_Type *base, spifi_dma_handle_t *handle)`

Aborts the receive data using DMA.

This function abort receive data which using DMA.

**Parameters**

- base – SPIFI peripheral base address.
- handle – Pointer to `spifi_dma_handle_t` structure

`status_t SPIFI_TransferGetSendCountDMA(SPIFI_Type *base, spifi_dma_handle_t *handle, size_t *count)`

Gets the transferred counts of send.

**Parameters**

- base – Pointer to QuadSPI Type.
- handle – Pointer to `spifi_dma_handle_t` structure.
- count – Bytes sent.

**Return values**

- `kStatus_Success` – Succeed get the transfer count.
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

`status_t SPIFI_TransferGetReceiveCountDMA(SPIFI_Type *base, spifi_dma_handle_t *handle, size_t *count)`

Gets the status of the receive transfer.

**Parameters**

- base – Pointer to QuadSPI Type.
- handle – Pointer to `spifi_dma_handle_t` structure
- count – Bytes received.

**Return values**

- `kStatus_Success` – Succeed get the transfer count.
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

`uint32_t SPIFI_GetInstance(SPIFI_Type *base)`

Get the SPIFI instance from peripheral base address.

**Parameters**

- base – SPIFI peripheral base address.

**Returns**

SPIFI instance.

`void SPIFI_Init(SPIFI_Type *base, const spifi_config_t *config)`

Initializes the SPIFI with the user configuration structure.

This function configures the SPIFI module with the user-defined configuration.

**Parameters**

- `base` – SPIFI peripheral base address.
- `config` – The pointer to the configuration structure.

`void SPIFI_GetDefaultConfig(spifi_config_t *config)`

Get SPIFI default configure settings.

**Parameters**

- `config` – SPIFI config structure pointer.

`void SPIFI_Deinit(SPIFI_Type *base)`

Deinitializes the SPIFI regions.

**Parameters**

- `base` – SPIFI peripheral base address.

`void SPIFI_SetCommand(SPIFI_Type *base, spifi_command_t *cmd)`

Set SPIFI flash command.

**Parameters**

- `base` – SPIFI peripheral base address.
- `cmd` – SPIFI command structure pointer.

`static inline void SPIFI_SetCommandAddress(SPIFI_Type *base, uint32_t addr)`

Set SPIFI command address.

**Parameters**

- `base` – SPIFI peripheral base address.
- `addr` – Address value for the command.

`static inline void SPIFI_SetIntermediateData(SPIFI_Type *base, uint32_t val)`

Set SPIFI intermediate data.

Before writing a command which needs specific intermediate value, users shall call this function to write it. The main use of this function for current serial flash is to select no-opcode mode and cancelling this mode. As dummy cycle do not care about the value, no need to call this function.

**Parameters**

- `base` – SPIFI peripheral base address.
- `val` – Intermediate data.

`static inline void SPIFI_SetCacheLimit(SPIFI_Type *base, uint32_t val)`

Set SPIFI Cache limit value.

SPIFI includes caching of previously-accessed data to improve performance. Software can write an address to this function, to prevent such caching at and above the address.

**Parameters**

- `base` – SPIFI peripheral base address.
- `val` – Zero-based upper limit of cacheable memory.

```
static inline void SPIFI_ResetCommand(SPIFI_Type *base)
```

Reset the command field of SPIFI.

This function is used to abort the current command or memory mode.

**Parameters**

- `base` – SPIFI peripheral base address.

```
void SPIFI_SetMemoryCommand(SPIFI_Type *base, spifi_command_t *cmd)
```

Set SPIFI flash AHB read command.

Call this function means SPIFI enters to memory mode, while users need to use command, a `SPIFI_ResetCommand` shall be called.

**Parameters**

- `base` – SPIFI peripheral base address.
- `cmd` – SPIFI command structure pointer.

```
static inline void SPIFI_EnableInterrupt(SPIFI_Type *base, uint32_t mask)
```

Enable SPIFI interrupt.

The interrupt is triggered only in command mode, and it means the command now is finished.

**Parameters**

- `base` – SPIFI peripheral base address.
- `mask` – SPIFI interrupt enable mask. It is a logic OR of members the enumeration :: `spifi_interrupt_enable_t`

```
static inline void SPIFI_DisableInterrupt(SPIFI_Type *base, uint32_t mask)
```

Disable SPIFI interrupt.

The interrupt is triggered only in command mode, and it means the command now is finished.

**Parameters**

- `base` – SPIFI peripheral base address.
- `mask` – SPIFI interrupt enable mask. It is a logic OR of members the enumeration :: `spifi_interrupt_enable_t`

```
static inline uint32_t SPIFI_GetStatusFlag(SPIFI_Type *base)
```

Get the status of all interrupt flags for SPIFI.

**Parameters**

- `base` – SPIFI peripheral base address.

**Returns**

SPIFI flag status

FSL\_SPIFI\_DMA\_DRIVER\_VERSION

SPIFI DMA driver version 2.0.3.

FSL\_SPIFI\_DRIVER\_VERSION

SPIFI driver version 2.0.3.

Status structure of SPIFI.

*Values:*

enumerator kStatus\_SPIFI\_Idle

SPIFI is in idle state

enumerator kStatus\_SPIFI\_Busy

SPIFI is busy

enumerator kStatus\_SPIFI\_Error

Error occurred during SPIFI transfer

enum \_spifi\_interrupt\_enable

SPIFI interrupt source.

*Values:*

enumerator kSPIFI\_CommandFinishInterruptEnable

Interrupt while command finished

enum \_spifi\_spi\_mode

SPIFI SPI mode select.

*Values:*

enumerator kSPIFI\_SPISckLow

SCK low after last bit of command, keeps low while CS high

enumerator kSPIFI\_SPISckHigh

SCK high after last bit of command and while CS high

enum \_spifi\_dual\_mode

SPIFI dual mode select.

*Values:*

enumerator kSPIFI\_QuadMode

SPIFI uses IO3:0

enumerator kSPIFI\_DualMode

SPIFI uses IO1:0

enum \_spifi\_data\_direction

SPIFI data direction.

*Values:*

enumerator kSPIFI\_DataInput

Data input from serial flash.

enumerator kSPIFI\_DataOutput

Data output to serial flash.

enum \_spifi\_command\_format

SPIFI command opcode format.

*Values:*

enumerator kSPIFI\_CommandAllSerial

All fields of command are serial.

enumerator kSPIFI\_CommandDataQuad

Only data field is dual/quad, others are serial.

enumerator kSPIFI\_CommandOpcodeSerial

Only opcode field is serial, others are quad/dual.

enumerator kSPIFI\_CommandAllQuad

All fields of command are dual/quad mode.

enum *\_spifi\_command\_type*

SPIFI command type.

*Values:*

enumerator kSPIFI\_CommandOpcodeOnly

Command only have opcode, no address field

enumerator kSPIFI\_CommandOpcodeAddrOneByte

Command have opcode and also one byte address field

enumerator kSPIFI\_CommandOpcodeAddrTwoBytes

Command have opcode and also two bytes address field

enumerator kSPIFI\_CommandOpcodeAddrThreeBytes

Command have opcode and also three bytes address field.

enumerator kSPIFI\_CommandOpcodeAddrFourBytes

Command have opcode and also four bytes address field

enumerator kSPIFI\_CommandNoOpcodeAddrThreeBytes

Command have no opcode and three bytes address field

enumerator kSPIFI\_CommandNoOpcodeAddrFourBytes

Command have no opcode and four bytes address field

SPIFI status flags.

*Values:*

enumerator kSPIFI\_MemoryCommandWriteFinished

Memory command write finished

enumerator kSPIFI\_CommandWriteFinished

Command write finished

enumerator kSPIFI\_InterruptRequest

CMD flag from 1 to 0, means command execute finished

typedef struct *\_spifi\_dma\_handle* spifi\_dma\_handle\_t

typedef void (\*spifi\_dma\_callback\_t)(SPIFI\_Type \*base, *spifi\_dma\_handle\_t* \*handle, *status\_t* status, void \*userData)

SPIFI DMA transfer callback function for finish and error.

typedef enum *\_spifi\_interrupt\_enable* spifi\_interrupt\_enable\_t

SPIFI interrupt source.

typedef enum *\_spifi\_spi\_mode* spifi\_spi\_mode\_t

SPIFI SPI mode select.

typedef enum *\_spifi\_dual\_mode* spifi\_dual\_mode\_t

SPIFI dual mode select.

typedef enum *\_spifi\_data\_direction* spifi\_data\_direction\_t

SPIFI data direction.

typedef enum *\_spifi\_command\_format* spifi\_command\_format\_t

SPIFI command opcode format.

```
typedef enum _spifi_command_type spifi_command_type_t
```

SPIFI command type.

```
typedef struct _spifi_command spifi_command_t
```

SPIFI command structure.

```
typedef struct _spifi_config spifi_config_t
```

SPIFI region configuration structure.

```
typedef struct _spifi_transfer spifi_transfer_t
```

Transfer structure for SPIFI.

```
static inline void SPIFI_EnableDMA(SPIFI_Type *base, bool enable)
```

Enable or disable DMA request for SPIFI.

#### Parameters

- base – SPIFI peripheral base address.
- enable – True means enable DMA and false means disable DMA.

```
static inline uint32_t SPIFI_GetDataRegisterAddress(SPIFI_Type *base)
```

Gets the SPIFI data register address.

This API is used to provide a transfer address for the SPIFI DMA transfer configuration.

#### Parameters

- base – SPIFI base pointer

#### Returns

data register address

```
static inline void SPIFI_WriteData(SPIFI_Type *base, uint32_t data)
```

Write a word data in address of SPIFI.

Users can write a page or at least a word data into SPIFI address.

#### Parameters

- base – SPIFI peripheral base address.
- data – Data need be write.

```
static inline void SPIFI_WriteDataByte(SPIFI_Type *base, uint8_t data)
```

Write a byte data in address of SPIFI.

Users can write a byte data into SPIFI address.

#### Parameters

- base – SPIFI peripheral base address.
- data – Data need be write.

```
void SPIFI_WriteDataHalfword(SPIFI_Type *base, uint16_t data)
```

Write a halfword data in address of SPIFI.

Users can write a halfword data into SPIFI address.

#### Parameters

- base – SPIFI peripheral base address.
- data – Data need be write.



```
static inline uint32_t SPIFI_ReadData(SPIFI_Type *base)
```

Read data from serial flash.

Users should notice before call this function, the data length field in command register shall larger than 4, otherwise a hardfault will happen.

#### Parameters

- base – SPIFI peripheral base address.

#### Returns

Data input from flash.

```
static inline uint8_t SPIFI_ReadDataByte(SPIFI_Type *base)
```

Read a byte data from serial flash.

#### Parameters

- base – SPIFI peripheral base address.

#### Returns

Data input from flash.

```
uint16_t SPIFI_ReadDataHalfword(SPIFI_Type *base)
```

Read a halfword data from serial flash.

#### Parameters

- base – SPIFI peripheral base address.

#### Returns

Data input from flash.

```
struct _spifi_dma_handle
```

*#include <fsl\_spifi\_dma.h>* SPIFI DMA transfer handle, users should not touch the content of the handle.

### Public Members

*dma\_handle\_t* \*dmaHandle

DMA handler for SPIFI send

*size\_t* transferSize

Bytes need to transfer.

*uint32\_t* state

Internal state for SPIFI DMA transfer

*spifi\_dma\_callback\_t* callback

Callback for users while transfer finish or error occurred

*void* \*userData

User callback parameter

```
struct _spifi_command
```

*#include <fsl\_spifi.h>* SPIFI command structure.

### Public Members

*uint16\_t* dataLen

How many data bytes are needed in this command.

bool isPollMode

For command need to read data from serial flash

*spifi\_data\_direction\_t* direction

Data direction of this command.

uint8\_t intermediateBytes

How many intermediate bytes needed

*spifi\_command\_format\_t* format

Command format

*spifi\_command\_type\_t* type

Command type

uint8\_t opcode

Command opcode value

struct \_\_spifi\_config

*#include <fsl\_spifi.h>* SPIFI region configuration structure.

### Public Members

uint16\_t timeout

SPI transfer timeout, the unit is SCK cycles

uint8\_t csHighTime

CS high time cycles

bool disablePrefetch

True means SPIFI will not attempt a speculative prefetch.

bool disableCachePrefech

Disable prefetch of cache line

bool isFeedbackClock

Is data sample uses feedback clock.

*spifi\_spi\_mode\_t* spiMode

SPIFI spi mode select

bool isReadFullClockCycle

If enable read full clock cycle.

*spifi\_dual\_mode\_t* dualMode

SPIFI dual mode, dual or quad.

struct \_\_spifi\_transfer

*#include <fsl\_spifi.h>* Transfer structure for SPIFI.

### Public Members

uint8\_t \*data

Pointer to data to transmit

size\_t dataSize

Bytes to be transmit

## 2.48 SPIFI DMA Driver

## 2.49 SPIFI Driver

## 2.50 USART: Universal Synchronous/Asynchronous Receiver/Transmitter Driver

## 2.51 USART DMA Driver

```
status_t USART_TransferCreateHandleDMA(USART_Type *base, usart_dma_handle_t *handle,  
                                       usart_dma_transfer_callback_t callback, void  
                                       *userData, dma_handle_t *txDmaHandle,  
                                       dma_handle_t *rxDmaHandle)
```

Initializes the USART handle which is used in transactional functions.

### Parameters

- base – USART peripheral base address.
- handle – Pointer to `usart_dma_handle_t` structure.
- callback – Callback function.
- userData – User data.
- txDmaHandle – User-requested DMA handle for TX DMA transfer.
- rxDmaHandle – User-requested DMA handle for RX DMA transfer.

```
status_t USART_TransferSendDMA(USART_Type *base, usart_dma_handle_t *handle,  
                               usart_transfer_t *xfer)
```

Sends data using DMA.

This function sends data using DMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

### Parameters

- base – USART peripheral base address.
- handle – USART handle pointer.
- xfer – USART DMA transfer structure. See `usart_transfer_t`.

### Return values

- `kStatus_Success` – if succeed, others failed.
- `kStatus_USART_TxBusy` – Previous transfer on going.
- `kStatus_InvalidArgument` – Invalid argument.

```
status_t USART_TransferReceiveDMA(USART_Type *base, usart_dma_handle_t *handle,  
                                  usart_transfer_t *xfer)
```

Receives data using DMA.

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

### Parameters

- base – USART peripheral base address.
- handle – Pointer to `usart_dma_handle_t` structure.

- `xfer` – USART DMA transfer structure. See `usart_transfer_t`.

**Return values**

- `kStatus_Success` – if succeed, others failed.
- `kStatus_USART_RxBusy` – Previous transfer on going.
- `kStatus_InvalidArgument` – Invalid argument.

`void USART_TransferAbortSendDMA(USART_Type *base, usart_dma_handle_t *handle)`

Aborts the sent data using DMA.

This function aborts send data using DMA.

**Parameters**

- `base` – USART peripheral base address
- `handle` – Pointer to `usart_dma_handle_t` structure

`void USART_TransferAbortReceiveDMA(USART_Type *base, usart_dma_handle_t *handle)`

Aborts the received data using DMA.

This function aborts the received data using DMA.

**Parameters**

- `base` – USART peripheral base address
- `handle` – Pointer to `usart_dma_handle_t` structure

`status_t USART_TransferGetReceiveCountDMA(USART_Type *base, usart_dma_handle_t *handle, uint32_t *count)`

Get the number of bytes that have been received.

This function gets the number of bytes that have been received.

**Parameters**

- `base` – USART peripheral base address.
- `handle` – USART handle pointer.
- `count` – Receive bytes count.

**Return values**

- `kStatus_NoTransferInProgress` – No receive in progress.
- `kStatus_InvalidArgument` – Parameter is invalid.
- `kStatus_Success` – Get successfully through the parameter `count`;

`status_t USART_TransferGetSendCountDMA(USART_Type *base, usart_dma_handle_t *handle, uint32_t *count)`

Get the number of bytes that have been sent.

This function gets the number of bytes that have been sent.

**Parameters**

- `base` – USART peripheral base address.
- `handle` – USART handle pointer.
- `count` – Sent bytes count.

**Return values**

- `kStatus_NoTransferInProgress` – No receive in progress.
- `kStatus_InvalidArgument` – Parameter is invalid.

- `kStatus_Success` – Get successfully through the parameter `count`;

`FSL_USART_DMA_DRIVER_VERSION`

USART dma driver version.

`typedef struct _usart_dma_handle usart_dma_handle_t`

`typedef void (*usart_dma_transfer_callback_t)(USART_Type *base, usart_dma_handle_t *handle, status_t status, void *userData)`

UART transfer callback function.

`struct _usart_dma_handle`

`#include <fsl_usart_dma.h>` UART DMA handle.

### Public Members

`USART_Type *base`

UART peripheral base address.

`usart_dma_transfer_callback_t` callback

Callback function.

`void *userData`

UART callback function parameter.

`size_t rxDataSizeAll`

Size of the data to receive.

`size_t txDataSizeAll`

Size of the data to send out.

`dma_handle_t *txDmaHandle`

The DMA TX channel used.

`dma_handle_t *rxDmaHandle`

The DMA RX channel used.

`volatile uint8_t txState`

TX transfer state.

`volatile uint8_t rxState`

RX transfer state

## 2.52 USART Driver

`status_t USART_Init(USART_Type *base, const usart_config_t *config, uint32_t srcClock_Hz)`

Initializes a USART instance with user configuration structure and peripheral clock.

This function configures the USART module with the user-defined settings. The user can configure the configuration structure and also get the default configuration by using the `USART_GetDefaultConfig()` function. Example below shows how to use this API to configure USART.

```
usart_config_t usartConfig;
usartConfig.baudRate_Bps = 115200U;
usartConfig.parityMode = kUSART_ParityDisabled;
usartConfig.stopBitCount = kUSART_OneStopBit;
USART_Init(USART1, &usartConfig, 20000000U);
```

**Parameters**

- base – USART peripheral base address.
- config – Pointer to user-defined configuration structure.
- srcClock\_Hz – USART clock source frequency in HZ.

**Return values**

- kStatus\_USART\_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus\_InvalidArgument – USART base address is not valid
- kStatus\_Success – Status USART initialize succeed

`void USART_Deinit(USART_Type *base)`

Deinitializes a USART instance.

This function waits for TX complete, disables TX and RX, and disables the USART clock.

**Parameters**

- base – USART peripheral base address.

`void USART_GetDefaultConfig(usart_config_t *config)`

Gets the default configuration structure.

This function initializes the USART configuration structure to a default value. The default values are: `usartConfig->baudRate_Bps = 115200U`; `usartConfig->parityMode = kUSART_ParityDisabled`; `usartConfig->stopBitCount = kUSART_OneStopBit`; `usartConfig->bitCountPerChar = kUSART_8BitsPerChar`; `usartConfig->loopback = false`; `usartConfig->enableTx = false`; `usartConfig->enableRx = false`;

**Parameters**

- config – Pointer to configuration structure.

`status_t USART_SetBaudRate(USART_Type *base, uint32_t baudrate_Bps, uint32_t srcClock_Hz)`

Sets the USART instance baud rate.

This function configures the USART module baud rate. This function is used to update the USART module baud rate after the USART module is initialized by the `USART_Init`.

```
USART_SetBaudRate(USART1, 115200U, 20000000U);
```

**Parameters**

- base – USART peripheral base address.
- baudrate\_Bps – USART baudrate to be set.
- srcClock\_Hz – USART clock source frequency in HZ.

**Return values**

- kStatus\_USART\_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus\_Success – Set baudrate succeed.
- kStatus\_InvalidArgument – One or more arguments are invalid.

`status_t USART_Enable32kMode(USART_Type *base, uint32_t baudRate_Bps, bool enableMode32k, uint32_t srcClock_Hz)`

Enable 32 kHz mode which USART uses clock from the RTC oscillator as the clock source.

Please note that in order to use a 32 kHz clock to operate USART properly, the RTC oscillator and its 32 kHz output must be manually enabled by user, by calling `RTC_Init` and setting

SYSCON\_RTCOSCCTRL\_EN bit to 1. And in 32kHz clocking mode the USART can only work at 9600 baudrate or at the baudrate that 9600 can evenly divide, eg: 4800, 3200.

#### Parameters

- base – USART peripheral base address.
- baudRate\_Bps – USART baudrate to be set..
- enableMode32k – true is 32k mode, false is normal mode.
- srcClock\_Hz – USART clock source frequency in HZ.

#### Return values

- kStatus\_USART\_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus\_Success – Set baudrate succeed.
- kStatus\_InvalidArgument – One or more arguments are invalid.

void USART\_Enable9bitMode(USART\_Type \*base, bool enable)

Enable 9-bit data mode for USART.

This function set the 9-bit mode for USART module. The 9th bit is not used for parity thus can be modified by user.

#### Parameters

- base – USART peripheral base address.
- enable – true to enable, false to disable.

static inline void USART\_SetMatchAddress(USART\_Type \*base, uint8\_t address)

Set the USART slave address.

This function configures the address for USART module that works as slave in 9-bit data mode. When the address detection is enabled, the frame it receives with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

---

**Note:** Any USART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

---

#### Parameters

- base – USART peripheral base address.
- address – USART slave address.

static inline void USART\_EnableMatchAddress(USART\_Type \*base, bool match)

Enable the USART match address feature.

#### Parameters

- base – USART peripheral base address.
- match – true to enable match address, false to disable.

```
static inline uint32_t USART_GetStatusFlags(USART_Type *base)
```

Get USART status flags.

This function get all USART status flags, the flags are returned as the logical OR value of the enumerators `_usart_flags`. To check a specific status, compare the return value with enumerators in `_usart_flags`. For example, to check whether the TX is empty:

```
if (kUSART_TxFifoNotFullFlag & USART_GetStatusFlags(USART1))
{
    ...
}
```

#### Parameters

- `base` – USART peripheral base address.

#### Returns

USART status flags which are ORed by the enumerators in the `_usart_flags`.

```
static inline void USART_ClearStatusFlags(USART_Type *base, uint32_t mask)
```

Clear USART status flags.

This function clear supported USART status flags. The mask is a logical OR of enumeration members. See `kUSART_AllClearFlags`. For example:

```
USART_ClearStatusFlags(USART1, kUSART_TxError | kUSART_RxError)
```

#### Parameters

- `base` – USART peripheral base address.
- `mask` – status flags to be cleared.

```
static inline void USART_EnableInterrupts(USART_Type *base, uint32_t mask)
```

Enables USART interrupts according to the provided mask.

This function enables the USART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See `_usart_interrupt_enable`. For example, to enable TX empty interrupt and RX full interrupt:

```
USART_EnableInterrupts(USART1, kUSART_TxLevelInterruptEnable | kUSART_
↳ RxLevelInterruptEnable);
```

#### Parameters

- `base` – USART peripheral base address.
- `mask` – The interrupts to enable. Logical OR of `_usart_interrupt_enable`.

```
static inline void USART_DisableInterrupts(USART_Type *base, uint32_t mask)
```

Disables USART interrupts according to a provided mask.

This function disables the USART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See `_usart_interrupt_enable`. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
USART_DisableInterrupts(USART1, kUSART_TxLevelInterruptEnable | kUSART_
↳ RxLevelInterruptEnable);
```

#### Parameters

- `base` – USART peripheral base address.
- `mask` – The interrupts to disable. Logical OR of `_usart_interrupt_enable`.



```
static inline uint32_t USART_GetEnabledInterrupts(USART_Type *base)
```

Returns enabled USART interrupts.

This function returns the enabled USART interrupts.

**Parameters**

- base – USART peripheral base address.

```
static inline void USART_EnableTxDMA(USART_Type *base, bool enable)
```

Enable DMA for Tx.

```
static inline void USART_EnableRxDMA(USART_Type *base, bool enable)
```

Enable DMA for Rx.

```
static inline void USART_EnableCTS(USART_Type *base, bool enable)
```

Enable CTS. This function will determine whether CTS is used for flow control.

**Parameters**

- base – USART peripheral base address.
- enable – Enable CTS or not, true for enable and false for disable.

```
static inline void USART_EnableContinuousSCLK(USART_Type *base, bool enable)
```

Continuous Clock generation. By default, SCLK is only output while data is being transmitted in synchronous mode. Enable this function, SCLK will run continuously in synchronous mode, allowing characters to be received on Un\_RxD independently from transmission on Un\_TXD).

**Parameters**

- base – USART peripheral base address.
- enable – Enable Continuous Clock generation mode or not, true for enable and false for disable.

```
static inline void USART_EnableAutoClearSCLK(USART_Type *base, bool enable)
```

Enable Continuous Clock generation bit auto clear. While enable this function, the Continuous Clock bit is automatically cleared when a complete character has been received. This bit is cleared at the same time.

**Parameters**

- base – USART peripheral base address.
- enable – Enable auto clear or not, true for enable and false for disable.

```
static inline void USART_SetRxFifoWatermark(USART_Type *base, uint8_t water)
```

Sets the rx FIFO watermark.

**Parameters**

- base – USART peripheral base address.
- water – Rx FIFO watermark.

```
static inline void USART_SetTxFifoWatermark(USART_Type *base, uint8_t water)
```

Sets the tx FIFO watermark.

**Parameters**

- base – USART peripheral base address.
- water – Tx FIFO watermark.

static inline void USART\_WriteByte(USART\_Type \*base, uint8\_t data)

Writes to the FIFOWR register.

This function writes data to the txFIFO directly. The upper layer must ensure that txFIFO has space for data to write before calling this function.

**Parameters**

- base – USART peripheral base address.
- data – The byte to write.

static inline uint8\_t USART\_ReadByte(USART\_Type \*base)

Reads the FIFORD register directly.

This function reads data from the rxFIFO directly. The upper layer must ensure that the rxFIFO is not empty before calling this function.

**Parameters**

- base – USART peripheral base address.

**Returns**

The byte read from USART data register.

static inline uint8\_t USART\_GetRxFifoCount(USART\_Type \*base)

Gets the rx FIFO data count.

**Parameters**

- base – USART peripheral base address.

**Returns**

rx FIFO data count.

static inline uint8\_t USART\_GetTxFifoCount(USART\_Type \*base)

Gets the tx FIFO data count.

**Parameters**

- base – USART peripheral base address.

**Returns**

tx FIFO data count.

void USART\_SendAddress(USART\_Type \*base, uint8\_t address)

Transmit an address frame in 9-bit data mode.

**Parameters**

- base – USART peripheral base address.
- address – USART slave address.

status\_t USART\_WriteBlocking(USART\_Type \*base, const uint8\_t \*data, size\_t length)

Writes to the TX register using a blocking method.

This function polls the TX register, waits for the TX register to be empty or for the TX FIFO to have room and writes data to the TX buffer.

**Parameters**

- base – USART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

**Return values**

- kStatus\_USART\_Timeout – Transmission timed out and was aborted.

- `kStatus_InvalidArgument` – Invalid argument.
- `kStatus_Success` – Successfully wrote all data.

`status_t` `USART_ReadBlocking(USART_Type *base, uint8_t *data, size_t length)`

Read RX data register using a blocking method.

This function polls the RX register, waits for the RX register to be full or for RX FIFO to have data and read data from the TX register.

#### Parameters

- `base` – USART peripheral base address.
- `data` – Start address of the buffer to store the received data.
- `length` – Size of the buffer.

#### Return values

- `kStatus_USART_FramingError` – Receiver overrun happened while receiving data.
- `kStatus_USART_ParityError` – Noise error happened while receiving data.
- `kStatus_USART_NoiseError` – Framing error happened while receiving data.
- `kStatus_USART_RxError` – Overflow or underflow rxFIFO happened.
- `kStatus_USART_Timeout` – Transmission timed out and was aborted.
- `kStatus_Success` – Successfully received all data.

`status_t` `USART_TransferCreateHandle(USART_Type *base, usart_handle_t *handle, usart_transfer_callback_t callback, void *userData)`

Initializes the USART handle.

This function initializes the USART handle which can be used for other USART transactional APIs. Usually, for a specified USART instance, call this API once to get the initialized handle.

#### Parameters

- `base` – USART peripheral base address.
- `handle` – USART handle pointer.
- `callback` – The callback function.
- `userData` – The parameter of the callback function.

`status_t` `USART_TransferSendNonBlocking(USART_Type *base, usart_handle_t *handle, usart_transfer_t *xfer)`

Transmits a buffer of data using the interrupt method.

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in the IRQ handler, the USART driver calls the callback function and passes the `kStatus_USART_TxIdle` as status parameter.

#### Parameters

- `base` – USART peripheral base address.
- `handle` – USART handle pointer.
- `xfer` – USART transfer structure. See `usart_transfer_t`.

#### Return values

- `kStatus_Success` – Successfully start the data transmission.

- `kStatus_USART_TxBusy` – Previous transmission still not finished, data not all written to TX register yet.
- `kStatus_InvalidArgument` – Invalid argument.

```
void USART__TransferStartRingBuffer(USART_Type *base, usart_handle_t *handle, uint8_t  
                                   *ringBuffer, size_t ringBufferSize)
```

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific USART handle.

When the RX ring buffer is used, data received are stored into the ring buffer even when the user doesn't call the `USART_TransferReceiveNonBlocking()` API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

---

**Note:** When using the RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, then only 31 bytes are used for saving data.

---

#### Parameters

- `base` – USART peripheral base address.
- `handle` – USART handle pointer.
- `ringBuffer` – Start address of the ring buffer for background receiving. Pass `NULL` to disable the ring buffer.
- `ringBufferSize` – size of the ring buffer.

```
void USART__TransferStopRingBuffer(USART_Type *base, usart_handle_t *handle)
```

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

#### Parameters

- `base` – USART peripheral base address.
- `handle` – USART handle pointer.

```
size_t USART__TransferGetRxRingBufferLength(usart_handle_t *handle)
```

Get the length of received data in RX ring buffer.

#### Parameters

- `handle` – USART handle pointer.

#### Returns

Length of received data in RX ring buffer.

```
void USART__TransferAbortSend(USART_Type *base, usart_handle_t *handle)
```

Aborts the interrupt-driven data transmit.

This function aborts the interrupt driven data sending. The user can get the `remainBytes` to find out how many bytes are still not sent out.

#### Parameters

- `base` – USART peripheral base address.
- `handle` – USART handle pointer.

```
status_t USART__TransferGetSendCount(USART_Type *base, usart_handle_t *handle, uint32_t  
                                     *count)
```

Get the number of bytes that have been sent out to bus.

This function gets the number of bytes that have been sent out to bus by interrupt method.

**Parameters**

- base – USART peripheral base address.
- handle – USART handle pointer.
- count – Send bytes count.

**Return values**

- kStatus\_NoTransferInProgress – No send in progress.
- kStatus\_InvalidArgument – Parameter is invalid.
- kStatus\_Success – Get successfully through the parameter count;

*status\_t* USART\_TransferReceiveNonBlocking(USART\_Type \*base, *usart\_handle\_t* \*handle, *usart\_transfer\_t* \*xfer, *size\_t* \*receivedBytes)

Receives a buffer of data using an interrupt method.

This function receives data using an interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough to read, the receive request is saved by the USART driver. When the new data arrives, the receive request is serviced first. When all data is received, the USART driver notifies the upper layer through a callback function and passes the status parameter `kStatus_USART_RxIdle`. For example, the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer. The 5 bytes are copied to the `xfer->data` and this function returns with the parameter `receivedBytes` set to 5. For the left 5 bytes, newly arrived data is saved from the `xfer->data[5]`. When 5 bytes are received, the USART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to the `xfer->data`. When all data is received, the upper layer is notified.

**Parameters**

- base – USART peripheral base address.
- handle – USART handle pointer.
- xfer – USART transfer structure, see `usart_transfer_t`.
- receivedBytes – Bytes received from the ring buffer directly.

**Return values**

- kStatus\_Success – Successfully queue the transfer into transmit queue.
- kStatus\_USART\_RxBusy – Previous receive request is not finished.
- kStatus\_InvalidArgument – Invalid argument.

*void* USART\_TransferAbortReceive(USART\_Type \*base, *usart\_handle\_t* \*handle)

Aborts the interrupt-driven data receiving.

This function aborts the interrupt-driven data receiving. The user can get the `remainBytes` to find out how many bytes not received yet.

**Parameters**

- base – USART peripheral base address.
- handle – USART handle pointer.

*status\_t* USART\_TransferGetReceiveCount(USART\_Type \*base, *usart\_handle\_t* \*handle, *uint32\_t* \*count)

Get the number of bytes that have been received.

This function gets the number of bytes that have been received.

**Parameters**

- base – USART peripheral base address.
- handle – USART handle pointer.
- count – Receive bytes count.

**Return values**

- kStatus\_NoTransferInProgress – No receive in progress.
- kStatus\_InvalidArgument – Parameter is invalid.
- kStatus\_Success – Get successfully through the parameter count;

void USART\_TransferHandleIRQ(USART\_Type \*base, usart\_handle\_t \*handle)  
USART IRQ handle function.

This function handles the USART transmit and receive IRQ request.

**Parameters**

- base – USART peripheral base address.
- handle – USART handle pointer.

FSL\_USART\_DRIVER\_VERSION  
USART driver version.

Error codes for the USART driver.

*Values:*

enumerator kStatus\_USART\_TxBusy  
Transmitter is busy.

enumerator kStatus\_USART\_RxBusy  
Receiver is busy.

enumerator kStatus\_USART\_TxIdle  
USART transmitter is idle.

enumerator kStatus\_USART\_RxIdle  
USART receiver is idle.

enumerator kStatus\_USART\_TxError  
Error happens on txFIFO.

enumerator kStatus\_USART\_RxError  
Error happens on rxFIFO.

enumerator kStatus\_USART\_RxRingBufferOverrun  
Error happens on rx ring buffer

enumerator kStatus\_USART\_NoiseError  
USART noise error.

enumerator kStatus\_USART\_FramingError  
USART framing error.

enumerator kStatus\_USART\_ParityError  
USART parity error.

enumerator kStatus\_USART\_BaudrateNotSupport  
Baudrate is not support in current clock source

enum \_usart\_sync\_mode

USART synchronous mode.

*Values:*

enumerator kUSART\_SyncModeDisabled

Asynchronous mode.

enumerator kUSART\_SyncModeSlave

Synchronous slave mode.

enumerator kUSART\_SyncModeMaster

Synchronous master mode.

enum \_usart\_parity\_mode

USART parity mode.

*Values:*

enumerator kUSART\_ParityDisabled

Parity disabled

enumerator kUSART\_ParityEven

Parity enabled, type even, bit setting: PE | PT = 10

enumerator kUSART\_ParityOdd

Parity enabled, type odd, bit setting: PE | PT = 11

enum \_usart\_stop\_bit\_count

USART stop bit count.

*Values:*

enumerator kUSART\_OneStopBit

One stop bit

enumerator kUSART\_TwoStopBit

Two stop bits

enum \_usart\_data\_len

USART data size.

*Values:*

enumerator kUSART\_7BitsPerChar

Seven bit mode

enumerator kUSART\_8BitsPerChar

Eight bit mode

enum \_usart\_clock\_polarity

USART clock polarity configuration, used in sync mode.

*Values:*

enumerator kUSART\_RxSampleOnFallingEdge

Un\_RXD is sampled on the falling edge of SCLK.

enumerator kUSART\_RxSampleOnRisingEdge

Un\_RXD is sampled on the rising edge of SCLK.

enum \_usart\_txfifo\_watermark

txFIFO watermark values

*Values:*

enumerator kUSART\_TxFifo0  
    USART tx watermark is empty

enumerator kUSART\_TxFifo1  
    USART tx watermark at 1 item

enumerator kUSART\_TxFifo2  
    USART tx watermark at 2 items

enumerator kUSART\_TxFifo3  
    USART tx watermark at 3 items

enumerator kUSART\_TxFifo4  
    USART tx watermark at 4 items

enumerator kUSART\_TxFifo5  
    USART tx watermark at 5 items

enumerator kUSART\_TxFifo6  
    USART tx watermark at 6 items

enumerator kUSART\_TxFifo7  
    USART tx watermark at 7 items

enum \_usart\_rxfifo\_watermark  
    rxFIFO watermark values

*Values:*

enumerator kUSART\_RxFifo1  
    USART rx watermark at 1 item

enumerator kUSART\_RxFifo2  
    USART rx watermark at 2 items

enumerator kUSART\_RxFifo3  
    USART rx watermark at 3 items

enumerator kUSART\_RxFifo4  
    USART rx watermark at 4 items

enumerator kUSART\_RxFifo5  
    USART rx watermark at 5 items

enumerator kUSART\_RxFifo6  
    USART rx watermark at 6 items

enumerator kUSART\_RxFifo7  
    USART rx watermark at 7 items

enumerator kUSART\_RxFifo8  
    USART rx watermark at 8 items

enum \_usart\_interrupt\_enable  
    USART interrupt configuration structure, default settings all disabled.

*Values:*

enumerator kUSART\_TxErrorInterruptEnable

enumerator kUSART\_RxErrorInterruptEnable

enumerator kUSART\_TxLevelInterruptEnable



enumerator kUSART\_RxLevelInterruptEnable

enumerator kUSART\_TxIdleInterruptEnable  
Transmitter idle.

enumerator kUSART\_CtsChangeInterruptEnable  
Change in the state of the CTS input.

enumerator kUSART\_RxBreakChangeInterruptEnable  
Break condition asserted or deasserted.

enumerator kUSART\_RxStartInterruptEnable  
Rx start bit detected.

enumerator kUSART\_FramingErrorInterruptEnable  
Framing error detected.

enumerator kUSART\_ParityErrorInterruptEnable  
Parity error detected.

enumerator kUSART\_NoiseErrorInterruptEnable  
Noise error detected.

enumerator kUSART\_AutoBaudErrorInterruptEnable  
Auto baudrate error detected.

enumerator kUSART\_AllInterruptEnables

enum \_usart\_flags

USART status flags.

This provides constants for the USART status flags for use in the USART functions.

*Values:*

enumerator kUSART\_TxError  
TXERR bit, sets if TX buffer is error

enumerator kUSART\_RxError  
RXERR bit, sets if RX buffer is error

enumerator kUSART\_TxFifoEmptyFlag  
TXEMPTY bit, sets if TX buffer is empty

enumerator kUSART\_TxFifoNotFullFlag  
TXNOTFULL bit, sets if TX buffer is not full

enumerator kUSART\_RxFifoNotEmptyFlag  
RXNOEMPTY bit, sets if RX buffer is not empty

enumerator kUSART\_RxFifoFullFlag  
RXFULL bit, sets if RX buffer is full

enumerator kUSART\_RxIdleFlag  
Receiver idle.

enumerator kUSART\_TxIdleFlag  
Transmitter idle.

enumerator kUSART\_CtsAssertFlag  
CTS signal high.

enumerator kUSART\_CtsChangeFlag  
CTS signal changed interrupt status.

enumerator `kUSART_BreakDetectFlag`

Break detected. Self cleared when rx pin goes high again.

enumerator `kUSART_BreakDetectChangeFlag`

Break detect change interrupt flag. A change in the state of receiver break detection.

enumerator `kUSART_RxStartFlag`

Rx start bit detected interrupt flag.

enumerator `kUSART_FramingErrorFlag`

Framing error interrupt flag.

enumerator `kUSART_ParityErrorFlag`

parity error interrupt flag.

enumerator `kUSART_NoiseErrorFlag`

Noise error interrupt flag.

enumerator `kUSART_AutobaudErrorFlag`

Auto baudrate error interrupt flag, caused by the baudrate counter timeout before the end of start bit.

enumerator `kUSART_AllClearFlags`

typedef enum `_usart_sync_mode` `usart_sync_mode_t`

USART synchronous mode.

typedef enum `_usart_parity_mode` `usart_parity_mode_t`

USART parity mode.

typedef enum `_usart_stop_bit_count` `usart_stop_bit_count_t`

USART stop bit count.

typedef enum `_usart_data_len` `usart_data_len_t`

USART data size.

typedef enum `_usart_clock_polarity` `usart_clock_polarity_t`

USART clock polarity configuration, used in sync mode.

typedef enum `_usart_txfifo_watermark` `usart_txfifo_watermark_t`

txFIFO watermark values

typedef enum `_usart_rxfifo_watermark` `usart_rxfifo_watermark_t`

rxFIFO watermark values

typedef struct `_usart_config` `usart_config_t`

USART configuration structure.

typedef struct `_usart_transfer` `usart_transfer_t`

USART transfer structure.

typedef struct `_usart_handle` `usart_handle_t`

typedef void (\*`usart_transfer_callback_t`)(`USART_Type` \*`base`, `usart_handle_t` \*`handle`, `status_t` `status`, void \*`userData`)

USART transfer callback function.

typedef void (\*`flexcomm_usart_irq_handler_t`)(`USART_Type` \*`base`, `usart_handle_t` \*`handle`)

Typedef for usart interrupt handler.

uint32\_t `USART_GetInstance(USART_Type *base)`

Returns instance number for USART peripheral base address.

USART\_FIFOTRIG\_TXLVL\_GET(base)

USART\_FIFOTRIG\_RXLVL\_GET(base)

UART\_RETRY\_TIMES

Retry times for waiting flag.

Defining to zero means to keep waiting for the flag until it is assert/deassert in blocking transfer, otherwise the program will wait until the UART\_RETRY\_TIMES counts down to 0, if the flag still remains unchanged then program will return kStatus\_USART\_Timeout. It is not advised to use this macro in formal application to prevent any hardware error because the actual wait period is affected by the compiler and optimization.

struct \_usart\_config

*#include <fsl\_usart.h>* USART configuration structure.

### Public Members

uint32\_t baudRate\_Bps

USART baud rate

usart\_parity\_mode\_t parityMode

Parity mode, disabled (default), even, odd

usart\_stop\_bit\_count\_t stopBitCount

Number of stop bits, 1 stop bit (default) or 2 stop bits

usart\_data\_len\_t bitCountPerChar

Data length - 7 bit, 8 bit

bool loopback

Enable peripheral loopback

bool enableRx

Enable RX

bool enableTx

Enable TX

bool enableContinuousSCLK

USART continuous Clock generation enable in synchronous master mode.

bool enableMode32k

USART uses 32 kHz clock from the RTC oscillator as the clock source.

bool enableHardwareFlowControl

Enable hardware control RTS/CTS

usart\_txfifo\_watermark\_t txWatermark

txFIFO watermark

usart\_rxfifo\_watermark\_t rxWatermark

rxFIFO watermark

usart\_sync\_mode\_t syncMode

Transfer mode select - asynchronous, synchronous master, synchronous slave.

usart\_clock\_polarity\_t clockPolarity

Selects the clock polarity and sampling edge in synchronous mode.

struct \_usart\_transfer

*#include <fsl\_usart.h>* USART transfer structure.

### Public Members

size\_t dataSize

The byte count to be transfer.

struct \_\_usart\_handle

*#include <fsl\_usart.h>* USART handle structure.

### Public Members

const uint8\_t \*volatile txData

Address of remaining data to send.

volatile size\_t txDataSize

Size of the remaining data to send.

size\_t txDataSizeAll

Size of the data to send out.

uint8\_t \*volatile rxData

Address of remaining data to receive.

volatile size\_t rxDataSize

Size of the remaining data to receive.

size\_t rxDataSizeAll

Size of the data to receive.

uint8\_t \*rxRingBuffer

Start address of the receiver ring buffer.

size\_t rxRingBufferSize

Size of the ring buffer.

volatile uint16\_t rxRingBufferHead

Index for the driver to store received data into ring buffer.

volatile uint16\_t rxRingBufferTail

Index for the user to get data from the ring buffer.

*usart\_transfer\_callback\_t* callback

Callback function.

void \*userData

USART callback function parameter.

volatile uint8\_t txState

TX transfer state.

volatile uint8\_t rxState

RX transfer state

uint8\_t txWatermark

txFIFO watermark

uint8\_t rxWatermark

rxFIFO watermark

union \_\_unnamed36\_\_

**Public Members**

uint8\_t \*data

The buffer of data to be transfer.

uint8\_t \*rxData

The buffer to receive data.

const uint8\_t \*txData

The buffer of data to be sent.

## 2.53 UTICK: MictoTick Timer Driver

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

Initializes an UTICK by turning its bus clock on.

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

Deinitializes a UTICK instance.

This function shuts down Utick bus clock

**Parameters**

- base – UTICK peripheral base address.

uint32\_t UTICK\_GetStatusFlags(UTICK\_Type \*base)

Get Status Flags.

This returns the status flag

**Parameters**

- base – UTICK peripheral base address.

**Returns**

status register value

void UTICK\_ClearStatusFlags(UTICK\_Type \*base)

Clear Status Interrupt Flags.

This clears intr status flag

**Parameters**

- base – UTICK peripheral base address.

**Returns**

none

void UTICK\_SetTick(UTICK\_Type \*base, *utick\_mode\_t* mode, uint32\_t count, *utick\_callback\_t* cb)

Starts UTICK.

This function starts a repeat/onetime countdown with an optional callback

**Parameters**

- base – UTICK peripheral base address.
- mode – UTICK timer mode (ie kUTICK\_onetime or kUTICK\_repeat)
- count – UTICK timer mode (ie kUTICK\_onetime or kUTICK\_repeat)
- cb – UTICK callback (can be left as NULL if none, otherwise should be a void func(void))

**Returns**

none

`void UTICK_HandleIRQ(UTICK_Type *base, utick_callback_t cb)`

UTICK Interrupt Service Handler.

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in `UTICK_SetTick()`). if no user callback is scheduled, the interrupt will simply be cleared.

**Parameters**

- `base` – UTICK peripheral base address.
- `cb` – callback scheduled for this instance of UTICK

**Returns**

none

`FSL_UTICK_DRIVER_VERSION`

UTICK driver version 2.0.5.

`enum _utick_mode`

UTICK timer operational mode.

*Values:*enumerator `kUTICK_Onetime`

Trigger once

enumerator `kUTICK_Repeat`

Trigger repeatedly

`typedef enum _utick_mode utick_mode_t`

UTICK timer operational mode.

`typedef void (*utick_callback_t)(void)`

UTICK callback function.

## 2.54 WWDT: Windowed Watchdog Timer Driver

`void WWDT_GetDefaultConfig(wwdt_config_t *config)`

Initializes WWDT configure structure.

This function initializes the WWDT configure structure to default value. The default value are:

```
config->enableWwdt = true;
config->enableWatchdogReset = false;
config->enableWatchdogProtect = false;
config->enableLockOscillator = false;
config->windowValue = 0xFFFFF0U;
config->timeoutValue = 0xFFFFF0U;
config->warningValue = 0;
```

**See also:**`wwdt_config_t`**Parameters**

- `config` – Pointer to WWDT config structure.

```
void WWDT_Init(WWDT_Type *base, const wwdt_config_t *config)
```

Initializes the WWDT.

This function initializes the WWDT. When called, the WWDT runs according to the configuration.

Example:

```
wwdt_config_t config;
WWDT_GetDefaultConfig(&config);
config.timeoutValue = 0x7ffU;
WWDT_Init(wwdt_base,&config);
```

#### Parameters

- base – WWDT peripheral base address
- config – The configuration of WWDT

```
void WWDT_Deinit(WWDT_Type *base)
```

Shuts down the WWDT.

This function shuts down the WWDT.

#### Parameters

- base – WWDT peripheral base address

```
static inline void WWDT_Enable(WWDT_Type *base)
```

Enables the WWDT module.

This function write value into WWDT\_MOD register to enable the WWDT, it is a write-once bit; once this bit is set to one and a watchdog feed is performed, the watchdog timer will run permanently.

#### Parameters

- base – WWDT peripheral base address

```
static inline void WWDT_Disable(WWDT_Type *base)
```

Disables the WWDT module.

#### Deprecated:

Do not use this function. It will be deleted in next release version, for once the bit field of WDEN written with a 1, it can not be re-written with a 0.

This function write value into WWDT\_MOD register to disable the WWDT.

#### Parameters

- base – WWDT peripheral base address

```
static inline uint32_t WWDT_GetStatusFlags(WWDT_Type *base)
```

Gets all WWDT status flags.

This function gets all status flags.

Example for getting Timeout Flag:

```
uint32_t status;
status = WWDT_GetStatusFlags(wwdt_base) & kWWDT_TimeoutFlag;
```

#### Parameters

- base – WWDT peripheral base address

**Returns**

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

```
void WWDT_ClearStatusFlags(WWDT_Type *base, uint32_t mask)
```

Clear WWDT flag.

This function clears WWDT status flag.

Example for clearing warning flag:

```
WWDT_ClearStatusFlags(wwdt_base, kWWDT_WarningFlag);
```

**Parameters**

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

```
static inline void WWDT_SetWarningValue(WWDT_Type *base, uint32_t warningValue)
```

Set the WWDT warning value.

The `WDWARNINT` register determines the watchdog timer counter value that will generate a watchdog interrupt. When the watchdog timer counter is no longer greater than the value defined by `WARNINT`, an interrupt will be generated after the subsequent `WDCLK`.

**Parameters**

- `base` – WWDT peripheral base address
- `warningValue` – WWDT warning value.

```
static inline void WWDT_SetTimeoutValue(WWDT_Type *base, uint32_t timeoutCount)
```

Set the WWDT timeout value.

This function sets the timeout value. Every time a feed sequence occurs the value in the TC register is loaded into the Watchdog timer. Writing a value below `0xFF` will cause `0xFF` to be loaded into the TC register. Thus the minimum time-out interval is `TWDCLK*256*4`. If `enableWatchdogProtect` flag is true in `wwdt_config_t` config structure, any attempt to change the timeout value before the watchdog counter is below the warning and window values will cause a watchdog reset and set the `WDTOF` flag.

**Parameters**

- `base` – WWDT peripheral base address
- `timeoutCount` – WWDT timeout value, count of WWDT clock tick.

```
static inline void WWDT_SetWindowValue(WWDT_Type *base, uint32_t windowValue)
```

Sets the WWDT window value.

The `WINDOW` register determines the highest TV value allowed when a watchdog feed is performed. If a feed sequence occurs when timer value is greater than the value in `WINDOW`, a watchdog event will occur. To disable windowing, set `windowValue` to `0xFFFFFFFF` (maximum possible timer value) so windowing is not in effect.

**Parameters**

- `base` – WWDT peripheral base address
- `windowValue` – WWDT window value.

```
void WWDT_Refresh(WWDT_Type *base)
```

Refreshes the WWDT timer.

This function feeds the WWDT. This function should be called before WWDT timer is in timeout. Otherwise, a reset is asserted.



### Parameters

- base – WWDT peripheral base address

FSL\_WWDT\_DRIVER\_VERSION

Defines WWDT driver version.

WWDT\_FIRST\_WORD\_OF\_REFRESH

First word of refresh sequence

WWDT\_SECOND\_WORD\_OF\_REFRESH

Second word of refresh sequence

enum \_wwdt\_status\_flags\_t

WWDT status flags.

This structure contains the WWDT status flags for use in the WWDT functions.

*Values:*

enumerator kWWDT\_TimeoutFlag

Time-out flag, set when the timer times out

enumerator kWWDT\_WarningFlag

Warning interrupt flag, set when timer is below the value WDWARNINT

typedef struct \_wwdt\_config wwdt\_config\_t

Describes WWDT configuration structure.

struct \_wwdt\_config

*#include <fsl\_wwdt.h>* Describes WWDT configuration structure.

### Public Members

bool enableWwdt

Enables or disables WWDT

bool enableWatchdogReset

true: Watchdog timeout will cause a chip reset false: Watchdog timeout will not cause a chip reset

bool enableWatchdogProtect

true: Enable watchdog protect i.e timeout value can only be changed after counter is below warning & window values false: Disable watchdog protect; timeout value can be changed at any time

bool enableLockOscillator

true: Disabling or powering down the watchdog oscillator is prevented Once set, this bit can only be cleared by a reset false: Do not lock oscillator

uint32\_t windowValue

Window value, set this to 0xFFFFFFFF if windowing is not in effect

uint32\_t timeoutValue

Timeout value

uint32\_t warningValue

Watchdog time counter value that will generate a warning interrupt. Set this to 0 for no warning

uint32\_t clockFreq\_Hz

Watchdog clock source frequency.



# Chapter 3

## Middleware

### 3.1 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\_pd\_bdm** uses JTAG or BDM debugging interface to access the target RAM directly while the CPU is running. Note that such approach can be used with any MCU application, even without any special driver code. The computer reads from and writes into the RAM directly without CPU intervention. The Packet-Driven BDM (PD-BDM) communication uses the same memory access to exchange command and response frames. With PD-BDM, the FreeMASTER tool is able to go beyond basic memory read/write operations and accesses also advanced features like Recorder, TSA, or Pipes. The typical transmission speed is around 10 kbps. A PD-BDM communication plug-in must be used in FreeMASTER and configured properly for the selected debugging interface. Note that this communication cannot be used while a debugging interface is used by a debugger session.
- **fmstr\_any** is a special example application which demonstrates how the NXP MCUXpresso Config Tools can be used to configure pins, clocks, peripherals, interrupts, and even the FreeMASTER “middleware” driver features in a graphical and user friendly way. The user can switch between the Serial, CAN, and other ways of communication and generate the required initialization code automatically.

**Zephyr sample applications** Zephyr sample applications demonstrate Kconfig and Device Tree configuration which configure the FreeMASTER middleware module for a selected communication option (Serial, CAN, Network or RTT).

Refer to *readme.md* files in each sample directory for description of configuration options required to implement FreeMASTER connectivity.

## Description

This section shows how to add the FreeMASTER Communication Driver into application and how to configure the connection to the FreeMASTER visualization tool.

**Features** The FreeMASTER driver implements the FreeMASTER protocol V4 and provides the following features which may be accessed using the FreeMASTER visualization tool:

- Read/write access to any memory location on the target.
- Optional password protection of the read, read/write, and read/write/flash access levels.

- Atomic bit manipulation on the target memory (bit-wise write access).
- Optimal size-aligned access to memory which is also suitable to access the peripheral register space.
- Oscilloscope access—real-time access to target variables. The sample rate may be limited by the communication speed.
- Recorder— access to the fast transient recorder running on the board as a part of the FreeMASTER driver. The sample rate is only limited by the MCU CPU speed. The length of the data recorded depends on the amount of available memory.
- Multiple instances of Oscilloscopes and Recorders without the limitation of maximum number of variables.
- Application commands—high-level message delivery from the PC to the application.
- TSA tables—describing the data types, variables, files, or hyperlinks exported by the target application. The TSA newly supports also non-memory mapped resources like external EEPROM or SD Card files.
- Pipes—enabling the buffered stream-oriented data exchange for a general-purpose terminal-like communication, diagnostic data streaming, or other data exchange.

The FreeMASTER driver features:

- Full FreeMASTER protocol V4 implementation with a new V4 style of CRC used.
- Layered approach supporting Serial, CAN, Network, PD-BDM, and other transports.
- Layered low-level Serial transport driver architecture enabling to select UART, LPUART, USART, and other physical implementations of serial interfaces, including USB-CDC.
- Layered low-level CAN transport driver architecture enabling to select FlexCAN, msCAN, MCAN, and other physical implementations of the CAN interface.
- Layered low-level Networking transport enabling to select TCP, UDP or J-Link RTT communication.
- TSA support to write-protect memory regions or individual variables and to deny the access to the unsafe memory.
- The pipe callback handlers are invoked whenever new data is available for reading from the pipe.
- Two Serial Single-Wire modes of operation are enabled. The “external” mode has the RX and TX shorted on-board. The “true” single-wire mode interconnects internally when the MCU or UART modules support it.

The following sections briefly describe all FreeMASTER features implemented by the driver. See the PC-based FreeMASTER User Manual for more details on how to use the features to monitor, tune, or control an embedded application.

**Board Detection** The FreeMASTER protocol V4 defines the standard set of configuration values which the host PC tool reads to identify the target and to access other target resources properly. The configuration includes the following parameters:

- Version of the driver and the version of the protocol implemented.
- MTU as the Maximum size of the Transmission Unit (for example; communication buffer size).
- Application name, description, and version strings.
- Application build date and time as a string.
- Target processor byte ordering (little/big endian).
- Protection level that requires password authentication.



- Number of the Recorder and Oscilloscope instances.
- RAM Base Address for optimized memory access commands.

**Memory Read** This basic feature enables the host PC to read any data memory location by specifying the address and size of the required memory area. The device response frame must be shorter than the MTU to fit into the outgoing communication buffer. To read a device memory of any size, the host uses the information retrieved during the Board Detection and splits the large-block request to multiple partial requests.

The driver uses size-aligned operations to read the target memory (for example; uses proper read-word instruction when an address is aligned to 4 bytes).

**Memory Write** Similarly to the Memory Read operation, the Memory Write feature enables to write to any RAM memory location on the target device. A single write command frame must be shorter than the MTU to fit into the target communication buffer. Larger requests must be split into smaller ones.

The driver uses size-aligned operations to write to the target memory (for example; uses proper write-word instruction when an address is aligned to 4 bytes).

**Masked Memory Write** To implement the write access to a single bit or a group of bits of target variables, the Masked Memory Write feature is available in the FreeMASTER protocol and it is supported by the driver using the Read-Modify-Write approach.

Be careful when writing to bit fields of volatile variables that are also modified in an application interrupt. The interrupt may be serviced in the middle of a read-modify-write operation and it may cause data corruption.

**Oscilloscope** The protocol and driver enables any number of variables to be read at once with a single request from the host. This feature is called Oscilloscope and the FreeMASTER tool uses it to display a real-time graph of variable values.

The driver can be configured to support any number of Oscilloscope instances and enable simultaneously running graphs to be displayed on the host computer screen.

**Recorder** The protocol enables the host to select target variables whose values are then periodically recorded into a dedicated on-board memory buffer. After such data sampling stops (either on a host request or by evaluating a threshold-crossing condition), the data buffer is downloaded to the host and displayed as a graph. The data sampling rate is not limited by the speed of the communication line, so it enables displaying the variable transitions in a very high resolution.

The driver can be configured to support multiple Recorder instances and enable multiple recorder graphs to be displayed on the host screen. Having multiple recorders also enables setting the recording point differently for each instance. For example; one instance may be recording data in a general timer interrupt while another instance may record at a specific control algorithm time in the PWM interrupt.

**TSA** With the TSA feature, data types and variables can be described directly in the application source code. Such information is later provided to the FreeMASTER tool which may use it instead of reading symbol data from the application ELF executable file.

The information is encoded as so-called TSA tables which become direct part of the application code. The TSA tables contain descriptors of variables that shall be visible to the host tool. The descriptors can describe the memory areas by specifying the address and size of the memory



block or more conveniently using the C variable names directly. Different set of TSA descriptors can be used to encode information about the structure types, unions, enumerations, or arrays.

The driver also supports special types of TSA table entries to describe user resources like external EEPROM and SD Card files, memory-mapped files, virtual directories, web URL hyperlinks, and constant enumerations.

**TSA Safety** When the TSA is enabled in the application, the TSA Safety can be enabled and validate the memory accesses directly by the embedded-side driver. When the TSA Safety is turned on, any memory request received from the host is validated and accepted only if it belongs to a TSA-described object. The TSA entries can be declared as Read-Write or Read-Only so that the driver can actively deny the write access to the Read-Only objects.

**Application commands** The Application Commands are high-level messages that can be delivered from the PC Host to the embedded application for further processing. The embedded application can either poll the status, or be called back when a new Application Command arrives to be processed. After the embedded application acknowledges that the command is handled, the host receives the Result Code and reads the other return data from memory. Both the Application Commands and the Result Codes are specific to a given application and it is user's responsibility to define them. The FreeMASTER protocol and the FreeMASTER driver only implement the delivery channel and a set of API calls to enable the Application Command processing in general.

**Pipes** The Pipes enable buffered and stream-oriented data exchange between the PC Host and the target application. Any pipe can be written to and read from at both ends (either on the PC or the MCU). The data transmission is acknowledged using the special FreeMASTER protocol commands. It is guaranteed that the data bytes are delivered from the writer to the reader in a proper order and without losses.

**Serial single-wire operation** The MCU Serial Communication Driver natively supports normal dual-wire operation. Because the protocol is half-duplex only, the driver can also operate in two single-wire modes:

- “External” single-wire operation where the Receiver and Transmitter pins are shorted on the board. This mode is supported by default in the MCU driver because the Receiver and Transmitter units are enabled or disabled whenever needed. It is also easy to extend this operation for the RS485 communication.
- “True” single-wire mode which uses only a single pin and the direction switching is made by the UART module. This mode of operation must be enabled by defining the FMSTR\_SERIAL\_SINGLEWIRE configuration option.

**Multi-session support** With networking interface it is possible for multiple clients to access the target MCU simultaneously. Reading and writing of target memory is processed atomically so there is no risk of data corruption. The state-full resources such as Recorders or Oscilloscopes are locked to a client session upon first use and access is denied to other clients until lock is released..

## Zephyr-specific

**Dedicated communication task** FreeMASTER communication may run isolated in a dedicated task. The task automates the FMSTR\_Init and FMSTR\_Poll calls together with periodic activities enabling the FreeMASTER UI to fetch information about tasks and CPU utilization. The task can be started automatically or manually, and it must be assigned a priority to be able to react on interrupts and other communication events. Refer to Zephyr FreeMASTER sample applications which all use this communication task.

**Zephyr shell and logging over FreeMASTER pipe** FreeMASTER implements a shell backend which may use FreeMASTER pipe as a I/O terminal and logging output. Refer to Zephyr FreeMASTER sample applications which all use this feature.

**Automatic TSA tables** TSA tables can be declared as “automatic” in Zephyr which make them automatically registered in the table list. This may be very useful when there are many TSA tables or when the tables are defined in different (often unrelated) libraries linked together. In this case user does not need to build a list of all tables manually.

**Driver files** The driver source files can be found in a top-level src folder, further divided into the sub-folders:

- **src/platforms** platform-specific folder—one folder exists for each supported processor platform (for example; 32-bit Little Endian platform). Each such folder contains a platform header file with data types and a code which implements the potentially platform-specific operations, such as aligned memory access.
- **src/common** folder—contains the common driver source files shared by the driver for all supported platforms. All the .c files must be added to the project, compiled, and linked together with the application.
  - *freemaster.h* - master driver header file, which declares the common data types, macros, and prototypes of the FreeMASTER driver API functions.
  - *freemaster\_cfg.h.example* - this file can serve as an example of the FreeMASTER driver configuration file. Save this file into a project source code folder and rename it to *freemaster\_cfg.h*. The FreeMASTER driver code includes this file to get the project-specific configuration options and to optimize the compilation of the driver.
  - *freemaster\_defcfg.h* - defines the default values for each FreeMASTER configuration option if the option is not set in the *freemaster\_cfg.h* file.
  - *freemaster\_protocol.h* - defines the FreeMASTER protocol constants used internally by the driver.
  - *freemaster\_protocol.c* - implements the FreeMASTER protocol decoder and handles the basic Get Configuration Value, Memory Read, and Memory Write commands.
  - *freemaster\_rec.c* - handles the Recorder-specific commands and implements the Recorder sampling and triggering routines. When the Recorder is disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
  - *freemaster\_scope.c* - handles the Oscilloscope-specific commands. If the Oscilloscope is disabled by the FreeMASTER driver configuration file, this file compiles as void.
  - *freemaster\_pipes.c* - implements the Pipes functionality when the Pipes feature is enabled.
  - *freemaster\_appcmd.c* - handles the communication commands used to deliver and execute the Application Commands within the context of the embedded application. When the Application Commands are disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.

- *freemaster\_tsa.c* - handles the commands specific to the TSA feature. This feature enables the FreeMASTER host tool to obtain the TSA memory descriptors declared in the embedded application. If the TSA is disabled by the FreeMASTER driver configuration file, this file compiles as void.
- *freemaster\_tsa.h* - contains the declaration of the macros used to define the TSA memory descriptors. This file is indirectly included into the user application code (via *freemaster.h*).
- *freemaster\_sha.c* - implements the SHA-1 hash code used in the password authentication algorithm.
- *freemaster\_private.h* - contains the declarations of functions and data types used internally in the driver. It also contains the C pre-processor statements to perform the compile-time verification of the user configuration provided in the *freemaster\_cfg.h* file.
- *freemaster\_serial.c* - implements the serial protocol logic including the CRC, FIFO queuing, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a character-oriented API exported by the specific low-level driver.
- *freemaster\_serial.h* - defines the low-level character-oriented Serial API.
- *freemaster\_can.c* - implements the CAN protocol logic including the CAN message preparation, signalling using the first data byte in the CAN frame, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a message-oriented API exported by the specific low-level driver.
- *freemaster\_can.h* - defines the low-level message-oriented CAN API.
- *freemaster\_net.c* - implements the Network protocol transport logic including multiple session management code.
- *freemaster\_net.h* - definitions related to the Network transport.
- *freemaster\_pdbdm.c* - implements the packet-driven BDM communication buffer and other communication-related operations.
- *freemaster\_utils.c* - aligned memory copy routines, circular buffer management and other utility functions
- *freemaster\_utils.h* - definitions related to utility code.
- **src/drivers/[sdk]/serial** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster\_serial\_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the UART, LPUART, USART, and other kinds of Serial communication modules.
- **src/drivers/[sdk]/can** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster\_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the FlexCAN, msCAN, MCAN, and other kinds of CAN communication modules.
- **src/drivers/[sdk]/network** - contains low-level code adapting the FreeMASTER Network transport to an underlying TCP/IP or RTT stack.
  - *freemaster\_net\_lwip\_tcp.c* and *\_udp.c* - default networking implementation of TCP and UDP transports using lwIP stack.
  - *freemaster\_net\_segger\_rtt.c* - implementation of network transport using Segger J-Link RTT interface

**Driver configuration** The driver is configured using a single header file (*freemaster\_cfg.h*). Create this file and save it together with other project source files before compiling the driver code. All FreeMASTER driver source files include the *freemaster\_cfg.h* file and use the macros defined here for the conditional and parameterized compilation. The C compiler must locate the configuration file when compiling the driver files. Typically, it can be achieved by putting this file into a folder where the other project-specific included files are stored.

As a starting point to create the configuration file, get the *freemaster\_cfg.h.example* file, rename it to *freemaster\_cfg.h*, and save it into the project area.

**Note:** It is NOT recommended to leave the *freemaster\_cfg.h* file in the FreeMASTER driver source code folder. The configuration file must be placed at a project-specific location, so that it does not affect the other applications that use the same driver.

**Configurable items** This section describes the configuration options which can be defined in *freemaster\_cfg.h*.

### Interrupt modes

```
#define FMSTR_LONG_INTR    [0|1]
#define FMSTR_SHORT_INTR   [0|1]
#define FMSTR_POLL_DRIVEN [0|1]
```

**Value Type** boolean (0 or 1)

**Description** Exactly one of the three macros must be defined to non-zero. The others must be defined to zero or left undefined. The non-zero-defined constant selects the interrupt mode of the driver. See [Driver interrupt modes](#).

- FMSTR\_LONG\_INTR — long interrupt mode
- FMSTR\_SHORT\_INTR — short interrupt mode
- FMSTR\_POLL\_DRIVEN — poll-driven mode

**Note:** Some options may not be supported by all communication interfaces. For example, the FMSTR\_SHORT\_INTR option is not supported by the USB\_CDC interface.

### Protocol transport

```
#define FMSTR_TRANSPORT [identifier]
```

**Value Type** Driver identifiers are structure instance names defined in FreeMASTER source code. Specify one of existing instances to make use of the protocol transport.

**Description** Use one of the pre-defined constants, as implemented by the FreeMASTER code. The current driver supports the following transports:

- FMSTR\_SERIAL - serial communication protocol
- FMSTR\_CAN - using CAN communication
- FMSTR\_PDBDM - using packet-driven BDM communication
- FMSTR\_NET - network communication using TCP or UDP protocol

**Serial transport** This section describes configuration parameters used when serial transport is used:

```
#define FMSTR_TRANSPORT FMSTR_SERIAL
```

**FMSTR\_SERIAL\_DRV** Select what low-level driver interface will be used when implementing the Serial communication.

```
#define FMSTR_SERIAL_DRV [identifier]
```

**Value Type** Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing serial driver instances.

**Description** When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/serial* implementation):

- **FMSTR\_SERIAL\_MCUX\_UART** - UART driver
- **FMSTR\_SERIAL\_MCUX\_LPUART** - LPUART driver
- **FMSTR\_SERIAL\_MCUX\_USART** - USART driver
- **FMSTR\_SERIAL\_MCUX\_MINIUSART** - miniUSART driver
- **FMSTR\_SERIAL\_MCUX\_QSCI** - DSC QSCI driver
- **FMSTR\_SERIAL\_MCUX\_USB** - USB/CDC class driver (also see code in the */support/mcuxsdk\_usb* folder)
- **FMSTR\_SERIAL\_56F800E\_EONCE** - DSC JTAG EOnCE driver

Other SDKs or BSPs may define custom low-level driver interface structure which may be used as **FMSTR\_SERIAL\_DRV**. For example:

- **FMSTR\_SERIAL\_DREG\_UART** - demonstrates the low-level interface implemented without the MCUXpresso SDK and using direct access to peripheral registers.

### FMSTR\_SERIAL\_BASE

```
#define FMSTR_SERIAL_BASE [address|symbol]
```

**Value Type** Optional address value (numeric or symbolic)

**Description** Specify the base address of the UART, LPUART, USART, or other serial peripheral module to be used for the communication. This value is not defined by default. User application should call `FMSTR_SetSerialBaseAddress()` to select the peripheral module.

### FMSTR\_COMM\_BUFFER\_SIZE

```
#define FMSTR_COMM_BUFFER_SIZE [number]
```

**Value Type** 0 or a value in range 32...255

**Description** Specify the size of the communication buffer to be allocated by the driver. Default value, which suits all driver features, is used when this option is defined as 0.

### FMSTR\_COMM\_RQUEUE\_SIZE

```
#define FMSTR_COMM_RQUEUE_SIZE [number]
```

**Value Type** Value in range 0...255

**Description** Specify the size of the FIFO receiver queue used to quickly receive and store characters in the FMSTR\_SHORT\_INTR interrupt mode. The default value is 32 B.

### FMSTR\_SERIAL\_SINGLEWIRE

```
#define FMSTR_SERIAL_SINGLEWIRE [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Set to non-zero to enable the “True” single-wire mode which uses a single MCU pin to communicate. The low-level driver enables the pin direction switching when the MCU peripheral supports it.

**CAN Bus transport** This section describes configuration parameters used when CAN transport is used:

```
#define FMSTR_TRANSPORT FMSTR_CAN
```

**FMSTR\_CAN\_DRV** Select what low-level driver interface will be used when implementing the CAN communication.

```
#define FMSTR_CAN_DRV [identifier]
```

**Value Type** Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing CAN driver instances.

**Description** When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/can implementation*):

- **FMSTR\_CAN\_MCUX\_FLEXCAN** - FlexCAN driver
- **FMSTR\_CAN\_MCUX\_MCAN** - MCAN driver
- **FMSTR\_CAN\_MCUX\_MSCAN** - msCAN driver
- **FMSTR\_CAN\_MCUX\_DSCFLEXCAN** - DSC FlexCAN driver
- **FMSTR\_CAN\_MCUX\_DSCMSCAN** - DSC msCAN driver

Other SDKs or BSPs may define the custom low-level driver interface structure which may be used as FMSTR\_CAN\_DRV.

### FMSTR\_CAN\_BASE

```
#define FMSTR_CAN_BASE [address|symbol]
```

**Value Type** Optional address value (numeric or symbolic)

**Description** Specify the base address of the FlexCAN, msCAN, or other CAN peripheral module to be used for the communication. This value is not defined by default. User application should call `FMSTR_SetCanBaseAddress()` to select the peripheral module.

#### FMSTR\_CAN\_CMDID

```
#define FMSTR_CAN_CMDID [number]
```

**Value Type** CAN identifier (11-bit or 29-bit number)

**Description** CAN message identifier used for FreeMASTER commands (direction from PC Host tool to target application). When declaring 29-bit identifier, combine the numeric value with `FMSTR_CAN_EXTID` bit. Default value is 0x7AA.

#### FMSTR\_CAN\_RSPID

```
#define FMSTR_CAN_RSPID [number]
```

**Value Type** CAN identifier (11-bit or 29-bit number)

**Description** CAN message identifier used for responding messages (direction from target application to PC Host tool). When declaring 29-bit identifier, combine the numeric value with `FMSTR_CAN_EXTID` bit. Note that both *CMDID* and *RSPID* values may be the same. Default value is 0x7AA.

#### FMSTR\_FLEXCAN\_TXMB

```
#define FMSTR_FLEXCAN_TXMB [number]
```

**Value Type** Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

**Description** Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame transmission. Default value is 0.

#### FMSTR\_FLEXCAN\_RXMB

```
#define FMSTR_FLEXCAN_RXMB [number]
```

**Value Type** Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

**Description** Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame reception. Note that the FreeMASTER driver may also operate with a common message buffer used by both TX and RX directions. Default value is 1.

**Network transport** This section describes configuration parameters used when Network transport is used:

```
#define FMSTR_TRANSPORT FMSTR_NET
```

**FMSTR\_NET\_DRV** Select network interface implementation.

```
#define FMSTR_NET_DRV [identifier]
```

**Value Type** Identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing NET driver instances.

**Description** When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/network implementation*):

- **FMSTR\_NET\_LWIP\_TCP** - TCP communication using lwIP stack
- **FMSTR\_NET\_LWIP\_UDP** - UDP communication using lwIP stack
- **FMSTR\_NET\_SEGGER\_RTT** - Communication using SEGGER J-Link RTT interface

Other SDKs or BSPs may define the custom networking interface which may be used as FMSTR\_CAN\_DRV.

Add another row below:

#### FMSTR\_NET\_PORT

```
#define FMSTR_NET_PORT [number]
```

**Value Type** TCP or UDP port number (short integer)

**Description** Specifies the server port number used by TCP or UDP protocols.

#### FMSTR\_NET\_BLOCKING\_TIMEOUT

```
#define FMSTR_NET_BLOCKING_TIMEOUT [number]
```

**Value Type** Timeout as number of milliseconds

**Description** This value specifies a timeout in milliseconds for which the network socket operations may block the execution inside *FMSTR\_Poll*. This may be set high (e.g. 250) when a dedicated RTOS task is used to handle FreeMASTER protocol polling. Set to a lower value when the polling task is also responsible for other operations. Set to 0 to attempt to use non-blocking socket operations.



**FMSTR\_NET\_AUTODISCOVERY**

```
#define FMSTR_NET_AUTODISCOVERY [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** This option enables the FreeMASTER driver to use a separate UDP socket to broadcast auto-discovery messages to network. This helps the FreeMASTER tool to discover the target device address, port and protocol options.

**Debugging options****FMSTR\_DISABLE**

```
#define FMSTR_DISABLE [0|1]
```

**Value Type** boolean (0 or 1)

**Description** Define as non-zero to disable all FreeMASTER features, exclude the driver code from build, and compile all its API functions empty. This may be useful to remove FreeMASTER without modifying any application source code. Default value is 0 (false).

**FMSTR\_DEBUG\_TX**

```
#define FMSTR_DEBUG_TX [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to enable the driver to periodically transmit test frames out on the selected communication interface (SCI or CAN). With the debug transmission enabled, it is simpler to detect problems in the baudrate or other communication configuration settings.

The test frames are transmitted until the first valid command frame is received from the PC Host tool. The test frame is a valid error status frame, as defined by the protocol format. On the serial line, the test frame consists of three printable characters (+©W) which are easy to capture using the serial terminal tools.

This feature requires the FMSTR\_Poll() function to be called periodically. Default value is 0 (false).

**FMSTR\_APPLICATION\_STR**

```
#define FMSTR_APPLICATION_STR
```

**Value Type** String.

**Description** Name of the application visible in FreeMASTER host application.

**Memory access**

**FMSTR\_USE\_READMEM**

```
#define FMSTR_USE_READMEM [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the Memory Read command and enable FreeMASTER to have read access to memory and variables. The access can be further restricted by using a TSA feature.  
Default value is 1 (true).

**FMSTR\_USE\_WRITEMEM**

```
#define FMSTR_USE_WRITEMEM [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the Memory Write command.  
The default value is 1 (true).

**Oscilloscope options****FMSTR\_USE\_SCOPE**

```
#define FMSTR_USE_SCOPE [number]
```

**Value Type** Integer number.

**Description** Number of Oscilloscope instances to be supported. Set to 0 to disable the Oscilloscope feature.  
Default value is 0.

**FMSTR\_MAX\_SCOPE\_VARS**

```
#define FMSTR_MAX_SCOPE_VARS [number]
```

**Value Type** Integer number larger than 2.

**Description** Number of variables to be supported by each Oscilloscope instance.  
Default value is 8.

**Recorder options****FMSTR\_USE\_RECORDER**

```
#define FMSTR_USE_RECORDER [number]
```

**Value Type** Integer number.

**Description** Number of Recorder instances to be supported. Set to 0 to disable the Recorder feature.  
Default value is 0.

#### FMSTR\_REC\_BUFF\_SIZE

```
#define FMSTR_REC_BUFF_SIZE [number]
```

**Value Type** Integer number larger than 2.

**Description** Defines the size of the memory buffer used by the Recorder instance #0.  
Default: not defined, user shall call 'FMSTR\_RecorderCreate()' API function to specify this parameter in run time.

#### FMSTR\_REC\_TIMEBASE

```
#define FMSTR_REC_TIMEBASE [time specification]
```

**Value Type** Number (nanoseconds time).

**Description** Defines the base sampling rate in nanoseconds (sampling speed) Recorder instance #0.

Use one of the following macros:

- FMSTR\_REC\_BASE\_SECONDS(x)
- FMSTR\_REC\_BASE\_MILLISEC(x)
- FMSTR\_REC\_BASE\_MICROSEC(x)
- FMSTR\_REC\_BASE\_NANOSEC(x)

Default: not defined, user shall call 'FMSTR\_RecorderCreate()' API function to specify this parameter in run time.

#### FMSTR\_REC\_FLOAT\_TRIG

```
#define FMSTR_REC_FLOAT_TRIG [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the floating-point triggering. Be aware that floating-point triggering may grow the code size by linking the floating-point standard library.  
Default value is 0 (false).

### Application Commands options

**FMSTR\_USE\_APPCMD**

```
#define FMSTR_USE_APPCMD [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the Application Commands feature. Default value is 0 (false).

**FMSTR\_APPCMD\_BUFF\_SIZE**

```
#define FMSTR_APPCMD_BUFF_SIZE [size]
```

**Value Type** Numeric buffer size in range 1..255

**Description** The size of the Application Command data buffer allocated by the driver. The buffer stores the (optional) parameters of the Application Command which waits to be processed.

**FMSTR\_MAX\_APPCMD\_CALLS**

```
#define FMSTR_MAX_APPCMD_CALLS [number]
```

**Value Type** Number in range 0..255

**Description** The number of different Application Commands that can be assigned a callback handler function using FMSTR\_RegisterAppCmdCall(). Default value is 0.

**TSA options****FMSTR\_USE\_TSA**

```
#define FMSTR_USE_TSA [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable the FreeMASTER TSA feature to be used. With this option enabled, the TSA tables defined in the applications are made available to the FreeMASTER host tool. Default value is 0 (false).

**FMSTR\_USE\_TSA\_SAFETY**

```
#define FMSTR_USE_TSA_SAFETY [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable the memory access validation in the FreeMASTER driver. With this option, the host tool is not able to access the memory which is not described by at least one TSA descriptor. Also a write access is denied for objects defined as read-only in TSA tables. Default value is 0 (false).

#### FMSTR\_USE\_TSA\_INROM

```
#define FMSTR_USE_TSA_INROM [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Declare all TSA descriptors as *const*, which enables the linker to put the data into the flash memory. The actual result depends on linker settings or the linker commands used in the project. Default value is 0 (false).

#### FMSTR\_USE\_TSA\_DYNAMIC

```
#define FMSTR_USE_TSA_DYNAMIC [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable runtime-defined TSA entries to be added to the TSA table by the FMSTR\_SetUpTsaBuff() and FMSTR\_TsaAddVar() functions. Default value is 0 (false).

### Pipes options

#### FMSTR\_USE\_PIPES

```
#define FMSTR_USE_PIPES [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable the FreeMASTER Pipes feature to be used. Default value is 0 (false).

#### FMSTR\_MAX\_PIPES\_COUNT

```
#define FMSTR_MAX_PIPES_COUNT [number]
```

**Value Type** Number in range 1..63.

**Description** The number of simultaneous pipe connections to support. The default value is 1.

**Driver interrupt modes** To implement the communication, the FreeMASTER driver handles the Serial or CAN module's receive and transmit requests. Use the *freemaster\_cfg.h* configuration file to select whether the driver processes the communication automatically in the interrupt service routine handler or if it only polls the status of the module (typically during the application idle time).

This section describes each of the interrupt mode in more details.

**Completely Interrupt-Driven operation** Activated using:

```
#define FMSTR_LONG_INTR 1
```

In this mode, both the communication and the FreeMASTER protocol decoding is done in the *FMSTR\_SerialIsr*, *FMSTR\_CanIsr*, or other interrupt service routine. Because the protocol execution may be a lengthy task (especially with the TSA-Safety enabled) it is recommended to use this mode only if the interrupt prioritization scheme is possible in the application and the FreeMASTER interrupt is assigned to a lower (the lowest) priority.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR\_SerialIsr* or *FMSTR\_CanIsr* functions from that handler.

**Mixed Interrupt and Polling Modes** Activated using:

```
#define FMSTR_SHORT_INTR 1
```

In this mode, the communication processing time is split between the interrupt routine and the main application loop or task. The raw communication is handled by the *FMSTR\_SerialIsr*, *FMSTR\_CanIsr*, or other interrupt service routine, while the protocol decoding and execution is handled by the *FMSTR\_Poll* routine. Call *FMSTR\_Poll* during the idle time in the application main loop.

The interrupt processing in this mode is relatively fast and deterministic. Upon a serial-receive event, the received character is only placed into a FIFO-like queue and it is not further processed. Upon a CAN receive event, the received frame is stored into a receive buffer. When transmitting, the characters are fetched from the prepared transmit buffer.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR\_SerialIsr* or *FMSTR\_CanIsr* functions from that handler.

When the serial interface is used as the serial communication interface, ensure that the *FMSTR\_Poll* function is called at least once per *N* character time periods. *N* is the length of the FreeMASTER FIFO queue (*FMSTR\_COMM\_QUEUE\_SIZE*) and the character time is the time needed to transmit or receive a single byte over the SCI line.

**Completely Poll-driven**

```
#define FMSTR_POLL_DRIVEN 1
```

In this mode, both the communication and the FreeMASTER protocol decoding are done in the *FMSTR\_Poll* routine. No interrupts are needed and the *FMSTR\_SerialIsr*, *FMSTR\_CanIsr*, and similar handlers compile to an empty code.

When using this mode, ensure that the *FMSTR\_Poll* function is called by the application at least once per the serial "character time" which is the time needed to transmit or receive a single character.

In the latter two modes (*FMSTR\_SHORT\_INTR* and *FMSTR\_POLL\_DRIVEN*), the protocol handling takes place in the *FMSTR\_Poll* routine. An application interrupt can occur in the middle of the

Read Memory or Write Memory commands' execution and corrupt the variable being accessed by the FreeMASTER driver. In these two modes, some issues or glitches may occur when using FreeMASTER to visualize or monitor volatile variables modified in interrupt servicing code.

The same issue may appear even in the full interrupt mode (FMSTR\_LONG\_INTR), if volatile variables are modified in the interrupt code with a priority higher than the priority of the communication interrupt.

**Data types** Simple portability was one of the main requirements when writing the FreeMASTER driver. This is why the driver code uses the privately-declared data types and the vast majority of the platform-dependent code is separated in the platform-dependent source files. The data types used in the driver API are all defined in the platform-specific header file.

To prevent name conflicts with the symbols used in the application, all data types, macros, and functions have the FMSTR\_ prefix. The only global variables used in the driver are the transport and low-level API structures exported from the driver-implementation layer to upper layers. Other than that, all private variables are declared as static and named using the fmstr\_ prefix.

**Communication interface initialization** The FreeMASTER driver does not perform neither the initialization nor the configuration of the peripheral module that it uses to communicate. It is the application startup code responsibility to configure the communication module before the FreeMASTER driver is initialized by the FMSTR\_Init call.

When the Serial communication module is used as the FreeMASTER communication interface, configure the UART receive and transmit pins, the serial communication baud rate, parity (no-parity), the character length (eight bits), and the number of stop bits (one) before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see [Driver interrupt modes](#)), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected serial peripheral module. Call the FMSTR\_SerialIsr function from the application handler.

When a CAN module is used as the FreeMASTER communication interface, configure the CAN receive and transmit pins and the CAN module bit rate before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see [Driver interrupt modes](#)), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected CAN peripheral module. Call the FMSTR\_CanIsr function from the application handler.

**Note:** It is not necessary to enable or unmask the serial nor the CAN interrupts before initializing the FreeMASTER driver. The driver enables or disables the interrupts and communication lines, as required during runtime.

**FreeMASTER Recorder calls** When using the FreeMASTER Recorder in the application (FMSTR\_USE\_RECORDER > 0), call the FMSTR\_RecorderCreate function early after FMSTR\_Init to set up each recorder instance to be used in the application. Then call the FMSTR\_Recorder function periodically in the code where the data recording should occur. A typical place to call the Recorder routine is at the timer or PWM interrupts, but it can be anywhere else. The example applications provided together with the driver code call the FMSTR\_Recorder in the main application loop.

In applications where FMSTR\_Recorder is called periodically with a constant period, specify the period in the Recorder configuration structure before calling FMSTR\_RecorderCreate. This setting enables the PC Host FreeMASTER tool to display the X-axis of the Recorder graph properly scaled for the time domain.

**Driver usage** Start using or evaluating FreeMASTER by opening some of the example applications available in the driver setup package.

Follow these steps to enable the basic FreeMASTER connectivity in the application:

- Make sure that all \*.c files of the FreeMASTER driver from the `src/common/platforms/[your_platform]` folder are a part of the project. See [Driver files](#) for more details.
- Configure the FreeMASTER driver by creating or editing the `freemaster_cfg.h` file and by saving it into the application project directory. See [Driver configuration](#) for more details.
- Include the `freemaster.h` file into any application source file that makes the FreeMASTER API calls.
- Initialize the Serial or CAN modules. Set the baud rate, parity, and other parameters of the communication. Do not enable the communication interrupts in the interrupt mask registers.
- For the FMSTR\_LONG\_INTR and FMSTR\_SHORT\_INTR modes, install the application-specific interrupt routine and call the FMSTR\_SerialIsr or FMSTR\_CanIsr functions from this handler.
- Call the FMSTR\_Init function early on in the application initialization code.
- Call the FMSTR\_RecorderCreate functions for each Recorder instance to enable the Recorder feature.
- In the main application loop, call the FMSTR\_Poll API function periodically when the application is idle.
- For the FMSTR\_SHORT\_INTR and FMSTR\_LONG\_INTR modes, enable the interrupts globally so that the interrupts can be handled by the CPU.

**Communication troubleshooting** The most common problem that causes communication issues is a wrong baud rate setting or a wrong pin multiplexer setting of the target MCU. When a communication between the PC Host running FreeMASTER and the target MCU cannot be established, try enabling the FMSTR\_DEBUG\_TX option in the `freemaster_cfg.h` file and call the FMSTR\_Poll function periodically in the main application task loop.

With this feature enabled, the FreeMASTER driver periodically transmits a test frame through the Serial or CAN lines. Use a logic analyzer or an oscilloscope to monitor the signals at the communication pins of the CPU device to examine whether the bit rate and signal polarity are configured properly.

## Driver API

This section describes the driver Application Programmers' Interface (API) needed to initialize and use the FreeMASTER serial communication driver.

**Control API** There are three key functions to initialize and use the driver.

### FMSTR\_Init

#### Prototype

```
FMSTR_BOOL FMSTR_Init(void);
```

- Declaration: `freemaster.h`
- Implementation: `freemaster_protocol.c`



**Description** This function initializes the internal variables of the FreeMASTER driver and enables the communication interface. This function does not change the configuration of the selected communication module. The hardware module must be initialized before the *FMSTR\_Init* function is called.

A call to this function must occur before calling any other FreeMASTER driver API functions.

## FMSTR\_Poll

### Prototype

```
void FMSTR_Poll(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_protocol.c*

**Description** In the poll-driven or short interrupt modes, this function handles the protocol decoding and execution (see *Driver interrupt modes*). In the poll-driven mode, this function also handles the communication interface with the PC. Typically, the *FMSTR\_Poll* function is called during the “idle” time in the main application task loop.

To prevent the receive data overflow (loss) on a serial interface, make sure that the *FMSTR\_Poll* function is called at least once per the time calculated as:

$$N * Tchar$$

where:

- *N* is equal to the length of the receive FIFO queue (configured by the *FMSTR\_COMM\_RQUEUE\_SIZE* macro). *N* is 1 for the poll-driven mode.
- *Tchar* is the character time, which is the time needed to transmit or receive a single byte over the SCI line.

**Note:** In the long interrupt mode, this function typically compiles as an empty function and can still be called. It is worthwhile to call this function regardless of the interrupt mode used in the application. This approach enables a convenient switching between the different interrupt modes only by changing the configuration macros in the *freemaster\_cfg.h* file.

## FMSTR\_SerialIsr / FMSTR\_CanIsr

### Prototype

```
void FMSTR_SerialIsr(void);
void FMSTR_CanIsr(void);
```

- Declaration: *freemaster.h*
- Implementation: *hw-specific low-level driver C file*

**Description** This function contains the interrupt-processing code of the FreeMASTER driver. In long or short interrupt modes (see *Driver interrupt modes*), this function must be called from the application interrupt service routine registered for the communication interrupt vector. On platforms where the communication module uses multiple interrupt vectors, the application should register a handler for all vectors and call this function at each interrupt.

**Note:** In a poll-driven mode, this function is compiled as an empty function and does not have to be used.

## Recorder API

### FMSTR\_RecorderCreate

#### Prototype

```
FMSTR_BOOL FMSTR_RecorderCreate(FMSTR_INDEX recIndex, FMSTR_REC_BUFF* buffCfg);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_rec.c*

**Description** This function registers a recorder instance and enables it to be used by the PC Host tool. Call this function for all recorder instances from 0 to the maximum number defined by the FMSTR\_USE\_RECORDER configuration option (minus one). An exception to this requirement is the recorder of instance 0 which may be automatically configured by FMSTR\_Init when the *freemaster\_cfg.h* configuration file defines the *FMSTR\_REC\_BUFF\_SIZE* and *FMSTR\_REC\_TIMEBASE* options.

For more information, see [Configurable items](#).

### FMSTR\_Recorder

#### Prototype

```
void FMSTR_Recorder(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_rec.c*

**Description** This function takes a sample of the variables being recorded using the FreeMASTER Recorder instance *recIndex*. If the selected Recorder is not active when the *FMSTR\_Recorder* function is being called, the function returns immediately. When the Recorder is active, the values of the variables being recorded are copied into the recorder buffer and the trigger conditions are evaluated.

If a trigger condition is satisfied, the Recorder enters the post-trigger mode, where it counts down the follow-up samples (number of *FMSTR\_Recorder* function calls) and de-activates the Recorder when the required post-trigger samples are finished.

The *FMSTR\_Recorder* function is typically called in the timer or PWM interrupt service routines. This function can also be called in the application main loop (for testing purposes).

### FMSTR\_RecorderTrigger

#### Prototype

```
void FMSTR_RecorderTrigger(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_rec.c*

**Description** This function forces the Recorder trigger condition to happen, which causes the Recorder to be automatically deactivated after the post-trigger samples are sampled. Use this function in the application code for programmatic control over the Recorder triggering. This can be useful when a more complex triggering conditions need to be used.

**Fast Recorder API** The Fast Recorder feature is not available in the FreeMASTER driver version 3. This feature was heavily dependent on the target platform and it was only available for the 56F8xxxx DSCs.

**TSA Tables** When the TSA is enabled in the FreeMASTER driver configuration file (by setting the FMSTR\_USE\_TSA macro to a non-zero value), it defines the so-called TSA tables in the application. This section describes the macros that must to be used to define the TSA tables.

There can be any number of TSA tables spread across the application source files. There must be always exactly one TSA Table List defined, which informs the FreeMASTER driver about the active TSA tables.

When there is at least one TSA table and one TSA Table List defined in the application, the TSA information automatically appears in the FreeMASTER symbols list. The symbols can then be used to create FreeMASTER variables for visualization or control.

**TSA table definition** The TSA table describes the static or global variables together with their address, size, type, and access-protection information. If the TSA-described variables are of a structure type, the TSA table may also describe this type and provide an access to the individual structure members of the variable.

The TSA table definition begins with the FMSTR\_TSA\_TABLE\_BEGIN macro with a *table\_id* identifying the table. The *table\_id* shall be a valid C-language symbol.

```
FMSTR_TSA_TABLE_BEGIN(table_id)
```

After this opening macro, the TSA descriptors are placed using these macros:

```
/* Adding variable descriptors */
FMSTR_TSA_RW_VAR(name, type) /* read/write variable entry */
FMSTR_TSA_RO_VAR(name, type) /* read-only variable entry */

/* Description of complex data types */
FMSTR_TSA_STRUCT(struct_name) /* structure or union type entry */
FMSTR_TSA_MEMBER(struct_name, member_name, type) /* structure member entry */

/* Memory blocks */
FMSTR_TSA_RW_MEM(name, type, address, size) /* read/write memory block */
FMSTR_TSA_RO_MEM(name, type, address, size) /* read-only memory block */
```

The table is closed using the FMSTR\_TSA\_TABLE\_END macro:

```
FMSTR_TSA_TABLE_END()
```

**TSA descriptor parameters** The TSA descriptor macros accept these parameters:

- *name* — variable name. The variable must be defined before the TSA descriptor references it.
- *type* — variable or member type. Only one of the pre-defined type constants may be used (see below).
- *struct\_name* — structure type name. The type must be defined (typedef) before the TSA descriptor references it.

- *member\_name* — structure member name.

**Note:** The structure member descriptors (FMSTR\_TSA\_MEMBER) must immediately follow the parent structure descriptor (FMSTR\_TSA\_STRUCT) in the table.

**Note:** To write-protect the variables in the FreeMASTER driver (FMSTR\_TSA\_RO\_VAR), enable the TSA-Safety feature in the configuration file.

**TSA variable types** The table lists *type* identifiers which can be used in TSA descriptors:

| Constant                           | Description                                                    |
|------------------------------------|----------------------------------------------------------------|
| FMSTR_TSA_UINTn                    | Unsigned integer type of size <i>n</i> bits (n=8,16,32,64)     |
| FMSTR_TSA_SINTn                    | Signed integer type of size <i>n</i> bits (n=8,16,32,64)       |
| FMSTR_TSA_FRACn                    | Fractional number of size <i>n</i> bits (n=16,32,64).          |
| FMSTR_TSA_FRAC_Q( <i>m,n</i> )     | Signed fractional number in general Q form (m+n+1 total bits)  |
| FMSTR_TSA_FRAC_UQ( <i>m,n</i> )    | Unsigned fractional number in general UQ form (m+n total bits) |
| FMSTR_TSA_FLOAT                    | 4-byte standard IEEE floating-point type                       |
| FMSTR_TSA_DOUBLE                   | 8-byte standard IEEE floating-point type                       |
| FMSTR_TSA_POINTER                  | Generic pointer type defined (platform-specific 16 or 32 bit)  |
| FM-STR_TSA_USERTYPE( <i>name</i> ) | Structure or union type declared with FMSTR_TSA_STRUCT record  |

**TSA table list** There shall be exactly one TSA Table List in the application. The list contains one entry for each TSA table defined anywhere in the application.

The TSA Table List begins with the FMSTR\_TSA\_TABLE\_LIST\_BEGIN macro and continues with the TSA table entries for each table.

```
FMSTR_TSA_TABLE_LIST_BEGIN()
```

```
FMSTR_TSA_TABLE(table_id)
FMSTR_TSA_TABLE(table_id2)
FMSTR_TSA_TABLE(table_id3)
...
```

The list is closed with the FMSTR\_TSA\_TABLE\_LIST\_END macro:

```
FMSTR_TSA_TABLE_LIST_END()
```

**TSA Active Content entries** FreeMASTER v2.0 and higher supports TSA Active Content, enabling the TSA tables to describe the memory-mapped files, virtual directories, and URL hyperlinks. FreeMASTER can access such objects similarly to accessing the files and folders on the local hard drive.

With this set of TSA entries, the FreeMASTER pages can be embedded directly into the target MCU flash and accessed by FreeMASTER directly over the communication line. The HTML-coded pages rendered inside the FreeMASTER window can access the TSA Active Content resources using a special URL referencing the *fmaster*: protocol.

This example provides an overview of the supported TSA Active Content entries:

```
FMSTR_TSA_TABLE_BEGIN(files_and_links)
```

```
/* Directory entry applies to all subsequent MEMFILE entries */
```

```
FMSTR_TSA_DIRECTORY("/text_files") /* entering a new virtual directory */
```

(continues on next page)

(continued from previous page)

```

/* The readme.txt file will be accessible at the fmstr://text_files/readme.txt URL */
FMSTR_TSA_MEMFILE("readme.txt", readme_txt, sizeof(readme_txt)) /* memory-mapped file */

/* Files can also be specified with a full path so the DIRECTORY entry does not apply */
FMSTR_TSA_MEMFILE("/index.htm", index, sizeof(index)) /* memory-mapped file */
FMSTR_TSA_MEMFILE("/prj/demo.pmp", demo_pmp, sizeof(demo_pmp)) /* memory-mapped file */

/* Hyperlinks can point to a local MEMFILE object or to the Internet */
FMSTR_TSA_HREF("Board's Built-in Welcome Page", "/index.htm")
FMSTR_TSA_HREF("FreeMASTER Home Page", "http://www.nxp.com/freemaster")

/* Project file links simplify opening the projects from any URLs */
FMSTR_TSA_PROJECT("Demonstration Project (embedded)", "/prj/demo.pmp")
FMSTR_TSA_PROJECT("Full Project (online)", "http://mycompany.com/prj/demo.pmp")

FMSTR_TSA_TABLE_END()

```

## TSA API

### FMSTR\_SetUpTsaBuff

#### Prototype

```
FMSTR_BOOL FMSTR_SetUpTsaBuff(FMSTR_ADDR buffAddr, FMSTR_SIZE buffSize);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_tsa.c*

#### Arguments

- *buffAddr* [in] - address of the memory buffer for the dynamic TSA table
- *buffSize* [in] - size of the memory buffer which determines the maximum number of TSA entries to be added in the runtime

**Description** This function must be used to assign the RAM memory buffer to the TSA subsystem when FMSTR\_USE\_TSA\_DYNAMIC is enabled. The memory buffer is then used to store the TSA entries added dynamically to the runtime TSA table using the FMSTR\_TsaAddVar function call. The runtime TSA table is processed by the FreeMASTER PC Host tool along with all static tables as soon as the communication port is open.

The size of the memory buffer determines the number of TSA entries that can be added dynamically. Depending on the MCU platform, one TSA entry takes either 8 or 16 bytes.

### FMSTR\_TsaAddVar

#### Prototype

```

FMSTR_BOOL FMSTR_TsaAddVar(FMSTR_TSATBL_STRPTR tsaName, FMSTR_TSATBL_STRPTR
↪ tsaType,
    FMSTR_TSATBL_VOIDPTR varAddr, FMSTR_SIZE32 varSize,
    FMSTR_SIZE flags);

```

- Declaration: *freemaster.h*

- Implementation: *freemaster\_tsa.c*

### Arguments

- *tsaName* [in] - name of the object
- *tsaType* [in] - name of the object type
- *varAddr* [in] - address of the object
- *varSize* [in] - size of the object
- *flags* [in] - access flags; a combination of these values:
  - *FMSTR\_TSA\_INFO\_RO\_VAR* — read-only memory-mapped object (typically a variable)
  - *FMSTR\_TSA\_INFO\_RW\_VAR* — read/write memory-mapped object
  - *FMSTR\_TSA\_INFO\_NON\_VAR* — other entry, describing structure types, structure members, enumerations, and other types

**Description** This function can be called only when the dynamic TSA table is enabled by the *FMSTR\_USE\_TSA\_DYNAMIC* configuration option and when the *FMSTR\_SetUpTsaBuff* function call is made to assign the dynamic TSA table memory. This function adds an entry into the dynamic TSA table. It can be used to register a read-only or read/write memory object or describe an item of the user-defined type.

See [TSA table definition](#) for more details about the TSA table entries.

## Application Commands API

### FMSTR\_GetAppCmd

#### Prototype

```
FMSTR_APPCMD_CODE FMSTR_GetAppCmd(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*

**Description** This function can be used to detect if there is an Application Command waiting to be processed by the application. If no command is pending, this function returns the *FMSTR\_APPCMDRESULT\_NOCMD* constant. Otherwise, this function returns the code of the Application Command that must be processed. Use the *FMSTR\_AppCmdAck* call to acknowledge the Application Command after it is processed and to return the appropriate result code to the host.

The *FMSTR\_GetAppCmd* function does not report the commands for which a callback handler function exists. If the *FMSTR\_GetAppCmd* function is called when a callback-registered command is pending (and before it is actually processed by the callback function), this function returns *FMSTR\_APPCMDRESULT\_NOCMD*.

### FMSTR\_GetAppCmdData

### Prototype

```
FMSTR_APPCMD_PDATA FMSTR_GetAppCmdData(FMSTR_SIZE* dataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*

### Arguments

- *dataLen* [out] - pointer to the variable that receives the length of the data available in the buffer. It can be NULL when this information is not needed.

**Description** This function can be used to retrieve the Application Command data when the application determines that an Application Command is pending (see [FMSTR\\_GetAppCmd](#)).

There is just a single buffer to hold the Application Command data (the buffer length is FMSTR\_APPCMD\_BUFF\_SIZE bytes). If the data are to be used in the application after the command is processed by the FMSTR\_AppCmdAck call, copy the data out to a private buffer.

### FMSTR\_AppCmdAck

#### Prototype

```
void FMSTR_AppCmdAck(FMSTR_APPCMD_RESULT resultCode);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*

### Arguments

- *resultCode* [in] - the result code which is to be returned to FreeMASTER

**Description** This function is used when the Application Command processing finishes in the application. The resultCode passed to this function is returned back to the host and the driver is re-initialized to expect the next Application Command.

After this function is called and before the next Application Command arrives, the return value of the FMSTR\_GetAppCmd function is FMSTR\_APPCMDRESULT\_NOCMD.

### FMSTR\_AppCmdSetResponseData

#### Prototype

```
void FMSTR_AppCmdSetResponseData(FMSTR_ADDR resultDataAddr, FMSTR_SIZE resultDataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*



### Arguments

- *resultDataAddr* [in] - pointer to the data buffer that is to be copied to the Application Command data buffer
- *resultDataLen* [in] - length of the data to be copied. It must not exceed the FMSTR\_APPCMD\_BUFF\_SIZE value.

**Description** This function can be used before the Application Command processing finishes, when there are data to be returned back to the PC.

The response data buffer is copied into the Application Command data buffer, from where it is accessed when the host requires it. Do not use FMSTR\_GetAppCmdData and the data buffer after FMSTR\_AppCmdSetResponseData is called.

**Note:** The current version of FreeMASTER does not support the Application Command response data.

### FMSTR\_RegisterAppCmdCall

#### Prototype

```
FMSTR_BOOL FMSTR_RegisterAppCmdCall(FMSTR_APPCMD_CODE appCmdCode, FMSTR_
↪PAPPCMDFUNC callbackFunc);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*

### Arguments

- *appCmdCode* [in] - the Application Command code for which the callback is to be registered
- *callbackFunc* [in] - pointer to the callback function that is to be registered. Use NULL to unregister a callback registered previously with this Application Command.

**Return value** This function returns a non-zero value when the callback function was successfully registered or unregistered. It can return zero when trying to register a callback function for more than FMSTR\_MAX\_APPCMD\_CALLS different Application Commands.

**Description** This function can be used to register the given function as a callback handler for the Application Command. The Application Command is identified using single-byte code. The callback function is invoked automatically by the FreeMASTER driver when the protocol decoder obtains a request to get the application command result code.

The prototype of the callback function is

```
FMSTR_APPCMD_RESULT HandlerFunction(FMSTR_APPCMD_CODE nAppcmd,
FMSTR_APPCMD_PDATA pData, FMSTR_SIZE nDataLen);
```

Where:

- *nAppcmd* -Application Command code
- *pData* —points to the Application Command data received (if any)
- *nDataLen* —information about the Application Command data length



The return value of the callback function is used as the Application Command Result Code and returned to FreeMASTER.

**Note:** The FMSTR\_MAX\_APPCMD\_CALLS configuration macro defines how many different Application Commands may be handled by a callback function. When FMSTR\_MAX\_APPCMD\_CALLS is undefined or defined as zero, the FMSTR\_RegisterAppCmdCall function always fails.

## Pipes API

### FMSTR\_PipeOpen

#### Prototype

```
FMSTR_HPIPE FMSTR_PipeOpen(FMSTR_PIPE_PORT pipePort, FMSTR_PPIPEFUNC pipeCallback,
    FMSTR_ADDR pipeRxBuff, FMSTR_PIPE_SIZE pipeRxSize,
    FMSTR_ADDR pipeTxBuff, FMSTR_PIPE_SIZE pipeTxSize,
    FMSTR_U8 type, const FMSTR_CHAR *name);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_pipes.c*

#### Arguments

- *pipePort* [in] - port number that identifies the pipe for the client
- *pipeCallback* [in] - pointer to the callback function that is called whenever a pipe data status changes
- *pipeRxBuff* [in] - address of the receive memory buffer
- *pipeRxSize* [in] - size of the receive memory buffer
- *pipeTxBuff* [in] - address of the transmit memory buffer
- *pipeTxSize* [in] - size of the transmit memory buffer
- *type* [in] - a combination of FMSTR\_PIPE\_MODE\_XXX and FMSTR\_PIPE\_SIZE\_XXX constants describing primary pipe data format and usage. This type helps FreeMASTER decide how to access the pipe by default. Optional, use 0 when undetermined.
- *name* [in] - user name of the pipe port. This name is visible to the FreeMASTER user when creating the graphical pipe interface.

**Description** This function initializes a new pipe and makes it ready to accept or send the data to the PC Host client. The receive memory buffer is used to store the received data before they are read out by the FMSTR\_PipeRead call. When this buffer gets full, the PC Host client denies the data transmission into this pipe until there is enough free space again. The transmit memory buffer is used to store the data transmitted by the application to the PC Host client using the FMSTR\_PipeWrite call. The transmit buffer can get full when the PC Host is disconnected or when it is slow in receiving and reading out the pipe data.

The function returns the pipe handle which must be stored and used in the subsequent calls to manage the pipe object.

The callback function (if specified) is called whenever new data are received through the pipe and available for reading. This callback is also called when the data waiting in the transmit buffer are successfully pushed to the PC Host and the transmit buffer free space increases. The prototype of the callback function provided by the user application must be as follows. The *PipeHandler* name is only a placeholder and must be defined by the application.

```
void PipeHandler(FMSTR_HPIPE pipeHandle);
```

## FMSTR\_PipeClose

### Prototype

```
void FMSTR_PipeClose(FMSTR_HPIPE pipeHandle);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_pipes.c*

### Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR\_PipeOpen function call

**Description** This function de-initializes the pipe object. No data can be received or sent on the pipe after this call.

## FMSTR\_PipeWrite

### Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeWrite(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,  
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE writeGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_pipes.c*

### Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR\_PipeOpen function call
- *pipeData* [in] - address of the data to be written
- *pipeDataLen* [in] - length of the data to be written
- *writeGranularity* [in] - size of the minimum unit of data which is to be written

**Description** This function puts the user-specified data into the pipe's transmit memory buffer and schedules it for transmission. This function returns the number of bytes that were successfully written into the buffer. This number may be smaller than the number of the requested bytes if there is not enough free space in the transmit buffer.

The *writeGranularity* argument can be used to split the data into smaller chunks, each of the size given by the *writeGranularity* value. The FMSTR\_PipeWrite function writes as many data chunks as possible into the transmit buffer and does not attempt to write an incomplete chunk. This feature can prove to be useful to avoid the intermediate caching when writing an array of integer values or other multi-byte data items. When making the *nGranularity* value equal to the *nLength* value, all data are considered as one chunk which is either written successfully as a whole or not at all. The *nGranularity* value of 0 or 1 disables the data-chunk approach.

## FMSTR\_PipeRead

## Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeRead(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE readGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_pipes.c*

## Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR\_PipeOpen function call
- *pipeData* [in] - address of the data buffer to be filled with the received data
- *pipeDataLen* [in] - length of the data to be read
- *readGranularity* [in] - size of the minimum unit of data which is to be read

**Description** This function copies the data received from the pipe from its receive buffer to the user buffer for further processing. The function returns the number of bytes that were successfully copied to the buffer. This number may be smaller than the number of the requested bytes if there is not enough data bytes available in the receive buffer.

The readGranularity argument can be used to copy the data in larger chunks in the same way as described in the FMSTR\_PipeWrite function.

**API data types** This section describes the data types used in the FreeMASTER driver. The information provided here can be useful when modifying or porting the FreeMASTER Communication Driver to new NXP platforms.

**Note:** The licensing conditions prohibit use of FreeMASTER and the FreeMASTER Communication Driver with non-NXP MPU or MCU products.

**Public common types** The table below describes the public data types used in the FreeMASTER driver API calls. The data types are declared in the *freemaster.h* header file.

| Type name                                                                                                                                                                                                                 | Description                                                                                                                         |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>FM-STR_ADDR</i><br>For example, this type is defined as long integer on the 56F8xxx platform where the 24-bit addresses must be supported, but the C-pointer may be only 16 bits wide in some compiler configurations. | Data type used to hold the memory address. On most platforms, this is normally a C-pointer, but it may also be a pure integer type. |
| <i>FM-STR_SIZE</i><br>It is required that this type is unsigned and at least 16 bits wide integer.                                                                                                                        | Data type used to hold the memory block size.                                                                                       |
| <i>FM-STR_BOOL</i><br>This type is used only in zero/non-zero conditions in the driver code.                                                                                                                              | Data type used as a general boolean type.                                                                                           |
| <i>FM-STR_APPCM</i><br>Generally, this is an unsigned 8-bit value.                                                                                                                                                        | Data type used to hold the Application Command code.                                                                                |
| <i>FM-STR_APPCM</i><br>Generally, this is an unsigned 8-bit value.                                                                                                                                                        | Data type used to create the Application Command data buffer.                                                                       |
| <i>FM-STR_APPCM</i><br>Generally, this is an unsigned 8-bit value.                                                                                                                                                        | Data type used to hold the Application Command result code.                                                                         |

**Public TSA types** The table describes the TSA-specific public data types. These types are declared in the *freemaster\_tsa.h* header file, which is included in the user application indirectly by the *freemaster.h* file.

|                       |                                                                                                                                                               |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>FM-STR_TSA_TII</i> | Data type used to hold a descriptor index in the TSA table or a table index in the list of TSA tables.<br>By default, this is defined as <i>FM-STR_SIZE</i> . |
| <i>FM-STR_TSA_TS</i>  | Data type used to hold a memory block size, as used in the TSA descriptors.<br>By default, this is defined as <i>FM-STR_SIZE</i> .                            |

**Public Pipes types** The table describes the data types used by the FreeMASTER Pipes API:

|                       |                                                                                                                |
|-----------------------|----------------------------------------------------------------------------------------------------------------|
| <i>FM-STR_HPIPE</i>   | Pipe handle that identifies the open-pipe object.<br>Generally, this is a pointer to a void type.              |
| <i>FM-STR_PIPE_PC</i> | Integer type required to hold at least 7 bits of data.<br>Generally, this is an unsigned 8-bit or 16-bit type. |
| <i>FM-STR_PIPE_SI</i> | Integer type required to hold at least 16 bits of data.<br>This is used to store the data buffer sizes.        |
| <i>FM-STR_PPIPEF</i>  | Pointer to the pipe handler function.<br>See <a href="#">FM-STR_PipeOpen</a> for more details.                 |

**Internal types** The table describes the data types used internally by the FreeMASTER driver. The data types are declared in the platform-specific header file and they are not available in the application code.

|                                                                                                          |                                                                         |
|----------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|
| <i>FMSTR_U8</i>                                                                                          | The smallest memory entity.                                             |
| On the vast majority of platforms, this is an unsigned 8-bit integer.                                    |                                                                         |
| On the 56F8xx DSP platform, this is defined as an unsigned 16-bit integer.                               |                                                                         |
| <i>FMSTR_U16</i>                                                                                         | Unsigned 16-bit integer.                                                |
| <i>FMSTR_U32</i>                                                                                         | Unsigned 32-bit integer.                                                |
| <i>FMSTR_S8</i>                                                                                          | Signed 8-bit integer.                                                   |
| <i>FMSTR_S16</i>                                                                                         | Signed 16-bit integer.                                                  |
| <i>FMSTR_S32</i>                                                                                         | Signed 32-bit integer.                                                  |
| <i>FMSTR_FLOAT</i>                                                                                       | 4-byte standard IEEE floating-point type.                               |
| <i>FMSTR_FLAGS</i>                                                                                       | Data type forming a union with a structure of flag bit-fields.          |
| <i>FMSTR_SIZE8</i>                                                                                       | Data type holding a general size value, at least 8 bits wide.           |
| <i>FMSTR_INDEX</i>                                                                                       | General for-loop index. Must be signed, at least 16 bits wide.          |
| <i>FMSTR_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>FMSTR_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**