



# MCUXpresso SDK Documentation

Release 25.06.00



NXP  
Jun 26, 2025



# Table of contents

|          |                                                                                     |            |
|----------|-------------------------------------------------------------------------------------|------------|
| <b>1</b> | <b>MIMXRT685-EVK</b>                                                                | <b>3</b>   |
| 1.1      | Overview                                                                            | 3          |
| 1.2      | Getting Started with MCUXpresso SDK Package                                         | 3          |
| 1.2.1    | Getting Started with Package                                                        | 3          |
| 1.3      | Getting Started with MCUXpresso SDK GitHub                                          | 43         |
| 1.3.1    | Getting Started with MCUXpresso SDK Repository                                      | 44         |
| 1.4      | Getting Started with MCUXpresso SDK Explorer                                        | 56         |
| 1.4.1    | Getting Started with MCUXpresso SDK Explorer                                        | 56         |
| 1.5      | Release Notes                                                                       | 100        |
| 1.5.1    | MCUXpresso SDK Release Notes                                                        | 100        |
| 1.6      | ChangeLog                                                                           | 107        |
| 1.6.1    | MCUXpresso SDK Changelog                                                            | 107        |
| 1.7      | Driver API Reference Manual                                                         | 176        |
| 1.8      | Middleware Documentation                                                            | 176        |
| 1.8.1    | Multicore                                                                           | 176        |
| 1.8.2    | MCU Boot                                                                            | 176        |
| 1.8.3    | eIQ                                                                                 | 176        |
| 1.8.4    | FreeMASTER                                                                          | 177        |
| 1.8.5    | AWS IoT                                                                             | 177        |
| 1.8.6    | NXP Wi-Fi                                                                           | 177        |
| 1.8.7    | FreeRTOS                                                                            | 177        |
| 1.8.8    | Wireless EdgeFast Bluetooth PAL                                                     | 177        |
| 1.8.9    | lwIP                                                                                | 177        |
| 1.8.10   | File systemFatfs                                                                    | 177        |
| 1.8.11   | DSP Audio Streamer                                                                  | 177        |
| <b>2</b> | <b>MIMXRT685S</b>                                                                   | <b>179</b> |
| 2.1      | ACMP: Analog Comparator Driver                                                      | 179        |
| 2.2      | CACHE: CACHE Memory Controller                                                      | 187        |
| 2.3      | CASPER: The Cryptographic Accelerator and Signal Processing Engine with RAM sharing | 191        |
| 2.4      | casper_driver                                                                       | 191        |
| 2.5      | casper_driver_pkha                                                                  | 195        |
| 2.6      | Clock Driver                                                                        | 197        |
| 2.7      | CRC: Cyclic Redundancy Check Driver                                                 | 228        |
| 2.8      | CTIMER: Standard counter/timers                                                     | 231        |
| 2.9      | DMA: Direct Memory Access Controller Driver                                         | 240        |
| 2.10     | DMIC: Digital Microphone                                                            | 257        |
| 2.11     | DMIC DMA Driver                                                                     | 257        |
| 2.12     | DMIC Driver                                                                         | 259        |
| 2.13     | DSP Driver                                                                          | 269        |
| 2.14     | FLEXCOMM: FLEXCOMM Driver                                                           | 269        |
| 2.15     | FLEXCOMM Driver                                                                     | 269        |
| 2.16     | FLEXSPI: Flexible Serial Peripheral Interface Driver                                | 270        |
| 2.17     | FLEXSPI DMA Driver                                                                  | 287        |
| 2.18     | FMEAS: Frequency Measure Driver                                                     | 289        |

|          |                                                                       |            |
|----------|-----------------------------------------------------------------------|------------|
| 2.19     | Hashcrypt: The Cryptographic Accelerator                              | 290        |
| 2.20     | Hashcrypt Background HASH                                             | 290        |
| 2.21     | Hashcrypt common functions                                            | 291        |
| 2.22     | Hashcrypt AES                                                         | 293        |
| 2.23     | Hashcrypt HASH                                                        | 297        |
| 2.24     | I2C: Inter-Integrated Circuit Driver                                  | 299        |
| 2.25     | I2C DMA Driver                                                        | 299        |
| 2.26     | I2C Driver                                                            | 301        |
| 2.27     | I2C Master Driver                                                     | 304        |
| 2.28     | I2C Slave Driver                                                      | 314        |
| 2.29     | I2S: I2S Driver                                                       | 323        |
| 2.30     | I2S_BRIDGE: I2S bridging and signal sharing configuration             | 323        |
| 2.31     | I2S DMA Driver                                                        | 325        |
| 2.32     | I2S Driver                                                            | 329        |
| 2.33     | I3C: I3C Driver                                                       | 338        |
| 2.34     | I3C Common Driver                                                     | 340        |
| 2.35     | I3C Master DMA Driver                                                 | 342        |
| 2.36     | I3C Master Driver                                                     | 345        |
| 2.37     | I3C Slave DMA Driver                                                  | 371        |
| 2.38     | I3C Slave Driver                                                      | 373        |
| 2.39     | IAP Boot Driver                                                       | 386        |
| 2.40     | IAP: In Application Programming Driver                                | 387        |
| 2.41     | IAP FlexSPI Driver                                                    | 387        |
| 2.42     | IAP OTP Driver                                                        | 400        |
| 2.43     | INPUTMUX: Input Multiplexing Driver                                   | 402        |
| 2.44     | IOPCTL: Input/Output Pad Controller                                   | 418        |
| 2.45     | Common Driver                                                         | 420        |
| 2.46     | LPADC: 12-bit SAR Analog-to-Digital Converter Driver                  | 432        |
| 2.47     | GPIO: General Purpose I/O                                             | 450        |
| 2.48     | MRT: Multi-Rate Timer                                                 | 453        |
| 2.49     | MU: Messaging Unit                                                    | 458        |
| 2.50     | OSTIMER: OS Event Timer Driver                                        | 465        |
| 2.51     | OTFAD: On The Fly AES-128 Decryption Driver                           | 468        |
| 2.52     | PINT: Pin Interrupt and Pattern Match Driver                          | 472        |
| 2.53     | Power Driver                                                          | 480        |
| 2.54     | POWERQUAD: PowerQuad hardware accelerator                             | 492        |
| 2.55     | PUF: Physical Unclonable Function                                     | 522        |
| 2.56     | Reset Driver                                                          | 524        |
| 2.57     | RTC: Real Time Clock                                                  | 529        |
| 2.58     | SCTimer: SCTimer/PWM (SCT)                                            | 535        |
| 2.59     | SEMA42: Hardware Semaphores Driver                                    | 551        |
| 2.60     | SPI: Serial Peripheral Interface Driver                               | 555        |
| 2.61     | SPI DMA Driver                                                        | 555        |
| 2.62     | SPI Driver                                                            | 559        |
| 2.63     | TRNG: True Random Number Generator                                    | 567        |
| 2.64     | USART: Universal Synchronous/Asynchronous Receiver/Transmitter Driver | 571        |
| 2.65     | USART DMA Driver                                                      | 571        |
| 2.66     | USART Driver                                                          | 574        |
| 2.67     | USDHC: Ultra Secured Digital Host Controller Driver                   | 590        |
| 2.68     | UTICK: MicroTick Timer Driver                                         | 619        |
| 2.69     | WWDT: Windowed Watchdog Timer Driver                                  | 621        |
| <b>3</b> | <b>Middleware</b>                                                     | <b>625</b> |
| 3.1      | Boot                                                                  | 625        |
| 3.1.1    | MCUXpresso SDK : mcuxsdk-middleware-mcuboot_opensource                | 625        |
| 3.1.2    | MCUboot                                                               | 626        |
| 3.2      | Motor Control                                                         | 627        |
| 3.2.1    | FreeMASTER                                                            | 627        |



|          |                                        |            |
|----------|----------------------------------------|------------|
| 3.3      | Multimedia . . . . .                   | 664        |
| 3.3.1    | Xtensa Audio Framework (XAF) . . . . . | 664        |
| <b>4</b> | <b>RTOS . . . . .</b>                  | <b>681</b> |
| 4.1      | FreeRTOS . . . . .                     | 681        |
| 4.1.1    | FreeRTOS kernel . . . . .              | 681        |
| 4.1.2    | FreeRTOS drivers . . . . .             | 681        |
| 4.1.3    | backoffalgorithm . . . . .             | 681        |
| 4.1.4    | corehttp . . . . .                     | 681        |
| 4.1.5    | corejson . . . . .                     | 681        |
| 4.1.6    | coremqtt . . . . .                     | 682        |
| 4.1.7    | coremqtt-agent . . . . .               | 682        |
| 4.1.8    | corepkcs11 . . . . .                   | 682        |
| 4.1.9    | freertos-plus-tcp . . . . .            | 682        |



This documentation contains information specific to the evkmimxrt685 board.

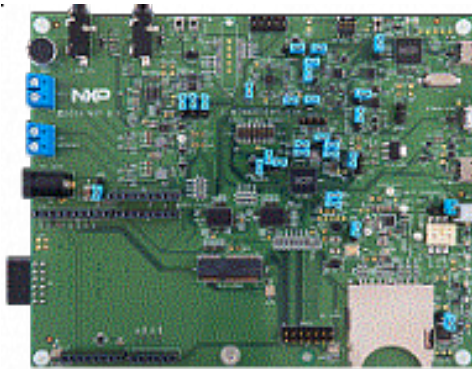


# Chapter 1

## MIMXRT685-EVK

### 1.1 Overview

The NXP MIMXRT685-EVK is a development board for the i.MX RT600 Crossover MCU with Arm Cortex-M33 and DSP Cores.



MCU device and part on board is shown below:

- Device: MIMXRT685S
- PartNumber: MIMXRT685SFVKB

### 1.2 Getting Started with MCUXpresso SDK Package

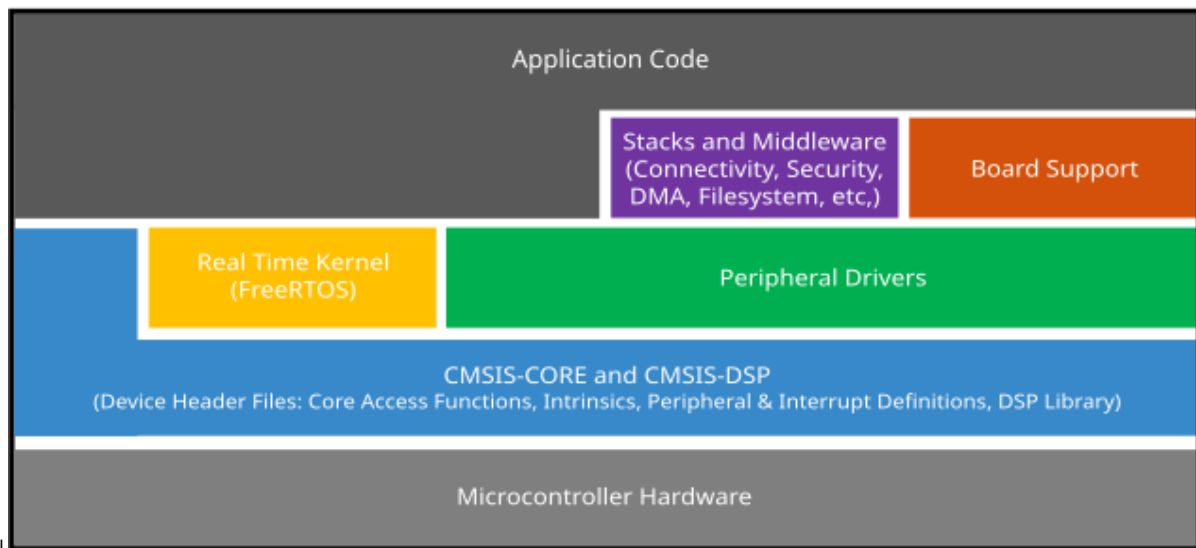
#### 1.2.1 Getting Started with Package

##### 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, a USB host and device stack, and various other middleware to support rapid development.

For supported toolchain versions, see *MCUXpresso SDK Release Notes for EVK-MIMXRT685* (document MCUXSDKMIMXRT6XXRN).

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



## MCUXpresso SDK board support package folders

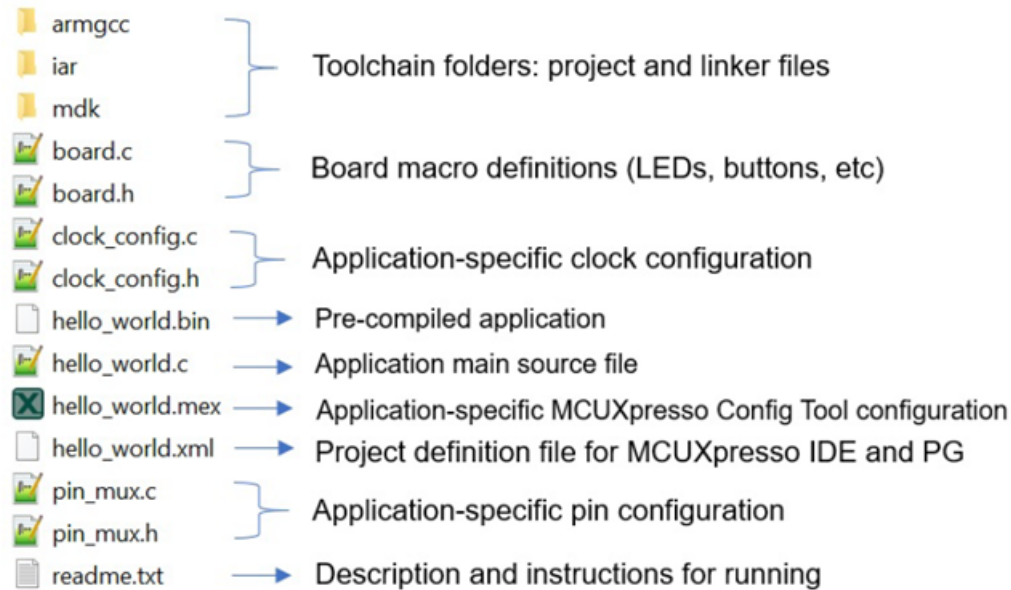
MCUXpresso SDK board support package provides example applications for NXP development and evaluation boards for Arm Cortex-M cores including Freedom, Tower System, i.mxrt EVK boards, and LPCXpresso boards. Board support packages are found inside the top-level boards folder and each supported board has 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):

- `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).
- `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

**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.

**Parent topic:** [MCUXpresso SDK board support package folders](#)

**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>/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

For examples containing an RTOS, there are references to the appropriate source code. 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.

**Parent topic:** [MCUXpresso SDK board support package folders](#)

## 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 MIMXRT685-EVK

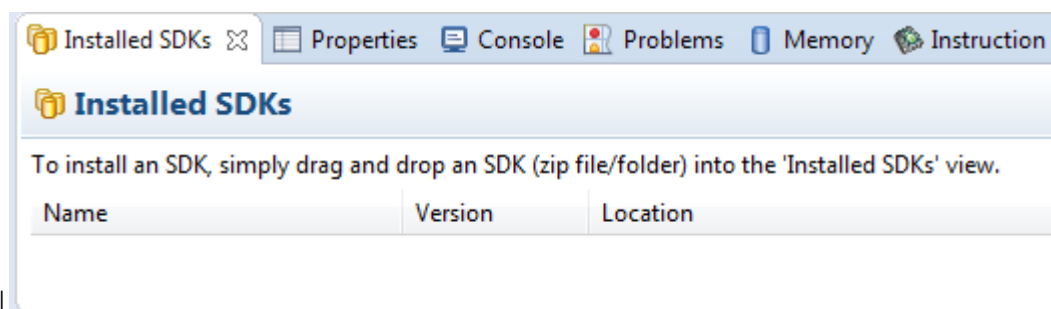
hardware platform is used as an example, though these steps can be applied to any example application in the MCUXpresso SDK.

**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.

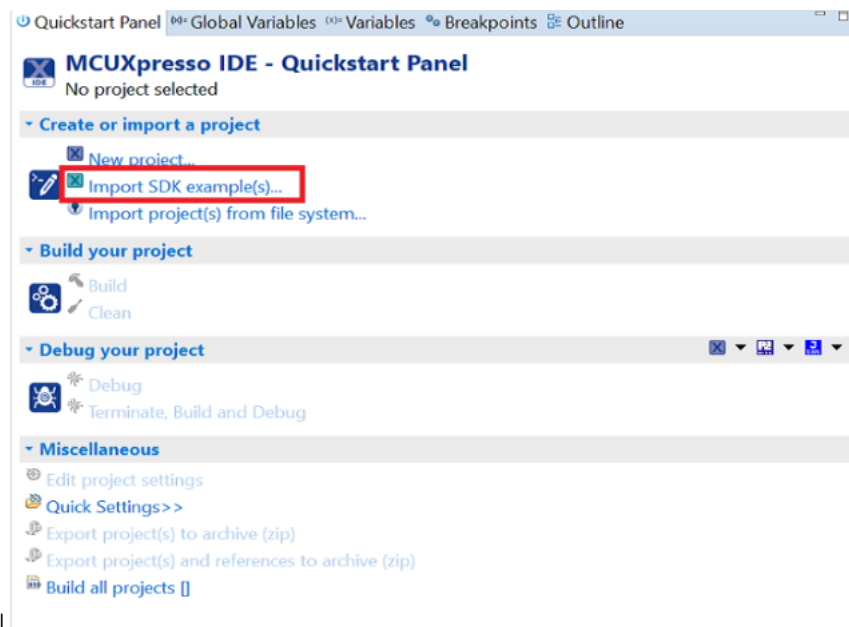
**Parent topic:**[Run a demo using MCUXpresso IDE](#)

**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.

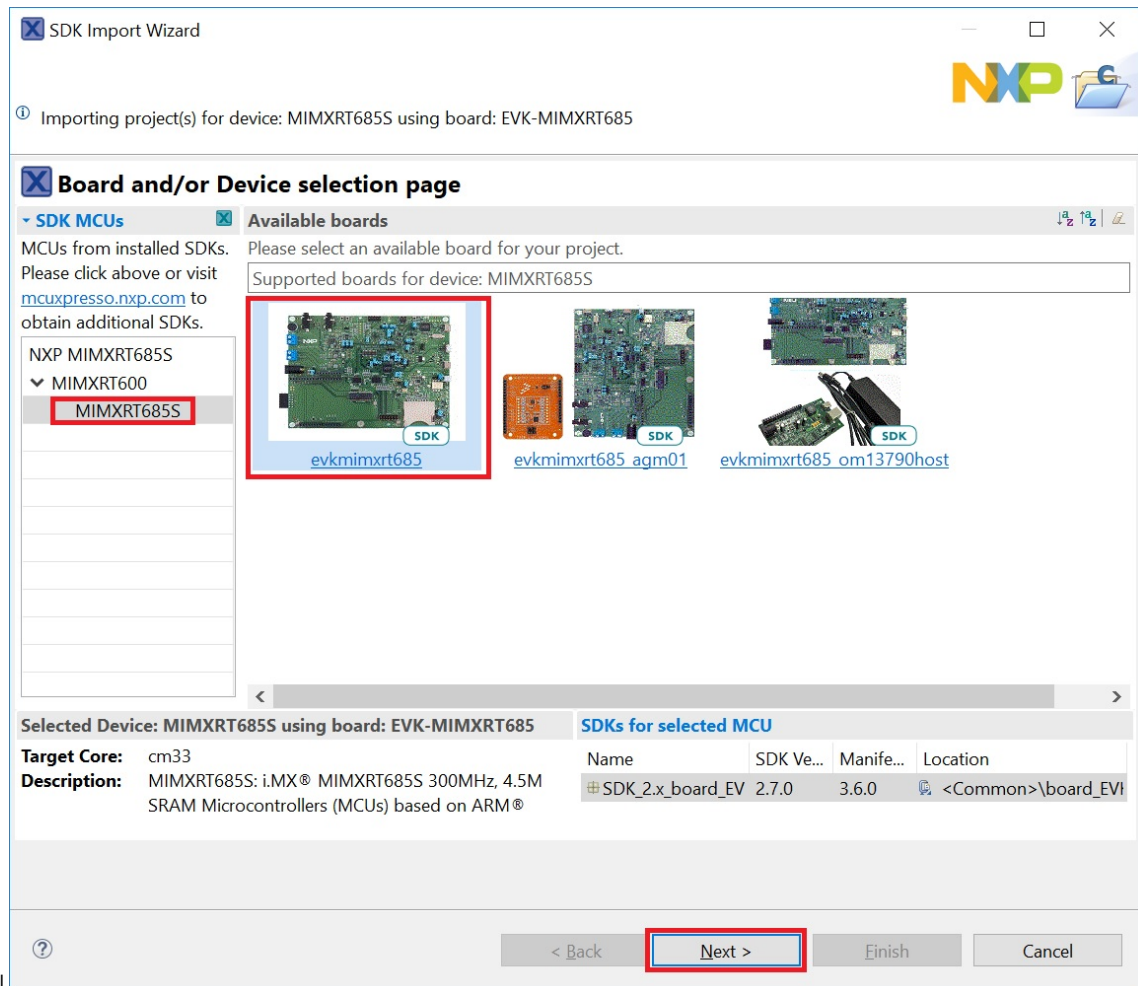


2. On the **Quickstart Panel**, click **Import SDK example(s)...**

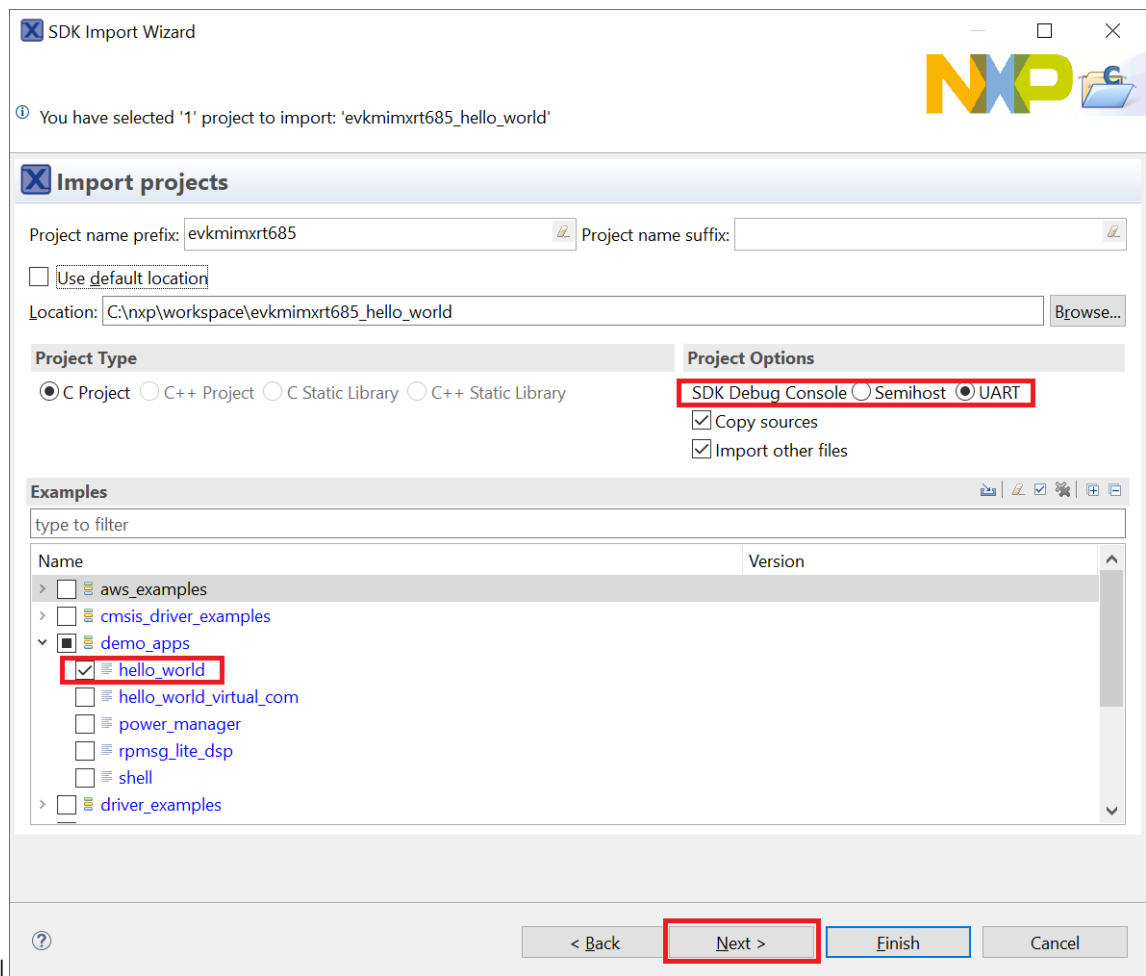


3. In the window that appears, expand the **MIMXRT600** folder and select **MIMXRT685S**. Then, select **evkmimxrt685** and click **Next**.

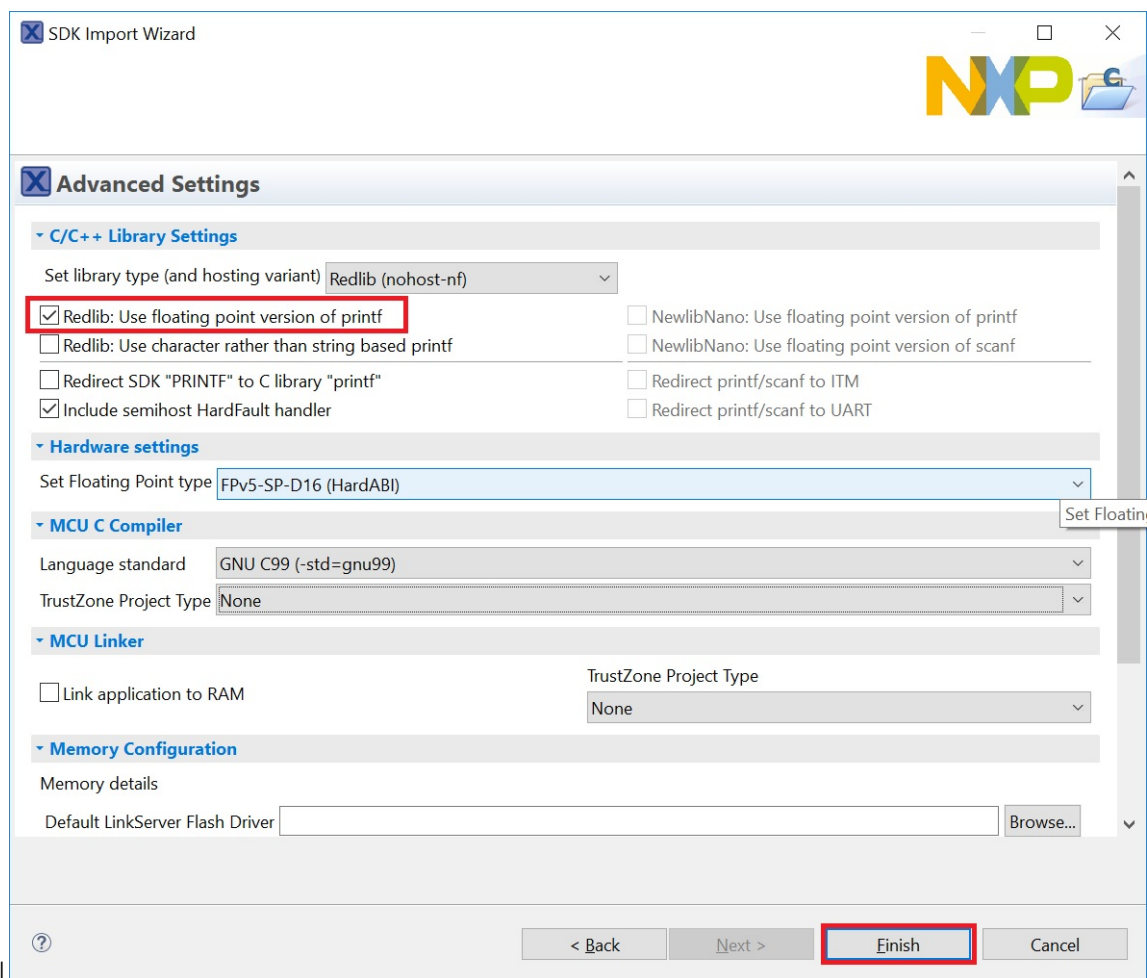




- Expand the demo\_apps folder and select hello\_world. Then, click **Next**.



5. Ensure **Redlib: Use floating-point version of printf** is selected if the example prints floating-point numbers on the terminal. Otherwise, it is not necessary to select this option. Then, click **Finish**.



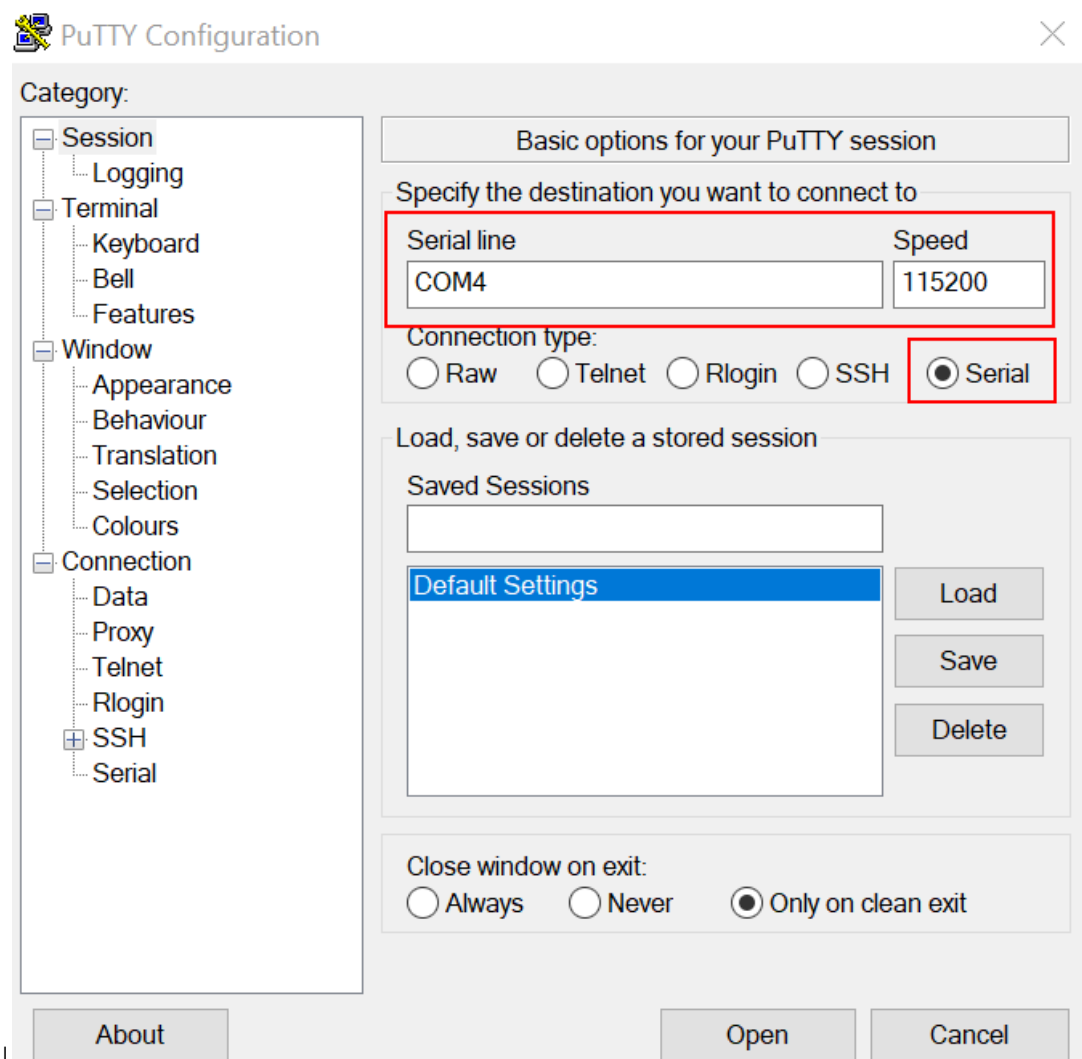
Parent topic: [Run a demo using MCUXpresso IDE](#)

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

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

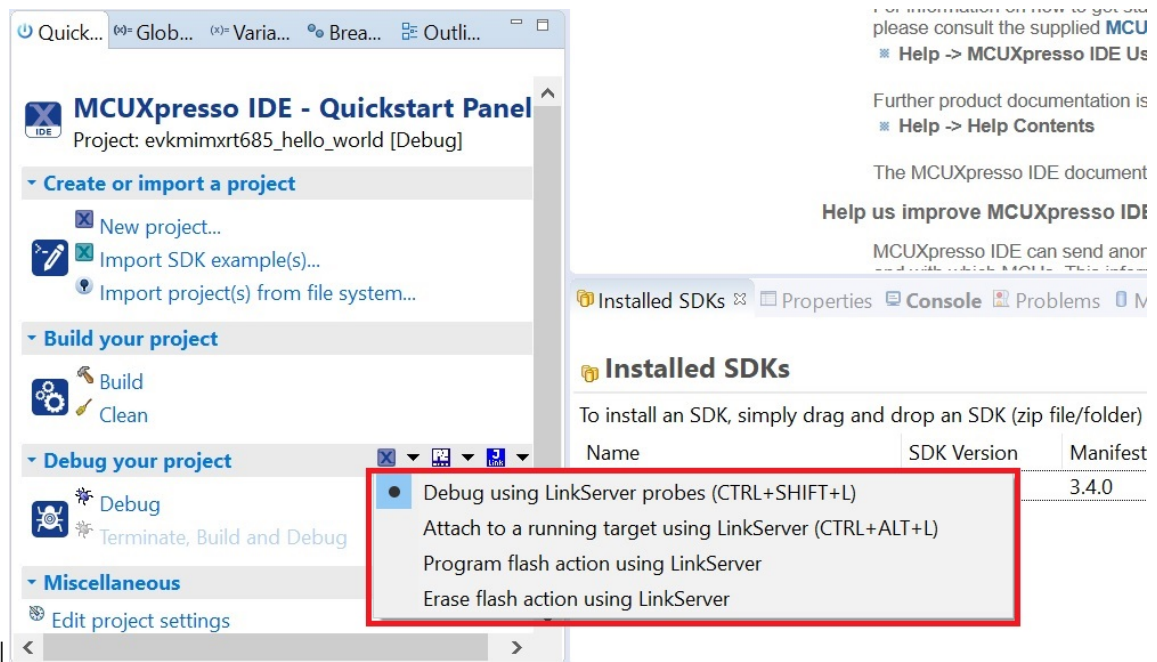
1. See the table in [Default debug interfaces](#) to determine the debug interface that comes loaded on your specific hardware platform.
  - For boards with CMSIS-DAP/mbd/DAPLink interfaces, visit [developer.mbed.org/handbook/Windows-serial-configuration](https://developer.mbed.org/handbook/Windows-serial-configuration) 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.
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

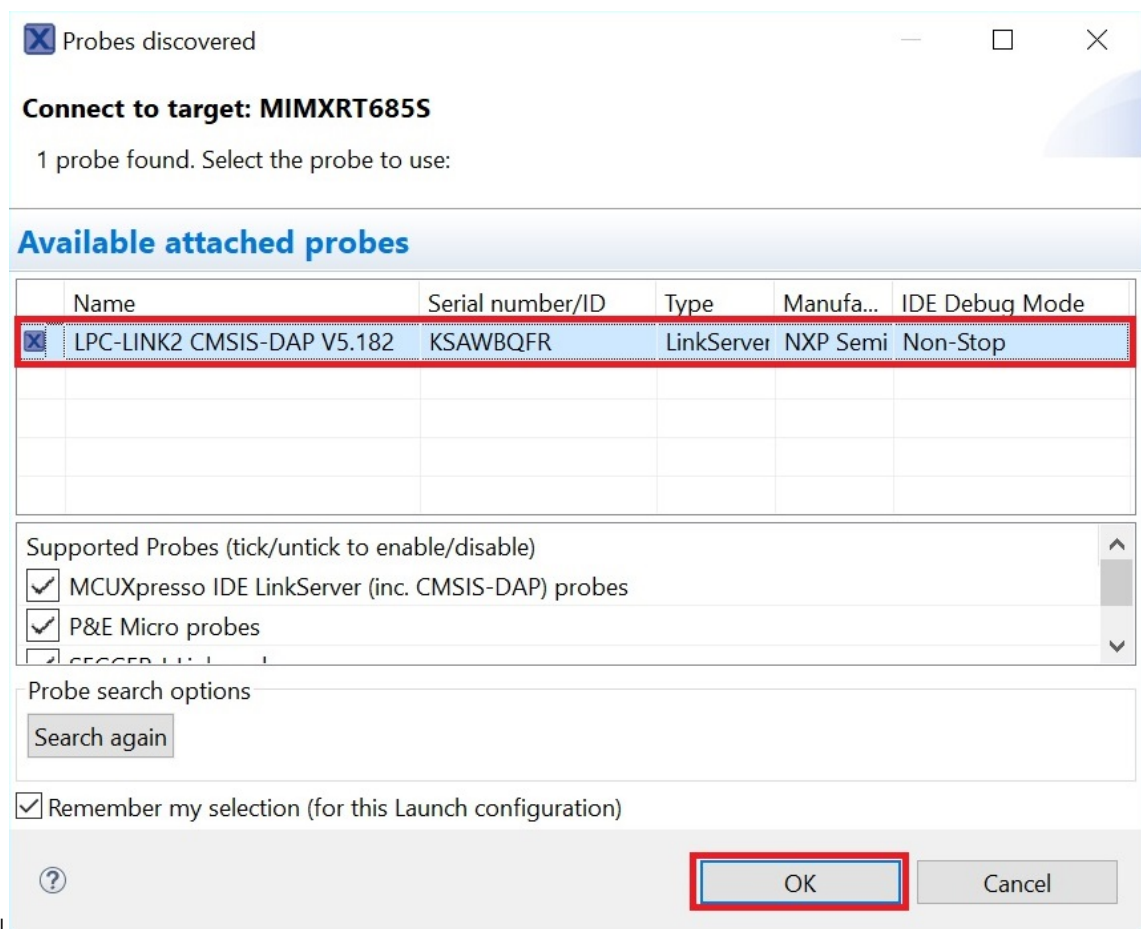


4. 1 stop bit

4. On the **Quickstart Panel**, click **Debug** evkmimxrt685\_hello\_world [Debug] to launch the debug session.



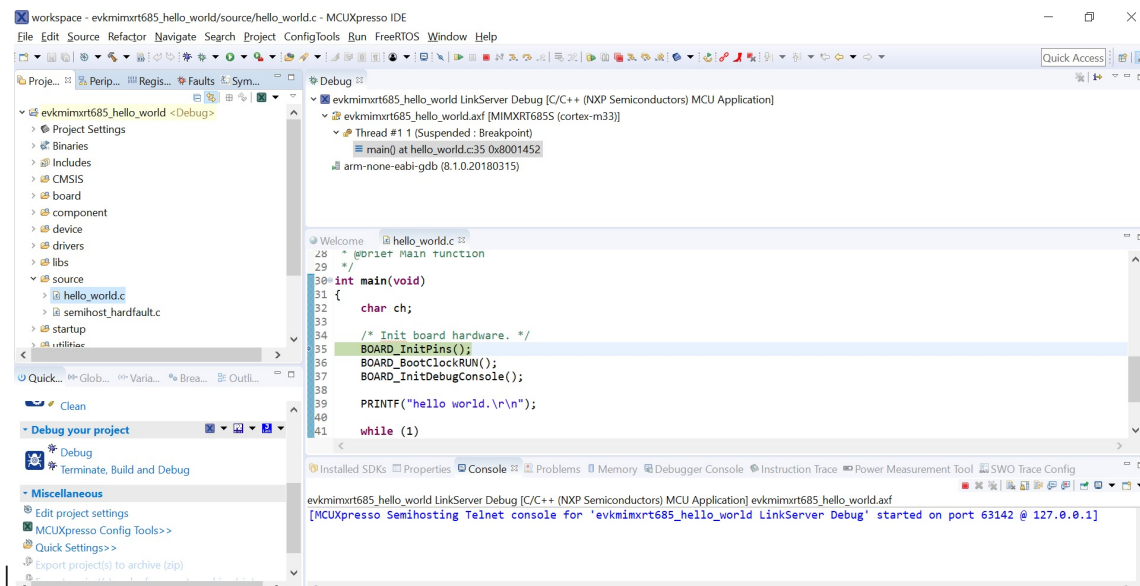
- The first time you debug a project, the **Debug Emulator Selection** dialog is displayed, showing all supported probes that are attached to your computer. Select the probe through which you want to debug and click **OK**. (For any future debug sessions, the stored probe selection is automatically used,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



**\*\*Note:\*\*** If the debug probe is CMSIS-DAP and the debugging application is running in flash, make sure that the board is set to FlexSPI flash boot mode (ISP2: ISP0 = ON, OFF, ON). Otherwise, you should not set to FlexSPI boot mode when debugging the application. If the debug probe is J-Link, **\*\*Reset before running\*\*** must be disabled. Double click the `<example> J-Link Debug.launch` file, and deselect this option under **\*\*JLink Debugger ->Additional Options\*\*** menu.



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.

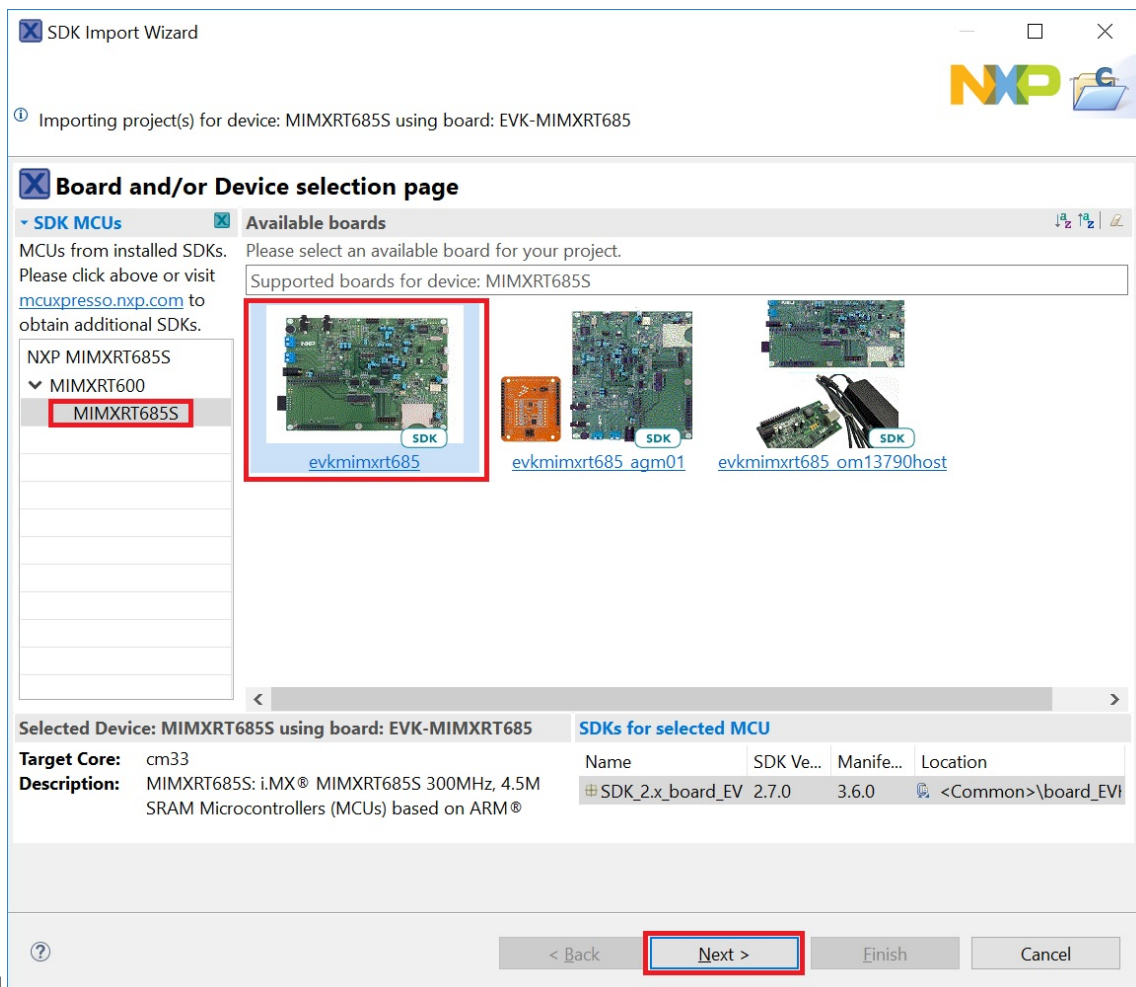




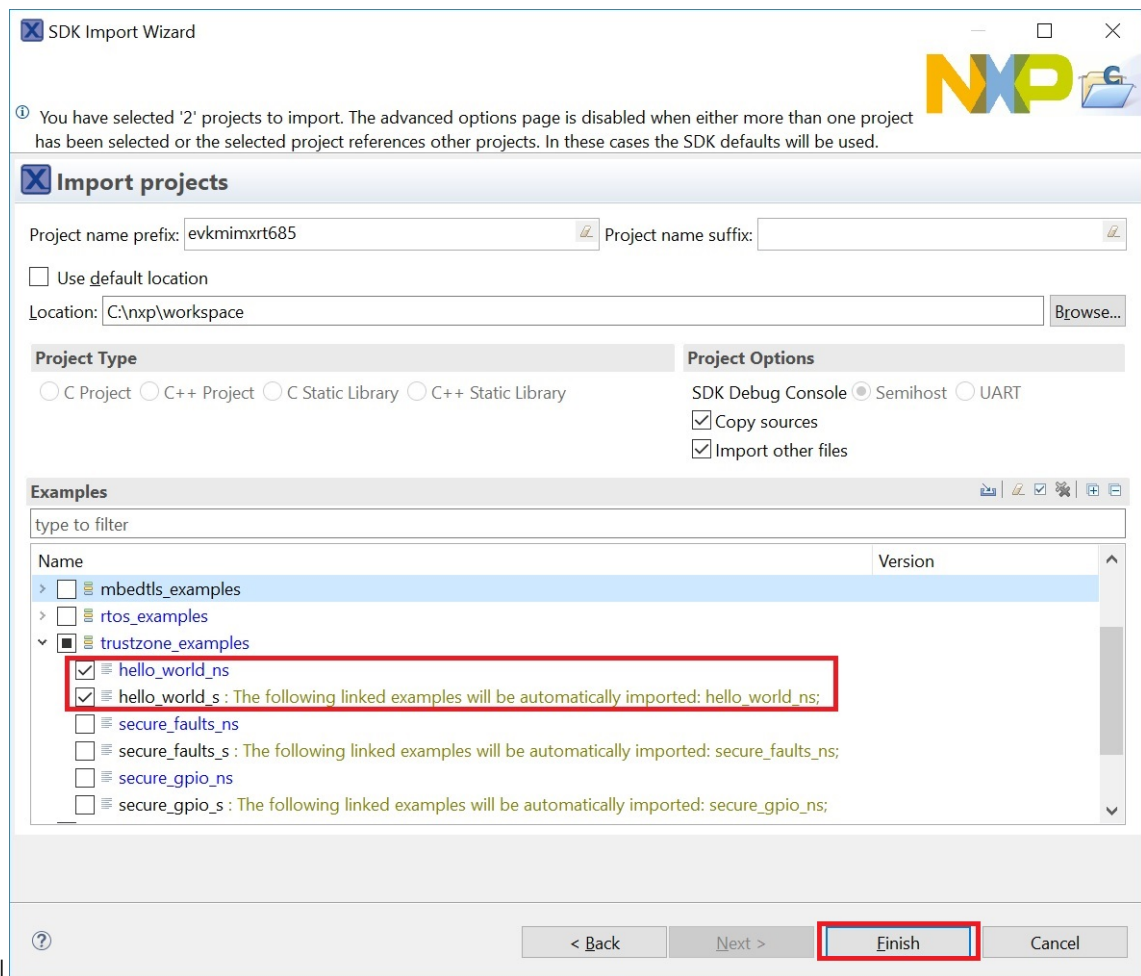
Parent topic: [Run a demo using MCUXpresso IDE](#)

**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 MIMXRT685-EVK hardware platform is used as an example, though these steps can be applied to any TrustZone example application in the MCUXpresso SDK.

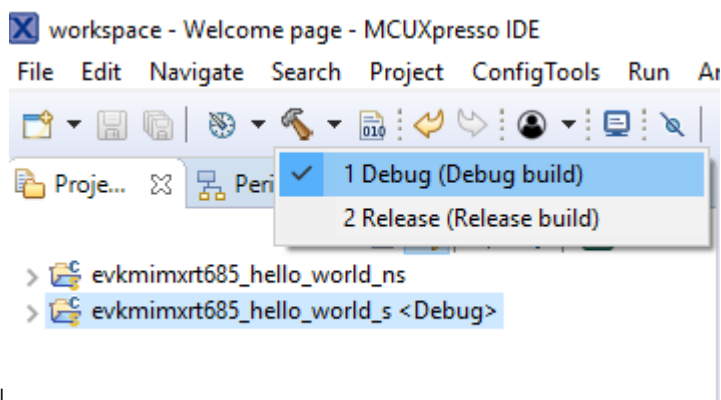
1. TrustZone examples are imported into the workspace in a similar way as single core applications. When the SDK zip package for MIMXRT685-EVK is installed and available in the **Installed SDKs** view, click **Import SDK example(s)...** on the Quickstart Panel. In the window that appears, expand the **MIMXRT600** folder and select **MIMXRT685S**. Then, select **evkmimxrt685** and click **Next**.



2. Expand the trustzone\_examples/ folder and select hello\_world\_s. Because TrustZone examples are linked together, the non-secure project is automatically imported with the secure project, and there is no need to select it explicitly. Then, click **Finish**.



- Now, two projects should be imported into the workspace. To start building the TrustZone application, highlight the `evkmimxrt685\_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 Figure 3. 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.

(continues on next page)



(continued from previous page)

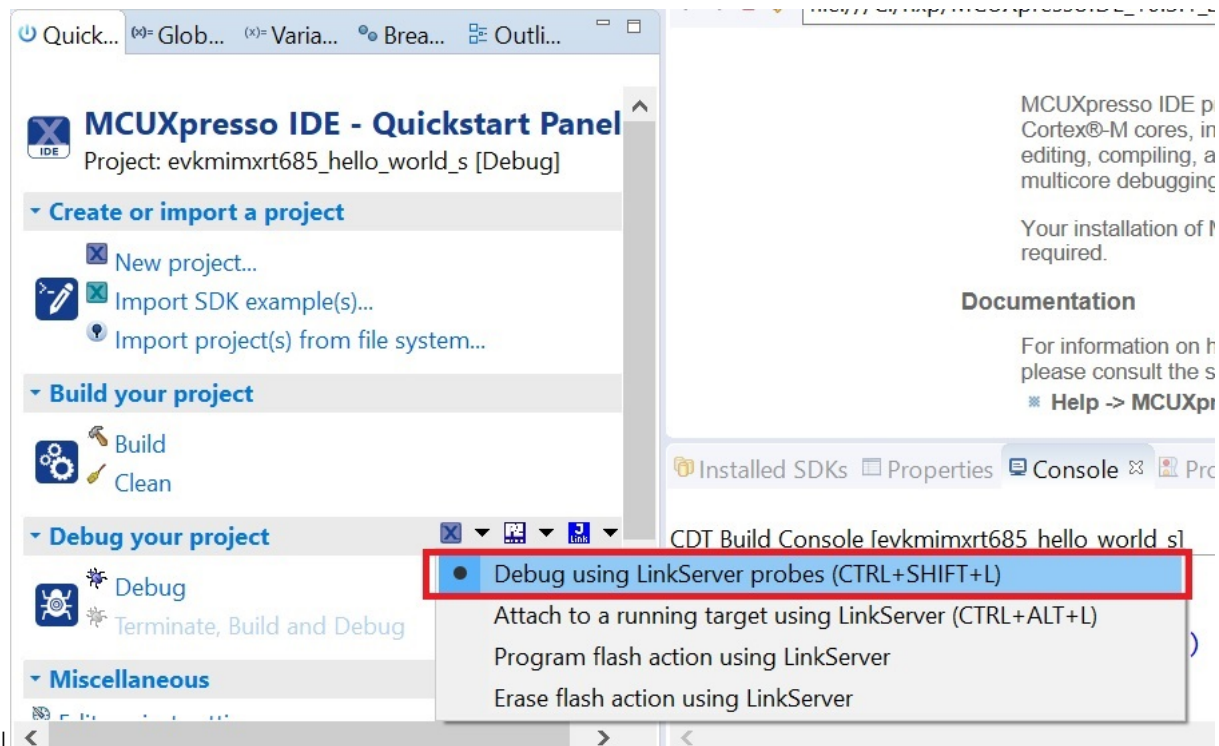
**Note:** When the **Release** build is requested, it is necessary to change the build configuration of both the secure and non-secure application projects first. To do this, select both projects in the Project Explorer view by clicking to select the first project, then using shift-click or control-click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

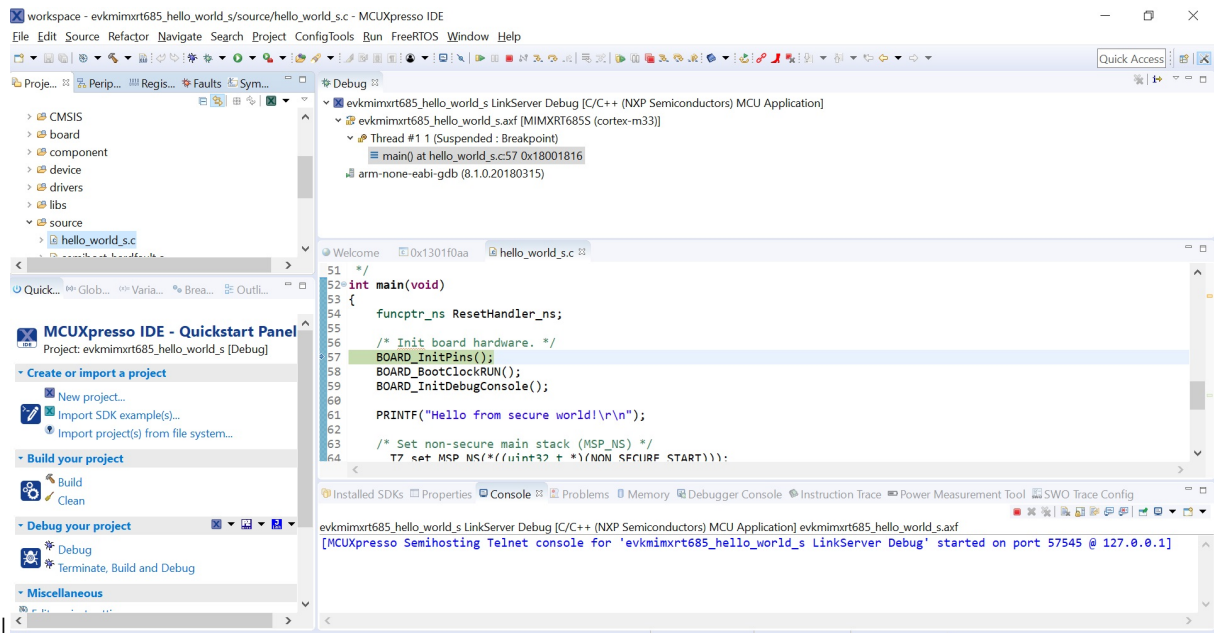
Switching TrustZone projects into the Release build configuration")

Parent topic: [Run a demo using MCUXpresso IDE](#)

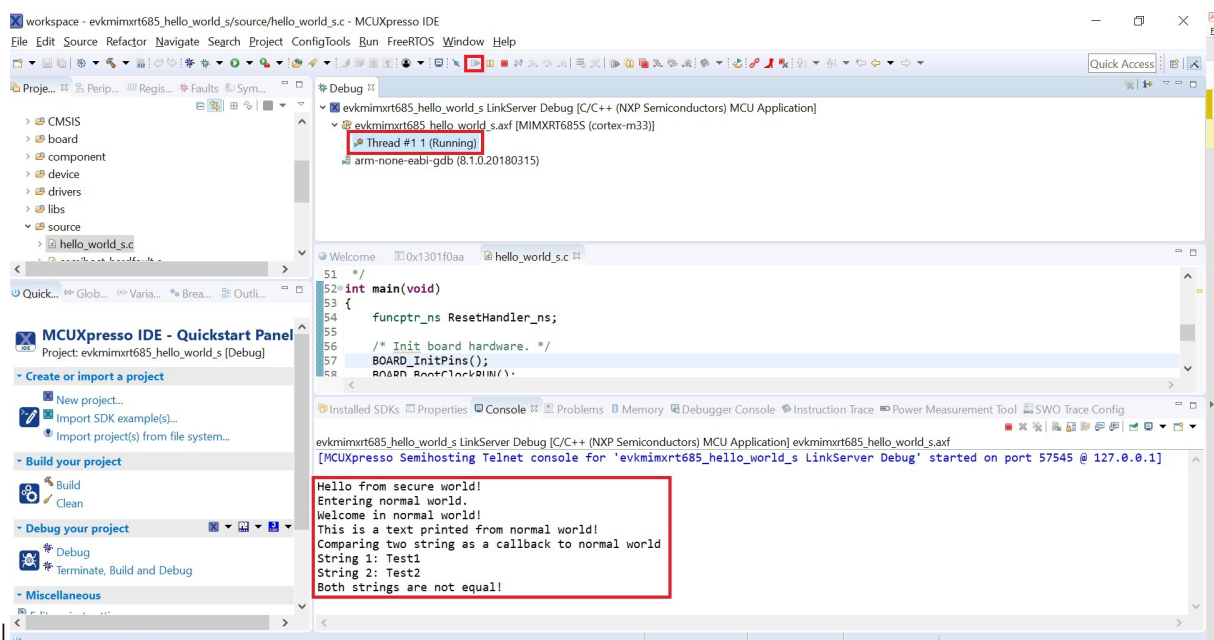
**Run a TrustZone example application** To download and run the application, perform all steps as described in Section 3.3, “Run an example application”. These steps are common for single core, and TrustZone applications, ensuring evkmimxrt685\_hello\_world\_s is selected for debugging.

In the Quickstart Panel, click **Debug evkmimxrt685\_hello\_world\_s [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.



Parent topic: [Run a demo using MCUXpresso IDE](#)

## 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 8.40.2 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* (document ID: MCUXSDKRN).

**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
```

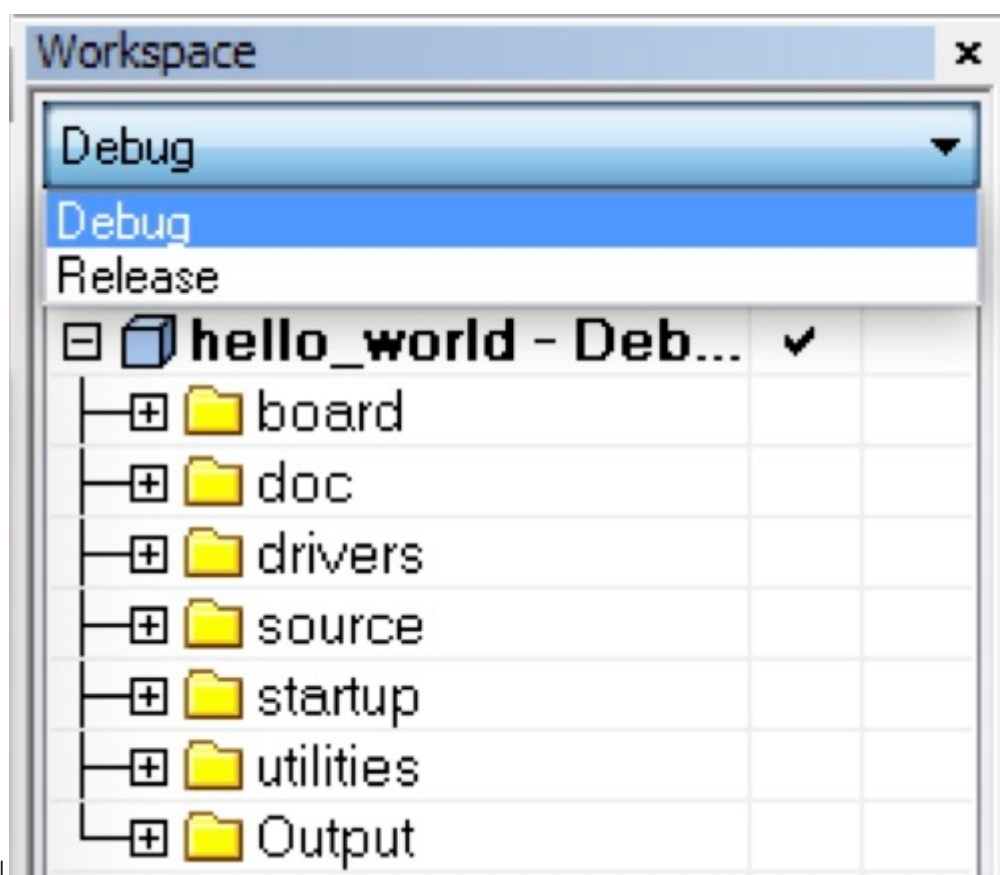
Using the MIMXRT685-EVK hardware platform as an example, the `hello_world` workspace is located in:

```
<install_dir>/boards/k32w061dk6/demo_apps/hello_world/iar/hello_world.eww
```

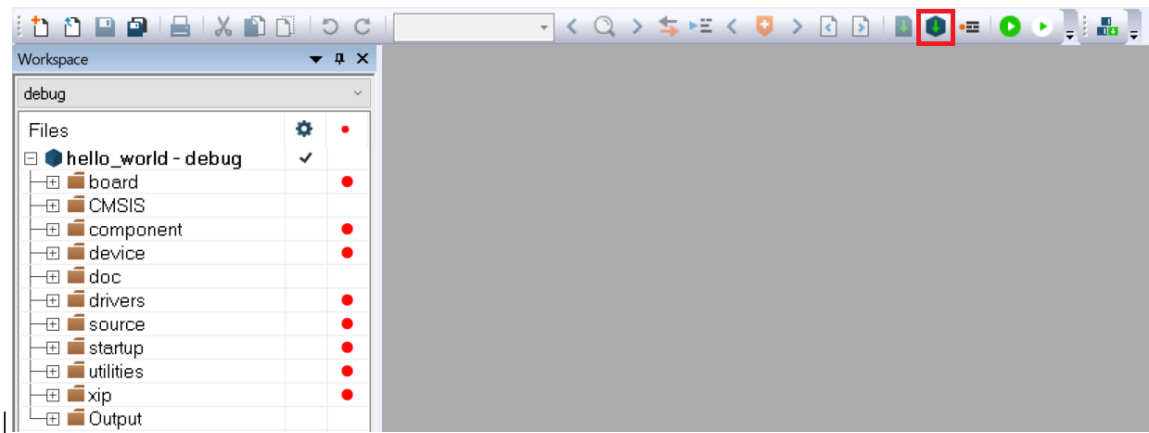
Other example applications may have additional folders in their path.

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 Figure 2.

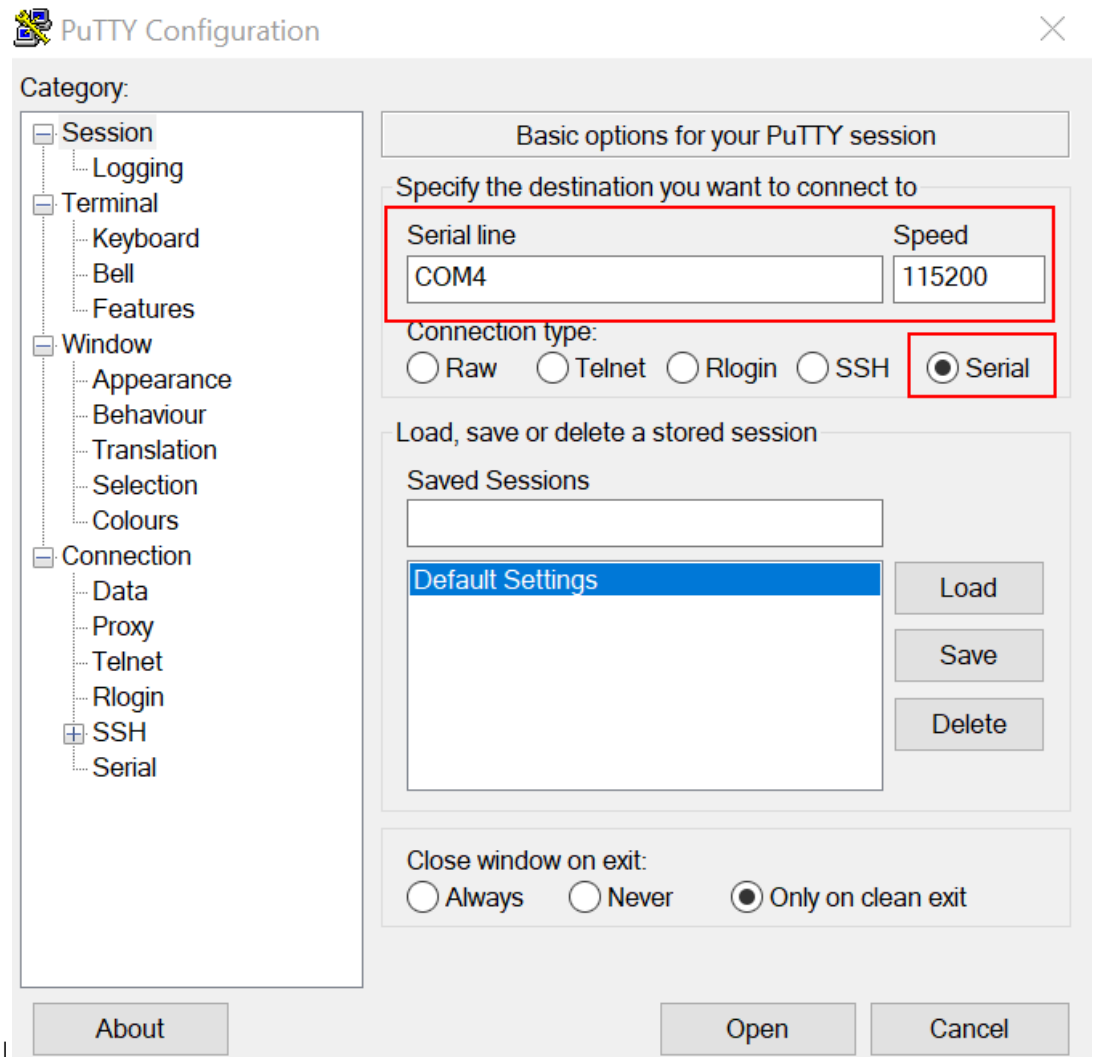


4. The build completes without errors.

**Parent topic:** [Run a demo application using IAR](#)

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

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

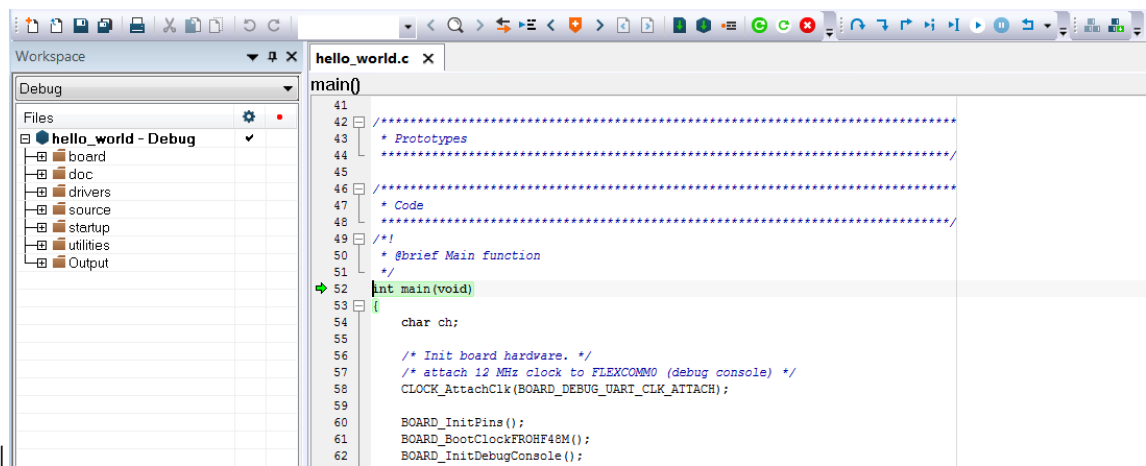


4. 1 stop bit

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.



Parent topic: [Run a demo application using IAR](#)

**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:

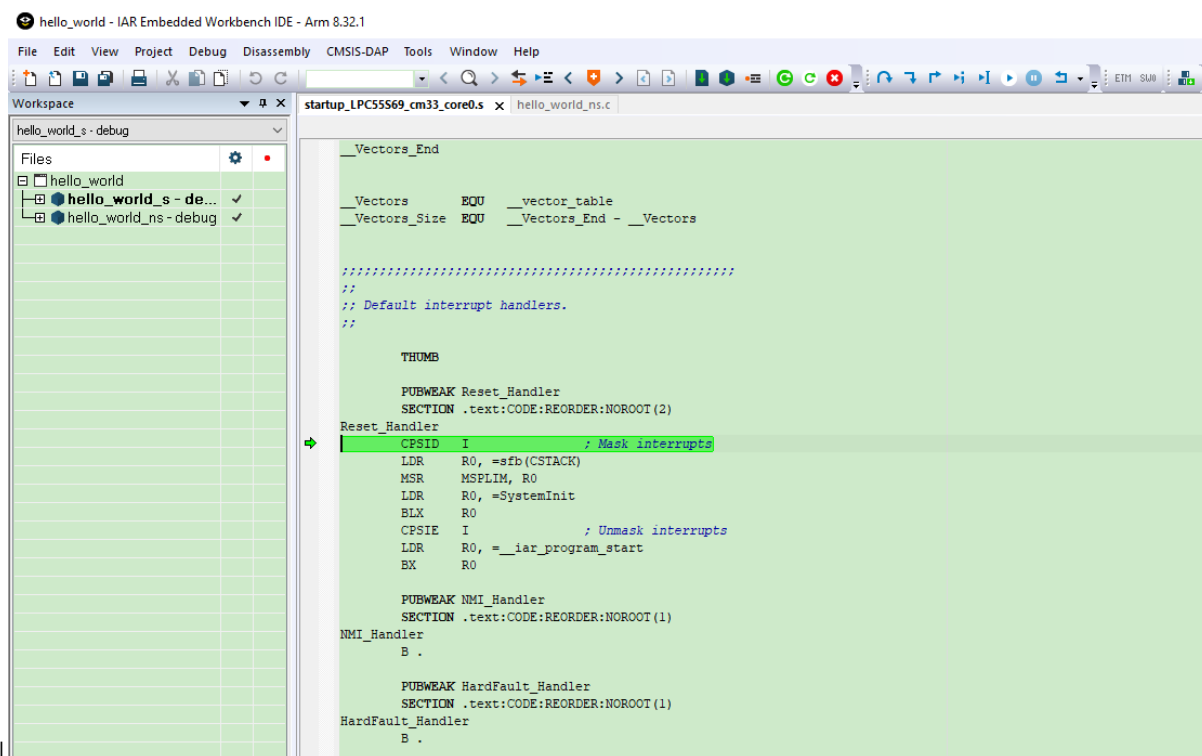
```
<install_dir>/boards/evkmimxrt685/trustzone_examples/hello_world/hello_world_s/iar/hello_world_s.  
→.eww
```

```
<install_dir>/boards/evkmimxrt685/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.

**Parent topic:** [Run a demo application using IAR](#)

**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 *Section 4.2, 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.

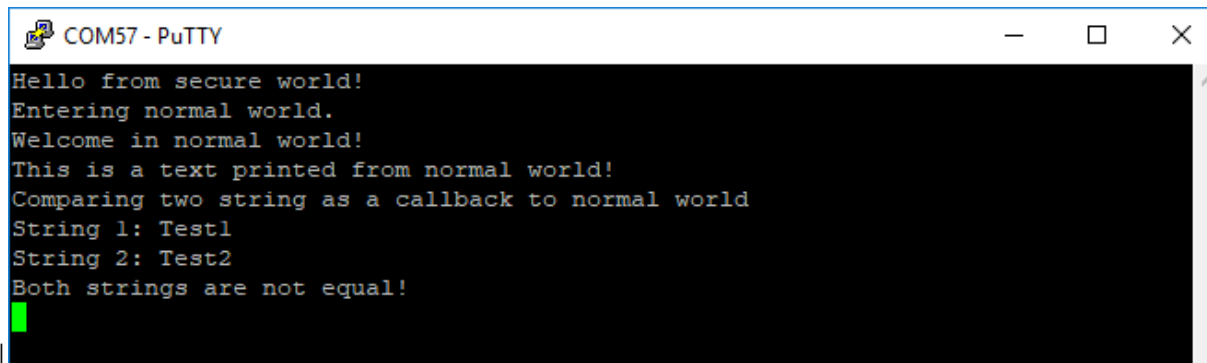


Run the code by clicking **Go** to start the application.



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.



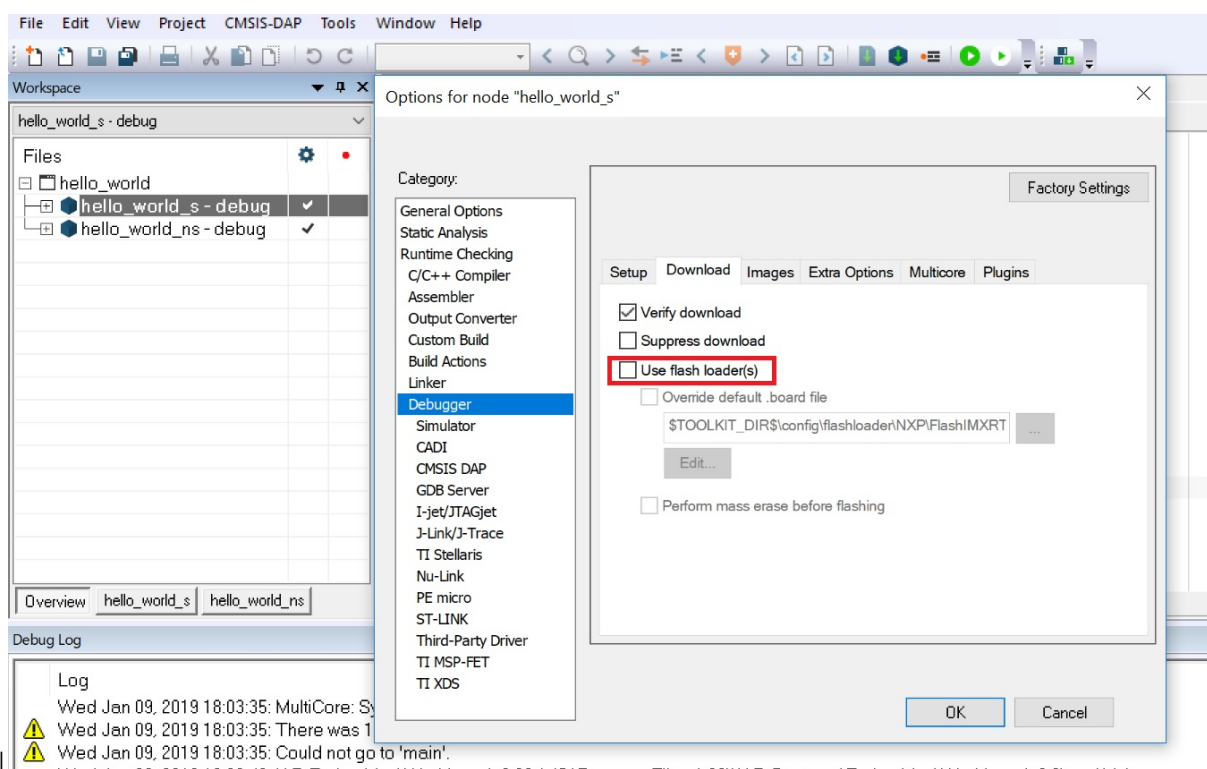


```

COM57 - PuTTY
Hello from secure world!
Entering normal world.
Welcome in normal world!
This is a text printed from normal world!
Comparing two string as a callback to normal world
String 1: Test1
String 2: Test2
Both strings are not equal!

```

**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.MXRT600.



**Parent topic:** [Run a demo application using IAR](#)

## 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 for the MIMXRT685-EVK hardware platform which is used as an example.

**Note:** ARMGCC version 8-2019-q3 is used as an example in this document. The latest GCC version for this package is as described in the *MCUXpresso SDK Release Notes Supporting MIMXRT685-EVK* (document MCUXSDKMIMXRT6XXRN).



**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*.

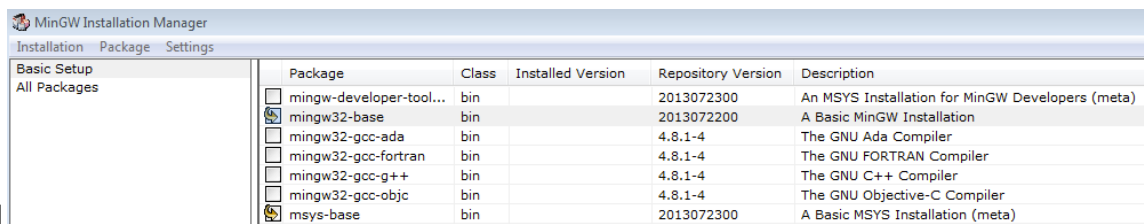
**Parent topic:**Set up toolchain

**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.

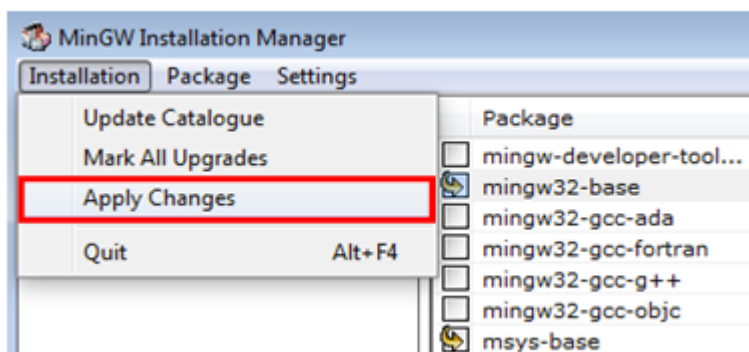
1. Download the latest MinGW mingw-get-setup installer from [MinGW](#).
2. Run the installer. The recommended installation path is C:\MinGW, however, you may install to any location.

**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.

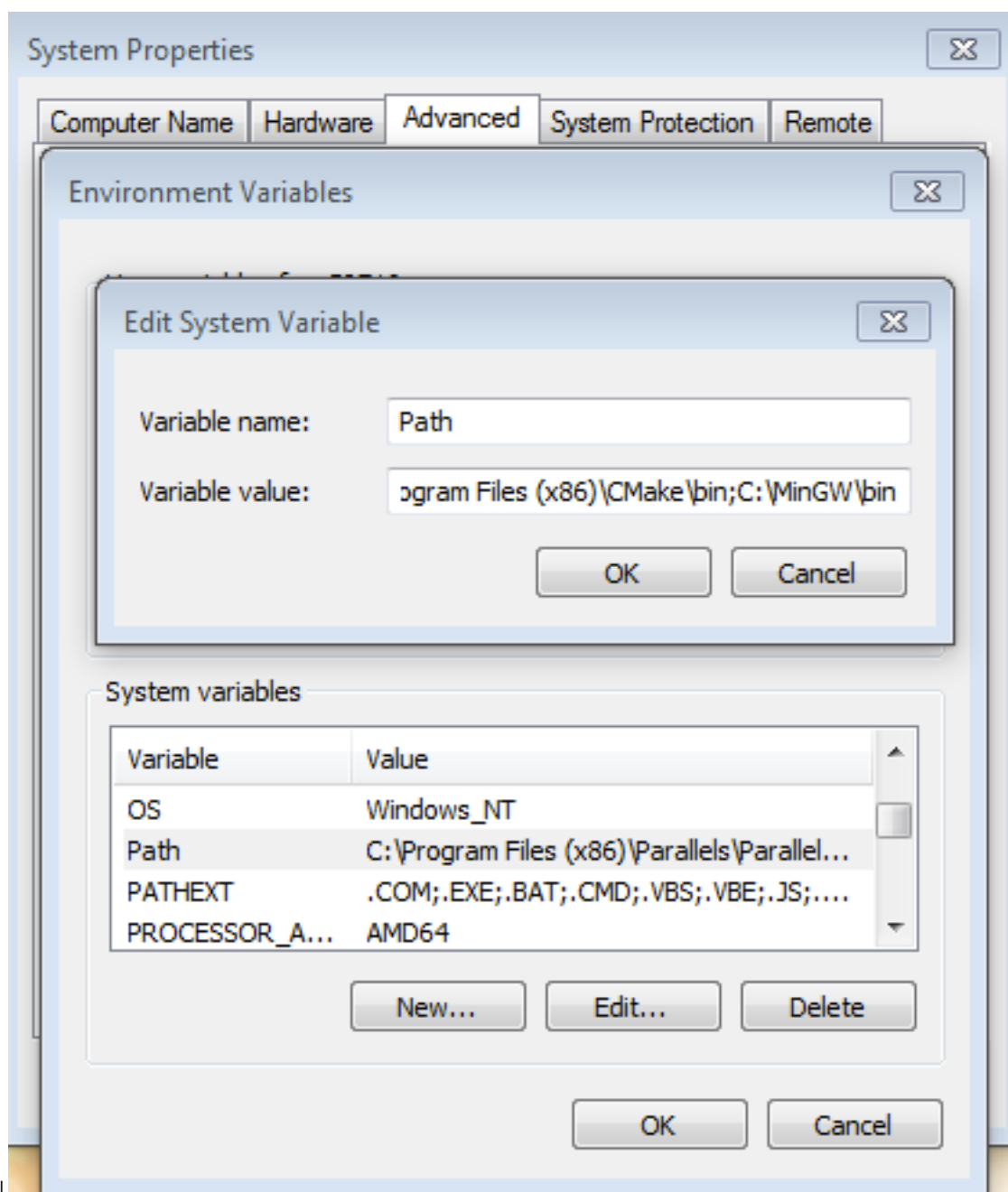


5. Add the appropriate item to the Windows operating system path environment variable. It can be found under **Control Panel->System and Security->System->Advanced System Settings** in the **Environment Variables...** section. The path is:

<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.



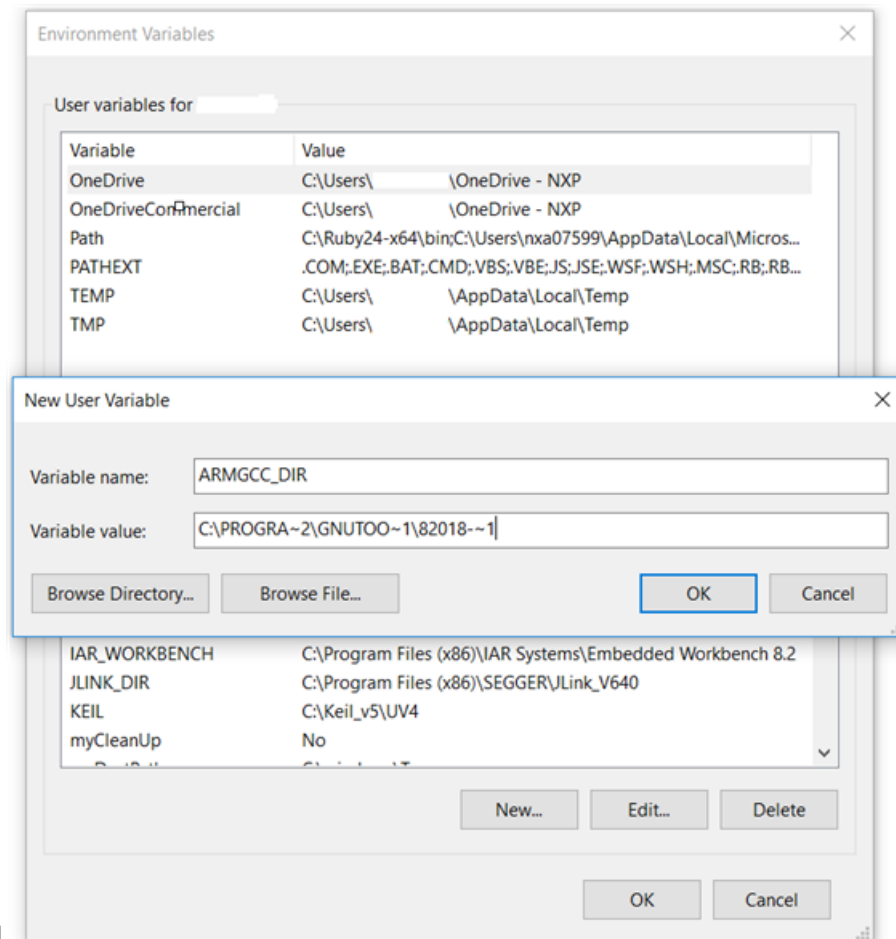
**Parent topic:** Set up toolchain

**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:

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

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

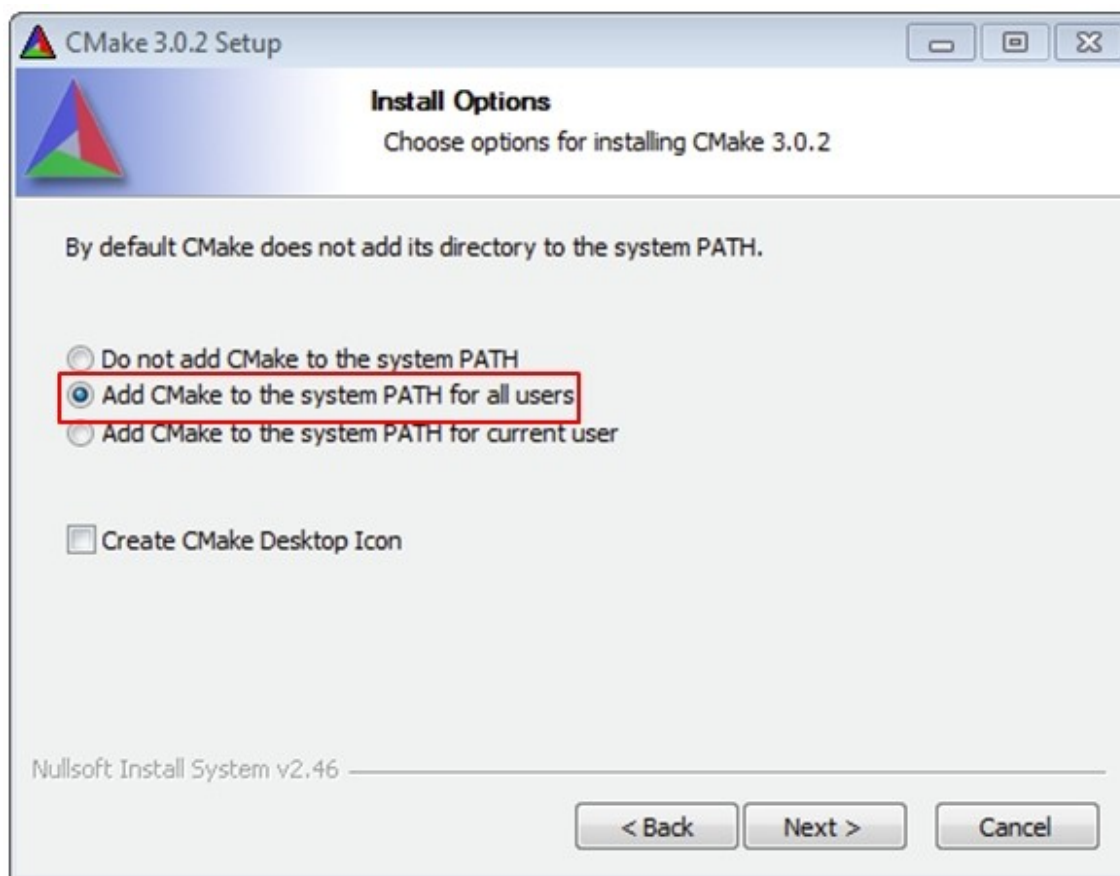
```
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>for %I in (.) do echo %~sI
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>echo C:\PROGRA~2\GNUTOO~1\82018~1
C:\PROGRA~2\GNUTOO~1\82018~1
```



**Parent topic:**Set up toolchain

## Install CMake

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.



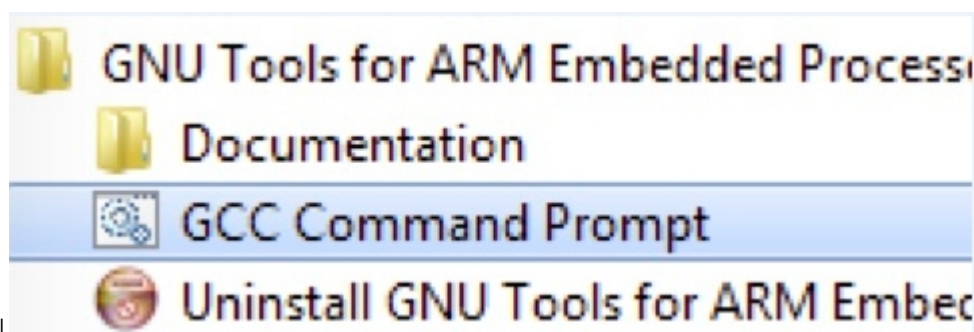
3. Follow the remaining instructions of the installer.
4. You may need to reboot your system for the PATH changes to take effect.
5. Make sure `sh.exe` is not in the Environment Variable PATH. This is a limitation of mingw32-make.

**Parent topic:**Set up toolchain

**Parent topic:**[Run a demo using Arm GCC](#)

**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:

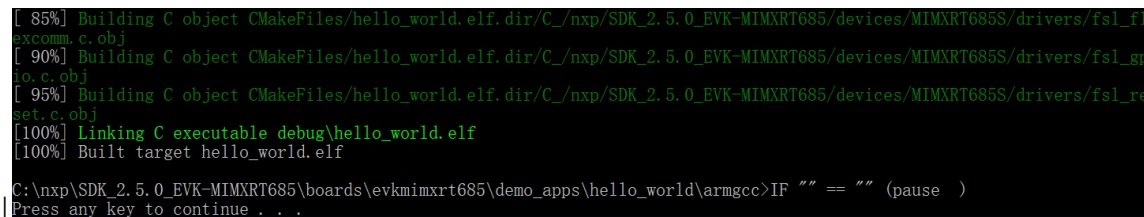
```
<install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc
```

For this example, the exact path is:

```
<install_dir>/examples/evkmimxrt685/demo_apps/hello_world/armgcc
```

**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 Figure 2.



```
[ 85%] Building C object CMakeFiles/hello_world.elf.dir/C:/npx/SDK_2.5.0_EVK-MIMXRT685/devices/MIMXRT685S/drivers/fsl_flexcomm.c.obj
[ 90%] Building C object CMakeFiles/hello_world.elf.dir/C:/npx/SDK_2.5.0_EVK-MIMXRT685/devices/MIMXRT685S/drivers/fsl_gpio.c.obj
[ 95%] Building C object CMakeFiles/hello_world.elf.dir/C:/npx/SDK_2.5.0_EVK-MIMXRT685/devices/MIMXRT685S/drivers/fsl_reset.c.obj
[100%] Linking C executable debug\hello_world.elf
[100%] Built target hello_world.elf
C:\npx\SDK_2.5.0_EVK-MIMXRT685\boards\evkmimxrt685\demo_apps\hello_world\armgcc>IF "" == "" (pause )
Press any key to continue . . .
```

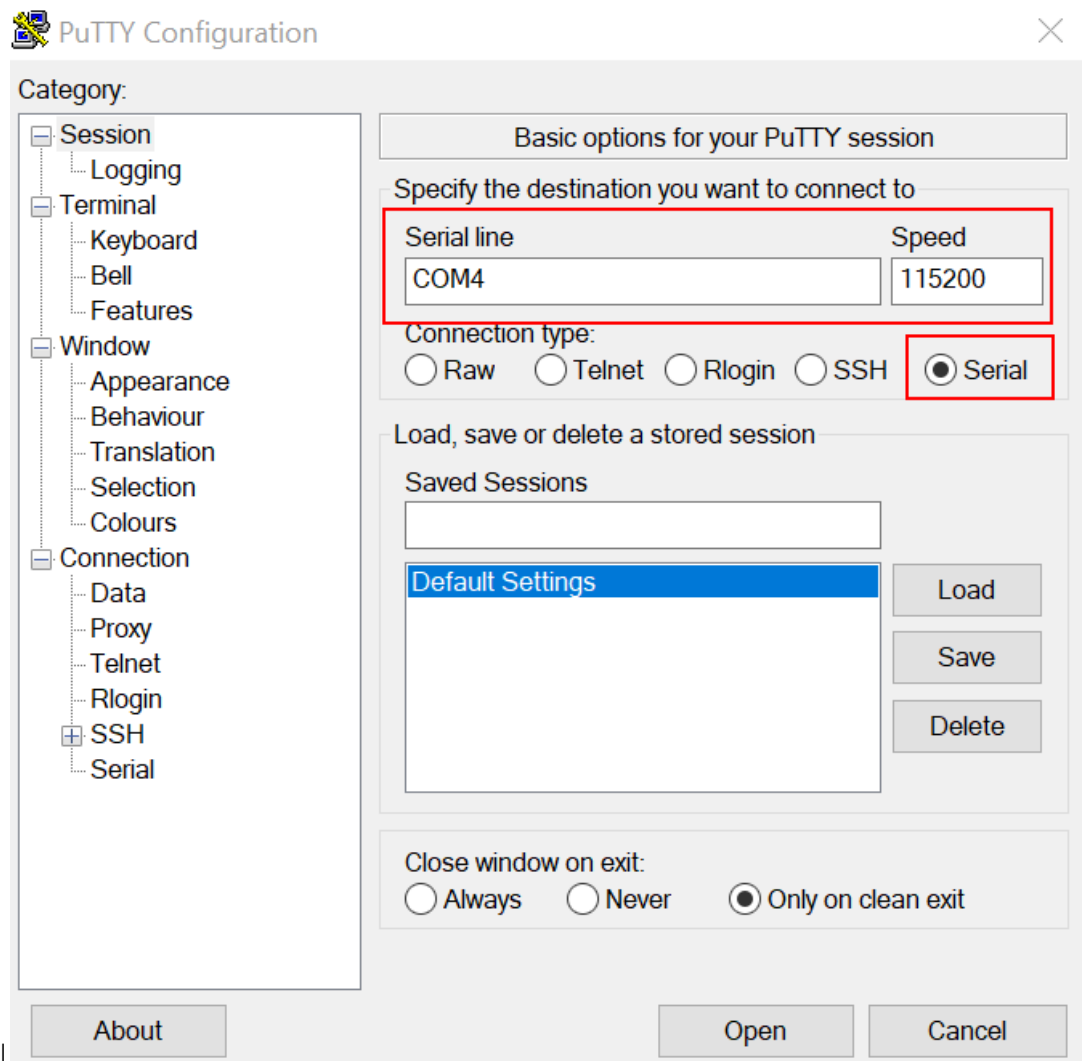
Parent topic: [Run a demo using Arm GCC](#)

**Run an example application** This section describes steps to run a demo application using J-Link GDB Server application. To update the on-board LPC-Link2 debugger to Jlink firmware, see [Updating debugger firmware](#).

**Note:** J-Link GDB Server application is not supported for TFM examples. Use CMSIS DAP instead of J-Link for flashing and debugging TFM examples.

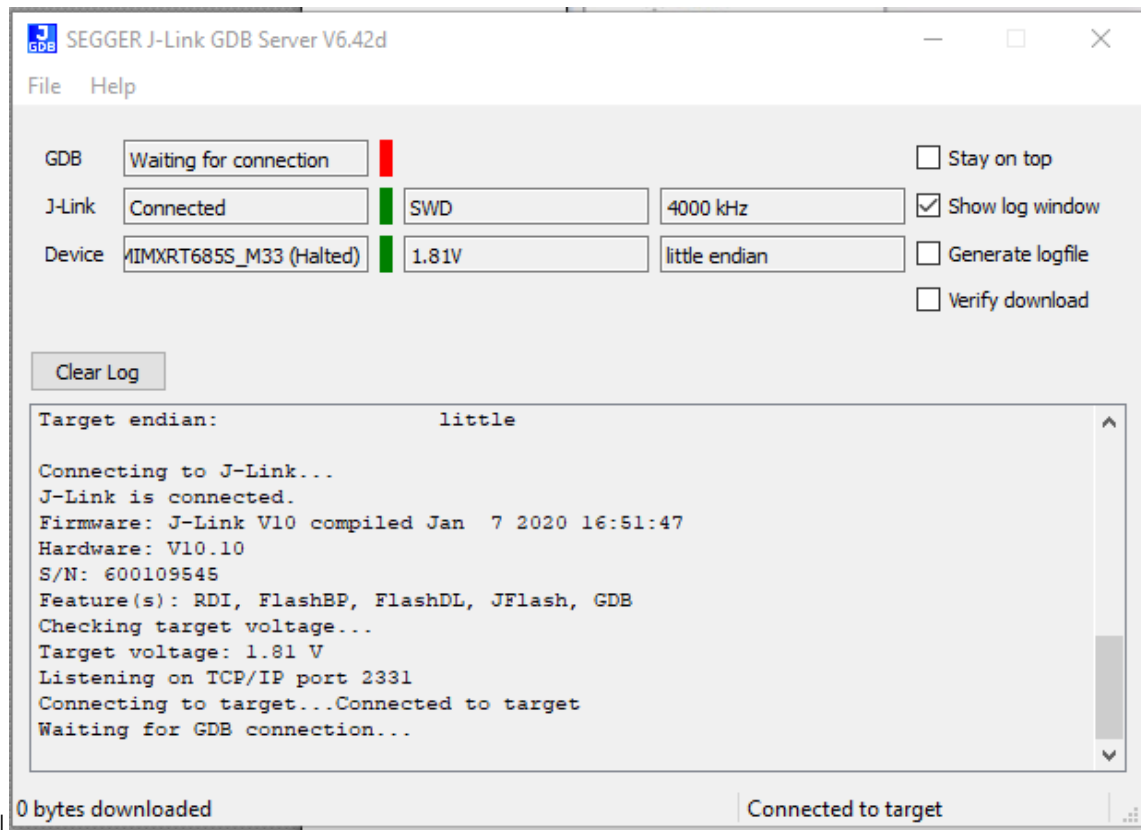
After the J-Link interface is configured and connected, follow these steps to download and run the demo applications:

1. Connect the development platform to your PC via USB cable between the LPC-Link2 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

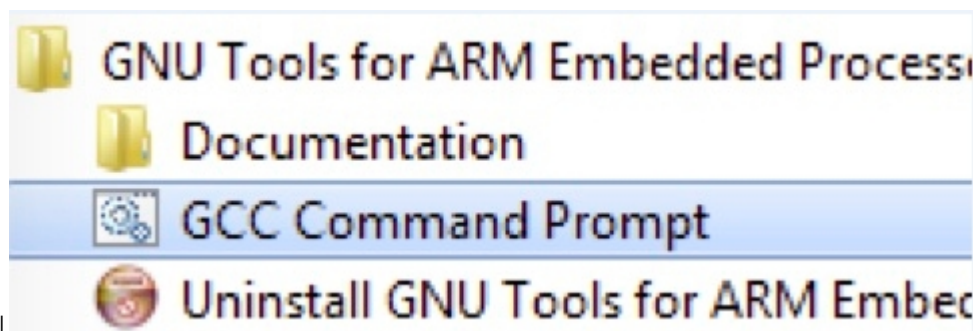


**\*\*Note:\*\*** Make sure that the board is set to FlexSPI flash boot mode \ (ISP2: ISP1: ISP0 = ON, OFF, ON\)  
 ↪ before use GDB debug.

3. Open the J-Link GDB Server application.
4. Open the J-Link GDB Server application. Go to the SEGGER install folder. For example, `C:\Program Files(x86)\SEGGER\JLink_Vxxx`. Open the command windows. Use the `JLinkGDBServer.exe -device MIMXRT685S_M33 -if SWD -scriptfile: <install_dir>/boards/<board_name>/<example_type>/<application_name>/evkmimxrt685.JLinkScript` command.
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
```

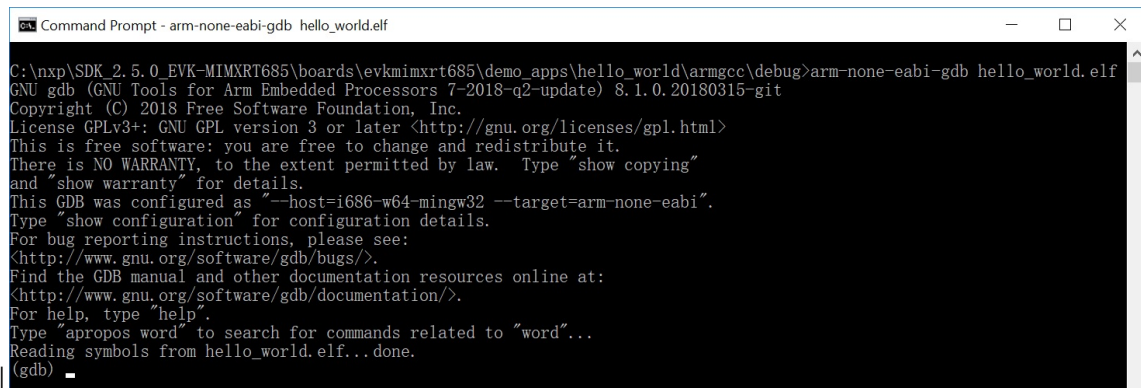
```
<install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc/release
```

For this example, the path is:

```
<install_dir>/boards/evkmimxrt685/demo_apps/hello_world/armgcc/debug
```

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





```

C:\nxp\SDK 2.5.0-EVK-MIMXRT685\boards\evkmimxrt685\demo_apps\hello_world\armgcc\debug>arm-none-eabi-gdb hello_world.elf
GNU gdb (GNU Tools for Arm Embedded Processors 7-2018-q2-update) 8.1.0.20180315-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 hello_world.elf...done.
(gdb) _

```

#### 9. Run these commands:

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

10. The application is now downloaded and halted at the watchpoint. 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.

```

Parent topic: [Run a demo using Arm GCC](#)

**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
```

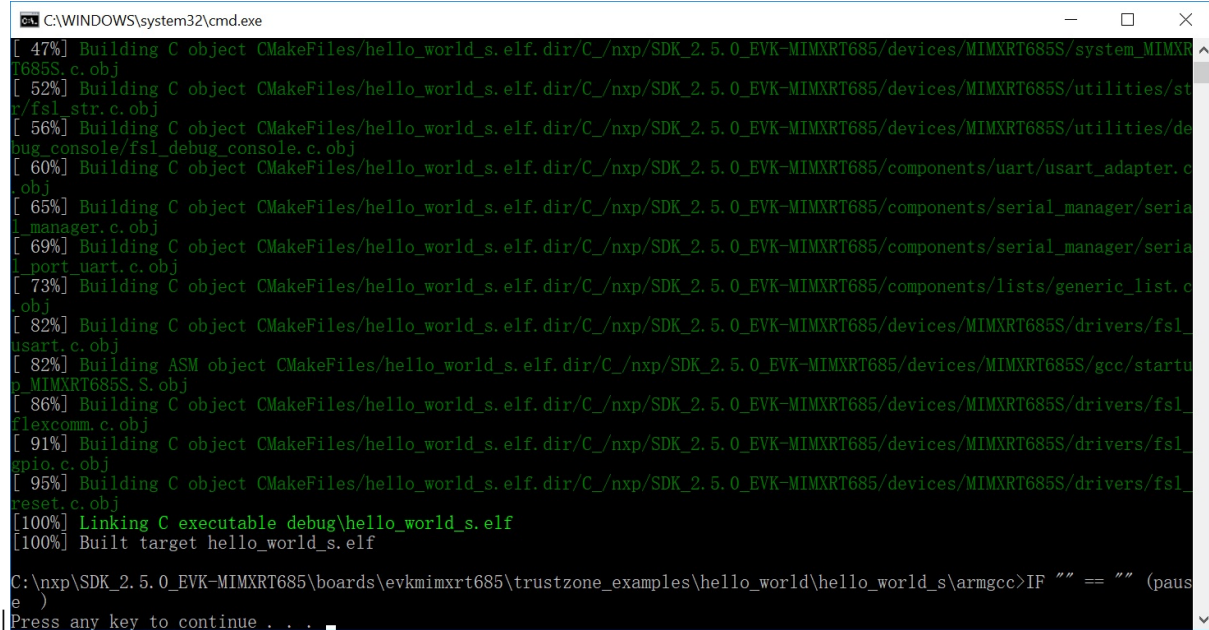
Begin with a simple TrustZone version of the Hello World application. The TrustZone Hello World GCC build scripts are located in this folder:



```
<install_dir>/boards/evkmimxrt685/trustzone_examples/hello_world/hello_world_ns/armgcc/build_debug.  
↪ bat
```

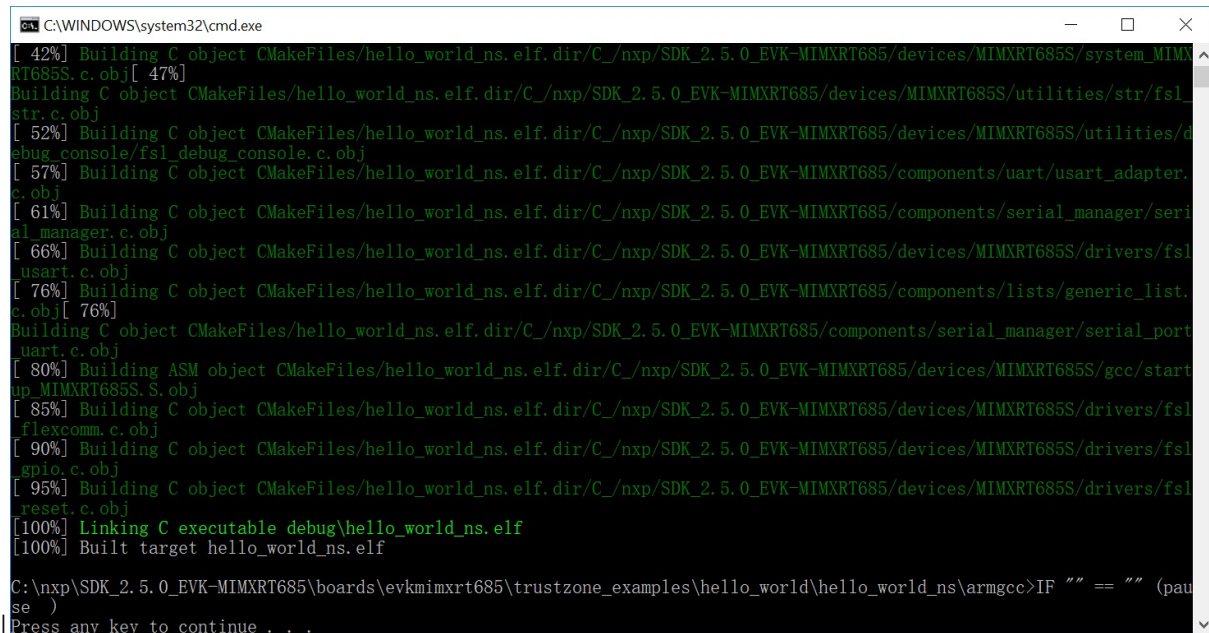
```
<install_dir>/boards/evkmimxrt685/trustzone_examples/hello_world/hello_world_s/armgcc/build_debug.  
↪ bat
```

Build both applications separately, following steps for single core examples as described in *Section 6.2, “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
[ 47%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/devices/MIMXRT685S/system_MIMXRT685S.c.obj
[ 52%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/devices/MIMXRT685S/utilities/string/fsl_str.c.obj
[ 56%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/devices/MIMXRT685S/utilities/debug_console/fsl_debug_console.c.obj
[ 60%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/components/uart/usart_adapter.c.obj
[ 65%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/components/serial_manager/serial_manager.c.obj
[ 69%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/components/serial_manager/serial_port_uart.c.obj
[ 73%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/components/lists/generic_list.c.obj
[ 82%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/devices/MIMXRT685S/drivers/fsl_usart.c.obj
[ 82%] Building ASM object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/devices/MIMXRT685S/gcc/startup_MIMXRT685S.S.obj
[ 86%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/devices/MIMXRT685S/drivers/fsl_flexcomm.c.obj
[ 91%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/devices/MIMXRT685S/drivers/fsl_gpio.c.obj
[ 95%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/devices/MIMXRT685S/drivers/fsl_reset.c.obj
[100%] Linking C executable debug\hello_world_s.elf
[100%] Built target hello_world_s.elf

C:\nxp\SDK_2.5.0_EVK-MIMXRT685\boards\evkmimxrt685\trustzone_examples\hello_world\hello_world_s\armgcc>IF "" == "" (pause)
Press any key to continue . . .
```



```
C:\WINDOWS\system32\cmd.exe
[ 42%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/devices/MIMXRT685S/system_MIMXRT685S.c.obj[ 47%]
Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/devices/MIMXRT685S/utilities/string/fsl_str.c.obj
[ 52%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/devices/MIMXRT685S/utilities/debug_console/fsl_debug_console.c.obj
[ 57%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/components/uart/usart_adapter.c.obj
[ 61%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/components/serial_manager/serial_manager.c.obj
[ 66%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/devices/MIMXRT685S/drivers/fsl_usart.c.obj
[ 76%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/components/lists/generic_list.c.obj[ 76%]
Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/components/serial_manager/serial_port_uart.c.obj
[ 80%] Building ASM object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/devices/MIMXRT685S/gcc/startup_MIMXRT685S.S.obj
[ 85%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/devices/MIMXRT685S/drivers/fsl_flexcomm.c.obj
[ 90%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/devices/MIMXRT685S/drivers/fsl_gpio.c.obj
[ 95%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.5.0_EVK-MIMXRT685/devices/MIMXRT685S/drivers/fsl_reset.c.obj
[100%] Linking C executable debug\hello_world_ns.elf
[100%] Built target hello_world_ns.elf

C:\nxp\SDK_2.5.0_EVK-MIMXRT685\boards\evkmimxrt685\trustzone_examples\hello_world\hello_world_ns\armgcc>IF "" == "" (pause)
Press any key to continue . . .
```

Parent topic:[Run a demo using Arm GCC](#)

**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 *Section 6.3, “Run an example application”*.

To download and run the TrustZone application, perform steps 1 to 10, as described in *Section 5.3, “Run an example application”*. These steps are common for both single core and TrustZone applications in Arm GCC.

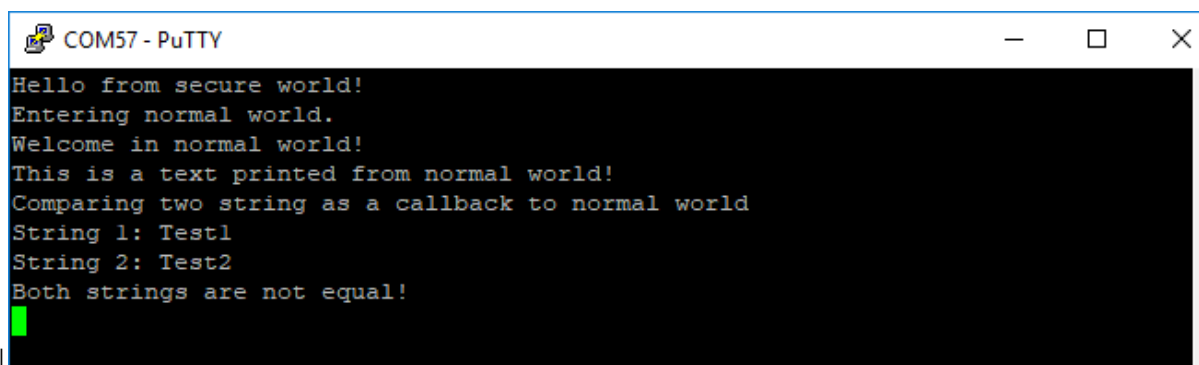
Then, run these commands:

1. arm-none-eabi-gdb.exe
2. target remote localhost:2331
3. monitor halt
4. monitor exec SetFlashDLNoRMWThreshold = 0x20000
5. load <install\\_dir>/boards/evkmimxrt685/trustzone\\_examples/hello\\_world/hello\\_world\\_ns/armgcc/debug/hello\\_world\\_ns.elf
6. load <install\\_dir>/boards/evkmimxrt685/trustzone\\_examples/hello\\_world/hello\\_world\\_s/armgcc/debug/hello\\_world\\_s.elf
7. monitor reset

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

```
$ arm-none-eabi-gdb.exe
GNU gdb (GNU Tools for Arm Embedded Processors 8-2019-q3-update) 8.3.0.20190703-git
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
warning: No executable has been specified and target does not support
determining executable automatically. Try using the "file" command.
0x0001c04a in ?? ()
(gdb) monitor halt
(gdb) load hello_world_ns/armgcc/debug/hello_world_ns.elf
Loading section .interrupts, size 0x130 lma 0x8041000
Loading section .text, size 0x1be8 lma 0x8041130
Loading section .ARM, size 0x8 lma 0x8042d18
Loading section .init_array, size 0x4 lma 0x8042d20
Loading section .fini_array, size 0x4 lma 0x8042d24
Loading section .data, size 0x60 lma 0x8042d28
Start address 0xc01e4, load size 7560
Transfer rate: 3691 KB/sec, 1260 bytes/write.
(gdb) load hello_world_s/armgcc/debug/hello_world_s.elf
Loading section .flash_config, size 0x1000 lma 0x18000000
Loading section .interrupts, size 0x130 lma 0x18001000
Loading section .text, size 0x5258 lma 0x18001130
Loading section .ARM, size 0x8 lma 0x18006388
Loading section .init_array, size 0x4 lma 0x18006390
Loading section .fini_array, size 0x4 lma 0x18006394
Loading section .data, size 0x70 lma 0x18006398
Loading section .gnu.sgstubs, size 0x20 lma 0x18040e00
Start address 0x100801e4, load size 25640
Transfer rate: 4173 KB/sec, 2848 bytes/write.
(gdb) monitor reset
Resetting target
(gdb) monitor go
(gdb) |
```



```
COM57 - PuTTY
Hello from secure world!
Entering normal world.
Welcome in normal world!
This is a text printed from normal world!
Comparing two string as a callback to normal world
String 1: Test1
String 2: Test2
Both strings are not equal!
```

Parent topic: [Run a demo using Arm GCC](#)

### 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 MIMXRT685S CMSIS pack.

1. Download the MIMXRT685S pack.
2. After downloading the DFP, double click to install it.

Parent topic: [Run a demo using Keil MDK/μVision](#)

### Build an example application

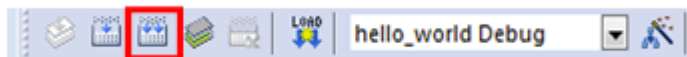
1. Open the desired example application workspace in:

```
<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:

```
<install_dir>/boards/evkmimxrt685s/demo_apps/hello_world/mdk/hello_world.uvmpw
```

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

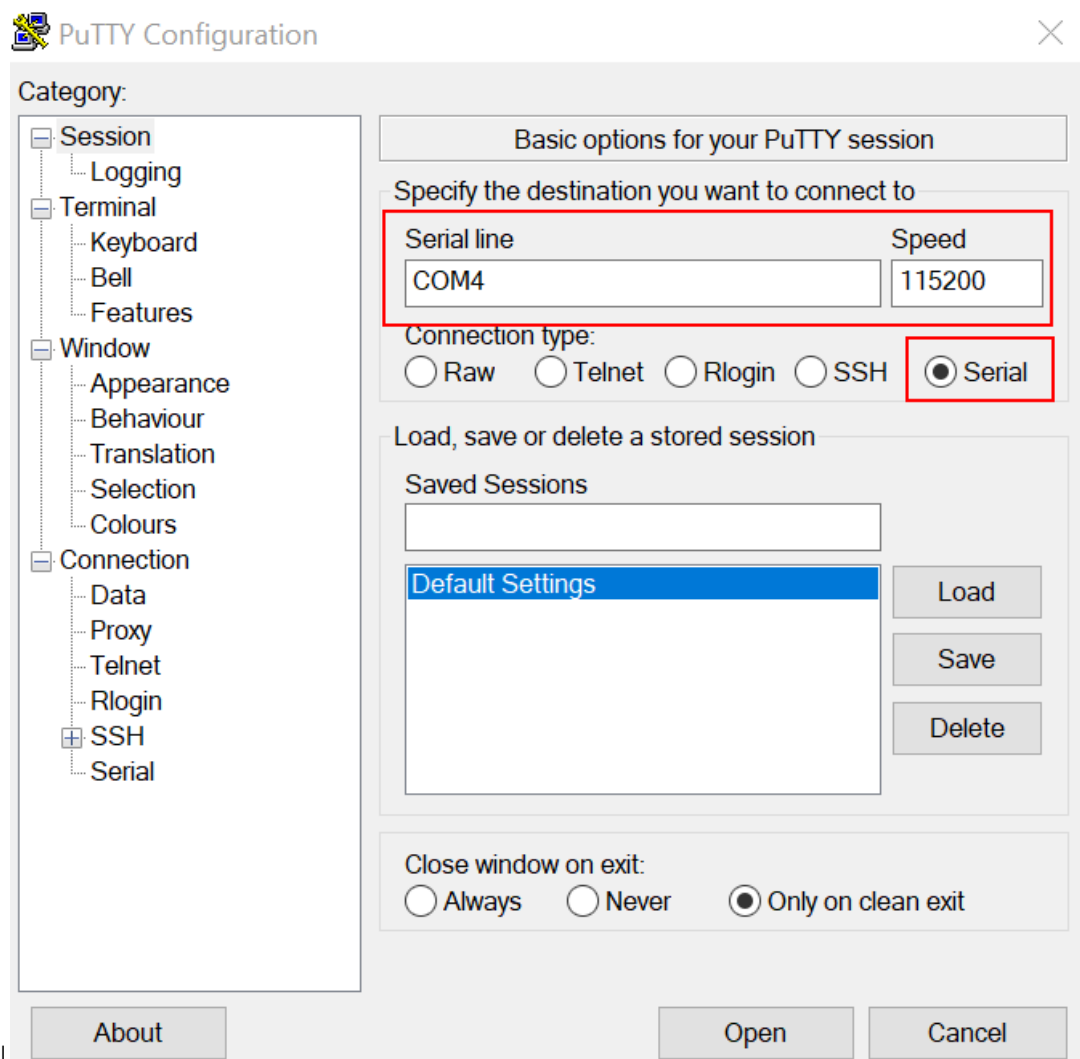


3. The build completes without errors.

Parent topic: [Run a demo using Keil MDK/μVision](#)

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

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

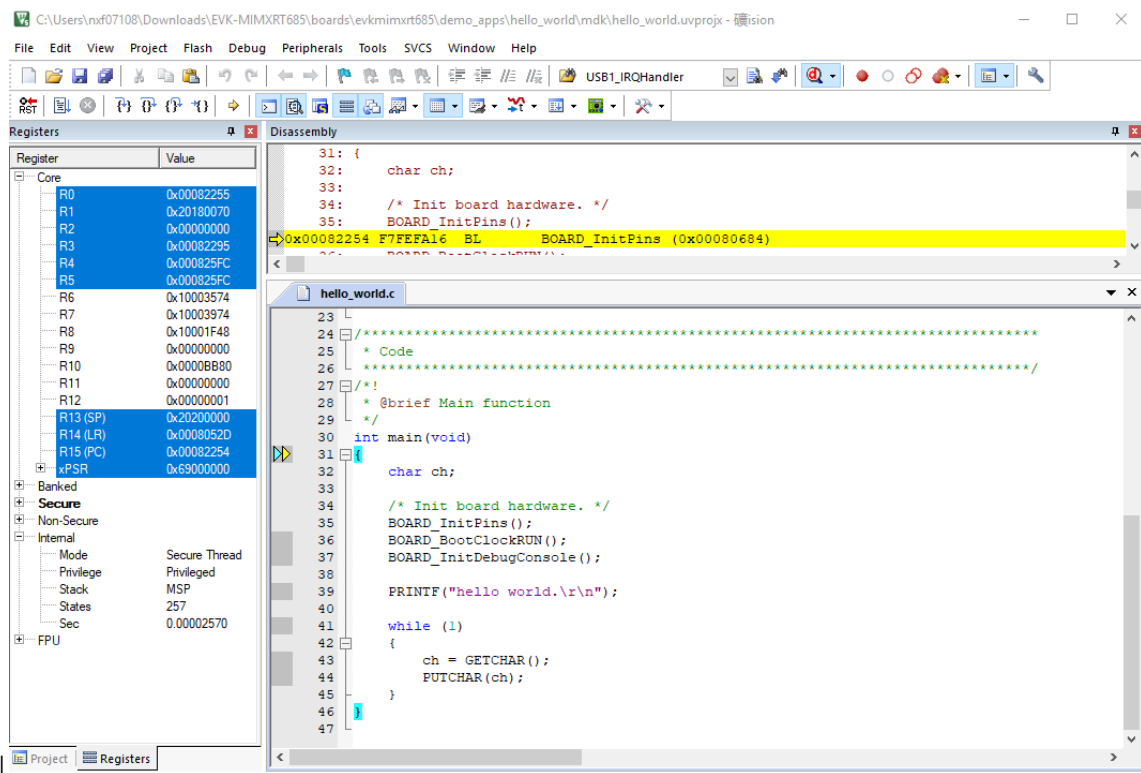


4. 1 stop bit

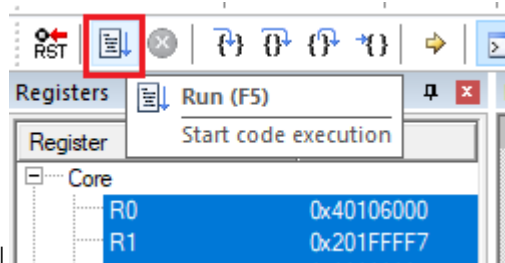
4. To debug the application, click **load** (or press the F8 key). Then, click the **Start/Stop Debug Session** button, highlighted in red in Figure 2. If using **J-Link** as the debugger, click **Project option > Debug > Settings > Debug > Port**, and select **SW**.

**Note:** When debugging with jlink, it expects one jlinkscript file named JLinkSettings.JLinkScript in the folder where the uVision project files are located. For details, see SEGGER Wiki. For the contents in this JLinkSettings.JLinkScript, use contents in evk-mimxrt1020\_sdram\_init.jlinkscript.

**Note:** Make sure that the board is set to FlexSPI flash boot mode (ISP2: ISP1: ISP0 = ON, OFF, ON) before using Keil debug.



5. Run the code by clicking **Run** to start the application, as shown in Figure 3.



The 'hello\_world' application is now running and a banner is displayed on the terminal, as shown in [Figure 4](run\_an\_example\_application\_001.md#S127DD02). If this is not true, check your terminal settings and connections.

```

```

**Parent topic:** [Run a demo using Keil MDK/μVision](#)

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

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



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

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

```
<install_dir>/boards/evkmimxrt595/trustzone_examples/hello_world/hello_world_ns/mdk/hello_world_ns.  
↳ uvmpw
```

```
<install_dir>/boards/evkmimxrt595/trustzone_examples/hello_world/hello_world_s/mdk/hello_world_s.  
↳ uvmpw
```

```
<install_dir>/boards/evkmimxrt595/trustzone_examples/hello_world/hello_world_s/mdk/hello_world.  
↳ uvmpw
```

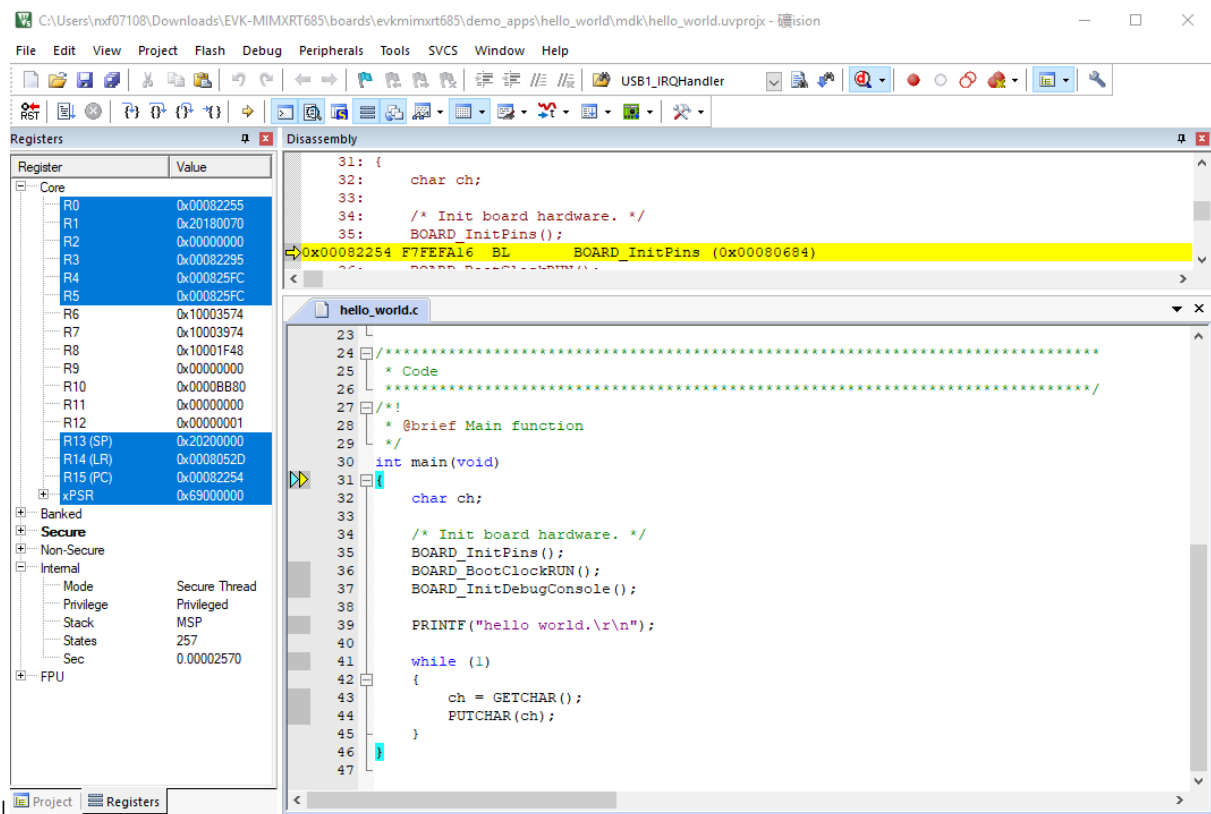
This project `hello_world.uvmpw` contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another.

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.

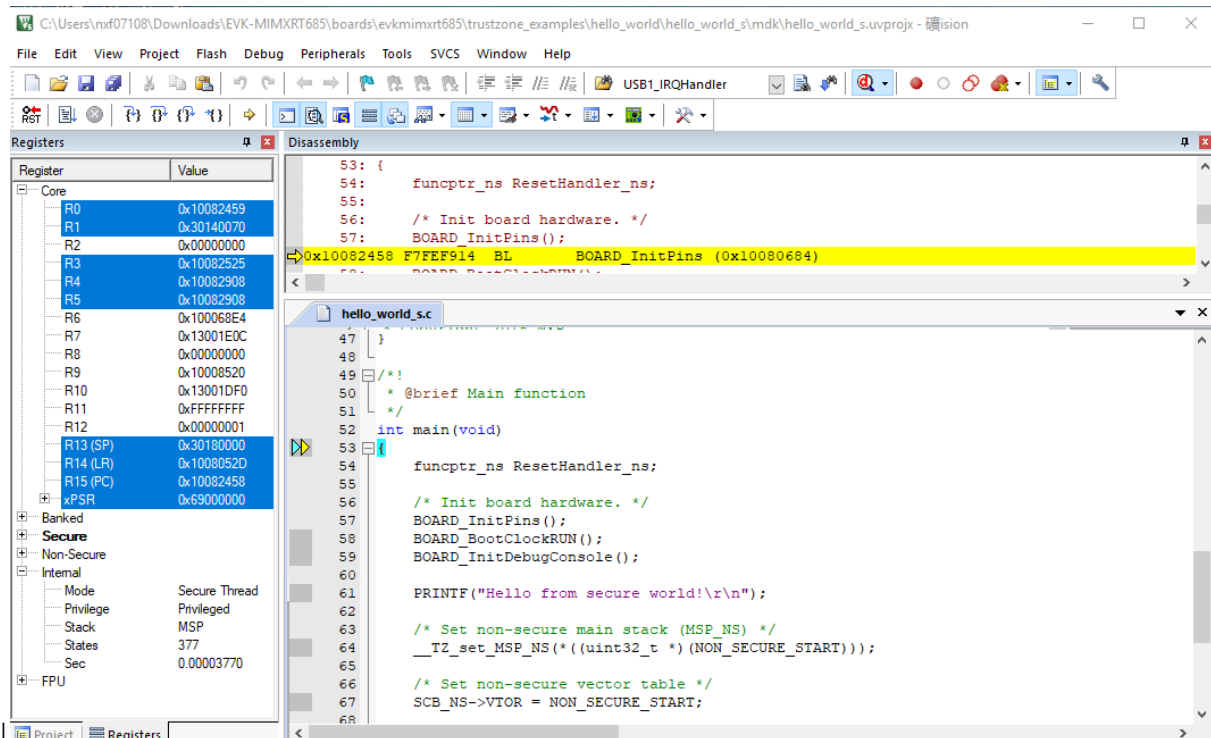
**Parent topic:** [Run a demo using Keil MDK/μVision](#)

**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.

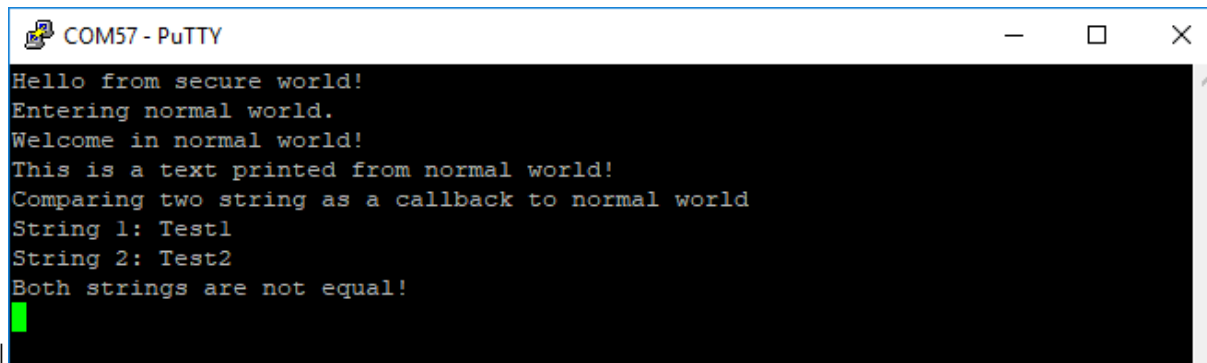
To download and run the TrustZone application, switch to the secure application project and perform steps as described in [Run a TrustZone example application](#). These steps are common for single core, dual-core, and TrustZone applications in μVision. After clicking **Download and Debug**, both the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



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



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



```
COM57 - PuTTY
Hello from secure world!
Entering normal world.
Welcome in normal world!
This is a text printed from normal world!
Comparing two string as a callback to normal world
String 1: Test1
String 2: Test2
Both strings are not equal!
```

**Parent topic:** [Run a demo using Keil MDK/μVision](#)

### Run a demo using the prebuilt binary

This section describes the steps to write a prebuilt binary in the SDK package to the external Flash and boot from the external Flash. The `hello_world` demo application is used as an example.

**Note:** The prebuilt binaries are provided for quick evaluation of the SDK and the board. Only parts of the examples are provided with prebuilt binary such as examples under `boards/<board_name>/demo_apps` and `boards/<board_name>/usb_examples`.

**Identify the load address of the binary** Several toolchains are supported by the MCUXpresso SDK, usually the prebuilt binaries are built from `armgcc flash_release` targets using Arm GCC. If there is no `flash_release` target, then `release` target is used.

Take `hello_world` demo application as an example, the `hello_world.bin` is located in:

```
<install_dir>/boards/<board_name>/<demo_apps>/hello_world/
```

And it was built from:

```
<install_dir>/boards/<board_name>/<demo_apps>/hello_world/armgcc/build_flash_release.sh
```

The load address can be identified by the linker file in the same folder. Such as `<device_name>_flash.ld`.

```
MEMORY
{
  m_flash_config      (RX) : ORIGIN = 0x08000400, LENGTH = 0x00000200
  m_interrupts        (RX) : ORIGIN = 0x08001000, LENGTH = 0x00000130
  m_text              (RX) : ORIGIN = 0x08001130, LENGTH = 0x001FEED0
  m_data              (RW) : ORIGIN = 0x20080000, LENGTH = 0x00180000
  m_usb_sram          (RW) : ORIGIN = 0x40140000, LENGTH = 0x00004000
```

So, the load address for `hello_world.bin` is `0x08000400`.

**Parent topic:** [Run a demo using the prebuilt binary](#)

**Write the binary to external flash** Take J-Link Commander as an example to write the `hello_world.bin` to external flash.

Run the J-Link Commander, run the following command to connect to the core.



```
J-Link>connect
J-Link>?
```

Select the correct device from the pop-up window **Target device settings**.

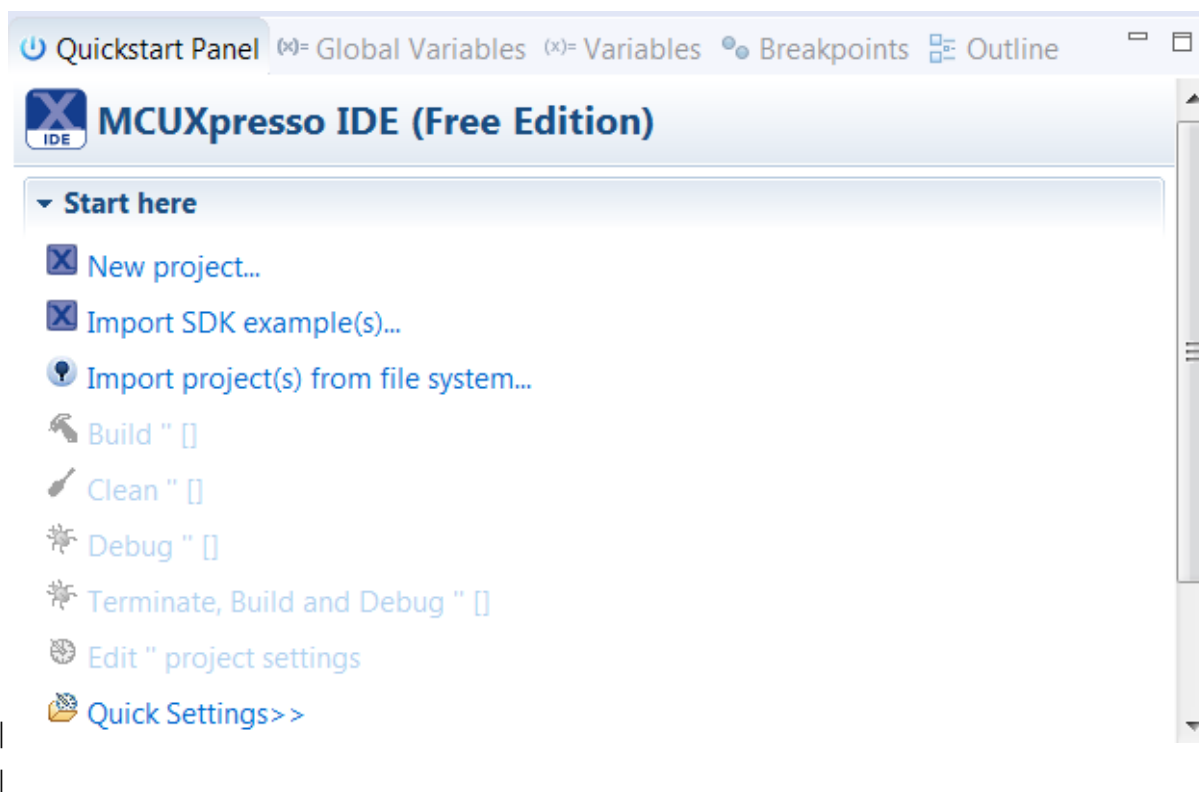
Specify targets interface as SWD and the target interface speed as per the guide in the command window. The **Cortex-M33 identified** appears when it is connected successfully.

**Parent topic:** [Run a demo using the prebuilt binary](#)

## MCUXpresso IDE New Project Wizard

MCUXpresso IDE features a new project wizard. The wizard provides functionality for the user to create new projects from the installed SDKs (and from pre-installed part support). It offers user the flexibility to select and change multiple builds. The wizard also includes a library and provides source code options. The source code is organized as software components, categorized as drivers, utilities, and middleware.

To use the wizard, start the MCUXpresso IDE. This is located in the **QuickStart Panel** at the bottom left of the MCUXpresso IDE window. Select **New project**, as shown in *Figure 1*.



For more details and usage of new project wizard, see the *MCUXpresso\_IDE\_User\_Guide.pdf* in the MCUXpresso IDE installation folder.

## How to determine COM port

This section describes the steps necessary to determine the debug COM port number of your NXP hardware development platform.

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 Cortex-A core debug console and the other is for Cortex M4.

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

## 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
}
```

## Default debug interfaces

The MCUXpresso SDK supports various hardware platforms that come loaded with various factory programmed debug interface configurations. *Table 1* lists the hardware platforms supported by the MCUXpresso SDK, their default debug interface, and any version information that helps differentiate a specific interface configuration.

**Note:** The *OpenSDA details* column in *Table 1* is not applicable to LPC.

| Hardware platform | Default interface     | OpenSDA details |
|-------------------|-----------------------|-----------------|
| EVK-MC56F83000    | P&E Micro OSJTAG      | N/A             |
| EVK-MIMXRT595     | CMSIS-DAP             | N/A             |
| EVK-MIMXRT685     | CMSIS-DAP             | N/A             |
| FRDM-K22F         | CMSIS-DAP/mbd/DAPLink | OpenSDA v2.1    |
| FRDM-K28F         | DAPLink               | OpenSDA v2.1    |
| FRDM-K32L2A4S     | CMSIS-DAP             | OpenSDA v2.1    |
| FRDM-K32L2B       | CMSIS-DAP             | OpenSDA v2.1    |
| FRDM-K32W042      | CMSIS-DAP             | N/A             |
| FRDM-K64F         | CMSIS-DAP/mbd/DAPLink | OpenSDA v2.0    |
| FRDM-K66F         | J-Link OpenSDA        | OpenSDA v2.1    |

continues on next page

Table 1 – continued from previous page

| Hardware platform   | Default interface     | OpenSDA details         |
|---------------------|-----------------------|-------------------------|
| FRDM-K82F           | CMSIS-DAP             | OpenSDA v2.1            |
| FRDM-KE15Z          | DAPLink               | OpenSDA v2.1            |
| FRDM-KE16Z          | CMSIS-DAP/mbd/DAPLink | OpenSDA v2.2            |
| FRDM-KL02Z          | P&E Micro OpenSDA     | OpenSDA v1.0            |
| FRDM-KL03Z          | P&E Micro OpenSDA     | OpenSDA v1.0            |
| FRDM-KL25Z          | P&E Micro OpenSDA     | OpenSDA v1.0            |
| FRDM-KL26Z          | P&E Micro OpenSDA     | OpenSDA v1.0            |
| FRDM-KL27Z          | P&E Micro OpenSDA     | OpenSDA v1.0            |
| FRDM-KL28Z          | P&E Micro OpenSDA     | OpenSDA v2.1            |
| FRDM-KL43Z          | P&E Micro OpenSDA     | OpenSDA v1.0            |
| FRDM-KL46Z          | P&E Micro OpenSDA     | OpenSDA v1.0            |
| FRDM-KL81Z          | CMSIS-DAP             | OpenSDA v2.0            |
| FRDM-KL82Z          | CMSIS-DAP             | OpenSDA v2.0            |
| FRDM-KV10Z          | CMSIS-DAP             | OpenSDA v2.1            |
| FRDM-KV11Z          | P&E Micro OpenSDA     | OpenSDA v1.0            |
| FRDM-KV31F          | P&E Micro OpenSDA     | OpenSDA v1.0            |
| FRDM-KW24           | CMSIS-DAP/mbd/DAPLink | OpenSDA v2.1            |
| FRDM-KW36           | DAPLink               | OpenSDA v2.2            |
| FRDM-KW41Z          | CMSIS-DAP/DAPLink     | OpenSDA v2.1 or greater |
| Hexiwear            | CMSIS-DAP/mbd/DAPLink | OpenSDA v2.0            |
| HVP-KE18F           | DAPLink               | OpenSDA v2.2            |
| HVP-KV46F150M       | P&E Micro OpenSDA     | OpenSDA v1              |
| HVP-KV11Z75M        | CMSIS-DAP             | OpenSDA v2.1            |
| HVP-KV58F           | CMSIS-DAP             | OpenSDA v2.1            |
| HVP-KV31F120M       | P&E Micro OpenSDA     | OpenSDA v1              |
| JN5189DK6           | CMSIS-DAP             | N/A                     |
| LPC54018 IoT Module | N/A                   | N/A                     |
| LPCXpresso54018     | CMSIS-DAP             | N/A                     |
| LPCXpresso54102     | CMSIS-DAP             | N/A                     |
| LPCXpresso54114     | CMSIS-DAP             | N/A                     |
| LPCXpresso51U68     | CMSIS-DAP             | N/A                     |
| LPCXpresso54608     | CMSIS-DAP             | N/A                     |
| LPCXpresso54618     | CMSIS-DAP             | N/A                     |
| LPCXpresso54628     | CMSIS-DAP             | N/A                     |
| LPCXpresso54S018M   | CMSIS-DAP             | N/A                     |
| LPCXpresso55s16     | CMSIS-DAP             | N/A                     |
| LPCXpresso55s28     | CMSIS-DAP             | N/A                     |
| LPCXpresso55s69     | CMSIS-DAP             | N/A                     |
| MAPS-KS22           | J-Link OpenSDA        | OpenSDA v2.0            |
| MIMXRT1170-EVK      | CMSIS-DAP             | N/A                     |
| TWR-K21D50M         | P&E Micro OSJTAG      | N/AOpenSDA v2.0         |
| TWR-K21F120M        | P&E Micro OSJTAG      | N/A                     |
| TWR-K22F120M        | P&E Micro OpenSDA     | OpenSDA v1.0            |
| TWR-K24F120M        | CMSIS-DAP/mbd         | OpenSDA v2.1            |
| TWR-K60D100M        | P&E Micro OSJTAG      | N/A                     |
| TWR-K64D120M        | P&E Micro OpenSDA     | OpenSDA v1.0            |
| TWR-K64F120M        | P&E Micro OpenSDA     | OpenSDA v1.0            |
| TWR-K65D180M        | P&E Micro OpenSDA     | OpenSDA v1.0            |
| TWR-K65D180M        | P&E Micro OpenSDA     | OpenSDA v1.0            |
| TWR-KV10Z32         | P&E Micro OpenSDA     | OpenSDA v1.0            |
| TWR-K80F150M        | CMSIS-DAP             | OpenSDA v2.1            |
| TWR-K81F150M        | CMSIS-DAP             | OpenSDA v2.1            |
| TWR-KE18F           | DAPLink               | OpenSDA v2.1            |
| TWR-KL28Z72M        | P&E Micro OpenSDA     | OpenSDA v2.1            |
| TWR-KL43Z48M        | P&E Micro OpenSDA     | OpenSDA v1.0            |

continues on next page

Table 1 – continued from previous page

| Hardware platform | Default interface  | OpenSDA details         |
|-------------------|--------------------|-------------------------|
| TWR-KL81Z72M      | CMSIS-DAP          | OpenSDA v2.0            |
| TWR-KL82Z72M      | CMSIS-DAP          | OpenSDA v2.0            |
| TWR-KM34Z75M      | P&E Micro OpenSDA  | OpenSDA v1.0            |
| TWR-KM35Z75M      | DAPLink            | OpenSDA v2.2            |
| TWR-KV10Z32       | P&E Micro OpenSDA  | OpenSDA v1.0            |
| TWR-KV11Z75M      | P&E Micro OpenSDA  | OpenSDA v1.0            |
| TWR-KV31F120M     | P&E Micro OpenSDA  | OpenSDA v1.0            |
| TWR-KV46F150M     | P&E Micro OpenSDA  | OpenSDA v1.0            |
| TWR-KV58F220M     | CMSIS-DAP          | OpenSDA v2.1            |
| TWR-KW24D512      | P&E Micro OpenSDA  | OpenSDA v1.0            |
| USB-KW24D512      | N/A External probe | N/A                     |
| USB-KW41Z         | CMSIS-DAP\DAPLink  | OpenSDA v2.1 or greater |

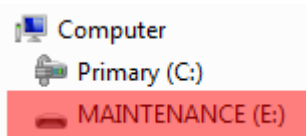
## Updating debugger firmware

**Updating OpenSDA firmware** Any NXP hardware platform that comes with an OpenSDA-compatible debug interface has the ability to update the OpenSDA firmware. 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.
- 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.

**Parent topic:** [Updating debugger firmware](#)

**Updating LPCXpresso board 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.

NXF 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 [www.nxp.com/lpcutilities](http://www.nxp.com/lpcutilities).

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 ([www.nxp.com/lpcutilities](http://www.nxp.com/lpcutilities), 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.

**Parent topic:** [Updating debugger firmware](#)

## 1.3 Getting Started with MCUXpresso SDK GitHub

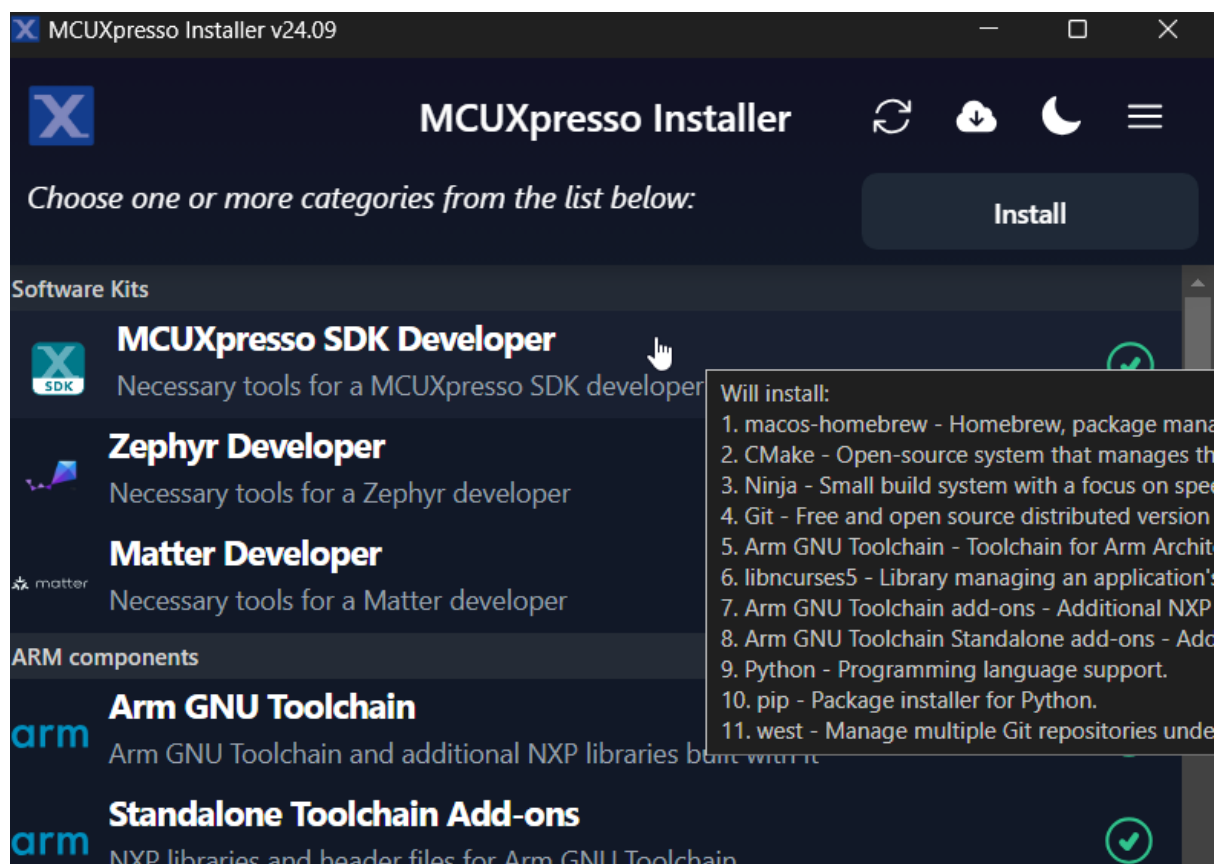
### 1.3.1 Getting Started with MCUXpresso SDK Repository

#### Installation

##### NOTE

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

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



#### Alternative: Manual Installation

##### Basic tools

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

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

Then configure your username and email using the commands:

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

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

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

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

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

## Build And Configuration System

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

For Windows, you can directly use the .msi installer like `cmake-3.31.4-windows-x86_64.msi` to install.

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

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

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

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

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

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

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

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

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

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



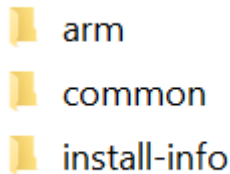
**Toolchain** MCUXpresso SDK supports all mainstream toolchains for embedded development. You can install your used or interested toolchains following the guides.

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

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

| Toolchain                    | Environment Variable | Example                                                                                                                                                | Cmd Line Argument        |
|------------------------------|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|
| Armgcc                       | ARMGCC_DIR           | C:\armgcc for windows/usr for Linux. Typically arm-none-eabi-* is installed under /usr/bin                                                             | – toolchain armgcc       |
| IAR                          | IAR_DIR              | C:\iar\ewarm-9.60.3 for Windows/opt/iarsystems/bxarm-9.60.3 for Linux                                                                                  | – toolchain iar          |
| MDK                          | MDK_DIR              | C:\Keil_v5 for Windows.MDK IDE is not officially supported with Linux.                                                                                 | – toolchain mdk          |
| Armclang                     | ARM-CLANG_DIR        | C:\ArmCompilerforEmbedded6.22 for Windows/opt/ArmCompilerforEmbedded6.21 for Linux                                                                     | – toolchain mdk          |
| Zephyr                       | ZEPHYR_DIR           | c:\NXP\zephyr-sdk-<version> for windows/opt/zephyr-sdk-<version> for Linux                                                                             | – toolchain zephyr       |
| CodeWarrior                  | CW_DIR               | C:\Freescall\CW MCU v11.2 for windowsCodeWarrior is not supported with Linux                                                                           | – toolchain code-warrior |
| Xtensa                       | XCC_DIR              | C:\xtensa\XtDevTools\install\tools\RI-2023.11-win32\XtensaTools for windows/opt/xtensa/XtDevTools/install/tools/RI-2023.11-Linux/XtensaTools for Linux | – toolchain xtensa       |
| NXP S32Compiler RISC-V Zen-V | RISCVL-LVM_DIR       | C:\riscv-llvm-win32_b298_b298_2024.08.12 for Windows/opt/riscv-llvm-Linux-x64_b298_b298_2024.08.12 for Linux                                           | – toolchain 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                                                                            |
|------------|----------------------------------------------------------------------------------------|
| mani-fests | 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.                                                                                                                                     |
| com-<br>po-<br>nents | Software components.                                                                                                                                                                          |
| de-<br>vices         | 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.                                                                                                                                                                           |
| ex-<br>am-<br>ples   | Various demos and examples, support files on different supported boards. For each board support, there are board configuration files.                                                         |
| mid-<br>dle-<br>ware | 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 west manifest config group.filter +optional and west update mcux-soc-svd to get this folder.                     |

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

Examples files are placed in folder of <example\_category>, these examples include (but are not limited to)

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

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

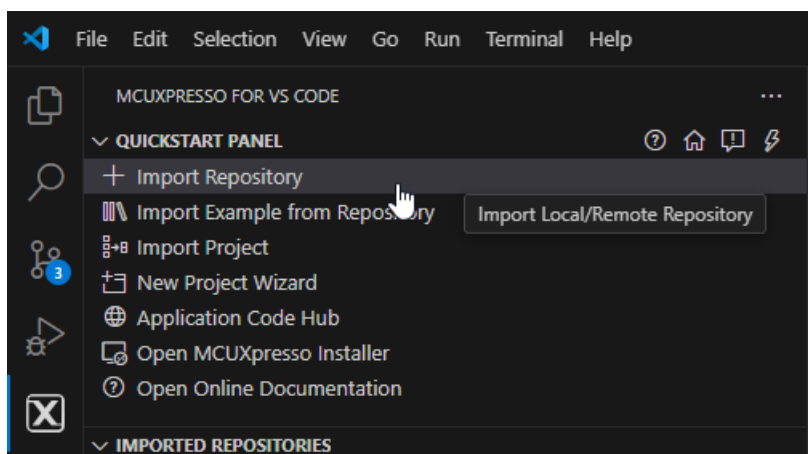
## Run a demo using MCUXpresso for VS Code

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

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

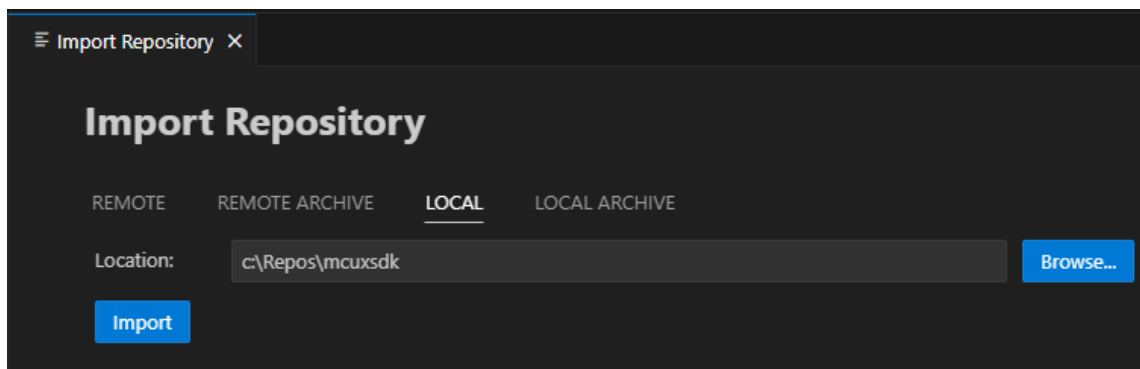
To build an example application:

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

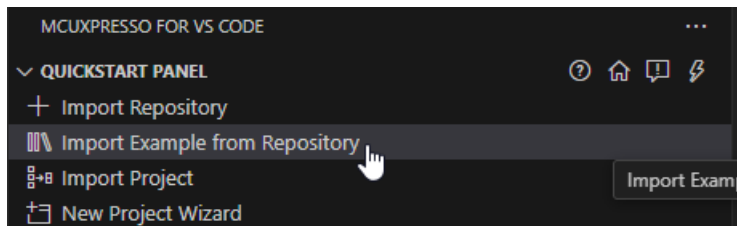


**Note:** You can import the SDK in several ways. Refer to [MCUXpresso for VS Code Wiki](#) for details.

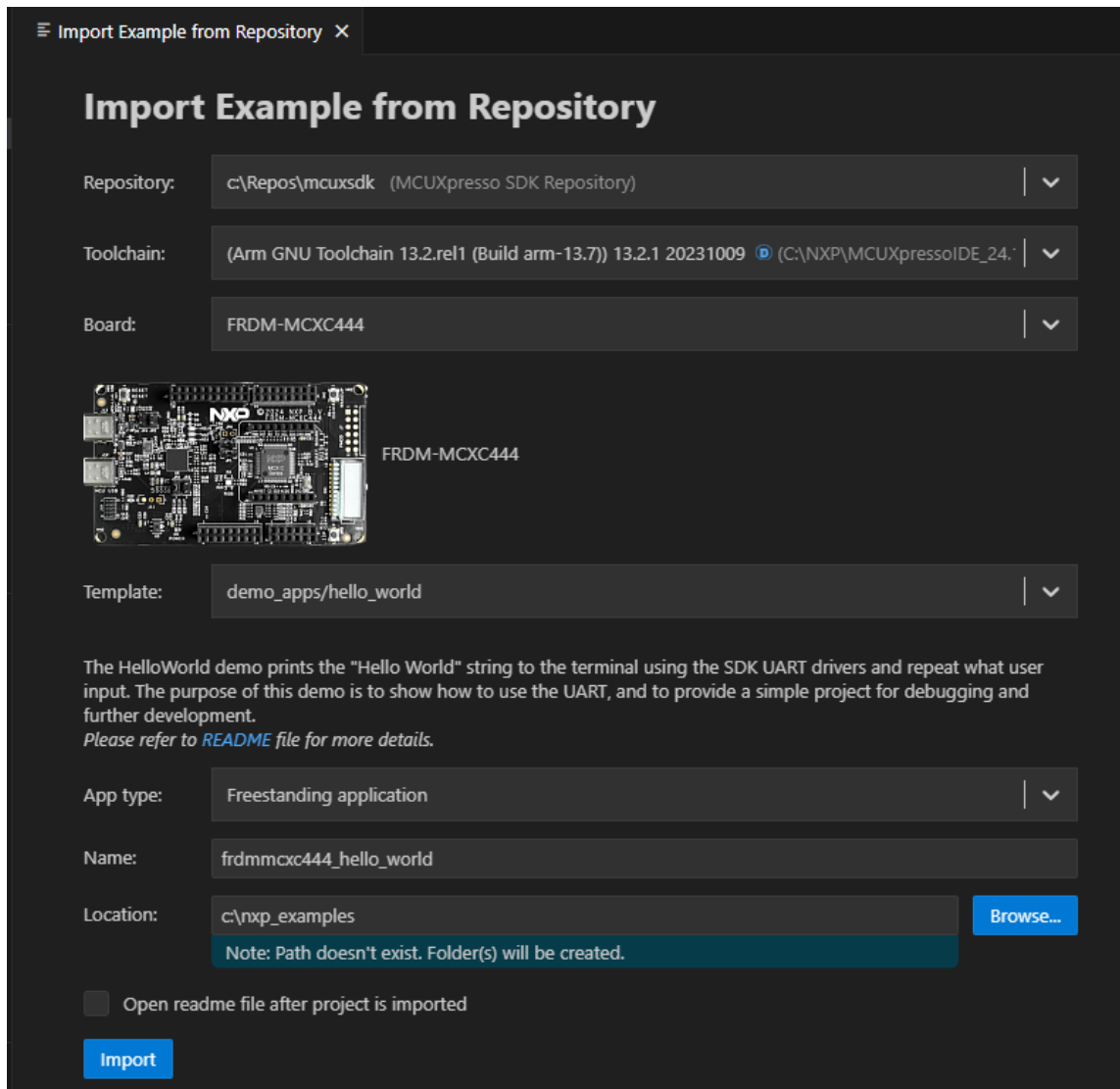
Select **Local** if you've already obtained the SDK as seen in [Get MCUXpresso SDK Repo](#). Select your location and click **Import**.



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

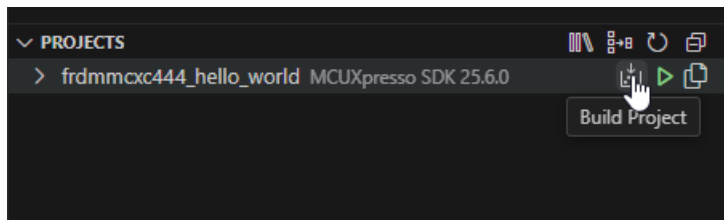


In the dropdown menu, select the MCUXpresso SDK, the Arm GNU Toolchain, your board, template, and application type. Click **Import**.

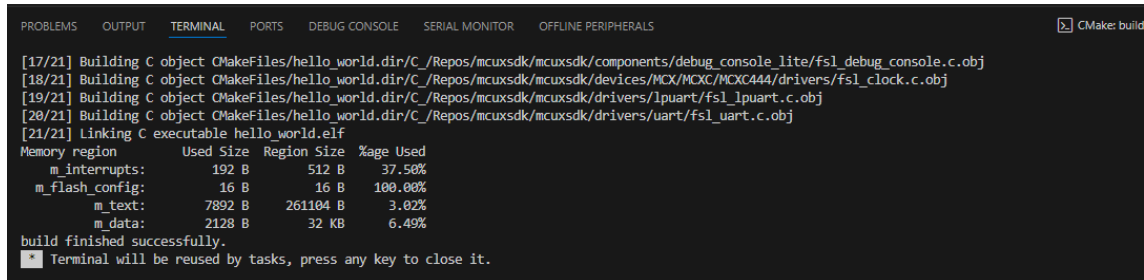


**Note:** The MCUXpresso SDK projects can be imported as **Repository applications** or **Free-standing applications**. The difference between the two is the import location. Projects imported as Repository examples will be located inside the MCUXpresso SDK, whereas Free-standing examples can be imported 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.

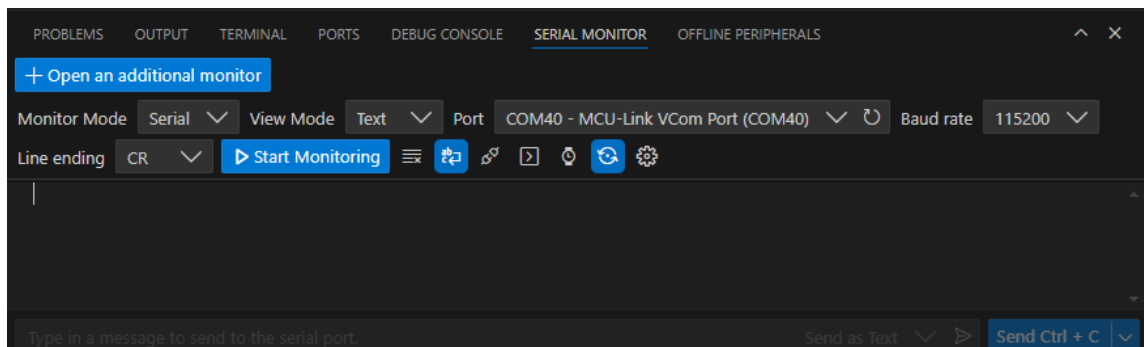


The integrated terminal will open at the bottom and will display the build output.

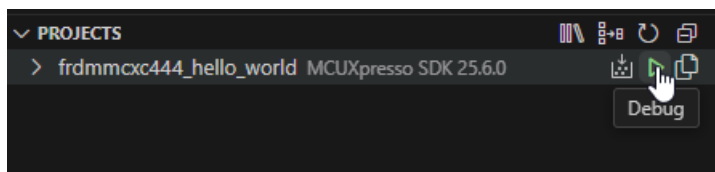


**Run an example application** **Note:** for full details on MCUXpresso for VS Code debug probe support, see [MCUXpresso for VS Code Wiki](#).

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.

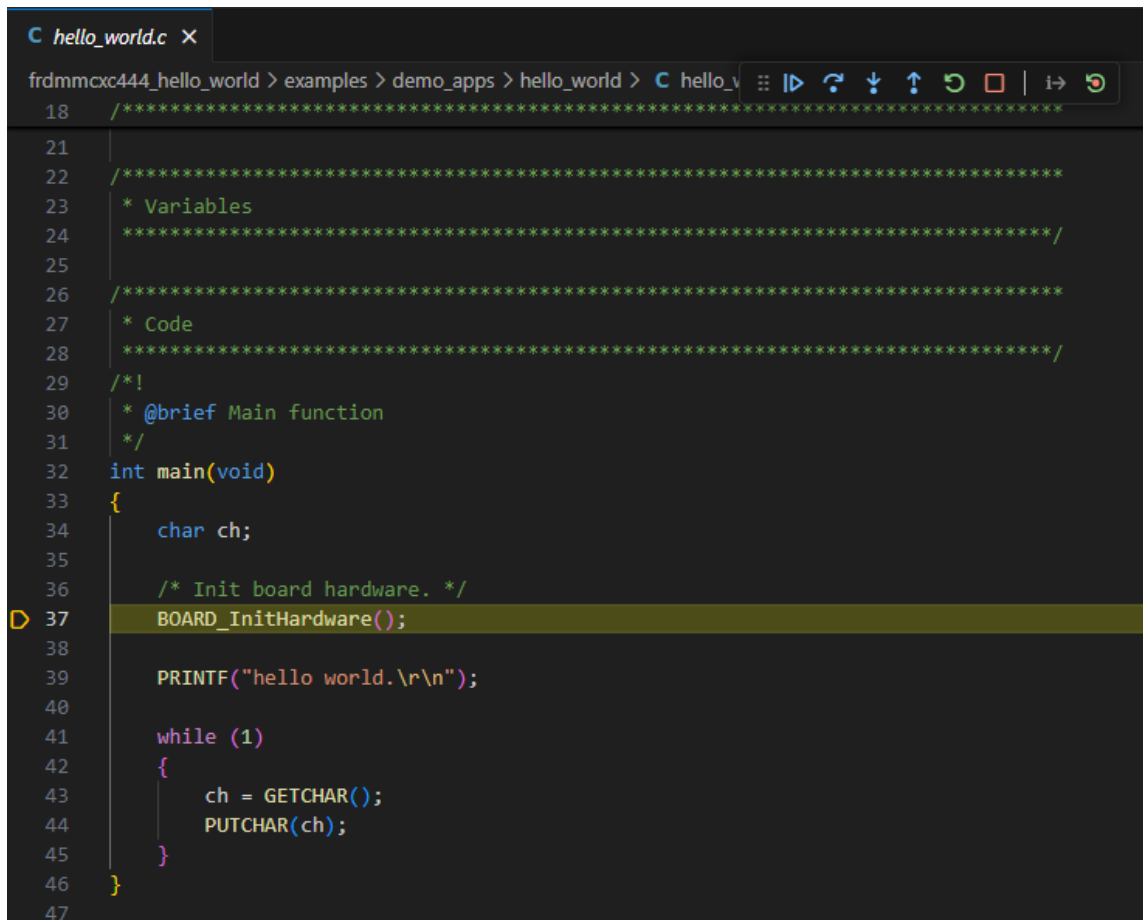


2. Navigate to the **PROJECTS** view and click the play button to initiate a debug session.



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



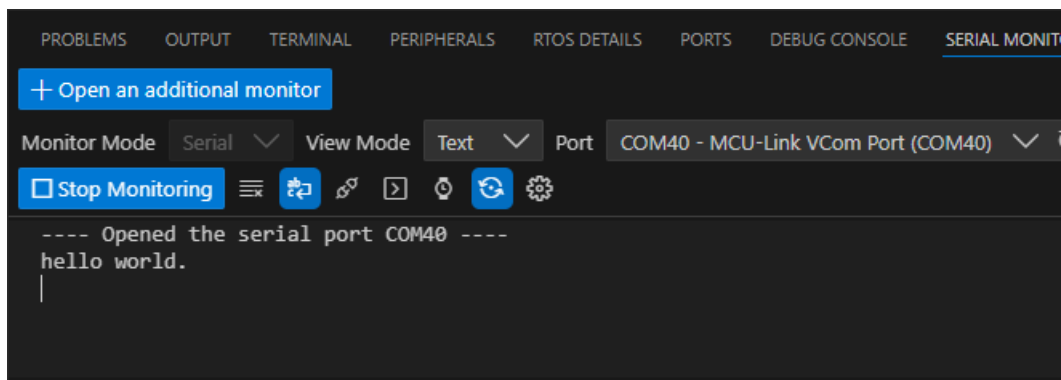


```

C hello_world.c x
frdm-mxc444_hello_world > examples > demo_apps > hello_world > C hello_v
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

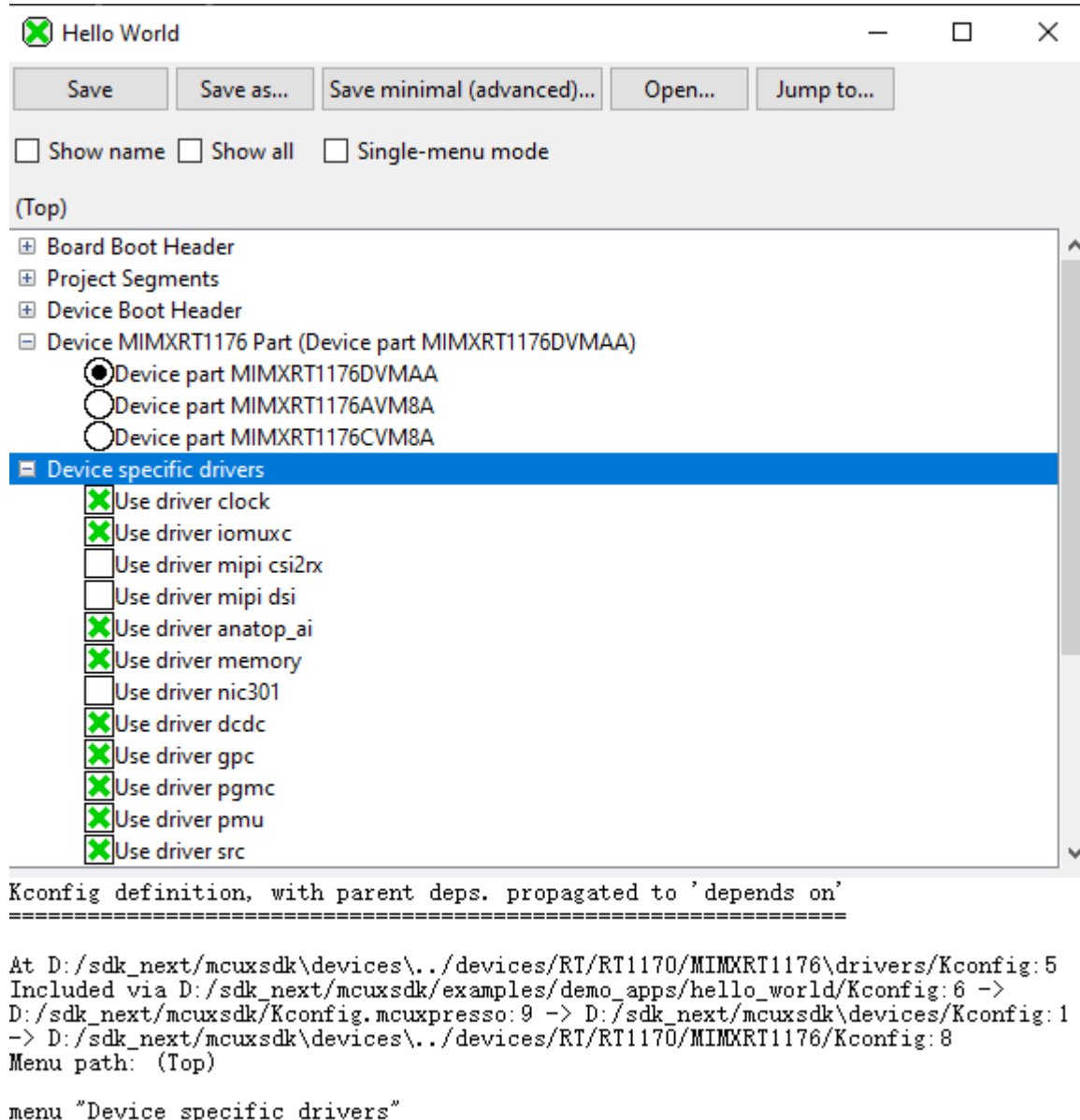
```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

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:

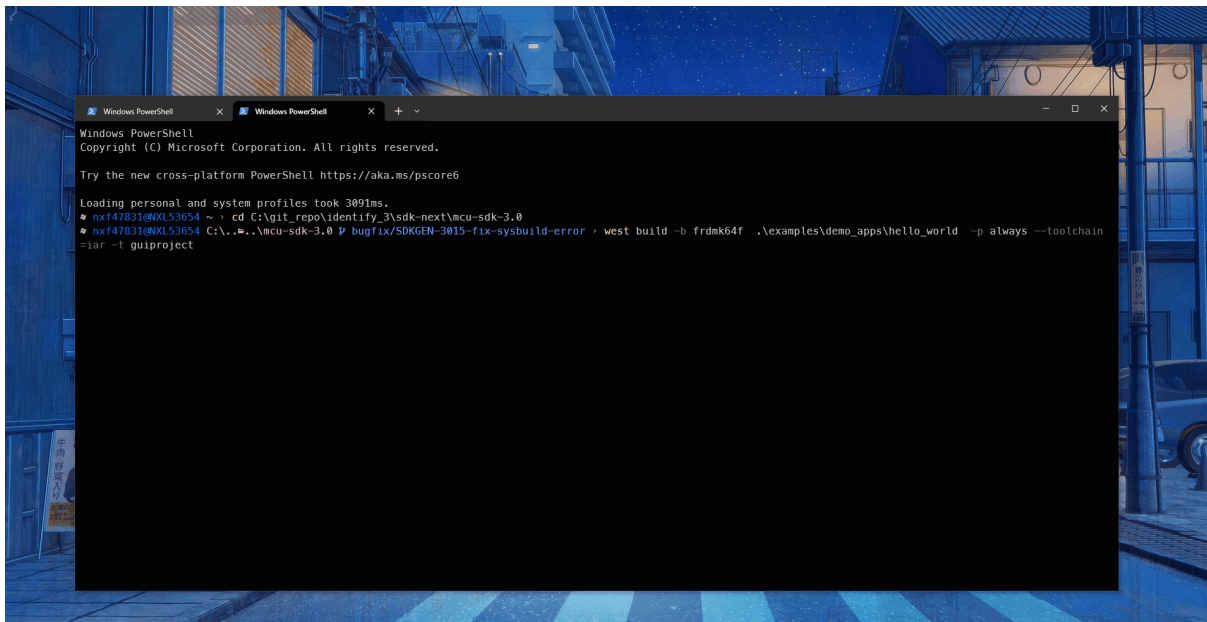
```
west debug -r linkserver
```

**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 Getting Started with MCUXpresso SDK Xplorer

### 1.4.1 Getting Started with MCUXpresso SDK Xplorer

#### Overview

Cadence Tensilica Xplorer is a complete development environment that helps users create application code for high-performance Tensilica processors. Xplorer is the interface to powerful

software development tools such as the XCC compiler, assembler, linker, debugger, code profiler, and full set of GUI tools.

Xplorer, including both GUI and command-line environment, is the only available development IDE for the DSP core of MIMXRT600.

## Install Xplorer Toolchains

This section provides information on Xtensa Software Tools Platform Support and steps to:

- Install the Xtensa Xplorer IDE and Tools
- Install License Key
- Install RT600 DSP Build Configuration
- Install Xtensa On Chip Debugger Daemon
- Program LPC-Link2 with SEGGER J-Link
- Install Xtensa Software Tools without IDE

**Xtensa Software Tools Platform Support** The Xtensa Software Tools are officially supported on the following platforms:

**Windows:** Win 10 64-bit, Win 8 64-bit, Win 7 64-bit.

**Linux:** RHEL 6 64-bit (with 'Desktop' package installed).

There may be compatibility issues with other versions of Linux or Windows, especially when using the IDE.

**Note:** The security-enhanced Linux (SELinux) is not a supported platform because the OS can prevent different shared libraries (including Xtensa Tools) from loading.

For more information on platform support and installation guidelines, see the Xtensa Development Tools Installation Guide.

**Parent topic:** [Install Xplorer Toolchains](#)

**Install the Xtensa Xplorer IDE and Tools** To install the Xtends Xplorer IDE and tools, perform the following steps:

1. Go to the URL <https://tensilicatools.com/download/rt600-download-page/> and log in. If you are accessing the site for the first time, make sure to register.
2. Make sure to use your corporate email address to register.



3. Once registered you should receive an email confirmation with an activation link from 'Tensilica Tools' [no-reply@tensilicatools.com](mailto:no-reply@tensilicatools.com). Make sure to check the spam folder if this email does not show up in the inbox.
4. To complete the registration, click the activation link.
5. Once registered, log in to see the material available for download:



1. Download and install the **Xplorer IDE V10.1.11** for your operating system (Windows or Linux).
2. Download the **DSP Configuration** for your operating system – installed later through the IDE, see Install RT600 DSP Build Configuration. **Note:** NXP recommends version **10.1.11** of the Xtensa Xplorer IDE and tools for use with the RT600 DSP.

**Parent topic:** [Install Xplorer Toolchains](#)

**Install License Key** Xtensa development tools use FLEXlm for license management. FLEXlm licensing is required for tools such as the Xtensa Xplorer IDE, Xtensa C and C++ compiler, and Instruction Set Simulator (ISS).

Currently RT600 supports node-locked license for Xtensa tools. A node-locked license permits tools to run on a specific computer, tied to the MAC address of the primary network interface permanently attached to the machine.

**Identify PC MAC Address** To generate the correct license file, you should first identify the appropriate MAC for the computer you plan to run Xtensa tools on. Remove ‘-’ or ‘:’ symbols in the MAC address.

**Windows:**

```
C:\Users>ipconfig /all

Wireless LAN adapter Wireless Network Connection:

Connection-specific DNS Suffix  . : us-sjo01.nxp.com
Description . . . . . : Intel(R) Dual Band Wireless-AC 8265
Physical Address. . . . . : 14-4F-8A-63-8C-33
DHCP Enabled. . . . . : Yes
```

**Linux:**

```
[user@rhel ~]$ ifconfig
eth0      Link encap:Ethernet  HWaddr 12:34:56:78:90:AB
```

**Note: For Linux:** MAC address must be associated with eth0 interface. If not, FLEXlm cannot perform the license checkout, compilation, or simulation of code is not possible. If the host has the MAC address associated with another interface, for example em1, the following approach may be used.

```
# Add udev rule for naming interface
$ sudo vim /etc/udev/rules.d/70-persistent-net.rules
# udev rule (replace 'XX' with the MAC address of your PC):
SUBSYSTEM=="net", ACTION=="add", ATTR{address}=="XX:XX:XX:XX:XX:XX", NAME="eth0"
# Change "em1" to "eth0" in your interfaces file.
$ sudo vim /etc/network/interfaces
# Restart udev or reboot machine
$ sudo reboot
```

Alternatively, you can use the approach recommended by your IT team to rename the interface to eth0.

**Parent topic:** Install License Key

**Download License Key** To download the license key:

1. Reload or return to the Tensilica URL: <https://tensilicatools.com/download/rt600-download-page/>.
2. Click the button and get the license key for RT600 SDK.



**CLICK TO GET A LICENSE KEY FOR RT600 SDK**

3. Provide the address.

### Get a License Key for RT600 SDK

Please Enter a MAC Address

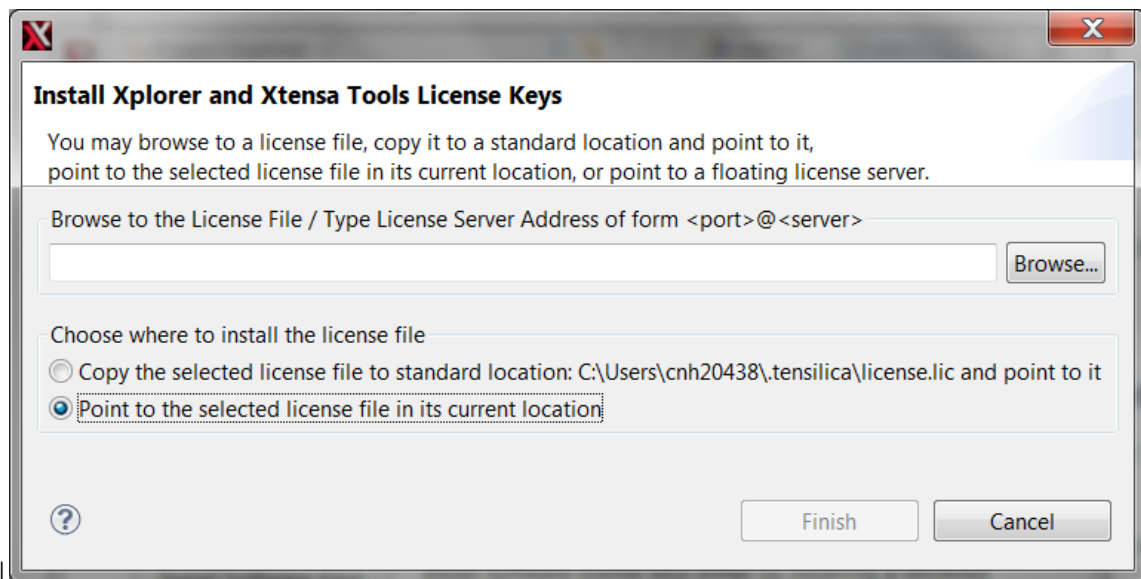
Your machine may have multiple MAC addresses. Please use one that is permanently installed. The license checker will confirm that the MAC address is present, even if that particular network interface isn't in use. For example, you can enter the MAC address for your LAN interface, and that will work even if you are using the WiFi interface for your development.

Accept

Cancel

4. Once the license file has been generated and downloaded, open your recently installed Xplorer V10.1.11.
5. Select menu **Help > Xplorer License Keys > Install Software Keys**.
6. Select the license key file.





7. Click the **Finish** button.

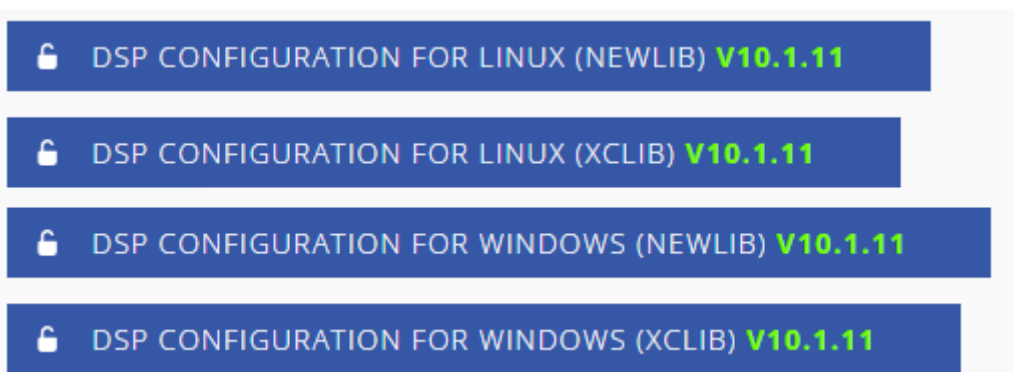
**Note:** The generated license file only supports debug/run on the RT600 device target. It does not support software simulation/Xplorer ISS. If you have special requirements to run the software simulations, contact Cadence directly.

**Parent topic:**Install License Key

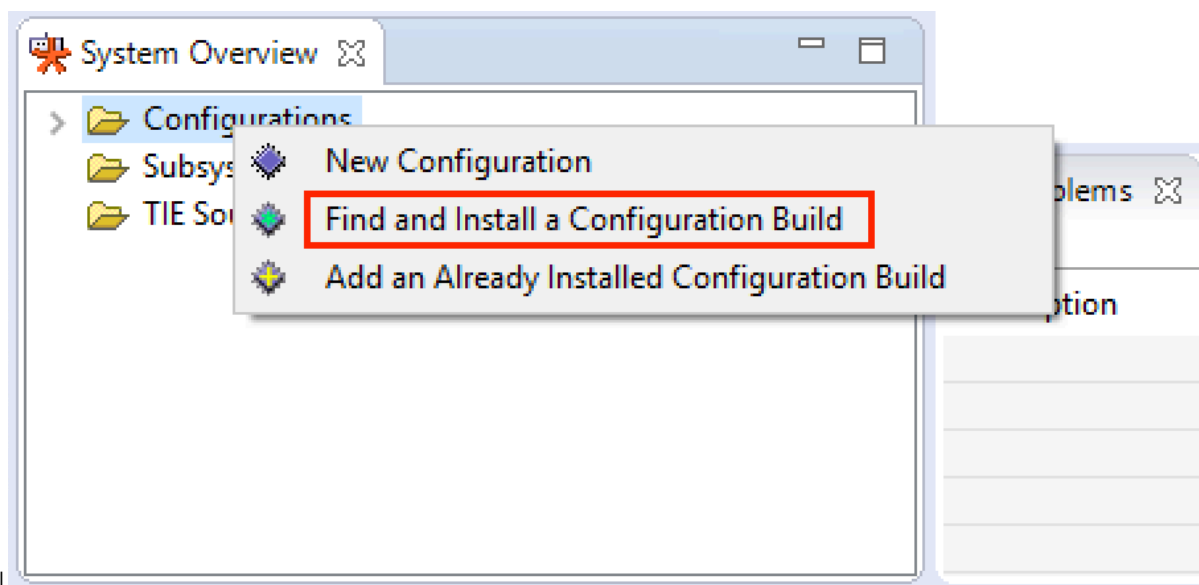
**Parent topic:**[Install Xplorer Toolchains](#)

**Install RT600 DSP Build Configuration** ‘*Build Configuration*’ is a term that describes all parameters and necessary build includes for the Tensilica processor implementation for development. It is mandatory to install a specific build configuration before starting development on RT600.

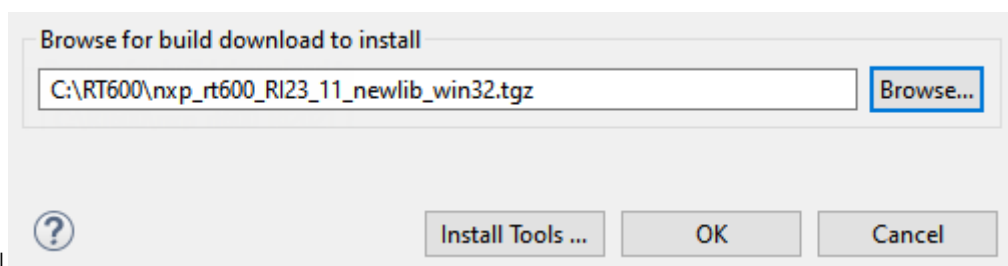
The build configuration is provided by NXP as a binary file and is imported into the Xplorer IDE. The binary file for the OS is available at the Tensilica URL: <https://tensilicatools.com/download/rt600-download-page/>.



The build configuration is installed into the IDE using the **System Overview** panel available in the lower left corner, by default. If the **System Overview** panel is not visible, toggle using the **Window > Show View > System Overview** menu item.



Click **OK** for build download to install.



Parent topic: [Install Xplorer Toolchains](#)

**Install Xtensa On Chip Debugger Daemon** The Xtensa On Chip Debugger Daemon (xt-ocd), is a gdb-based debugging tool but is not installed by default with the Xplorer IDE. For installation, a self-extracting executable installer is included with the IDE, which can be found at the following location:

#### Windows:

C:\usr\xtensa\XtDevTools\downloads\RI-2023.11\tools\xt-ocd-14.11-windows64-installer.exe

#### Linux:

~/xtensa/XtDevTools/downloads/RI-2023.11/tools/xt-ocd-14.11-linux64-installer

Currently, xt-ocd supports J-Link and Arm RVI/DSTREAM probes over Serial Wire Debug (SWD) for RT600. xt-ocd installs support for J-Link probes but does not install the required J-Link drivers which must be installed separately. Make sure that the latest version of J-Link software is installed.

**Note: For Linux:** When installing xt-ocd on Linux, ensure that the symlink is manually added to the installed J-Link driver: `ln -s <jlink-install-dir>libjlinkarm.so.6 <xocd-install-dir>/modules/libjlinkarm.so.6.`

xt-ocd is configured with an XML input file 'topology.xml' that modifies to fit the debugger hardware. Using J-link as an example, use the content below to replace the original template.

**Note:** It is mandatory to replace the 'usbser' section with the 9-digits number J-Link serial number on the back of the J-Link hardware in use.

```

<configuration>
  <controller id='Controller0' module='jlink' usbser='600100000' type='swd' speed='1000000' locking='1' />
  <driver id='XtensaDriver0' dap='1' xdm-id='12' module='xtensa' step-intr='mask,stepover,setps' />
  <chain controller='Controller0'>
    <tap id='TAP0' irwidth='4' />
  </chain>
  <system module='jtag'>
    <component id='Component0' tap='TAP0' config='trax' />
  </system>
  <device id='Xtensa0' component='Component0' driver='XtensaDriver0' ap-sel='3' />
  <application id='GDBStub' module='gdbstub' port='20000' sys-reset='0'>
    <target device='Xtensa0' />
  </application>
</configuration>

```

Below is another topology.xml example if Arm RealView ICE (RVI) and DSTREAM debug probes is used.

```

<configuration>
  <controller id='Controller0' module='rvi' />
  <driver id='XtensaDriver0' debug='' inst-verify='mem' module='xtensa' step-intr='mask,stepover,setps' />
  <driver id='TraxDriver0' module='trax' />
  <chain controller='Controller0'>
    <tap id='TAP0' irwidth='4' />
  </chain>
  <system module='jtag'>
    <component id='Component0' tap='TAP0' config='trax' />
  </system>
  <device id='Xtensa0' component='Component0' driver='XtensaDriver0' xdm-id='12' />
  <device id='Trax0' component='Component0' driver='TraxDriver0' xdm-id='12' />
  <application id='GDBStub' module='gdbstub' port='20000'>
    <target device='Xtensa0' />
  </application>
  <application id='TraxApp' module='traxapp' port='11444'>
    <target device='Trax0' />
  </application>
</configuration>

```

Congratulations! All Xplorer toolchains are installed.

For more details on Xtensa software tools, build configurations, or xt-ocd daemon, see the full set of documents in Xplorer menu **Help > PDF Documentation**.

**Parent topic:** [Install Xplorer Toolchains](#)

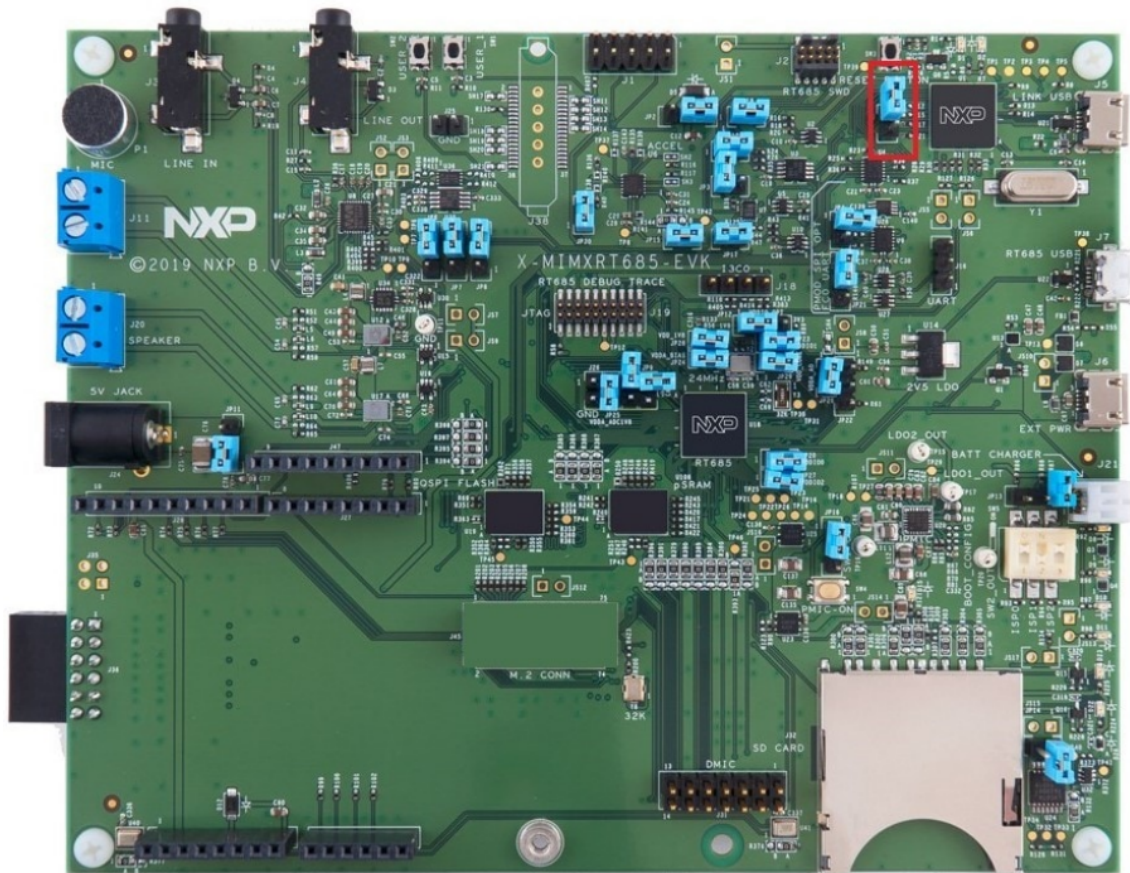
**Program LPC-Link2 with SEGGER J-Link** In addition to standalone probes, the onboard LPC-Link2 debug probe is used to debug HiFi4 DSP over SWD ports. The RT600 EVK has an LPC4300 MCU (top right corner on EVK) and by default has been pre-programmed as CMSIS-DAP probe. CMSIS-DAP is not compatible with HiFi4. Therefore, a few extra steps are required to flash it with J-Link firmware and support both Arm CM33 core and HiFi4 DSP core.

The steps are:

1. Install LPCScript, a command-line tool for programming onboard LPC-Link2 debug probe with the latest CMSIS-DAP and J-Link firmware. LPCScript is available for download from: <https://www.nxp.com/design/microcontrollers-developer-resources/lpc-microcontroller-utilities/lpcscript-v2.1.0:LPCSCRIPT>.
2. After the download is complete, run the installer.

**Note:** During the installation, the DFU, and VCOM drivers are automatically installed for all platforms.

3. To update the LPC-Link2 debug circuit firmware, unplug the USB cable on J5 and then connect to the DFULink jumper. In the MIMXRT685-EVK, JP1 is the LPCXpresso DFU jumper as shown in Figure 1.
4. Connect JP1 using the jumper.



5. Reconnect the board to the host computer over the debug link USB connector J5.
6. Launch LPCScript by double-clicking the *Boot LPCScript* file in the LPCScript install location: C:\ProgramData\Microsoft\Windows\Start Menu\Programs\LPCScript.

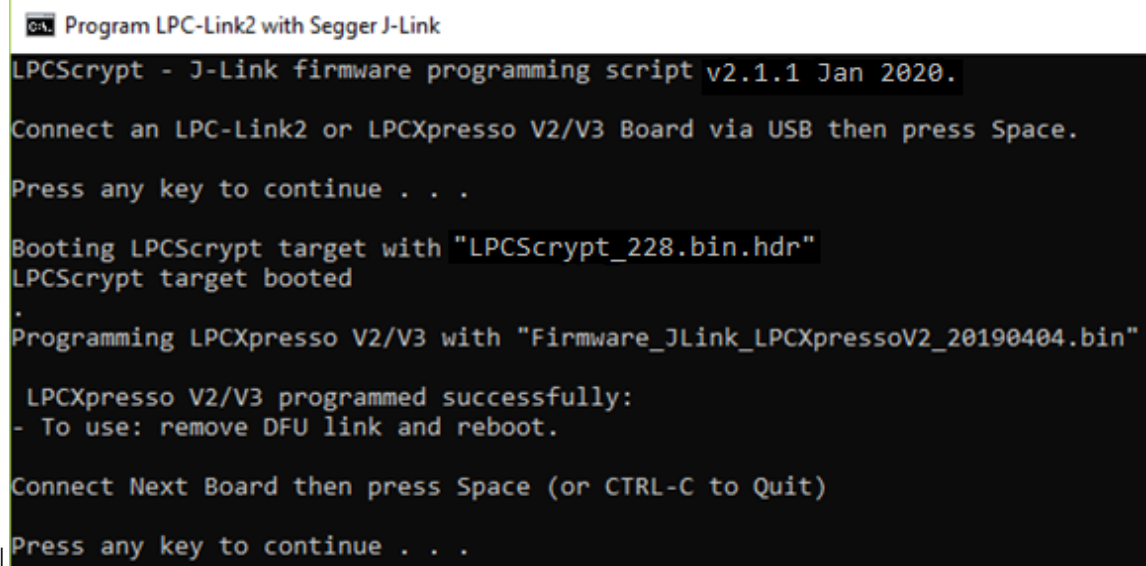
| Name                                 | Date modified      | Type              | Size |
|--------------------------------------|--------------------|-------------------|------|
| Boot LPCScript                       | 9/10/2018 10:14 AM | Shortcut          | 2 KB |
| LPCScript on the Web                 | 9/10/2018 10:14 AM | Internet Shortcut | 1 KB |
| Open LPCScript simple CLI            | 9/10/2018 10:14 AM | Shortcut          | 1 KB |
| Program LPC-Link2 with CMSIS-DAP     | 9/10/2018 10:14 AM | Shortcut          | 1 KB |
| Program LPC-Link2 with Segger J-Link | 9/10/2018 10:14 AM | Shortcut          | 1 KB |
| Uninstall LPCScript                  | 9/10/2018 10:14 AM | Shortcut          | 1 KB |

7. In that command shell, run the program\_JLINK script to install the JLink debug firmware.

```
<LPCScript Intall Dir>scripts\program_JLINK
```

**\*\*Note:\*\*** File paths in this document use the Windows directory separator, on Linux or Mac OSX. The file paths must be replaced with '/' For Windows users, shortcuts to the scripts are available from the LPCScript entry on the Start menu.

8. Verify once you select the firmware (in this case J-Link), LPCScript. The console appears as shown in Figure 4.



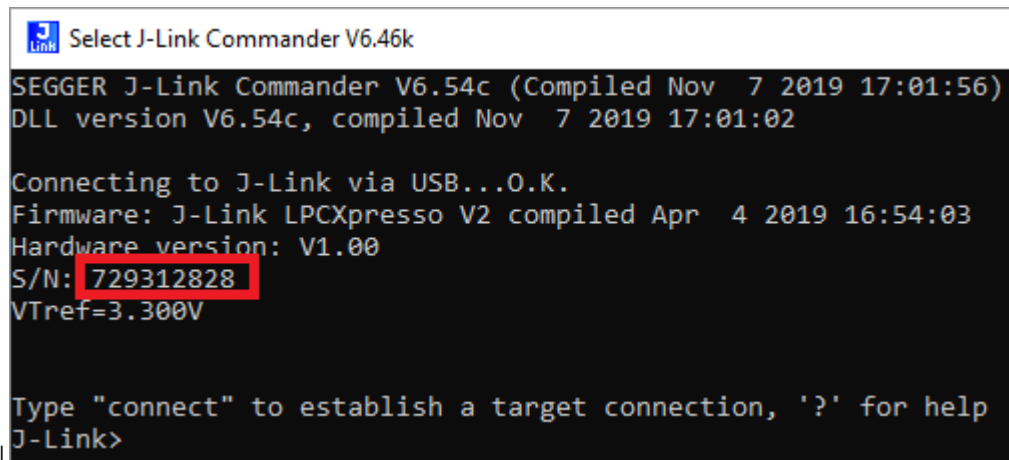
```

Program LPC-Link2 with Segger J-Link
LPCScript - J-Link firmware programming script v2.1.1 Jan 2020.
Connect an LPC-Link2 or LPCXpresso V2/V3 Board via USB then press Space.
Press any key to continue . . .
Booting LPCScript target with "LPCScript_228.bin.hdr"
LPCScript target booted
Programming LPCXpresso V2/V3 with "Firmware_JLink_LPCXpressoV2_20190404.bin"
LPCXpresso V2/V3 programmed successfully:
- To use: remove DFU link and reboot.
Connect Next Board then press Space (or CTRL-C to Quit)
Press any key to continue . . .

```

9. Open/ Disconnect JP1 and power cycle the board. The onboard LPC-Link2 is ready to be used as SEGGER J-Link probe.

Every EVK/ LPC-Link2 has a different J-Link S/N. Therefore, make sure to write down the S/N for xt-ocd and topology.xml as indicated in Install Xtensa On Chip Debugger Daemon.

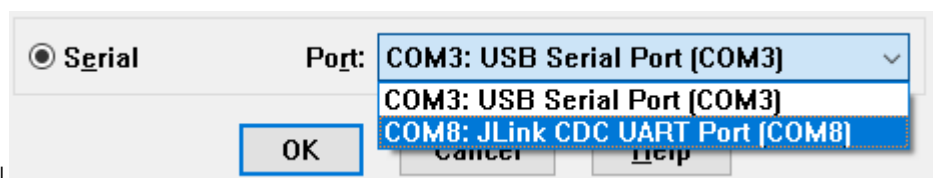


```

Select J-Link Commander V6.46k
SEGGER J-Link Commander V6.54c (Compiled Nov  7 2019 17:01:56)
DLL version V6.54c, compiled Nov  7 2019 17:01:02
Connecting to J-Link via USB...O.K.
Firmware: J-Link LPCXpresso V2 compiled Apr  4 2019 16:54:03
Hardware version: V1.00
S/N: 729312828
VTref=3.300V
Type "connect" to establish a target connection, '?' for help
J-Link>

```

10. Another benefit is that LPC-Link2 debug probe creates a virtual serial port over USB, so the extra UART2USB cable for debugging is not required.



The download link provides the document, demo videos and details on LPC-Link2. If you have any questions, [or have difficulties to program the probe, see \[https://www.nxp.com/design/microcontrollers-developer-](https://www.nxp.com/design/microcontrollers-developer-)

(continues on next page)



(continued from previous page)

↪resources/lpc-microcontroller-utilities/lpcscrypt-v2.1.0:LPCSCRYPT](https://www.nxp.com/design/  
 ↪microcontrollers-developer-resources/lpc-microcontroller-utilities/lpcscrypt-v2.1.0:LPCSCRYPT).

Parent topic:[Install Xplorer Toolchains](#)

**Install Xtensa Software Tools without IDE** The Xtensa Software Tools can also be installed without the use of the IDE. The installation without the IDE is useful in a command-line only Linux environment, or for better compatibility with an unsupported Linux environment.

The command-line tools package is available as a redistributable zip file that is extracted with an Xplorer IDE install. To gain access to the tools package, the IDE must be installed once in the organization. The tools package is available at: `~/xtensa/XtDevTools/downloads/RI-2023.11/tools/XtensaTools_RI_2023_11_linux.tgz..`

With the tools package and the DSP Build Configuration package available from the [Tensilica Tools download site](#), the toolchain can be set up as follows:

```
# Create Xtensa install root
mkdir -p ~/xtensa/tools
mkdir -p ~/xtensa/builds
# Set up the configuration-independent Xtensa Tool:
tar zxvf XtensaTools_RI_2023_11_linux.tgz -C ~/xtensa/tools
# Set up the configuration-specific core files:
tar zxvf nxp_rt600_RI23_11_newlib_linux_redist.tgz -C ~/xtensa/builds
# Install the Xtensa development toolchain:
cd ~/xtensa
./builds/RI-2023.11-linux/nxp_rt600_RI23_11_newlib/install \
--xtensa-tools./tools/RI-2023.11-linux/XtensaTools \
--registry ./tools/RI-2023.11-linux/XtensaTools/configg
```

Parent topic:[Install Xplorer Toolchains](#)

## Install MCUXpresso SDK

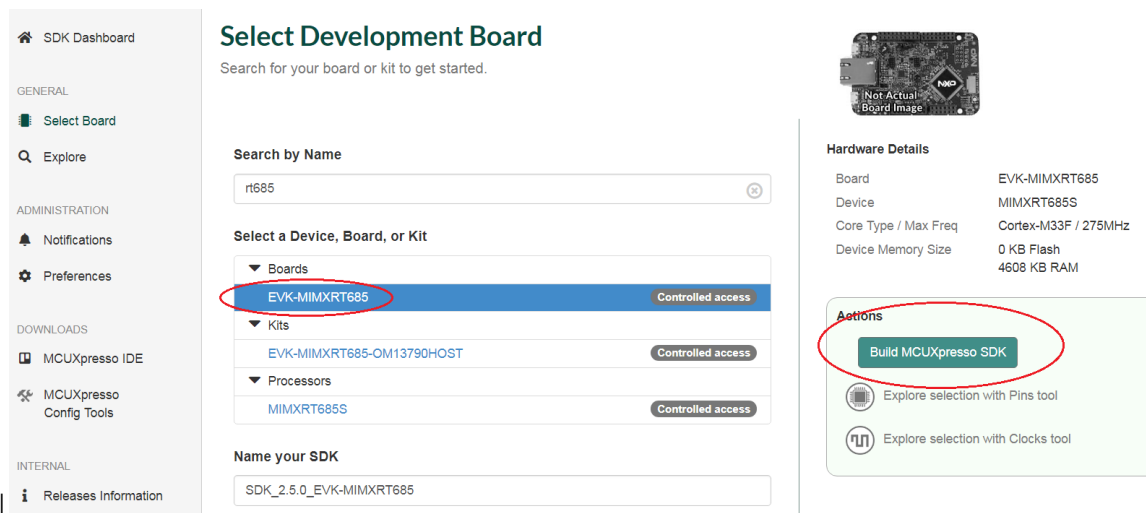
This section list the steps to:

- Download MCUXpresso SDK for RT600
- Enable MCUXpresso SDK DSP
- Initialize DSP Core
- Link DSP Profiles

**Download MCUXpresso SDK for RT600** DSP enablement for RT600, including drivers, middleware libraries, and demo applications are included with the latest RT600 SDK available for download from <https://mcuxpresso.nxp.com>. If you are accessing the site for the first time, make sure to register.

Once logged in, perform the following steps to use the SDK builder.

1. Click the link **Select Board /Processor** on the left panel.
2. In the **Search by Name** field, enter the name of the board. For example, *RT685*.
3. From the search results, select *EVK-MIMXRT685*.
4. Click the **Build MCUXpresso SDK** button on the right panel.



5. In the Build SDK for EVK-MIMXRT685 page, select the required environment settings.
6. Click the **Download SDK** button at the bottom of the page.
7. Save the archive in a local directory.

**Parent topic:** [Install MCUXpresso SDK](#)

**Enable MCUXpresso SDK DSP** The following DSP-specific enablements are available inside the MCUXpresso SDK release package for RT600.

- 

<SDK\_ROOT>/devices/MIMXRT685S/

Unified device and peripheral driver source code that can be compiled for both Arm and DSP cores.

**\*\*Note:\*\*** Only a limited subset of peripheral drivers and components is supported on the DSP.

~ ~ ~  
<SDK\_ROOT>/boards/evkmimxrt685/dsp\_examples/

DSP example applications

- 

<SDK\_ROOT>/middleware/multicore/rpmsg\_lite/

Unified RPMsg-Lite multicore communication library, with porting layers for Arm and DSP cores

~ ~ ~  
<SDK\_ROOT>/middleware/dsp/audio\_framework/

Xtensa Audio Framework \(\XAF\) for DSP core

- 

<SDK\_ROOT>/middleware/dsp/audio\_framework/libxa\_af\_hostless/



Source code and documentation for the core XAF framework

```
-   \
<SDK_ROOT>/middleware/dsp/audio_framework/testxa_af_hostless/
```

- Utilities and tests for developing applications with the XAF framework

```
<SDK_ROOT>/middleware/dsp/audio_framework/testxa_af_hostless/test/plugins
```

- XAF components and codec binaries

- 

```
<SDK_ROOT>/middleware/dsp/naturedsp/hifi4/
```

NatureDSP Math Library for HiFi4 DSP

**\*\*Parent topic:\*\***[Install MCUXpresso SDK](../topics/install\_mcuxpresso\_sdk.md)

**Initialize DSP Core** To minimize the power consumption, the DSP core is not powered when RT600 boots up. To initialize, run, or debug DSP applications, you must execute some code on the Arm core.

The DSP management interface library provided in the SDK is available at the location: <SDK\_ROOT>/devices/MIMXRT685S/drivers/fsl\_dsp.c.

```
/* Initialize DSP core. */
void DSP_Init(void);
/* Deinit DSP core. */
void DSP_Deinit(void);
/* Copy DSP image to destination address. */
void DSP_CopyImage(dsp_copy_image_t *dspCopyImage);
/* Start DSP core. */
void DSP_Start(void);
/* Stop DSP core. */
void DSP_Stop(void);
```

The SDK includes a helper function used by the DSP example applications at: <SDK\_ROOT>/boards/evkmimxrt685/dsp\_examples/dsp\_support.c.

```
/* Prepare DSP core for code execution:
- Setup PMIC for DSP
- Initialize DSP clock and core
- (Optional) Copy DSP binary image into RAM
- Start DSP core
*/
void BOARD_DSP_Init(void);
```

After executing this function during your Arm application startup, the DSP is initialized and ready to run. From here, code is loaded and debugged on the DSP with Xplorer IDE and tools.

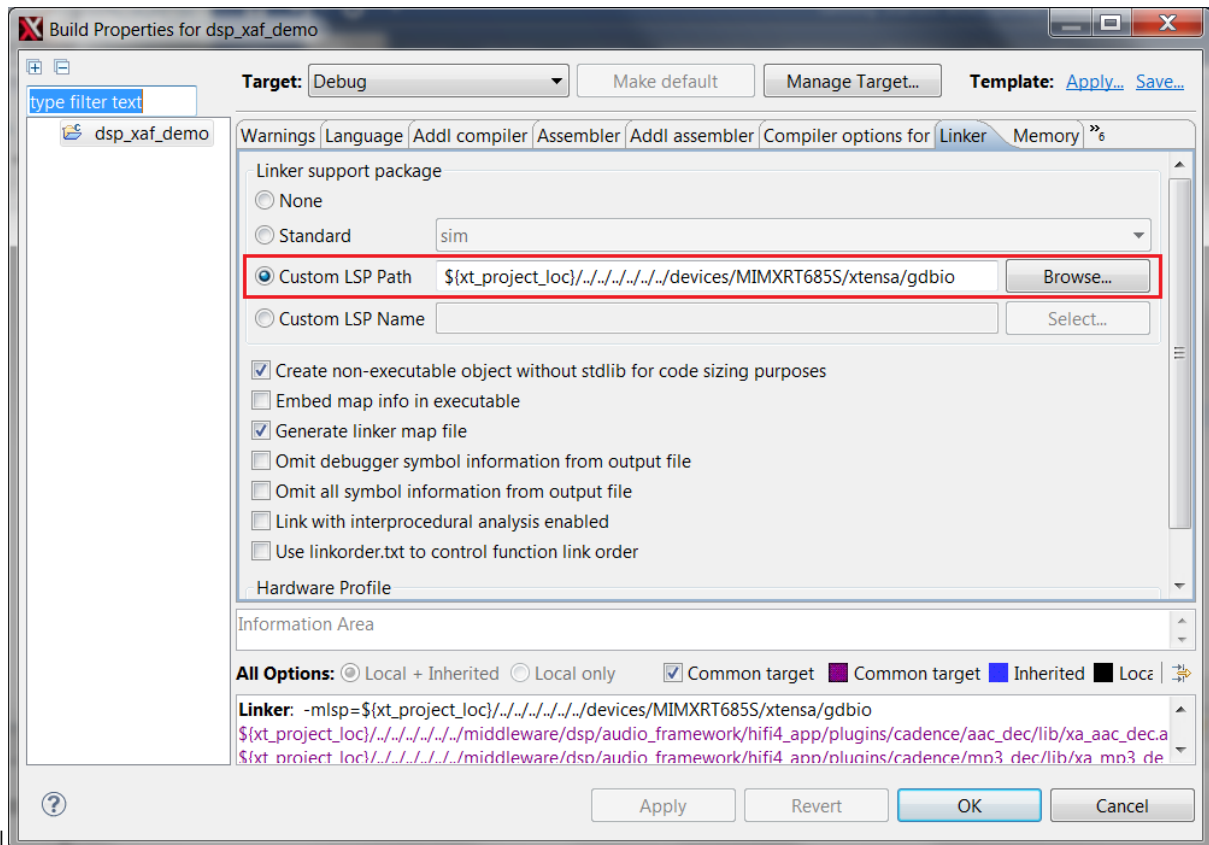
**Parent topic:**[Install MCUXpresso SDK](#)

**Link DSP Profiles** The Xtensa Software Tools use the linker support packages (LSPs) to link a HiFi4 DSP application for the RT600. An LSP includes both a system memory map and a collection

of libraries to include into the final binary. The LSPs are provided in the MCUXpresso SDK and are available at: `<SDK_ROOT>/devices/MIMXRT685S/xtensa/`.

DSP sample applications are configured to link against one of these custom LSPs. By default, *Debug* targets are linked against the *gdbio* LSP intended for use with an attached debugger and captures I/O requests (`printf`) through *gdb*. The *Release* target links against the *min-rt* LSP, which includes minimal runtime support.

To change the LSP used by a project target in the Xplorer IDE, use the **Linker** tab in the project **Build Properties** dialog box.



The MCUXpresso SDK ships with other standard LSPs for RT600. For more information on LSPs and creating a custom memory map using Xtensa software tools, see the Cadence Linker Support Packages (LSPs) Reference Manual.

**Parent topic:** [Install MCUXpresso SDK](#)

## Run and Debug DSP Demo using Xplorer IDE

This section lists the steps to:

- Prepare Arm Core for 'Hello World'
- Start Xtensa Debugger Daemon
- Prepare DSP Core for 'Hello World'
- Run and Debug DSP Audio Framework
- Launch DSP Application from Arm Core

**Prepare Arm Core for ‘Hello World’** Each of the DSP demos included in the MCUXpresso SDK consists of two separate applications that run on the Arm core and DSP core. The Arm core application initializes the DSP core in the manner described in Initialize DSP Core and executes other application-specific functionality.

To debug the ‘Hello World’ DSP application, you must first set up and execute the Arm application using an environment of your choice.

- Build and execute the ‘Hello World’ Arm demo located in: `<SDK_ROOT>/boards/evkmimxrt685/dsp_examples/hello_world_usart/cm33/`.

Preparing an Arm core development environment is outside the scope of this document. For information on how to use the SDK for Arm core development, see the *Getting Started with MCUXpresso SDK for MIMXRT600.pdf* document at `<SDK_ROOT>/docs/`.

**Note:** IAR Embedded Workbench may require a patch to enable compatibility with RT600. For details on the patch, contact NXP directly.

**Note:** If you are using MCUXpresso, it is highly recommended to upgrade to latest version of the SDK and match the latest EVK board.

**Parent topic:** [Run and Debug DSP Demo using Xplorer IDE](#)

**Start Xtensa Debugger Daemon** To debug DSP applications on RT600, ensure that the xt-ocd daemon is running. This application runs a gdb server that the Xtensa core debugger connects to.

Go to the command-line window and change directory (cd) to xt-ocd daemon installation path. The default path on Windows is `C:\Program Files (x86)\Tensilica\Xtensa OCD Daemon 14.08`.

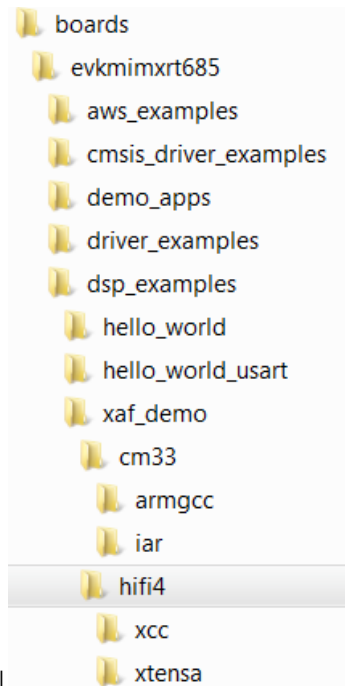
Execute the daemon with the custom topology.

```
**xt-ocd.exe -c topology.xml**
XOCD <VERSION_INFO>
(c) 1999-2021 Cadence Design Systems Inc. All rights reserved.
[Debug Log 2021-06-17 08:12:29]
Loading module "gdbstub" <VERSION_INFO>
Loading module "jlink" <VERSION_INFO>
Using JLINK lib <VERSION_INFO>
Jlink USB Serial Number: XXXXXXXXXX
Connected to Jlink Device:
  Name:'J-Link LPCXpresso V2 compiled Apr  4 2019 16:54:03'
  S/N:XXXXXXXX
  Firmware: J-Link LPCXpresso V2 compiled Apr  4 2019 16:54:03
  Requested/Set TCK: 2000kHz/65534kHz
Jlink: Select SWD
SWD-DP with ID 0x6BA02477
Loading module "jtag" <VERSION_INFO>
Loading module "xtensa" <VERSION_INFO>
Starting thread 'GDBStub'
Opened GDB socket at port 20000
Initialize XDM driver
Warning: Warning: DAP Reset request failed! Ignoring...
```

**Note:** Some warning messages are expected and can be ignored. If you receive an error initializing the XDM driver, initialize and start the DSP core before debugging. For details on initializing and debugging, see Initialize DSP Core and Link DSP Profiles. For details on xt-ocd runtime options and configuration, see Chapter 7 of the Xtensa Debug Guide, available in **Help > PDF Documentation**.

**Parent topic:** [Run and Debug DSP Demo using Xplorer IDE](#)

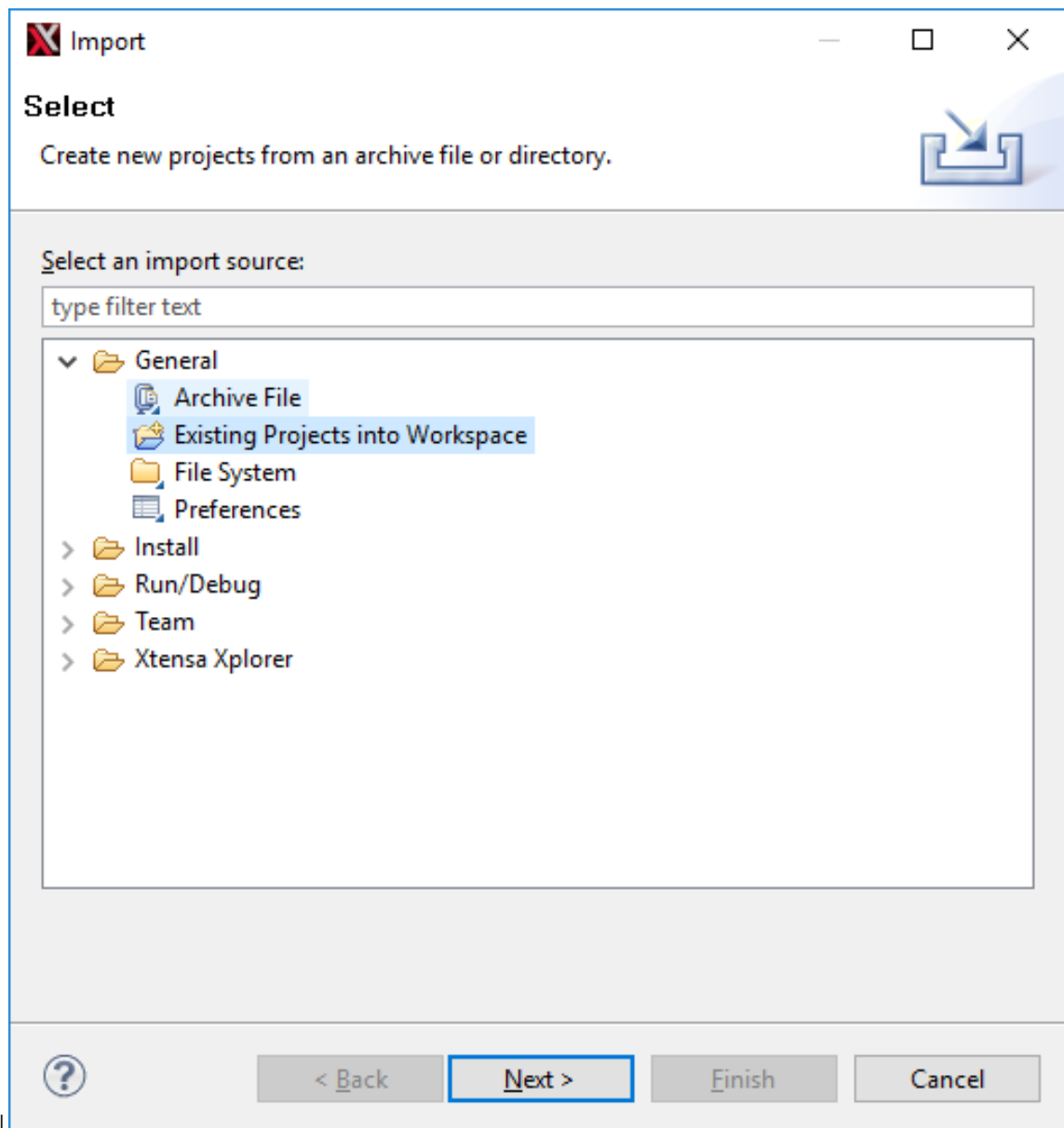
**Prepare DSP Core for ‘Hello World’** The RT600 SDK provides a collection of DSP example applications located in `boards/evkmimxrt685/dsp_examples/`. Each DSP example has two source directories, one for the Arm Cortex-M33 core (‘cm33’) and one for the DSP HiFi4 core (‘dsp’).



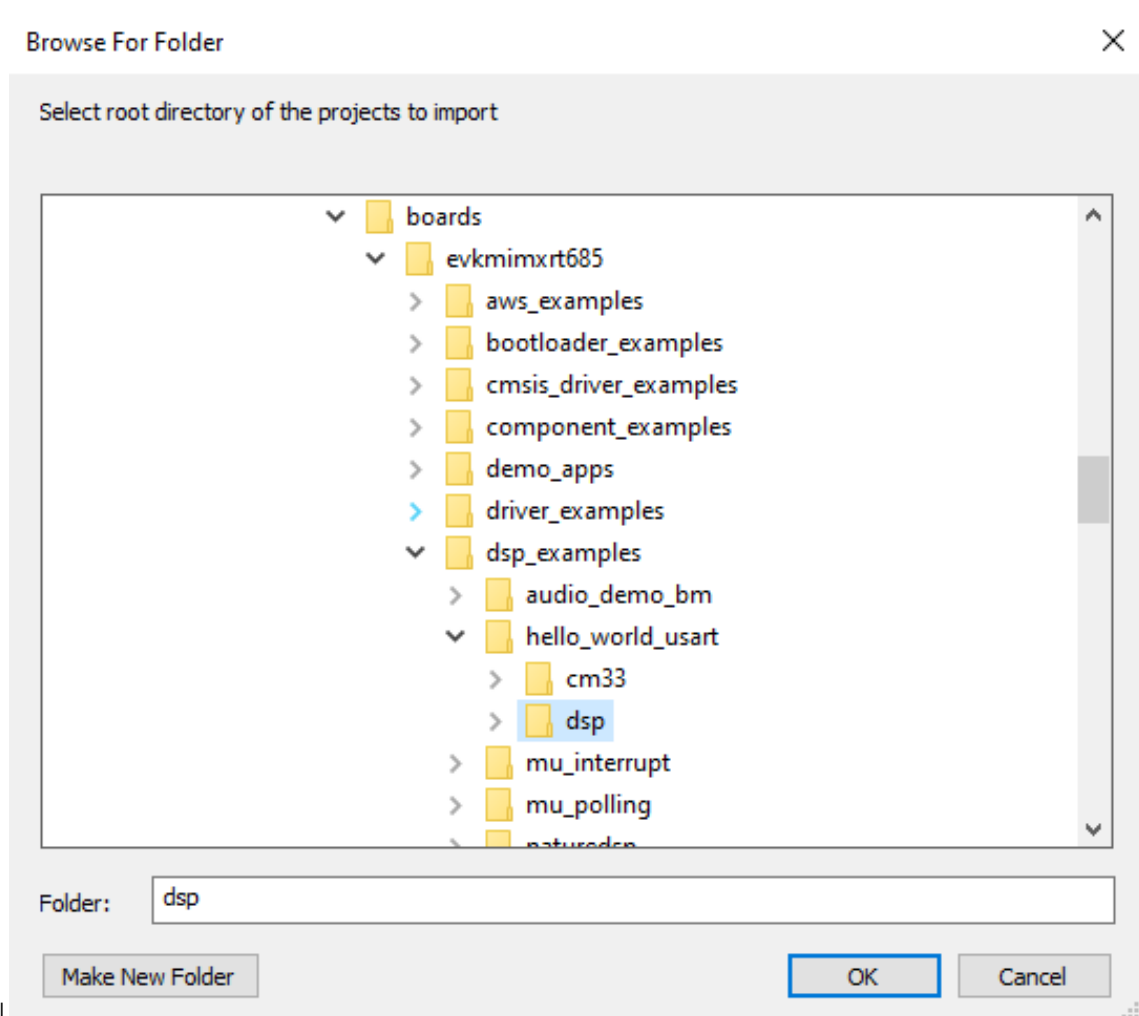
The projects for different supported toolchains are build in the above directories. For the DSP example above, the ‘xcc’ project builds on the command line and the ‘xtensa’ directory is an Xplorer IDE project.

To run the ‘Hello World’ demo, import the SDK sources into the Xplorer IDE.

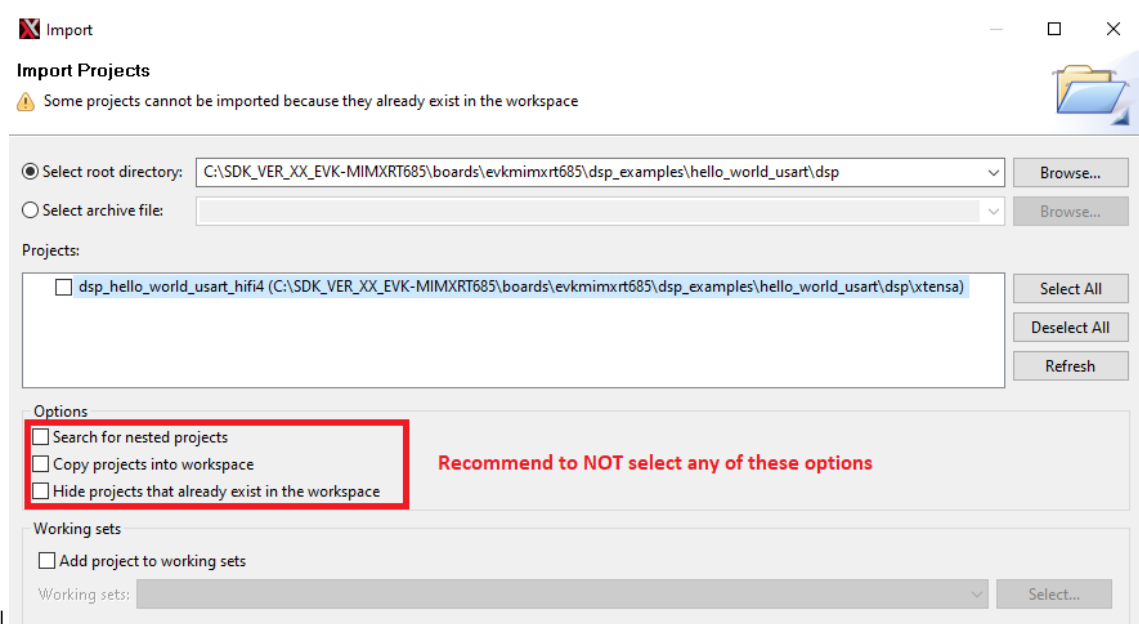
1. Select **File > Import > General > Existing Projects into Workspace**.



2. Select the SDK directory `<SDK_ROOT>\boards\evkmimxrt685\dsp_examples\hello_world_usart\dsp\xtensa` as root directory.

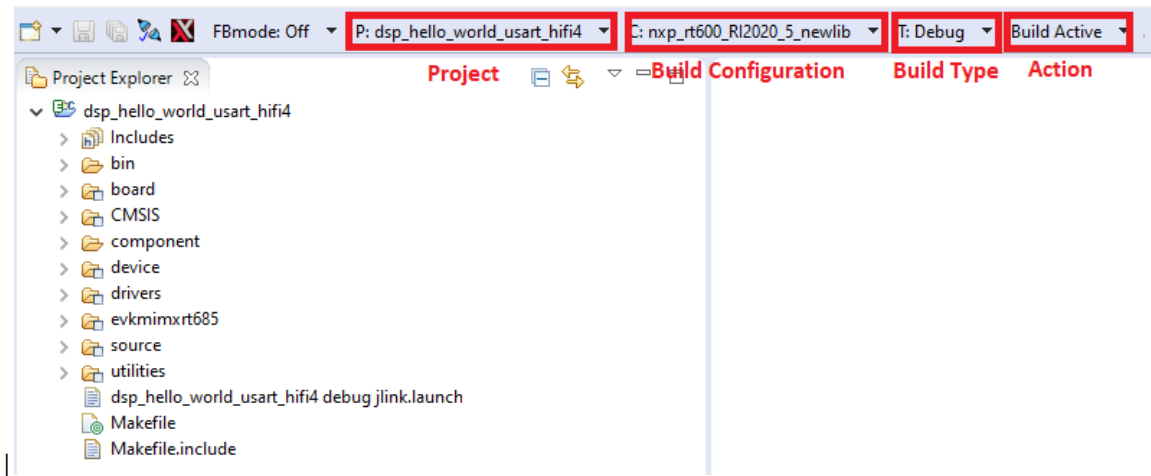


3. Leave all the other check boxes blank.

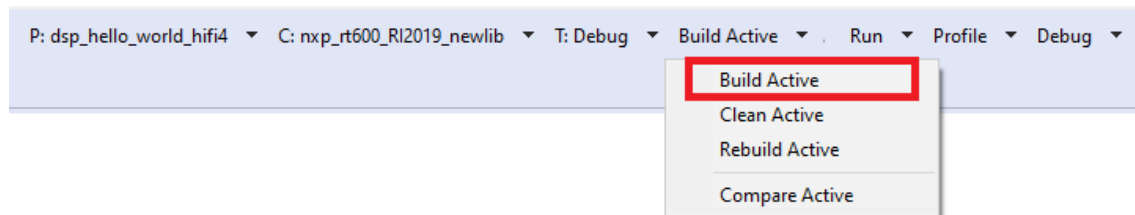


Once imported, the 'dsp\\_hello\\_world\\_usart\\_hifi4' appears in the Project Explorer.

4. To make a build selection for the project and hardware target configuration, use the drop-down buttons on the menu bar.

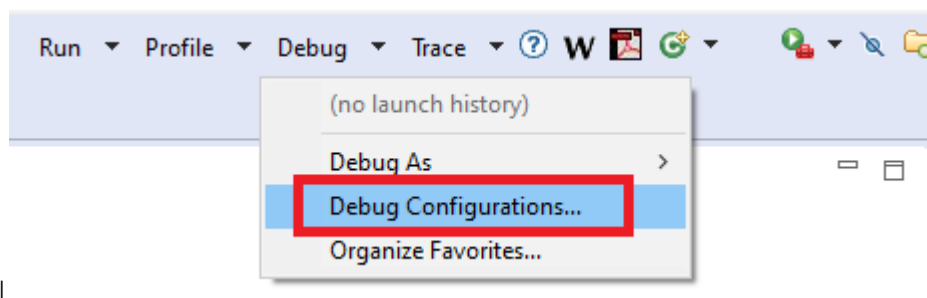


5. To build the project for debug, profile, or trace, use the action buttons on the right side of the menu bar.



A default debug configuration is provided by the SDK project, which utilizes the on-chip debugger.

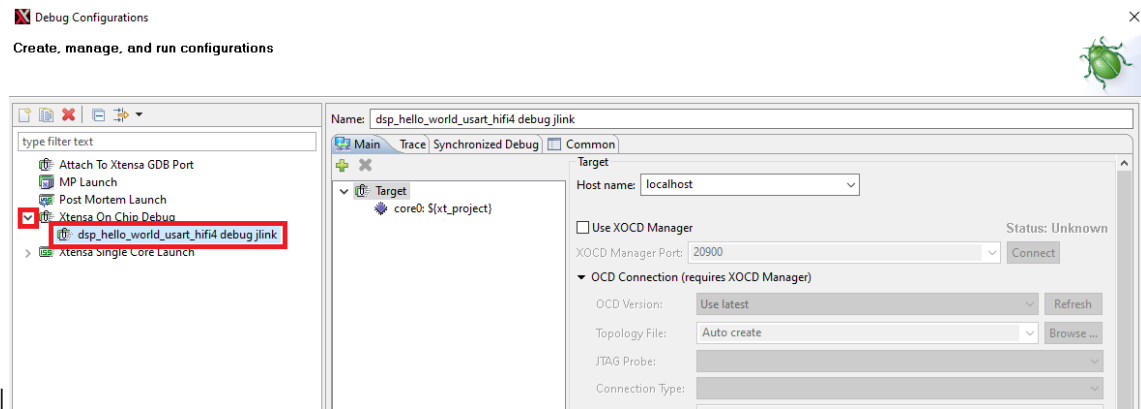
6. To choose the configuration, select the **Debug Configurations** menu item.



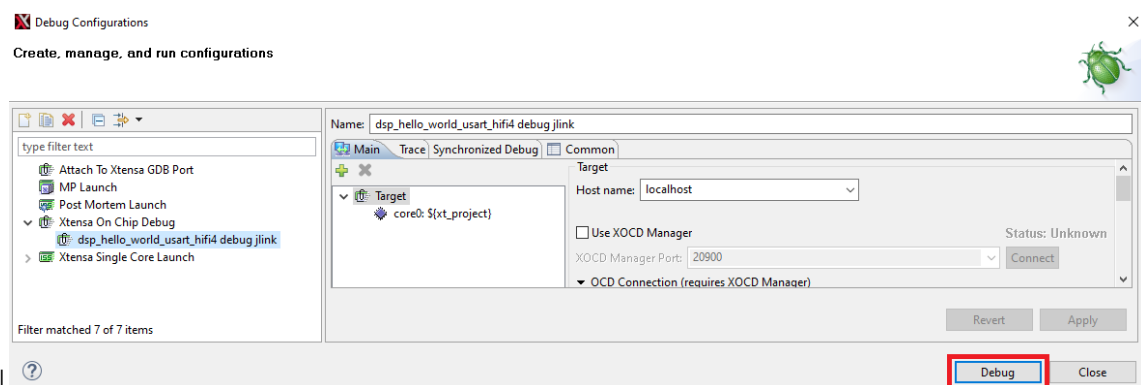
The **\*\*Debug Configurations\*\*** dialog box appears.

7. Expand the **Xtensa On Chip Debug** tree option and select 'dsp\_hello\_world\_hifi4\_debug\_jlink'.



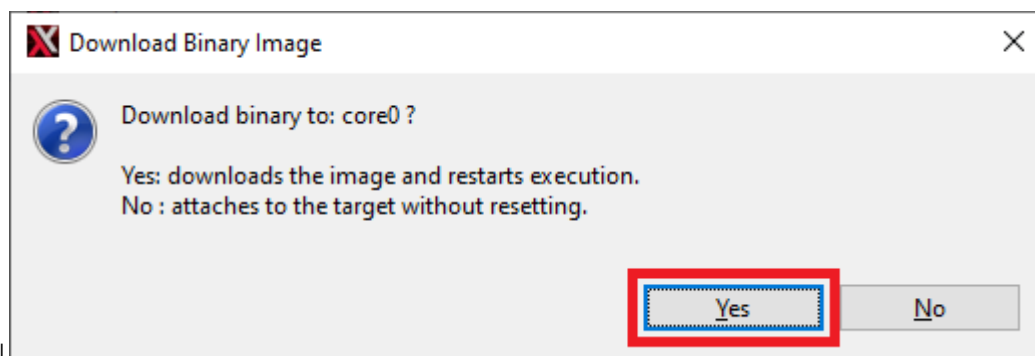


8. Click the **Debug** button. The actual debug on the chip initiates.



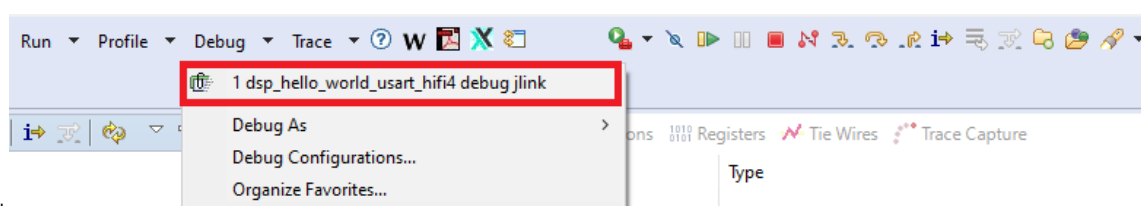
The Xplorer IDE prompts whether binaries should download to the hardware.

9. Select **Yes**.



10. The Xplorer IDE transitions to the **Debug** perspective after the binary download.

**Note:** After initial configuration, it is possible to select the same debug configuration as in Figure 8.



The **Debug** perspective appears as in [Figure 9](prepare\_dsp\_core\_for\_hello\_world.md#GVDVWQC).



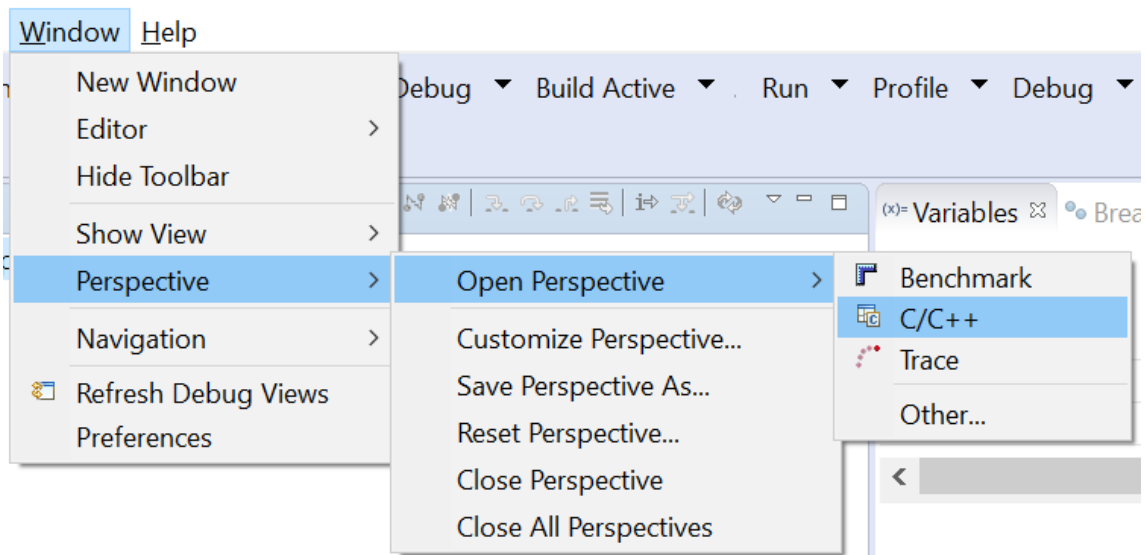
The program stops at the start of main\(\) function. To run the program, click the **Resume** or **Stepping through** icon button as shown in [Figure 10](prepare\_dsp\_core\_for\_hello\_world.md#LKWEUGG).



- After resuming / stepping through the 'printf' statement, the following output appears in the **Console** view of the IDE.

```
Hello World running on core nxp_rt600_RI2020_5_newlib
```

To return to the default IDE layout after the debugging completes, select the previous code perspective.



**Parent topic:** [Run and Debug DSP Demo using Xplorer IDE](#)

**Run and Debug DSP Audio Framework** The DSP audio framework demo consists of separate applications that run on the Arm core and DSP core. The Arm application runs a command shell and relays the input requests to the DSP application using RPMsg-Lite.

**EVK Board Setup for Audio Demo** The DSP audio demo is tested against EVK-MIMXRT685 Rev E and requires UART for serial console.

For the codec to output audio properly, attach one jumper on the board as follows:

JP7-1 <==> JP8-2

**Note:** DSP operates DMA and audio peripherals same as CM33 side. Jumper settings, I2S configs, or pin mux settings are all same with CM33 side. For information on steps to operate I2S, DMIC and DMA, see the SDK CM33 side driver examples. SDK\boards\evkmimxrt685\driver\_examples\i2s;

**Note:** Jumper settings are different on different version of EVKs or validation boards. The above mentioned setting is for SDK 2.8.0 and Rev E EVK or later only. For old version of boards or SDKs, double check the SDK driver examples for correct settings for the pin mux and jumper settings.

The demo uses the UART for console input and output. Connect the EVK board to a PC via the USB debug interface (J5) and open up a serial interface on your PC using a terminal tool such as PuTTY on Windows or screen on Linux.

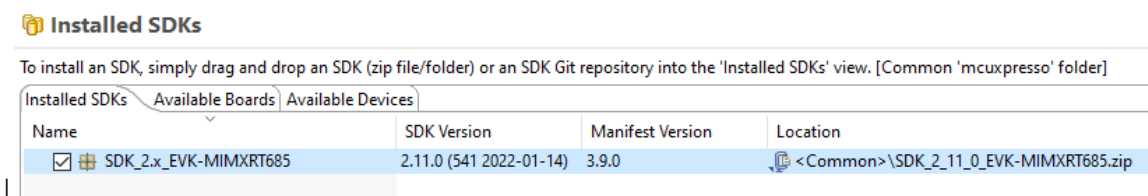
**Parent topic:** Run and Debug DSP Audio Framework

**Debug Audio Demo** To debug this DSP application, set up and execute the Arm application in a desired environment. For more information on Arm development environment options, see ‘Getting Started with MCUXpresso SDK for EVK-MIMXRT685.pdf.’

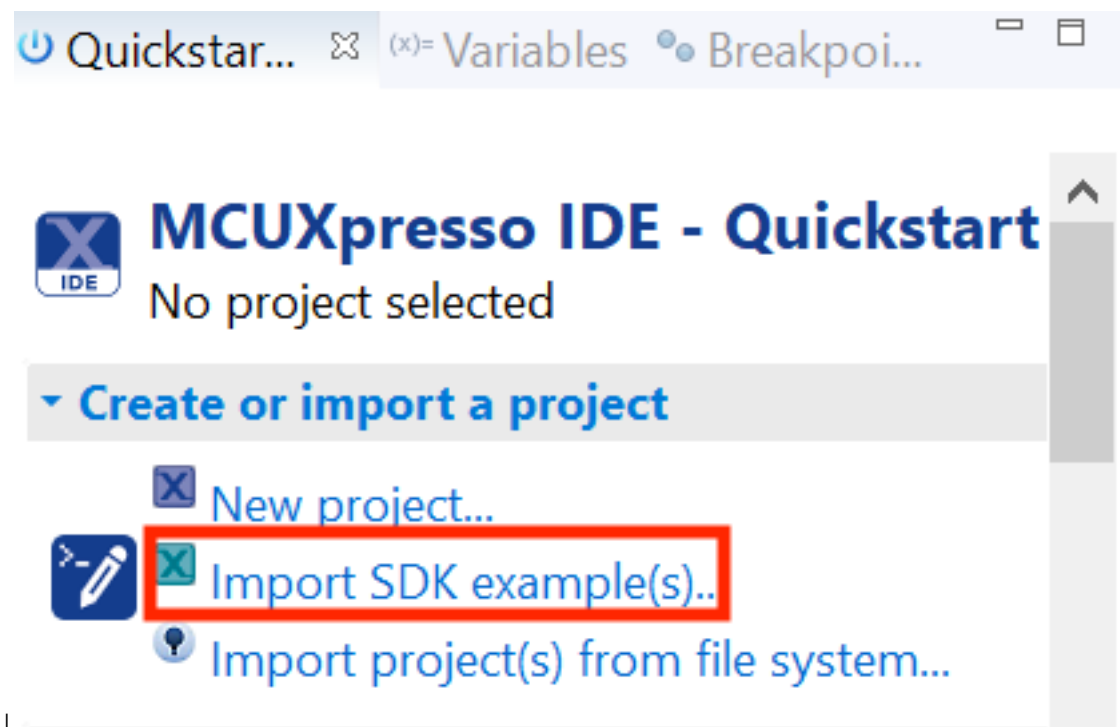
The following example uses the NXP MCUXpresso IDE for the Arm environment.

To debug the audio demo, perform the following steps.

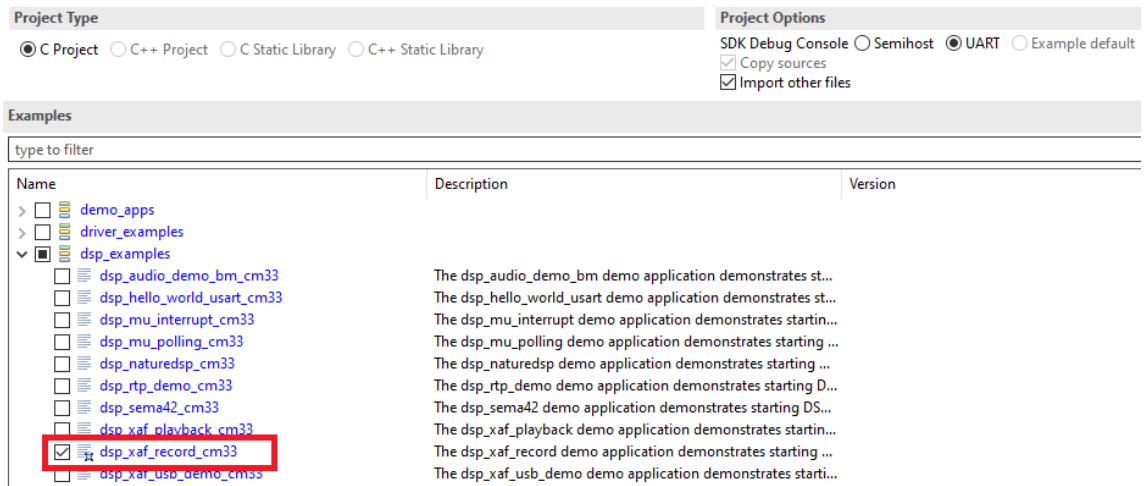
1. Install the MCUXpresso SDK for RT600 into the MCUXpresso IDE using the Installed SDK panel as shown in



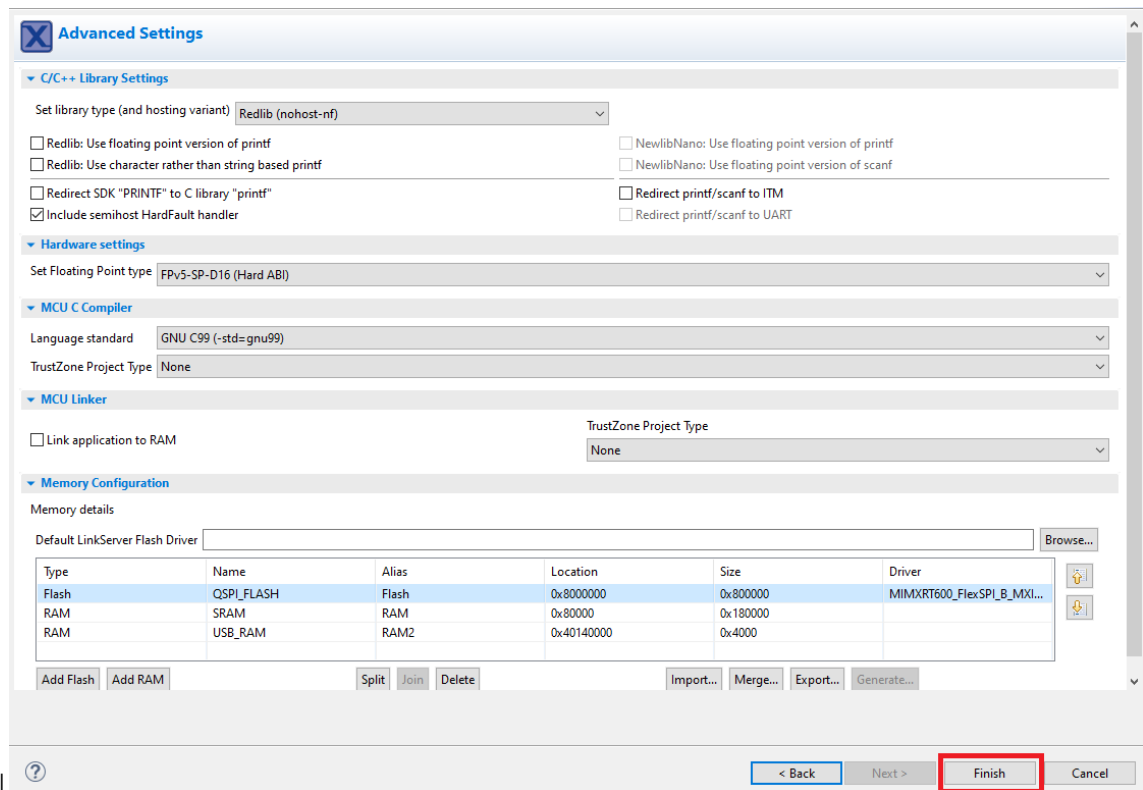
2. To import an example from the installed SDK., use the **Import SDK examples** link in the **Quickstart** panel on the lower left of the screen.



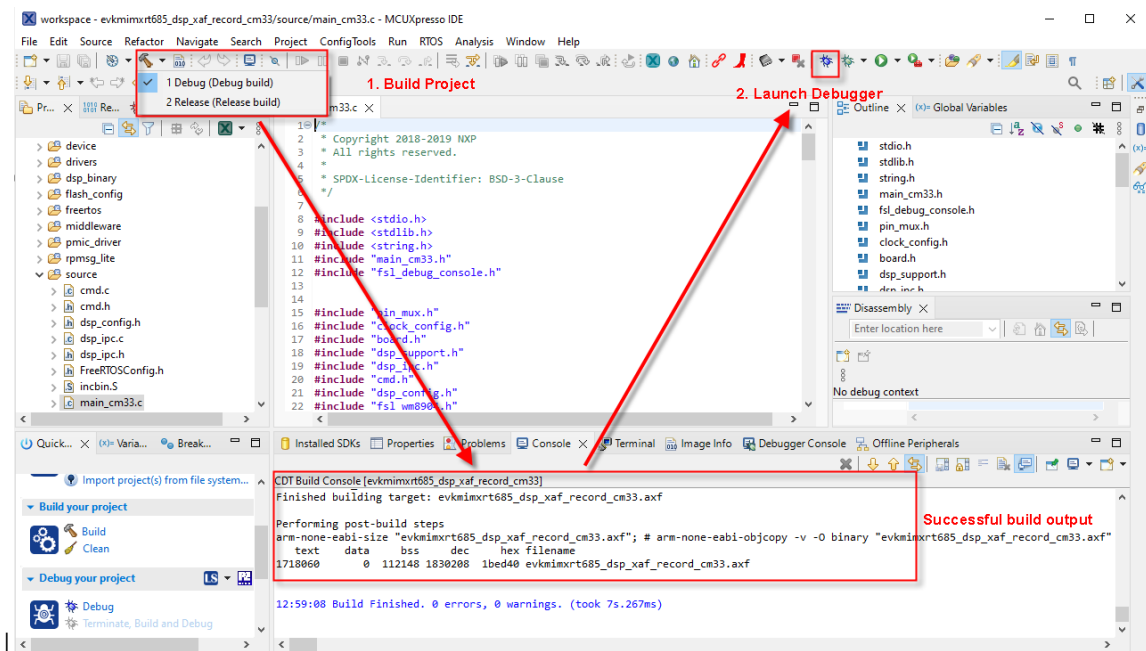
3. Select the ‘*dsp\_xaf\_record\_cm33*’ example for Cortex-M33 core.



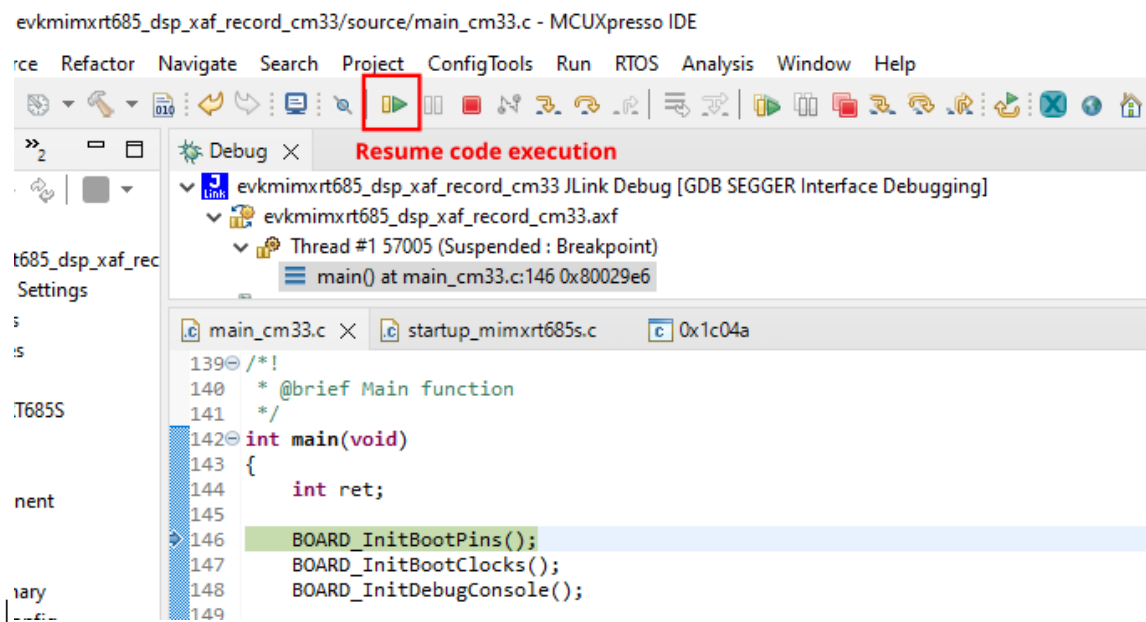
4. Click the **Finish** button.



5. Build the project and launch the debugger on success.



6. Use the debug toolbar to resume the code execution.



7. Observe serial terminal output with shell prompt.

```
*****
DSP audio framework demo start
*****
Configure WM8904 codec
DSP image copied to DSP TCM
[APP_DSP_IPC_Task] start
[APP_Shell_Task] start
Copyright 2022 NXP
>>
```

8. Using the Xplorer IDE, load and execute xaf\_record using the steps described in Prepare DSP Core for 'Hello World'.

- After the DSP application runs, use the serial shell to invoke the 'record\_dmic [language]' command. For information on the supported languages, check VIT ReleaseNotes.txt. Using the serial shell creates an audio pipeline that captures microphone audio, perform voice recognition (VIT), and playback via the codec speaker line out (J4 on the EVK).

```
>> help
"help": List all the registered commands
"exit": Exit program
"version": Query DSP for component versions
"record_dmic": Record DMIC audio , perform voice recognition (VIT) and playback on codec
USAGE: record_dmic [language]
```

For voice recognition say supported the wake-word and in 3 s say the frame supported command. If selected model contains strings, then the wake-word and the list of commands appears in the console.

**Note:** this command does not return to the shell

- The serial terminal shows the IPC communications to set up an audio device..

```
>> record_dmic en
Command not recognized. Enter 'help' to view a list of available commands.
>> record_dmic en
[CM33 CMD] Setting VIT language to en
[DSP Main] Number of channels 2, sampling rate 16000, PCM width 16
[CM33 CMD] [APP_DSP_IPC_Task] response from DSP, cmd: 19, error: 0
[DSP Record] Audio Device Ready
[CM33 CMD] DSP DMIC Recording started
[DSP VoiceSeeker] VoiceSeekerLight_GetRequiredHeapMemoryBytes: 51120 bytes
[CM33 CMD] To see VIT functionality say wakeword and command
[DSP VoiceSeeker] VoiceSeekerLight_Create: voiceseeker_status = 0
[DSP VoiceSeeker] VoiceSeekerLight_GetLibVersion: v0.6.0
```

The VIT wake-word and supported commands appear in the serial console.



The serial console prints the detected wake-word and commands.



For more information on configuration and using the Audio Framework demo, see the file ``<SDK_ROOT>\boards\evkmimxrt685\dsp_examples\xaf_record\cm33\readme.txt`.`

For more details about VIT which supports English and Mandarin models, see ``<SDK_ROOT>\middleware\vit\HIFI4\Doc\VIT_Integration_Guide.pdf`.`

**Parent topic:**Run and Debug DSP Audio Framework

**Parent topic:**[Run and Debug DSP Demo using Xplorer IDE](#)

**Launch DSP Application from Arm Core** In the previous example, the Arm application and DSP application are independently loaded and debugged. This section list the steps to produce one Arm application binary that includes and starts the DSP application without the use of a debugger or a loader.

The Arm core application for each DSP demo uses a global preprocessor macro to control loading of the DSP binary application:

## DSP\_IMAGE\_COPY\_TO\_RAM

When this macro is set to '1'/TRUE (default value), it instructs the DSP demo application to do the following:

- Link the DSP application binary images into the Arm binary
- Copy the DSP application images into RAM on program boot
- Initialize the DSP to run from the RAM image

**Note:** When modifying the Arm project, make sure to supply the global preprocessor macro to both the C compiler and assembler.

To build the DSP application image for the Arm application, select the **Release** target option in the Xplorer IDE. For more information on building with min-rt LSP, see section Link DSP Profiles.



Three DSP binaries are generated and are loaded into different TCM or SRAM address segments:

- <SDK\_ROOT>/boards/evkmimxrt685/dsp\_examples/xaf\_record/dsp/binary/\*\*dsp\\_data\\_release.bin\*\*
- <SDK\_ROOT>/boards/evkmimxrt685/dsp\_examples/xaf\_record/dsp/binary/\*\*dsp\\_text\\_release.bin\*\*
- <SDK\_ROOT>/boards/evkmimxrt685/dsp\_examples/xaf\_record/dsp/binary/\*\*dsp\\_ncache\\_release.bin\*\*

**Note:** Depending on the environment, there might be a need to copy the binary images into the Arm application workspace.

**Parent topic:** [Run and Debug DSP Demo using Xplorer IDE](#)

## Run and Debug from Command-Line Environment / LINUX

RT600 SDK has been configured to be as flexible to support multiple toolchains, including the Arm GCC and XCC command-line environments. The principles and essentials are still the same as with the IDE. The command-line environment settings are nearly identical between WIN32 and LINUX, with few different path settings.

**Build and Debug Arm Application** The Arm application requires the GNU Arm Embedded Toolchain and CMake version 3.x for command-line compile and linking. For more information on installation and configuration of required build tools for command-line development, see the *Getting Started with MCUXpresso SDK for EVK\_MIMXRT685.pdf* in the <SDK\_ROOT>/docs/ directory.

To build and debug:

1. Launch a command prompt / terminal and change directory to the xaf\_record application.



```

user@linux:~/SDK/boards/evkmimxrt685/dsp_examples/xaf_record/cm33/armgcc$ ls -l
build_all.bat
build_all.sh
build_debug.bat
build_debug.sh
build_flash_debug.bat
build_flash_debug.sh
build_flash_release.bat
build_flash_release.sh
build_release.bat
build_release.sh
clean.bat
clean.sh
CMakeLists.txt

```

## 2. Use .bat files to build the configuration on Windows, and .sh files on Linux/UNIX.

```

user@linux:~/SDK/boards/evkmimxrt685/dsp_examples/xaf_record/cm33/armgcc$ ./build_debug.sh
...
[100%] Linking C executable debug/dsp_xaf_record_cm33.elf
[100%] Built target dsp_xaf_record_cm33.elf

```

## 3. Launch the GDB server.

```

user@linux:/opt/JLink$ ./JLinkGDBServerCLExe -device MIMXRT685S_M33 -if SWD
SEGGER J-Link GDB Server V6.46j Command Line Version
...
Listening on TCP/IP port 2331
Connecting to target...Connected to target
Waiting for GDB connection...

```

## 4. Connect with GDB to the device and load Arm application.

```

user@jlinux:~/SDK/boards/evkmimxrt685/dsp_examples/xaf_record/cm33/armgcc$ arm-none-eabi-
→gdb debug/dsp_xaf_record_cm33.elf
...
Reading symbols from debug/dsp_xaf_record_cm33.elf...
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
0x1301ec7a in ?? ()
(gdb) mon reset
Resetting target
(gdb) load
Loading section .flash_config, size 0x200 lma 0x7f400
Loading section .interrupts, size 0x130 lma 0x80000
Loading section .text, size 0xe330 lma 0x80130
Loading section CodeQuickAccess, size 0x52c lma 0x8e460
Loading section .ARM, size 0x8 lma 0x8e98c
Loading section .init_array, size 0x4 lma 0x8e994
Loading section .fini_array, size 0x4 lma 0x8e998
Loading section .data, size 0x104 lma 0x8e99c
Start address 0x801e4, load size 60576
Transfer rate: 272 KB/sec, 5506 bytes/write.
(gdb) b main
Breakpoint 1 at 0x808f2: file /SDK/boards/src/dsp_examples/xaf_record/cm33/main_cm33.c, line 161.
(gdb) c
Continuing.
Breakpoint 1, main ()
    at /SDK/boards/src/dsp_examples/xaf_record/cm33/main_cm33.c:161
161     BOARD_InitHardware();

```

Parent topic: [Run and Debug from Command-Line Environment / LINUX](#)

**Build and Debug DSP Application** The Xtensa command-line toolchain is installed as part of the Xplorer IDE. The tools can be optionally installed on a new Windows or Linux system without the IDE using the redistributable compressed file found at: <XTENSA\_ROOT>/XtDevTools/downloads/RI-2023.11/tools/. For more information, see [Install Xtensa On Chip Debugger Daemon](#).

In order to use the command-line tools, some environment variables must be set up for use with the cmake build scripts:

```
# Add tools binaries to PATH. Assume ~/xtensa/ is install root - please adjust accordingly.
export PATH=$PATH:~/xtensa/XtDevTools/install/tools/RI-2023.11-linux/XtensaTools/bin
# (Optional) Use environment variable to control license file
# NOTE: ~/.flexlmrc will override this selection. Please delete that file before proceeding.
export LM_LICENSE_FILE=~/.RT600.lic
# Setup env vars needed for compile and linking
export XCC_DIR=~/.xtensa/XtDevTools/install/tools/RI-2023.11-linux/XtensaTools
export XTENSA_SYSTEM=~/.xtensa/XtDevTools/install/builds/RI-2023.11-linux/
npx_rt600_RI23_11_newlib/configexport XTENSA_CORE=npx_rt600_RI23_11_newlibb
```

**Note:** On Windows, you can use the ‘setx’ command instead of the ‘export’ command to set the environment variables.

1. Use the batch/shell script to build out the DSP application from the command line, in the ‘xcc’ directory:

```
user@linux:~/SDK/boards/evkmimxrt685/dsp_examples/xaf_record/dsp/xcc$ ./build_debug.sh
...
[100%] Built target dsp_xaf_record_hifi4.elf
```

**Note:** Some warnings during the linking process (floating-point ABI) may appear. However, the warnings may be ignored.

2. Launch xt-ocd debugging server (replace topology.xml with your custom version – see section 1.5 of this document):

```
user@linux:/opt/Tensilica/xocd-14.11$ ./xt-ocd.exe -c topology.xml
```

**Note:** If the Arm core fails to initialize the DSP, the xt-ocd daemon may fail to start. Therefore, the Arm core must initialize the DSP first.

3. Connect with Xtensa GDB to the device and execute the DSP application:

```
user@linux:/SDK/boards/evkmimxrt685/dsp_examples/xaf_record/dsp/xcc$ xt-gdb
debug/dsp_xaf_record_hifi4.elf
GNU gdb (GDB) 7.11.1 <Xtensa Tools VERSION NUMBER>
...
Reading symbols from debug/dsp_xaf_record_hifi4.elf...done.
(xt-gdb)
(xt-gdb) target remote localhost:20000
Remote debugging using localhost:20000
__DoubleExceptionVector ()
    at /home/xpgcust/tree/RI-2023.11/ib/tools/swtools-x86_64-linux/xtensa-elf/src/xos/src/xos_
↳ vectors.S:216
216 /home/xpgcust/tree/RI-2023.11/ib/tools/swtools-x86_64-linux/xtensa-elf/src/xos/src/xos_
↳ vectors.S: No such file or directory.
(xt-gdb) reset
__ResetVector ()
    at /home/xpgcust/tree/RI-2023.11/ib/tools/swtools-x86_64-linux/xtensa-elf/src/xtos/xea2/reset-
↳ vector-xea2.S:71
71 /home/xpgcust/tree/RI-2023.11/ib/tools/swtools-x86_64-linux/xtensa-elf/src/xtos/xea2/reset-
↳ vector-xea2.S: No such file or directory.
(xt-gdb) load
Loading section .rtos.rodata, size 0x80 lma 0x200000
Loading section .rodata, size 0x17d50 lma 0x200080
```

(continues on next page)

(continued from previous page)

```

Loading section .text, size 0x633f0 lma 0x217dd0
Loading section .rtos.percpu.data, size 0x4 lma 0x27b1c0
Loading section .data, size 0x110c lma 0x27b1d0
Loading section NonCacheable, size 0x2960 lma 0x20040000
Loading section .Level3InterruptVector.literal, size 0x4 lma 0x24000000
Loading section .DebugExceptionVector.literal, size 0x4 lma 0x24000004
Loading section .NMIExceptionVector.literal, size 0x4 lma 0x24000008
Loading section .ResetVector.text, size 0x13c lma 0x24020000
Loading section .WindowVectors.text, size 0x16c lma 0x24020400
Loading section .Level2InterruptVector.text, size 0x1c lma 0x2402057c
Loading section .Level3InterruptVector.text, size 0xc lma 0x2402059c
Loading section .DebugExceptionVector.text, size 0xc lma 0x240205bc
Loading section .NMIExceptionVector.text, size 0xc lma 0x240205dc
Loading section .KernelExceptionVector.text, size 0xc lma 0x240205fc
Loading section .UserExceptionVector.text, size 0x18 lma 0x2402061c
Loading section .DoubleExceptionVector.text, size 0x8 lma 0x2402063c
Start address 0x24020000, load size 520016
Transfer rate: 8 KB/sec, 10612 bytes/write.
(xt-gdb) b main
Breakpoint 1 at 0x21ab5b: file /SDK/boards/src/dsp_examples/xaf_record/hifi4/xaf_main_hifi4.c,
↳ line 366.
(xt-gdb) c
Continuing.
Breakpoint 1, main ()
    at /SDK/boards/src/dsp_examples/xaf_record/hifi4/xaf_main_hifi4.c:366
366  xos_start_main("main", 7, 0);
(xt-gdb) c
Continuing.

```

**Note:** You can use the gdb command ‘set substitute-path’ to map the missing symbols from the toolchain libraries. For example:

```

set substitute-path /home/xpgcust/tree/RI-2023.11/ib/tools/swtools-x86_64-linux ~/xtensa/tools/RI-
↳ 2023.11-linux/XtensaTools

```

For more information on xt-gdb, see the Cadence GNU Debugger User’s Guide and the Cadence Xtensa Debug Guide. The documents are available at:

- [~/xtensa/XtDevTools/downloads/RI-2023.11/docs/gnu\\_gdb\\_ug.pdf](#)
- [~/xtensa/XtDevTools/downloads/RI-2023.11/docs/xtensa\\_debug\\_guide.pdf](#)

**Parent topic:** [Run and Debug from Command-Line Environment / LINUX](#)

## HiFi4 System Programming

This section provides more examples, tips, and some best practices about HiFi4 programming on RT600 EVKs. It focuses more on RT6xx and SDK. For general HiFi programming, see the Xtensa IDE documents. You can find them in the directory **Xtensa Xplorer IDE menu Help > PDF Documentation**.

The following are some frequently used references:

- Xtensa Instruction Set Architecture (ISA) Reference Manual: Architecture/ high-level overview.
- HiFi 4 DSP User’s Guide: Most useful reference manual for DSP programmer. It has all details about HiFi4 instructions and intrinsics, as well as some algorithm optimization techniques.
- Xtensa XOS Reference Manual: XOS is the default and native embedded kernel for Xtensa HiFi cores. SDK examples use XOS as well.

- Xtensa System Software Reference Manual: XTOS, also known as the basic runtime and handlers, has been depreciated and moved toward XOS. However, the Xtensa Processor Hardware Abstraction Layer/ HAL is still very useful in many perspectives. SDK examples use HAL functions.
- Xtensa Linker Support Packages (LSPs) Reference Manual and GNU Linker User's Guide: Both documents are useful for understanding the HiFi program linker and the memory map.

Specifically for RT6xx, the user manual and the data sheet are the most important references. All product specifications are found in the user guide and data sheet documents.

**HiFi4 Boot Loader and Memory Map** This section provides an overview of:

- HiFi4 Boot Loader
- Linker and Memory Map
- Cache and Data Exchange Memory Partitions
- Boot or Run from Flash

**HiFi4 Boot Loader** For better power efficiency, by default the HiFi4 DSP is powered off when RT6xx powers up. To boot up, Cortex M33 acts as master core to configure DSP local memories, clocks, load DSP images, and so on. SDK has wrapped up this part as HiFi4 Boot Loader or the DSP driver. The essentials are located in: <SDK path>\devices\MIMXRT685S\drivers\fsl\_dsp.h & fsl\_dsp.c.

For each DSP example, it also provides two more implementation files for detailed configurations: <SDK path>\boards\evkmimxrt685\dsp\_examples\<any dsp example>\cm33\dsp\_support.h & dsp\_support.c

Below is the DSP boot procedure with more elaborations.

To run DSP at full power, for example, 600 MHz, SoC Vddcore is set to 1.1 V. If full power is not required, for example, to run DSP at the half speed at 300 MHz, then Vddcore only requires 0.8 V. RT6xx EVK integrates NXP PCA9420 PMIC for power management and by default Vddcore is set to 1.0 V.

Therefore, PMIC is initialized for better power management. For details on Vddcore and DSP frequency operating conditions, see any dsp\_example\cm33\pmic\_support.c BOARD\_SetPmicVoltageForFreq(), or see the data sheet, section 13.1, General Operating Conditions.

```
/* Initialize PMIC PCA9420 */
BOARD_InitPmic();
/* Configure PMIC Vddcore value according to main/dsp clock. */
BOARD_SetPmicVoltageForFreq(CLOCK_GetMainClkFreq(), CLK_600MHZ);
```

It is possible to clock DSP from various clock sources. The DSP PLL for full power and can also run at FFRO low-frequency clocks to save the power.

```
/* Enable DSP PLL clock 594MHz. */
CLOCK_InitSysPfd(kCLOCK_Pfd1, 16);
/* Let DSP run on DSP PLL clock with divider 1 (594Mhz). */
CLOCK_AttachClk(kDSP_PLL_to_DSP_MAIN_CLK);
CLOCK_SetClkDiv(kCLOCK_DivDspCpuClk, 1);
```

As Cortex M33 and SRAM are clocked at lower speed/ max frequency at 300 MHz, set the DSP AHB bus clock divider as 2.

```
CLOCK_SetClkDiv(kCLOCK_DivDspRamClk, 2);
```

If DSP clock is running at 300 MHz or lower, it is more efficient to use divider as 1. For divider as 1, note that an extra register SYSCTL0\_PACKERENABLE is set. For more details, see the user manual section 4.5.2.18 DSP Main Ram Clock Divider and section 4.5.5.3 Packer Enable.

```
/* This is a quick register setting example for secure mode */
/* SYSCTL0->PACKERENABLE = 0x4 */
```

Power up TCM/ DSP local memories and cache, supply clock, and reset peripherals.

```
/* Initializing DSP core */
DSP_Init();
```

For SDK DSP examples, split the DSP images into three parts.

One is for vectors and critical sections sitting on TCM/ DSP local memories. The second one is for normal code and data sections sitting on SRAM, and the final is for non-cached DSP initialized data in SRAM.

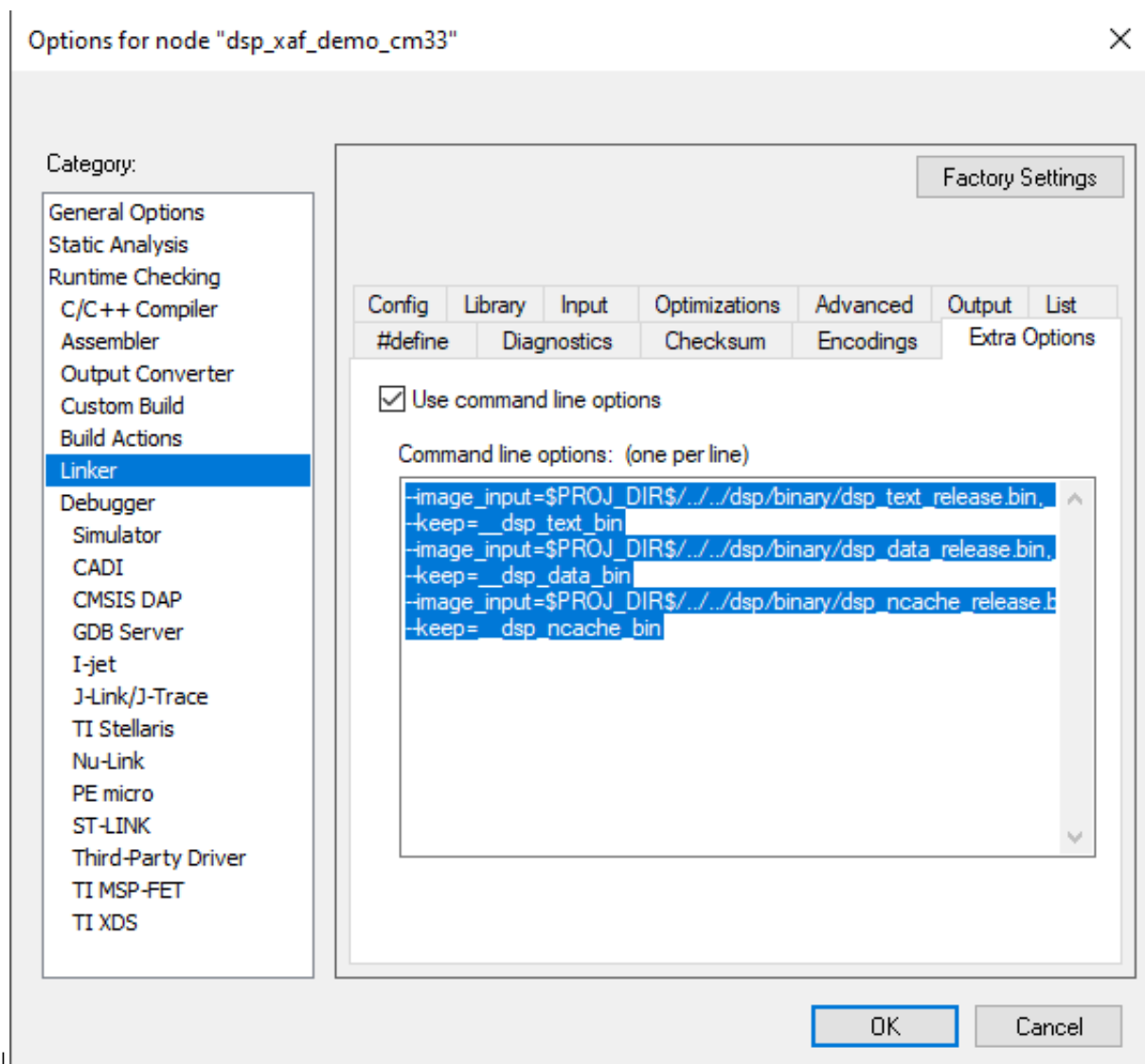
Here, Cortex M33 load those binaries to its destination. When the DSP program is debugged, it is possible to load DSP binaries from the Xtensa Xplorer IDE, as described in Prepare DSP Core for 'Hello World'. To load binaries, remove the DSP\_IMAGE\_COPY\_TO\_RAM compilation flag or set it to 0. By default, the compilation flag is set to 1 and always load the DSP images.

```
#if DSP_IMAGE_COPY_TO_RAM
/* Copy application from RAM to DSP_TCM */
DSP_CopyImage(&tcm_image);
/* Copy application from RAM to DSP_RAM */
DSP_CopyImage(&sram_image);
/* Copy application from RAM to DSP_Uncached RAM */
DSP_CopyImage(&ncache_image);
#endif
```

The DSP stall register SYSCTL0\_DSPSTALL controls the HiFi4 operation. Start the DSP and run it.

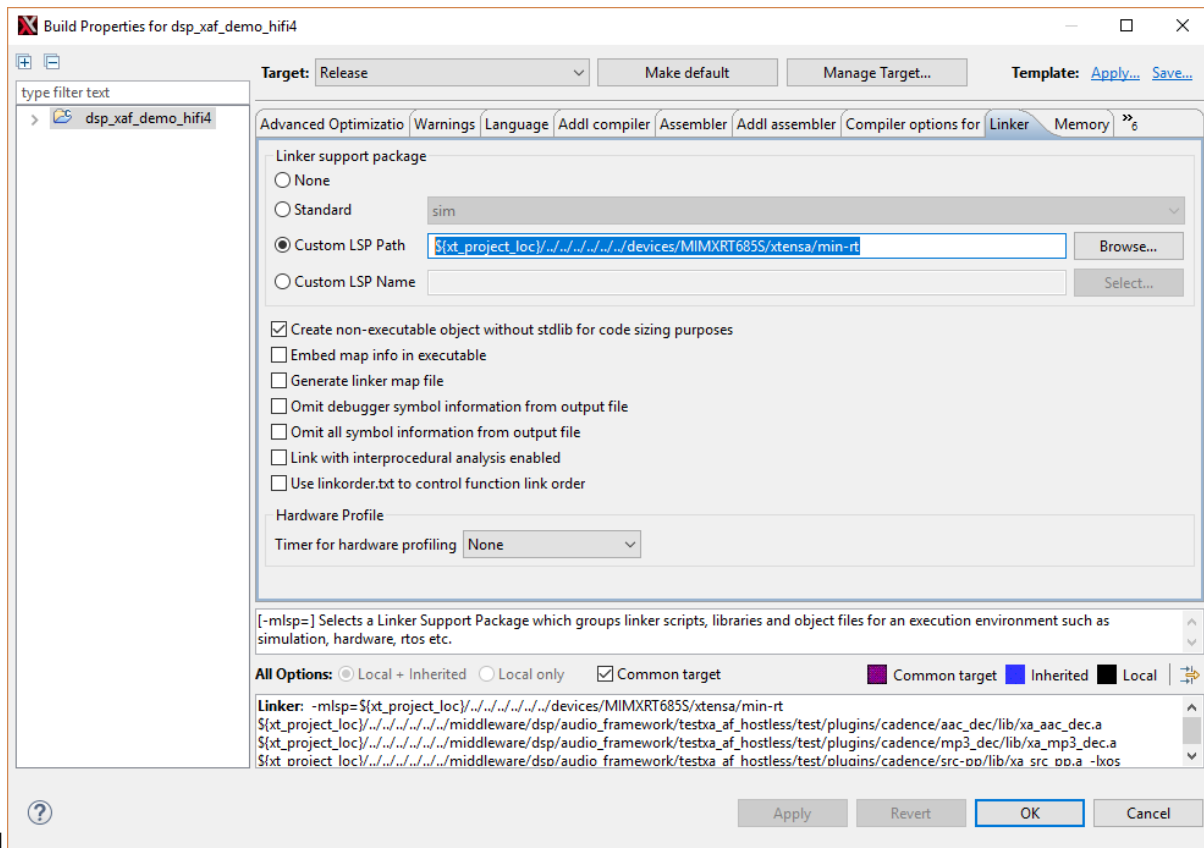
```
/* Run DSP core */
DSP_Start();
```

The post build scripts create the DSP images. For details, see Makefile.include in any DSP example. To reduce the image size and make image copy more efficient, they are split into SRAM part; TCM part, and uncached SRAM part. DSP images are set and linked into Cortex M33 side. For IAR, the linker is set in **Project Options > Linker > Extra Options**. For ArmGCC or Linux environment, set in any DSP example\cm33\incbin\_gcc.S.



**Parent topic:** HiFi4 Boot Loader and Memory Map

**Linker and Memory Map** When importing the SDK DSP examples, by default the mode is set as Release. The default mode means that the images must be built in Release mode with 'min-rt' Linker Support Package/ LSP. To double-check, open **SDK DSP examples > Build Properties > Linker**.



SDK provides three different LSPs.

- **‘min-rt’ for Release mode** - ‘min-rt’ eliminates all unnecessary debug info and reduces image size.
- **‘gdbio’ for Debug mode** - gdbio’ support standard ‘printf’/log output back to Xtensa Xplorer debug console, as well as other debug utilities, perfect for debug purpose but not appropriate for official deployment nor loading directly from Cortex M33 side.
- **‘sim’ for simulations** - ‘sim’ only works for software simulation and does not fit on device debugging.

The memory map is identical for different LSPs. It sits with linker scripts in SDK path\devices\MIMXRT685S\xtensa\‘LSP name’\ldscripts\elf32xtensa.x. It specifies how HiFi4 DSP organizes image sections on the memory. For example:

- 0x0020 0000 ~ 0x0048 0000, size 2.5 M bytes, for code and data.
- Stack and Heap are at the top of the segment and count top down from 0x0048 0000 to lower.
- 0x2400 0000 ~ 0x2400 FFFF, size 64 K bytes, for Data TCM. By default, it is empty.
- 0x2402 0000 ~ 0x2402 FFFF, size 64 K bytes, for instruction TCM. By default it only contains essential vectors and left around 62 K for applications.
- 0x2004 0000 ~ 0x2007 FFFF, size 256 K bytes. This is non-cached area for Cortex M33 and HiFi4 DSP data exchange.

Note that both Cortex M33 and HiFi4 DSP have access to all SRAM partitions. It means that a unified memory map is necessary at system level and both cores must not affect each other’s memory map. For SDK examples, see that HiFi4 memory map starts from 0x0020 0000 and Cortex M33 side sits under this address. Using IAR environment, as an example, its memory map sits in SDK path\boards\evkmimxrt685\dsp\_examples\ any example\cm33\iar\MIMXRT685Sxxxx\_ram.icf.



- 0x0008 0000 ~ 0x0017 FFFF, for interrupt vectors and code.
- 0x0018 0000 ~ 0x001F FFFF, for data.

The memory map is flexible and can be adjusted as per the requirement of the application. Note that modifying the core's memory map might affect another. Changes to both the cores must be made accordingly. For example, when allocating more SRAM partitions to DSP, you must reduce the memory taken at Cortex M33 side. Otherwise, Cortex M33 might not work properly. Also, when loading DSP image directly from Cortex M33 side, the image still sits in Cortex M33 data section before booting up. Therefore, it raises the bar for application data section requirements. You may consider running the application from FLASH.

**Parent topic:**HiFi4 Boot Loader and Memory Map

**Cache and Data Exchange Memory Partitions** You may have noticed that HiFi4 DSP has a small non-cached area that starts from 0x20040000. The non-cached are used for data exchange between two cores. As both M33 and HiFi DSP have shared access to all SRAM partitions, shared memory access is the most effective way to exchange data between two cores. The given physical addresses are read/ written by both cores at same address, no memory mapping or address converting is required. For example, if a piece of data array is passed from Cortex M33 to HiFi4 DSP, only the start pointer and the size of the array is passed, and conversely. It is convenient for system programming and simplifies the inter-core communications.

Consider cache here. Cortex M33 has no cache, the entire SRAM is considered as its local memory. Therefore, any memory write is flushed immediately. HiFi4 has 32 K instruction cache and 64 K data cache, and both cache are enabled by default. Therefore, the memory write is not flushed immediately. To make a tradeoff between performance and IPC convenience, set the non-cached area for data exchange memory partitions.

The above memory map has specified the non-cached region, and in DSP code, and HAL functions are called to disable the cache. For details, see the audio framework example in SDK path\boards\evkmimxrt685\dsp\_examples\xaf\_demo\dsp\xaf\_main\_dsp.c. For more details about HAL cache function, see the Xtensa System Software Reference Manual, section 3.11 Cache

```
/* Disable DSP cache for RPSMsg-Lite shared memory. */
xthal_set_region_attribute((void *)RPSMSG_LITE_SHMEM_BASE, RPSMSG_LITE_SHMEM_SIZE,
↪XCHAL_CA_BYPASS, 0);
/* Disable DSP cache for noncacheable sections. */
xthal_set_region_attribute((uint32_t *)&NonCacheable_start,
                          (uint32_t *)&NonCacheable_end - (uint32_t *)&NonCacheable_start, XCHAL_CA_
↪BYPASS, 0);
```

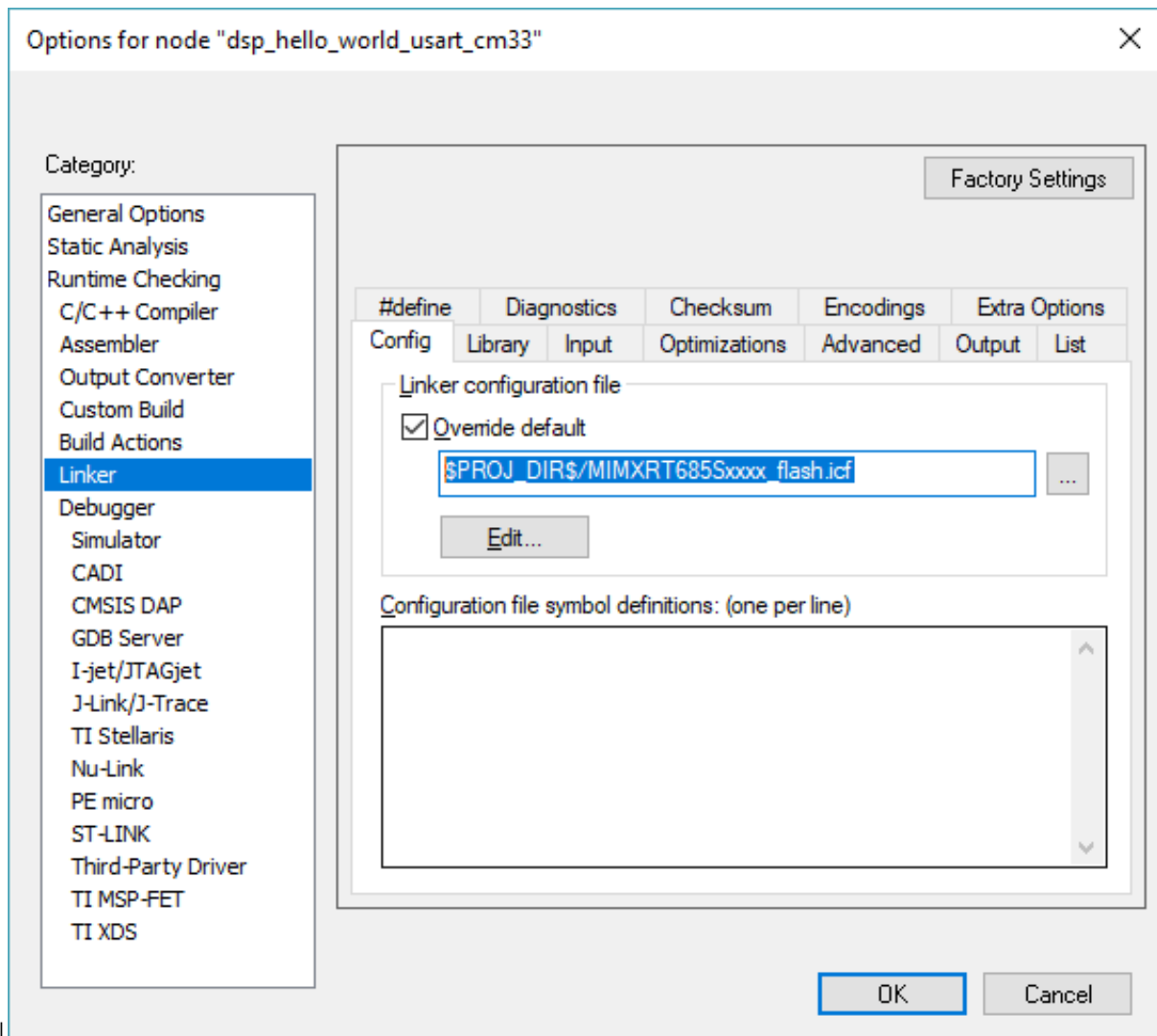
Note that the XHAL call sets cache attribute of the whole region/ 512 M bytes even if the set size is passed. This is also one reason why non-cached attribute is set on the overlapping SRAM address and starts from 0x2000 0000. This also distinguished the physical SRAM addresses starting from 0x0000 0000, which is cacheable area for HiFi4.

Data exchange memory partitions are flexible and can be configured as per application's requirement. However, to mitigate the possible AHB arbitration between the two cores, use of the first eight 32 K and following four 64 K memory partitions is recommended. DSP Data TCM is also used as data exchange area for those data have high demand on timing performance. You must avoid accessing same partition at same time for frequent data exchange between two cores. You can keep one core in Sleep or Wait for Interrupt while another core operating, or set up a ping pong data exchange/ DMA such that when one core fills one partition, another core fetches another partition, and conversely.

For more details about the RT6xx memory map, see the user manual, section 2 Memory Map, and section 2.1.11 HiFi4 memory map.

**Parent topic:**HiFi4 Boot Loader and Memory Map

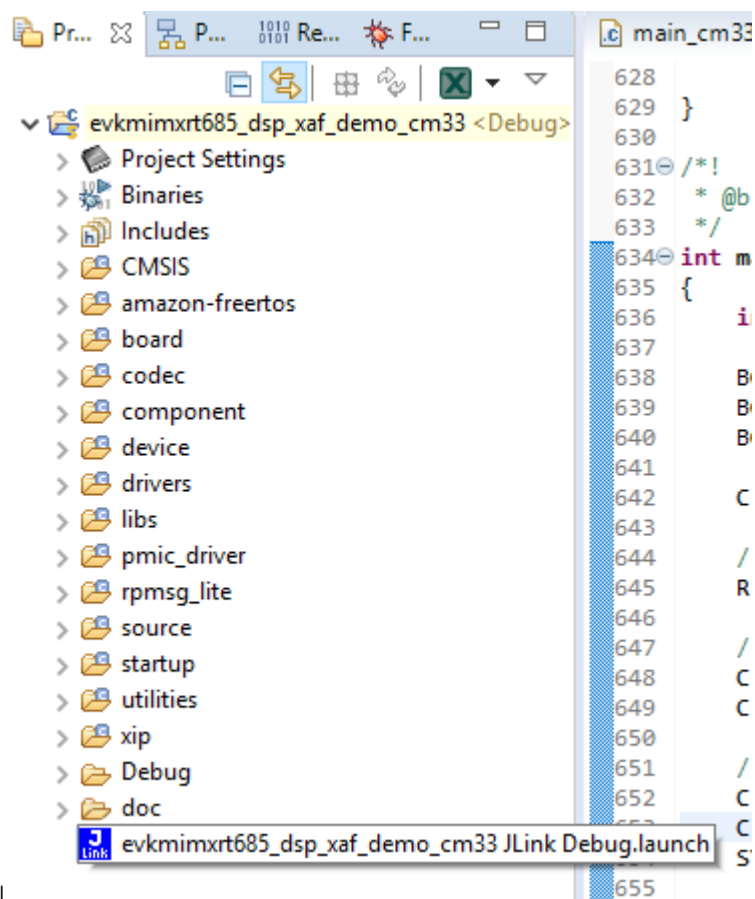
**Boot or Run from Flash** Boot from Flash is straight forward when using IAR environment. SDK provides either two or four different build configurations based on the project: debug (from SRAM)/ release (from SRAM)/ flash\_debug/ flash\_release. Flash configurations use different memory map in project linker options. For details, see Figure 1.



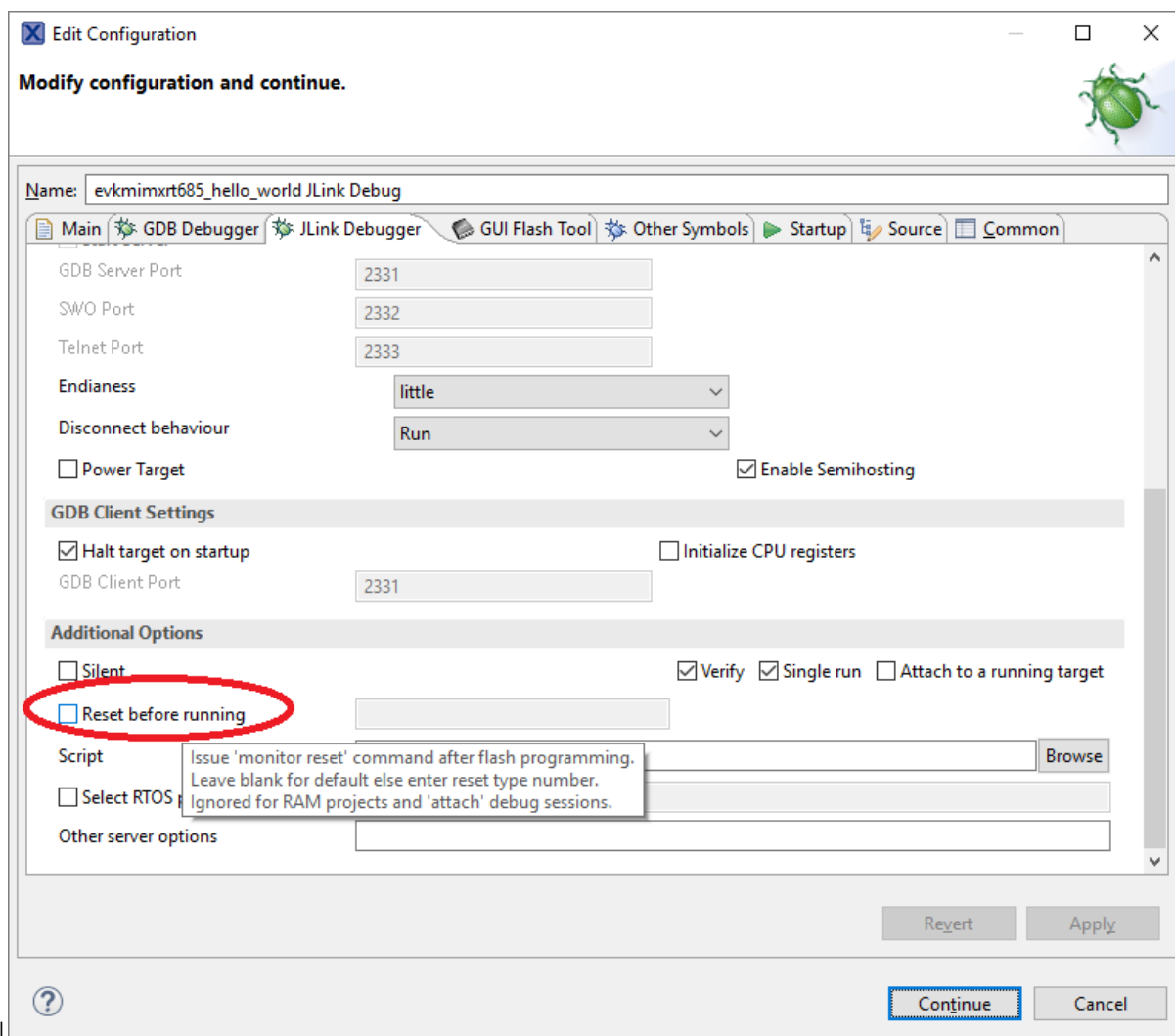
To enable booting from flash, change the ISP mode/ SW5 switches on the EVK.

If using MCUXpresso/ armgcc build environment, note that by default build environment is set to boot from flash. Using MCUXpresso as an example:

1. Make sure to use the latest version of MCUXpresso.
2. Import the SDK examples.
3. Once completed, double click the last file/ J-Link Debug.



4. Launch to modify J-Link debugger setting.
5. Make sure to deselect **Reset before running**. This helps the flash-based program get into main function.



- Make sure to:
  - Modify `DSP_IMAGE_COPY_TO_RAM` and Define to 1 in **Project settings > C/C++ General > Paths and Symbols > Symbols**.
  - Have the right compilation flag as C/C++. Compilation flags do not work on \*.S files.
  - Have `#define DSP_IMAGE_COPY_TO_RAM 1` as the first line of source/incbin.S to include the DSP binaries.
  - Use the correct DSP binaries at correct path (must be release binaries).
  - Provide the right image path to incbin.S.
- To run/boot from flash, make sure that the board is set to FlexSPI flash boot mode (ISP2: ISP1: ISP0 = ON, OFF, ON).

**Parent topic:** HiFi4 Boot Loader and Memory Map

**Parent topic:** [HiFi4 System Programming](#)

**Peripheral Drivers and Interrupts** This section provides information on the basics of peripheral drivers, interrupts, DMA, and how peripheral drivers work on HiFi4 DSP.

**Basics** As both cores use shared SRAM and all digital peripherals, SDK drivers work the same on HiFi4 DSP. User Manual section 1.3 Block Diagram shows the system architect on this perspective. For applications, it is the same to use SDK drivers on HiFi4 DSP with Cortex M33 core. SDK has a DSP hello world UART example showing how easy to use UART on DSP side, same as Cortex M33 programming:

```
#include <xtensa/config/core.h>
#include "fsl_debug_console.h"
/* Init board hardware. */
BOARD_InitDebugConsole();
PRINTF("Hello World running on DSP core '%s', %s_%s\r\n", XCHAL_CORE_ID);
```

Compare with Hello world programming on Cortex M33 side. There are two major differences:

- Pin initialization is not necessary for DSP. It is doable but it is highly recommended managing all pins from Cortex M33 side to make pins all-in-one place. It is also to avoid possible conflicts when you set pins on two different cores;
- Clock setting is not necessary for DSP. It is also doable but same it is highly recommended managing all clock sources all-in-one place;

**Parent topic:**Peripheral Drivers and Interrupts

**DMA** RT6xx has two DMA controllers. Each has the same DMA request and trigger input possibilities. The two intended scenarios for their use are:

- One DMA controller (DMA0) is used by the CM33, the other (DMA1) is used by the HiFi4. This case can apply to systems where there is no need to differentiate security between the CPUs or between different tasks.
- One DMA controller (DMA0) is secured and has access to secure spaces and peripherals. The other (DMA1) is not secured and does not have access to secure spaces and peripherals. In this scenario, only the secure code running on the CM33 has access to the secure DMA controller. The other code and the HiFi4 share the non-secure DMA controller (if needed).

HiFi4 DSP always uses DMA1 controller. Again, it is the same to call DMA drivers at HiFi4 DSP side. The DMA destination buffer and DMA descriptor have to be in non-cached area to ensure that each transaction flushes to memory immediately. The below example code showing how to create a DMIC DMA channel:

```
AT_NONCACHEABLE_SECTION_ALIGN(
    static uint8_t s_buffer[BUFFER_SIZE * BUFFER_NUM], 4
);
AT_NONCACHEABLE_SECTION_ALIGN(
    dma_descriptor_t s_dmaDescriptorPingpong[2], 16
);
#define DEMO_DMA (DMA1)
#define DEMO_DMIC_RX_CHANNEL DMAREQ_DMIC0
DMA_Init(DEMO_DMA);
DMA_EnableChannel(DEMO_DMA, DEMO_DMIC_RX_CHANNEL);
DMA_SetChannelPriority(DEMO_DMA, DEMO_DMIC_RX_CHANNEL, kDMA_ChannelPriority2);
DMA_CreateHandle(&s_dmicRxDmaHandle, DEMO_DMA, DEMO_DMIC_RX_CHANNEL);
```

Macro AT\_NONCACHEABLE\_SECTION\_ALIGN is defined in SDK driver layer and refer to non-cached area specified by memory map.

For more details about DMA, see the User Manual Chapter 11 RT6xx DMA Controller and SDK DMA examples in <SDK path>\boards\evkmimxrt685\driver\_examples\dma.

For audio peripherals/ DMIC/ I2S DMA, see <SDK path>\boards\evkmimxrt685\driver\_examples\dmic and <SDK path>\boards\evkmimxrt685\driver\_examples\i2s.

**Parent topic:**Peripheral Drivers and Interrupts

**Interrupts** HiFi4 DSP has total 32 interrupts and 4 interrupt levels. Besides the first five interrupts, the rest interrupt 5 ~ 31 are multiplexed to allow more control over priorities and more general flexibility. To see the full list, see the User Manual section 8.6.3 DSP Interrupt Input Multiplexers.

It only requires few lines of code to enable an interrupt by calling XOS function calls, and peripheral controls are identical as Cortex M33 side. For example, the below code shows how to enable a UTick timer at DSP side:

First, include XOS system header files and libraries to enable necessary XOS functions.

To include libraries, go to Build **Properties** > **Libraries** > **Add libraries** > **Select 'xos' & 'xtutil'** and **OK** to accept.

```
#include <xtensa/config/core.h>
#include <xtensa/xos.h>
```

Set up the UTick callback and delay functions. This part is identical with Cortex M33 side. For Cortex M33 side implementation, see SDK path\\boards\\evkmimxrt685\\driver\_examples\\utick

```
#define UTICK_TIME_1S      (1000000UL)
#define EXAMPLE_UTICK      UTICK0
static volatile bool utickExpired;
static void UTickCallback(void)
{
    utickExpired = true;
}
static void UTickDelay(uint32_t usec)
{
    /* Set the UTICK timer to wake up the device from reduced power mode */
    UTICK_SetTick(EXAMPLE_UTICK, kUTICK_Onetime, usec - 1, UTickCallback);
    while (!utickExpired)
    {
    }
    utickExpired = false;
}
```

Initialize the XOS and start UTick timer. The XOS function calls are the differences with Cortex M33 side:

```
/* Initialize XOS thread and start scheduler. Priority 7*/
xos_start_main("main", 7, 0);
/* Init board hardware. */
CLOCK_AttachClk(kLPOSC_to_UTICK_CLK);
CLOCK_EnableClock(kCLOCK_InputMux);
UTICK_Init(EXAMPLE_UTICK);
INPUTMUX_AttachSignal(INPUTMUX, 10U, kINPUTMUX_Utick0ToDspInterrupt);
/* To register interrupt callback */
xos_register_interrupt_handler(15, (XosIntFunc *)UTICK0_DriverIRQHandler, NULL);
/* To enable the interrupt */
xos_interrupt_enable(15);
while (1)
{
    UTickDelay(UTICK_TIME_1S);
    PRINTF("DSP UTICK TIMER every 1s\r\n");
}
```

Pay attention to the instant numbers. Pick interrupt 15, which maps to DSP\_INT0\_SEL10 as a L1 interrupts, lowest priority level. For more details about RT6xx HiFi4 DSP interrupt configuration, see User Manual section 51.7 Interrupt. Therefore, for Inputmux, attach UTick interrupt to 10. 15-5=10 as first five interrupts are reserved, not user configurable. To get it to highest priority level, for example L3, configure interrupt as follows.

```
INPUTMUX_AttachSignal(INPUTMUX, 24U,
kINPUTMUX_Utick0ToDspInterrupt);
xos_register_interrupt_handler(29, (XosIntFunc
*)UTICK0_DriverIRQHandler, NULL);
xos_interrupt_enable(29);
```

kINPUTMUX\_Utick0ToDspInterrupt specifies DSP interrupt multiplexing value. It has been defined in SDK and matching User Manual section 8.6.3 DSP Interrupt Input Multiplexers.

For more details about XOS interrupt handling, see the Xtensa XOS Reference Manual Chapter 18 Interrupt and Exception Handling, and Chapter 26 Interrupt Handler Restrictions.

**Parent topic:**Peripheral Drivers and Interrupts

**Complete Example** To present better how peripheral drivers work on HiFi4 DSP, below list a bare-metal DSP example program that transfers data from DMIC to codec on RT6xx EVKs. The full workspace located in <SDK path>\boards\evkmimxrt685\dsp\_examples\audio\_demo\_bm. This example is derived from Cortex M33 driver <SDK path>\boards\evkmimxrt685\driver\_examples\dmic\dmic\_i2s\_dma with below slightly modifications to adapt to HiFi4 DSP.

- Move DMA buffer and descriptors into non-cached memory partitions;
- Using DMA1 for DSP;
- Calling XOS functions to enable interrupts.
- This example does not contain any pin mux initializing or clock configurations. See the above Cortex M33 example dmic\_i2s\_dma to set up Cortex M33 side. Once Cortex M33 side ready, this example is compiled and run same as any other SDK DSP examples.

Connect headphone/earphone on audio out of the board, speak on DMIC, or play song nearby the DMIC, you can hear sound on the left channel of headphone/earphone.

```
/* Start XOS */
xos_start_main("main", 7, 0);
/* Disable DSP cache for noncacheable sections. DMA MUST run on none-cacheable/ cache bypass area*/
xthal_set_region_attribute((uint32_t *)&NonCacheable_start,
                           (uint32_t *)&NonCacheable_end - (uint32_t *)&NonCacheable_start, XCHAL_CA_
↳BYPASS, 0);
xthal_set_region_attribute((uint32_t *)&NonCacheable_init_start,
                           (uint32_t *)&NonCacheable_init_end - (uint32_t *)&NonCacheable_init_start, XCHAL_
↳CA_BYPASS,
                           0);
PRINTF("Configure DMA\r\n");
/* DSP_INT0_SEL18 = DMA1 */
INPUTMUX_AttachSignal(INPUTMUX, 18U, kINPUTMUX_Dmac1ToDspInterrupt);
/* Map DMA IRQ handler to INPUTMUX selection DSP_INT0_SEL18
 * EXTINT19 = DSP INT 23 */
xos_register_interrupt_handler(XCHAL_EXTINT19_NUM, (XosIntFunc *)DMA_IRQHandle, DMA1);
xos_interrupt_enable(XCHAL_EXTINT19_NUM);
/* The rest DMA & DMIC operations are identical with CM33 side */
```

**Parent topic:**Peripheral Drivers and Interrupts

**Parent topic:**[HiFi4 System Programming](#)

**Messaging Unit, Semaphore, and IPC** Message Unit/ MU and Semaphore/ SEMA42 are essential for DSP programming. The inter-core communications a.k.a IPC on RT600 is based on MU and SEMA42. SDK provides three simple examples about standalone MU and SEMA 42. SDK path \boards\evkmimxrt685\dsp\_examples\mu\_interrupt, mu\_polling, and sema42. These bare-metal examples show how IPC work between two cores. Furthermore, the audio framework



demo/ SDK path \boards\evkmimxrt685\dsp\_examples\xaf\_demo is complete. It uses rpmsg\_lite as IPC protocol, which based on MU to transfer messages between the two cores.

First Cortex M33 side initializes rpmsg master, main\_cm33.c in app\_task()

```
g_my_rpmsg = rpmsg_lite_master_init((void *)RPMSG_LITE_SHMEM_BASE, RPMSG_LITE_
↳SHMEM_SIZE, RPMSG_LITE_LINK_ID, RL_NO_FLAGS);
g_my_queue = rpmsg_queue_create(g_my_rpmsg);
g_my_ept = rpmsg_lite_create_ept(g_my_rpmsg, LOCAL_EPT_ADDR, rpmsg_queue_rx_cb, g_my_
↳queue);
```

Rpmsg initialize MU interrupts for MUA/ master in rpmsg\_platform.c platform\_init\_interrupt()

```
/* Register ISR to environment layer */
env_register_isr(vector_id, isr_data);
env_lock_mutex(platform_lock);
RL_ASSERT(0 <= isr_counter);
if (isr_counter == 0)
{
    MU_EnableInterrupts(APP_MU, (1UL << 27U) >> RPMSG_MU_CHANNEL);
}
```

For DSP side, it also initializes rpmsg client and tries to hook with the master, in xaf\_main\_dsp.c:DSP\_Main()

```
/* Initialize standard SDK demo application pins */
my_rpmsg = rpmsg_lite_remote_init((void *)RPMSG_LITE_SHMEM_BASE, RPMSG_LITE_LINK_ID,
↳RL_NO_FLAGS, &rpmsg_ctxt);
while (!rpmsg_lite_is_link_up(my_rpmsg))
{
}
```

It has rpmsg porting as well, initialize MU interrupts for MUB/ client in rpmsg\_platform.c platform\_init\_interrupt() and enable it in platform\_interrupt\_enable()

```
xos_register_interrupt_handler(6, MU_B_IRQHandler, ((void *)0));
xos_interrupt_enable(6);
```

Once they hooked up both sides are ready for message exchange. SDK defines various SRTM message structures to pass the commands/ messages. In SDK example, it shows how to pass an MP3 decoding message, a AAC decoding message, a DMIC recording message and so on. Using MP3 decoding message as an example, It fills SRTM structure with input buffer address pointer, input buffer size, output buffer address pointer, output buffer size and so on. And then calling rpmsg to send it to DSP and waiting for the response.

```
rpmsg_lite_send(g_my_rpmsg, g_my_ept, g_remote_addr, (char *)msg, sizeof(THE_MESSAGE), RL_
↳BLOCK);
rpmsg_queue_recv(g_my_rpmsg, g_my_queue, (unsigned long *)&g_remote_addr, (char *)msg,
↳sizeof(THE_MESSAGE), len,
    RL_BLOCK);
```

DSP side listens to rpmsg event.

```
my_ept = rpmsg_lite_create_ept(my_rpmsg, DSP_EPT_ADDR, my_ept_read_cb, (**void**) &rpmsg_
↳user_data, &my_ept_context);
```

Once received, it handles message and proceeds the command. For MP3 decoding case, it decodes the MP3 data in the input buffer, and flushes the output buffer. It also fills out the SRTM message structure with actual read data size and actual write data size, and eventually sends the response message back to Cortex M33 side.

```
/*Send response message*/
/rpmsg_lite_send(my_rpmsg, my_ept, remote_addr, (**char**) &msg, **sizeof**(THE_MESSAGE),
↳RL_DONT_BLOCK);
```

For this example, DSP is considered as a coprocessor and rpmsg has been used for a light-weight IPC for both cores. All modules are open-sourced and could be easily adapted to whatever application needs.

**Parent topic:** [HiFi4 System Programming](#)

**NatureDSP Library** To facilitate application development on RT6xx HiFi4 DSP, NXP licensed NatureDSP signal processing library and embedded as is in source code format. It is found at <SDK path>\middleware\dsp\naturedsp\_hifi4.

This is an extensive library, containing the most commonly used signal processing functions: FFT, FIR, vector, matrix, and common mathematics. API and programing guide in .\doc\NatureDSP\_Signal\_Library\_Reference\_HiFi4.pdf, and performance data in .\doc\NatureDSP\_Signal\_Library\_Performance\_HiFi4.pdf

As this is a huge library, it is impossible to build an all-in-one example on the RT6xx hardware. Fortunately, this library is in source code format and each function/ filter are wrapped in standalone source file. They could be integrated to any application as needed.

**Parent topic:** [HiFi4 System Programming](#)

**System Optimization** Performance and power efficiency are key for embedded systems. The following sections list some tips and best practices for system optimization from this perspective.

**Profiling** The Xtensa Xplorer IDE tool can run software simulation and profile the application directly. Both simulation and profiling are cycle accurate. This is convenient for algorithm or heavy application developers.

Figure 1 shows profiling result of the helloworld program on simulation console.

| Events                        | Number          | Number<br>per 100<br>instrs  |         |
|-------------------------------|-----------------|------------------------------|---------|
| Committed instructions        | 3492 ( 100.00 ) |                              |         |
| Instruction fetches           | 4066 ( 116.44 ) |                              |         |
| From IRAM                     | 2545 ( 72.88 )  |                              |         |
| ICache fetches                | 1521 ( 43.56 )  |                              |         |
| ICache misses                 | 43 ( 1.23 )     | 2.83% of ICache fetches      |         |
| Loop buffer hits              | 338 ( 9.68 )    | 8.31% of Instruction fetches |         |
| Taken branches                | 702 ( 20.10 )   |                              |         |
| Exceptions                    | 35 ( 1.00 )     |                              |         |
| WindowUnderflow               | 17 ( 0.49 )     |                              |         |
| WindowOverflow                | 18 ( 0.52 )     |                              |         |
| Loads                         | 481 ( 13.77 )   |                              |         |
| From IRAM                     | 16 ( 0.46 )     |                              |         |
| DCache loads                  | 465 ( 13.32 )   |                              |         |
| DCache load misses            | 9 ( 0.26 )      | 1.94% of DCache loads        |         |
| Stores                        | 308 ( 8.82 )    |                              |         |
| DCache stores                 | 308 ( 8.82 )    |                              |         |
| DCache store misses           | 10 ( 0.29 )     | 3.25% of DCache stores       |         |
| DCache castouts               | 15 ( 0.43 )     |                              |         |
| PIF transfers (16 bytes each) | 1552 ( 44.44 )  |                              |         |
| IFetch reads                  | 496 ( 14.20 )   |                              |         |
| Data reads                    | 304 ( 8.71 )    |                              |         |
| Data writes                   | 240 ( 6.87 )    |                              |         |
| ICache prefetches             | 416 ( 11.91 )   |                              |         |
| DCache prefetches             | 96 ( 2.75 )     |                              |         |
| ICache prefetch hits          | 12 ( 0.34 )     | 27.91% of ICache allocates   |         |
| Cycles: total = 9078          |                 |                              |         |
|                               |                 | Summed                       | Summed  |
|                               | CPI             | CPI   % Cycle                | % Cycle |
| Committed instructions        | 3492 ( 1.0000   | 1.0000   38.47               | 38.47 ) |
| Taken branches                | 2121 ( 0.6074   | 1.6074   23.36               | 61.83 ) |
| Pipeline interlocks           | 357 ( 0.1022    | 1.7096   3.93                | 65.76 ) |
| ICache misses                 | 519 ( 0.1486    | 1.8582   5.72                | 71.48 ) |

Figure 1 shows the profiling chart, partially.

| Profile (Cycles)   Call-Graph   Graph   Saved Output   Pipeline   Dynamic ISA Profile   Console |              |              |              |             |               |              |
|-------------------------------------------------------------------------------------------------|--------------|--------------|--------------|-------------|---------------|--------------|
| Function Name                                                                                   | Function (%) | Function (F) | Children (C) | Total (F+C) | Called        | Size (bytes) |
| <TOTAL>                                                                                         | 100.00       | 9,078        | N/A          | N/A         | N/A           | N/A          |
| ResetHandler                                                                                    | 45.96        | 4,173        | 0            | 4,173       | <spontaneous> | 246          |
| xthal_dcache_all_writeback                                                                      | 9.77         | 887          | 0            | 887         | 1             | 25           |
| _vprintf_r                                                                                      | 4.40         | 400          | 1,776        | 2,176       | 1             | 8,306        |
| _malloc_r                                                                                       | 3.39         | 308          | 101          | 409         | 1             | 1,318        |
| _WindowUnderflow8                                                                               | 2.61         | 237          | 0            | 237         | 16            | 29           |
| _WindowOverflow8                                                                                | 2.48         | 226          | 0            | 226         | 16            | 29           |
| _fclose_r                                                                                       | 2.40         | 218          | 501          | 719         | 3             | 162          |
| _sinit                                                                                          | 2.03         | 185          | 170          | 355         | 1             | 267          |
| _start                                                                                          | 1.78         | 162          | 4,688        | 4,850       | 1             | 150          |
| _sfvwrite_r                                                                                     | 1.56         | 142          | 385          | 527         | 1             | 951          |
| memset                                                                                          | 1.56         | 142          | 0            | 142         | 3             | 143          |
| _clibrary_init                                                                                  | 1.50         | 137          | 165          | 302         | 1             | 35           |
| _fflush_r                                                                                       | 1.48         | 135          | 310          | 445         | 4             | 41           |
| _sflush_r                                                                                       | 1.46         | 133          | 127          | 260         | 4             | 377          |

The generated license file only supports debug/run on the RT6xx device target. It does not support software simulation/Xplorer ISS. If there are special requirements to run the software simulations, contact Cadence directly.

It is also common to measure exact cycle counts for specific processing/ timing measurements. The following is an example code to show how to do the cycle counts.

```
/* Cycle counts inline function */
static unsigned long inline get_ccount(void)
{
    unsigned long r;
    __asm__ volatile ("rsr.ccount %0" : "=r" (r));
    return r;
}
tic = get_ccount();
processing_function();
toc = get_ccount();
printf("Processing takes %d cycles \r\n", toc - tic);
```

**Parent topic:**System Optimization

**Using Local Memories** RT6xx HiFi4 DSP has 64 K data TCM and 64 K instruction TCM. The TCM is filled with less than 2 K of kernel vectors and the rest is available for application needs. They are the fastest RAM available with no access latency, and can improve critical data/ instruction performance considerably. Consider using TCM as much as possible.

To program code/ data to TCM area:

1. Define macros for sections in TCM. Reuse existing .dram0 and .iram0 sections. Both sections are default TCM sections for every Xplorer project/ memory map. The following are the sections in fsl\_common.h.

```
#define DRAM0_DATA __attribute__((section(".dram0.data")))
#define DRAM0_BSS __attribute__((section(".dram0.bss")))
#define IRAM0_TEXT __attribute__((section(".iram0.text")))
#define ALIGNED(alignbytes) __attribute__((aligned(alignbytes)))
```

2. Use macros to declare code and data to be placed in TCM. The data and bss/ uninitialized data are different sections. The following is an example for an FFT function call.

```
DRAM0_DATA const static int32_t fft_in_ref[FFT_LENGTH] = {
DRAM0_DATA const static int32_t fft_out_ref[FFT_LENGTH] = {
DRAM0_BSS static int32_t fft_out[FFT_LENGTH];
IRAM0_TEXT int TEST_FFT()
{
    ...
    fft_cplx32x32(fft_out, fft_in, FFT_HANDLE, 3);
    ...
}
```

3. The TCM addresses start from 0x2400 0000, which is too far away to main(). Therefore, the project enables long calls to ensure that main() is able to call sub functions. To enable, select **Build Properties > Optimization > Enable long calls** and select the **Yes** checkbox. Alternatively, add -mlongcalls to the compiler flags.

**Parent topic:**System Optimization

**Power Efficiency** RT6xx HiFi4 DSP has been equipped as a powerful processing core that can run at 600 MHz, but it may not be necessary at all time. It is always recommended to optimize the power efficiency at system level.

Voltage level and core frequency have huge impact on the power efficiency. Using FFT processing as example, continuous FFT may take ~130 mW at Vddcore 1.1 V / 452 MHz, and ~4 mW at Vddcore 0.7 V / 29 MHz, at room temperature. Profile or do cycle counts for software workload. If it only requires 300 MHz at peak, then the run at full power is not required. Instead, it can run with Vddcore 0.8 V 300 MHz. For more information on the Vddcore and DSP frequency operating conditions, see any *dsp example|cm33|pmic\_support.c BOARD\_SetPmicVoltageForFreq()*, or see the data sheet, section 13.1 General Operating Conditions.

Some other tips for better power efficiency:

- Turn off DSP/ set DSPSTALL, if needed.
- If possible, make DSP clock adapts to its workload.
- Call XT\_WAITI when DSP is in while loop waiting for interrupt. Similar to Arm side \_\_WFI(), it suspends some processor operations to reduce the power consumption.

```
#include <xtensa/tie/xt_interrupt.h >
extern void XT_WAITI(immediate s);
```

The immediate value passed is the interrupt level or lower to be IGNORED. For example, if you call XT\_WAITI(2) both L1 and L2 interrupts are ignored and only L3 interrupts can wake up DSP.

Current can further reduce when clocks are turned off to unused memory partitions. For more information, see User Manual 4.5.5.13 DSP SRAM access disable, Register SYSCTL0\_DSP\_SRAM\_ACCESS\_DISABLE.

- PLLs consume power. Consider FFRO for low-power use cases.

**Parent topic:**System Optimization

**Parent topic:**[HiFi4 System Programming](#)

### Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## 1.5 Release Notes

### 1.5.1 MCUXpresso SDK Release Notes

#### Overview

The MCUXpresso SDK is a comprehensive software enablement package designed to simplify and accelerate application development with Arm Cortex-M-based devices from NXP, including its general purpose, crossover and Bluetooth-enabled MCUs. MCUXpresso SW and Tools for DSC further extends the SDK support to current 32-bit Digital Signal Controllers. The MCUXpresso SDK includes production-grade software with integrated RTOS (optional), integrated enabling software technologies (stacks and middleware), reference software, and more.

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

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

#### MCUXpresso SDK

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

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

#### Development tools

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

- MCUXpresso IDE, Rev. 25.06.xx
- IAR Embedded Workbench for Arm, version is 9.60.4
- Keil MDK, version is 5.41
- MCUXpresso for VS Code v25.06
- GCC Arm Embedded Toolchain 14.2.x
- Xtensa Xplorer, version is 10.1.11
- Xtensa C Compiler, version is RI-2023.11

## Supported development systems

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

| Devel-<br>opment<br>boards | MCU devices                                            |                                     |                                            |
|----------------------------|--------------------------------------------------------|-------------------------------------|--------------------------------------------|
| <b>EVK-MIMXRT685</b>       | MIMXRT685SFAWBR,<br>MIMXRT685SVFVKB,<br>MIMXRT633SFVKB | MIMXRT685SFFOB,<br>MIMXRT633SFAWBR, | <b>MIMXRT685SFVKB</b> ,<br>MIMXRT633SFFOB, |

## 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

**Wireless EdgeFast Bluetooth PAL** For more information, see the MCUXpresso SDK EdgeFast Bluetooth Protocol Abstraction Layer User's Guide.

**Ethermind BT/BLE Stack** nxp\_bt\_ble\_stack

**coreHTTP** coreHTTP



**wpa\_supplicant-rtos** NXP Wi-Fi WPA Supplicant

**NXP Wi-Fi** The MCUXpresso SDK provides driver for NXP Wi-Fi external modules. The Wi-Fi driver is integrated with LWIP TCPIP stack and demonstrated with several network applications (iperf and AWS IoT).

For more information, see Getting Started with NXP based Wireless Modules and i.MX RT Platform Running on RTOS (document: UM11441).

**Voice Seeker (no AEC)** VoiceSeeker is a multi-microphone voice control audio front-end signal processing solution. VoiceSeeker is not featuring acoustic echo cancellation (AEC).

**Voice intelligent technology library** Voice Intelligent Technology (VIT) Library provides wake word and voice command engine for voice control

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

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

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

**TF-M** Trusted Firmware - M Library

**PSA Test Suite** Arm Platform Security Architecture Test Suite

**Mbed Crypto** Mbed Crypto library

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

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

**Multicore** Multicore Software Development Kit

**MCU Boot** Open source MCU Bootloader.

**mbedTLS** mbedtls SSL/TLS library v3.x

**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 TCPIP Stack and MCUXpresso SDK Integration User's Guide* (document MCUXSDKLWIPUG).

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

**eIQ** The package contains several example applications using the eIQ TensorFlow Lite for Microcontrollers library.

eIQ machine learning SDK containing:

- Arm CMSIS-NN library (neural network kernels optimized for Cortex-M cores)
- Inference engines:
  - TensorFlow Lite Micro
  - DeepView RT
- Example code for TensorFlow Lite Micro, Glow, and DeepView RT

**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.

**DSP Neural Networks** DSP Neural Networks Framework based on Xtensa Neural Networks Library from Cadence Design Systems for Xtensa HiFi Audio Engines.

**NatureDSP for Hifi4** Digital Signal Processing for Xtensa Hifi4 DSP Audio Engines.

**DSP Audio Streamer** DSP Audio Streamer Framework based on Xtensa Audio Framework from Cadence Design Systems for Xtensa DSP Audio Engines.

**DSP Codecs for HiFi4** DSP Codecs for HiFi4

**cJSON** Ultralightweight JSON parser in ANSI C

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

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

## 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.

### New Project Wizard compile failure

The following components request the user to manually select other components that they depend upon in order to compile.

These components depend on several other components and the New Project Wizard (NPW) is not able to decide which one is needed by the user.

**Note:** xxx means core variants, such as, cm0plus, cm33, cm4, cm33\_nodsp.

**\*\*Components:\*\***issdk\_mag3110, issdk\_host, systick, gpio\_kinetis, gpio\_lpc, issdk\_mpl3115, sensor\_fusion\_agm01, sensor\_fusion\_agm01\_lpc, issdk\_mma845x, issdk\_mma8491q, issdk\_mma865x, issdk\_mma9553, and CMSIS\_RTOS2.CMSIS\_RTOS2, and components which include cache driver, such as enet\_qos.

Also for low-level adapter components, currently the different types of the same adapter cannot be selected at the same time.

For example, if there are two types of timer adapters, gpt\_adapter and pit\_adapter, only one can be selected as timer adapter

in one project at a time. Duplicate implementation of the function results in an error.

**Note:** Most of middleware components have complex dependencies and are not fully supported in new project wizard. Adding a middleware component may result in compile failure.

### CMSIS PACK new project compile failure

The generated configuration cannot be applied globally. The components, serial\_manager\_usb\_cdc\_virtual and serial\_manager\_usb\_cdc\_virtual\_xxx (xxx means core variants like cm0plus, cm33, cm4, and cm33\_nodsp) are unsupported for new project wizard of CMSIS pack and will lead to compile failure if selected while creating new project(s).

### Low speed devices not supported

The host examples cannot support low-speed devices.

### IAR cannot debug RAM application with J-Link

Currently, IAR will call J-Link reset after the application is downloaded to SRAM, but such operation will cause SRAM data lost.

Here is a workaround to avoid real reset, with the cost of no any reset during the debugging, and hardware status uncleared.

1. Build and debug IAR project once and see the settings folder created.
2. Create the \_JLinkScript file in the settings folder with the following contents.

```
void ResetTarget(void) {  
    JLINK_TARGET_Halt();  
}
```

3. Debug the project again and now it can work.

### DSP\_rtp\_demo failure

The dsp\_rtp\_demo streaming fails after several minutes with timeout from the DSP side. The issue is probably caused by the incompatibility of the codec and the new XAF version.

### lwip\_httpsrv\_ota\_wifi example fails to accept the new image on EVKMIMXRT685

After uploading the new image over https, and rebooting the board the new image starts. However, reboot starts it is not possible to accept the update and make it permanent. This issue is specific to combination of EVKMIMXRT685 board and AW-NM191NF-uSD Wi-Fi module. Other configurations are not affected.

### Board may reset itself when running SD card related cases

Board may reset itself when running SD card related cases. However, the issue is not reproduced if more power supply is connected.

### Log output may be mixed in shell/hfp example

When multiple tasks print the log, the serial port terminal output has the probability to appear mixed.

### LE encryption failure causes connection to fail

There can be a corner case when LE link encryption can fail. This occurs when device under test (DUT); RT Bluetooth controller here, while waiting for the response to LL\_SLAVE\_FEATURES\_REQ, instead receives the LL\_ENC\_REQ response from a remote device.

This causes deadlock scenario where DUT and remote devices are stuck waiting for response from peer.

### Connection disconnects with 7.5 ms connection interval

When wireless example works as a peripheral, the central role with 7.5 ms connection interval connects to the wireless example. However, when using the 6th/7th/8th central to connect the wireless example, all the previous connections with the previous centrals are disconnected.

When wireless example works as central and the connection interval is 7.5 ms, the 4th peripheral is not scanned.

### a2dp sink demo: Noise may occur when phone plays music with other operations

Noise may occur when phone plays music with other operations.

For example:

- Switches the WIFI network of the phone when playing music
- Switches music when playing the online music

### Wireless EdgeFast Bluetooth PAL

For more information, see the MCUXpresso SDK EdgeFast Bluetooth Protocol Abstraction Layer User's Guide.

### Examples hello\_world\_ns, secure\_faults\_ns, and secure\_faults\_trdc\_ns have incorrect library path in GUI projects

When the affected examples are generated as GUI projects, the library linking the secure and non-secure worlds has an incorrect path set. This causes linking errors during project compilation.

**Examples:** hello\_world\_ns, hello\_world\_s, secure\_faults\_ns, secure\_faults\_s, secure\_faults\_trdc\_ns, secure\_faults\_trdc\_s

**Affected toolchains:** mdk, iar

**Workaround:** In the IDE project settings for the non-secure (\_ns) project, find the linked library (named hello\_world\_s\_CMSE\_lib.o, or similar, depending on the example project) and replace the path to the library with <build\_directory>/<secure\_world\_project\_folder>/<IDE>/, replacing the subdirectory names with the build directory, the secure world project name, and IDE name.

## 1.6 ChangeLog

### 1.6.1 MCUXpresso SDK Changelog

#### Board Support Files

##### board

###### [25.06.00]

- Initial version

##### clock\_config

###### [25.06.00]

- Initial version

##### pin\_mux

###### [25.06.00]

- Initial version
- 

#### ACMP

###### [2.3.0]

- Improvements
  - Expose C0 register FILTER\_CNT bitfield and FPR bitfield to the user.

###### [2.2.0]

- Improvements
  - Updated feature macros for roundrobin mode, window mode, filter mode, and 3V domain removes.

###### [2.1.0]

- New Feature
  - Supported the platforms which don't have hysteresis mode.

###### [2.0.6]

- Bug Fixes
  - Fixed the wrong comments, the DAC value should range from 0 to 255.

#### [2.0.5]

- Bug Fixes
  - Fixed the out-of-bounds error of Coverity caused by missing an assert sentence to avoid the return value of `ACMP_GetInstance()` exceeding the array bounds.
  - Fixed the violations of MISRA C-2012 rules:
    - \* Rule 10.1, 14.4, 16.4, 17.7.

#### [2.0.4]

- Bug Fixes
  - Avoided changing w1c bit in `ACMP_SetRoundRobinPreState()`.

#### [2.0.3]

- New Features
  - Added feature functions for usage of different power domains(1.8 V and 3 V). These functions are first enabled in ULP1. They are about:
    - \* `ACMP_EnableLinkToDAC()`
    - \* `ACMP_SetDiscreteModeConfig()`
    - \* `ACMP_GetDefaultDiscreteModeConfig()`

#### [2.0.2]

- Other Changes
  - Changed coding style of peripheral base address from “`s_acmpBases`” to “`s_acmpBase`”.

#### [2.0.1]

- Bug Fixes
    - Fixed bug regarding the function “`ACMP_SetRoundRobinConfig`”. It will not continue execution but returns directly after disabling round robin mode.
- 

### CACHE64

#### [2.0.11]

- Bug Fixes
  - Fixed CERT INT30-C violations: check and guarantee address plus size is equal or smaller than `UINT32_MAX`.

#### [2.0.10]

- Improvements
  - Updated `CACHE64_InvalidateCacheByRange()`, `CACHE64_CleanCacheByRange()` and `CACHE64_CleanInvalidateCacheByRange()` to support some platforms that multiple regions in the memory map are remapped to create a continuous address space.



**[2.0.9]**

- Improvements
  - Removed assert(false) in CACHE64\_GetInstanceByAddr.

**[2.0.8]**

- Improvements
  - Updated function CACHE64\_GetInstanceByAddr() to support some devices that provide alias of cacheable memory section.

**[2.0.7]**

- Improvements
  - Check input parameter “size\_byte” must be larger than 0.

**[2.0.6]**

- Bug Fixes
  - Fixed overflow for CACHE64\_GetInstanceByAddr()/CACHE64\_CleanCacheByRange()/CACHE64\_Invalid APIs.

**[2.0.5]**

- Improvement
  - Made use of FSL\_FEATURE\_CACHE64\_CTRL\_HAS\_NO\_WRITE\_BUF feature

**[2.0.4]**

- Improvement
  - Disable cache policy feature on SoC without CACHE64\_POLSEL IP.
- Bug Fixes
  - Fixed doxygen issue.

**[2.0.3]**

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

**[2.0.2]**

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 10.1, 10.3, 10.4 and 14.4.
  - Fixed doxygen issue.

#### [2.0.1]

- Improvements
  - Moved CLCR register configuration out of the while loop, it's unnecessary to repeat this operation.

#### [2.0.0]

- Initial version.
- 

### CASSPER

#### [2.2.4]

- Fix MISRA-C 2012 issue.

#### [2.2.3]

- Added macro into CASPER\_Init and CASPER\_Deinit to support devices without clock and reset control.

#### [2.2.2]

- Enable hardware interleaving to RAMX0 and RAMX1 for CASPER by feature macro FSL\_FEATURE\_CASPER\_RAM\_HW\_INTERLEAVE

#### [2.2.1]

- Fix MISRA C-2012 issue.

#### [2.2.0]

- Rework driver to support multiple curves at once.

#### [2.1.0]

- Add ECC NIST P-521 elliptic curve.

#### [2.0.10]

- Fix MISRA C-2012 issue.

#### [2.0.9]

- Remove unused function Jac\_oncurve().
- Fix ECC384 build.

#### [2.0.8]

- Add feature macro for CASPER\_RAM\_OFFSET.

**[2.0.7]**

- Fix MISRA C-2012 issue.

**[2.0.6]**

- Bug Fixes
  - Fix IAR Pa082 warning

**[2.0.5]**

- Bug Fixes
  - Fix sign-compare warning

**[2.0.4]**

- For GCC compiler, enforce O1 optimize level, specifically to remove strict-aliasing option. This driver is very specific and requires -fno-strict-aliasing.

**[2.0.3]**

- Bug Fixes
  - Fixed the bug for KPSDK-28107 RSUB, FILL and ZERO operations not implemented in enum\_casper\_operation.

**[2.0.2]**

- Bug Fixes
  - Fixed KPSDK-25015 CASPER\_MEMCPY hard-fault on LPC55xx when both source and destination buffers are outside of CASPER\_RAM.

**[2.0.1]**

- Bug Fixes
  - Fixed the bug that KPSDK-24531 double\_scalar\_multiplication() result may be all zeroes for some specific input.

**[2.0.0]**

- Initial version.
- 
- 

**COMMON****[2.6.0]**

- Bug Fixes
  - Fix CERT-C violations.

#### [2.5.0]

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

#### [2.4.3]

- Improvements
  - Enable 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\_ChannelsBusy 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\_DisableChannelGlobalSync

### [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****[2.4.1]**

- 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
- 
- 

## 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.
- 

## FLEXSPI

### [2.7.0]

- New Features
  - Added new API to support address remapping.

### [2.6.4]

- Improvements
  - Added new macro “FSL\_SDK\_ENABLE\_FLEXSPI\_RESET\_CONTROL” to support driver level reset control.



### [2.6.3]

- Bug Fixes
  - Fixed an issue which cause IPCR1[IPAREN] cleared by mistake.

### [2.6.2]

- Bug Fixes
  - Wait Bus IDLE before operation of FLEXSPI\_SoftwareReset(), FLEXSPI\_TransferBlocking() and FLEXSPI\_TransferNonBlocking().

### [2.6.1]

- Bug Fixes
  - Updated code of reset peripheral.
  - Updated FLEXSPI\_UpdateLUT() to check if input lut address is not in Flexspi AMBA region.
  - Updated FLEXSPI\_Init() to check if input AHB buffer size exceeded maximum AHB size.

### [2.6.0]

- New Features
  - Added new API to set AHB memory-mapped flash base address.
  - Added support of DLLxCR[REFPHASEGAP] bit field, it is recommended to set it as 0x2 if DLL calibration is enabled.

### [2.5.1]

- Bugfixes
  - Fixed handling of W1C bits in the INTR register
  - Removed FIFO resets from FLEXSPI\_CheckAndClearError
  - FLEXSPI\_TransferBlocking is observing IPCMDDONE and then fetches the final status of the transfer
  - Fixed issue that FLEXSPI2\_DriverIRQHandler not defined.

### [2.5.0]

- Improvements
  - Supported word un-aligned access for write/read blocking/non-blocking API functions.
  - Fixed dead loop issue in DLL update function when using FRO clock source.
  - Fixed violations of the MISRA C-2012 Rule 10.3.

### [2.4.0]

- Improvements
  - Isolated IP command parallel mode and AHB command parallel mode using feature MACRO.
  - Supported new column address shift feature for external memory.

#### [2.3.5]

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

#### [2.3.4]

- Bug Fixes
  - Updated `flexspi_config_t` structure and `FlexSPI_Init` to support new feature `FSL_FEATURE_FLEXSPI_HAS_NO_MCR0_COMBINATION`.

#### [2.3.3]

- Bug Fixes
  - Removed feature `FSL_FEATURE_FLEXSPI_DQS_DELAY_PS` for DLL delay setting. Changed to use feature `FSL_FEATURE_FLEXSPI_DQS_DELAY_MIN` to set slave delay target as 0 for DLL enable and clock frequency higher than 100MHz.

#### [2.3.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 8.4, 8.5, 10.1, 10.3, 10.4, 11.6 and 14.4.

#### [2.3.1]

- Bug Fixes
  - Wait for bus to be idle before using it as access to external flash with new setting in `FLEXSPI_SetFlashConfig()` API.
  - Fixed the potential buffer overread and Tx FIFO overwrite issue in `FLEXSPI_WriteBlocking`.

#### [2.3.0]

- New Features
  - Added new API `FLEXSPI_UpdateDllValue` for users to update DLL value after updating `flexspi` root clock.
  - Corrected grammatical issues for comments.
  - Added support for new feature `FSL_FEATURE_FLEXSPI_DQS_DELAY_PS` in DLL configuration.

#### [2.2.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 10.1, 10.3 and 10.4.
  - Updated `_flexspi_command` from named enumerator into anonymous enumerator.

### [2.2.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 10.1, 10.3, 10.4, 10.8, 11.9, 14.4, 15.7, 16.4, 17.7, 7.3.
  - Fixed IAR build warning Pe167.
  - Fixed the potential buffer overwrite and Rx FIFO overread issue in FLEXSPI\_ReadBlocking.

### [2.2.0]

- Bug Fixes
  - Fixed flag name typos: kFLEXSPI\_IpTxFifoWatermarkEmpltyFlag to kFLEXSPI\_IpTxFifoWatermarkEmptyFlag; kFLEXSPI\_IpCommandExcutionDoneFlag to kFLEXSPI\_IpCommandExecutionDoneFlag.
  - Fixed comments typos such as sequencen->sequence, levle->level.
  - Fixed FLSHCR2[ARDSEQID] field clean issue.
  - Updated flexspi\_config\_t structure and FlexSPI\_Init to support new feature FSL\_FEATURE\_FLEXSPI\_HAS\_NO\_MCR0\_ATDFEN and FSL\_FEATURE\_FLEXSPI\_HAS\_NO\_MCR0\_ARDFEN.
  - Updated flexspi\_flags\_t structure to support new feature FSL\_FEATURE\_FLEXSPI\_HAS\_INTEN\_AHBBUSERROREN.

### [2.1.1]

- Improvements
  - Defaulted enable prefetch for AHB RX buffer configuration in FLEXSPI\_GetDefaultConfig, which is align with the reset value in AHBRXBUFxCR0.
  - Added software workaround for ERR011377 in FLEXSPI\_SetFlashConfig; added some delay after DLL lock status set to ensure correct data read/write.

### [2.1.0]

- New Features
  - Added new API FLEXSPI\_UpdateRxSampleClock for users to update read sample clock source after initialization.
  - Added reset peripheral operation in FLEXSPI\_Init if required.

### [2.0.5]

- Bug Fixes
  - Fixed FLEXSPI\_UpdateLUT cannot do partial update issue.

### [2.0.4]

- Bug Fixes
  - Reset flash size to zero for all ports in FLEXSPI\_Init; fixed the possible out-of-range flash access with no error reported.

### [2.0.3]

- Bug Fixes
  - Fixed AHB receive buffer size configuration issue. The FLEXSPI\_AHBRXBUFCRO\_BUFSZ field should configure 64 bits size, and currently the AHB receive buffer size is in bytes which means 8-bit, so the correct configuration should be `config->ahbConfig.buffer[i].bufferSize / 8`.

### [2.0.2]

- New Features
  - Supported DQS write mask enable/disable feature during set FLEXSPI configuration.
  - Provided new API FLEXSPI\_TransferUpdateSizeEDMA for users to update eDMA transfer size(SSIZE/DSIZE) per DMA transfer.
- Bug Fixes
  - Fixed invalid operation of FLEXSPI\_Init to enable AHB bus Read Access to IP RX FIFO.
  - Fixed incorrect operation of FLEXSPI\_Init to configure IP TX FIFO watermark.

### [2.0.1]

- Bug Fixes
  - Fixed the flag clear issue and AHB read Command index configuration issue in FLEXSPI\_SetFlashConfig.
  - Updated FLEXSPI\_UpdateLUT function to update LUT table from any index instead of previous command index.
  - Added bus idle wait in FLEXSPI\_SetFlashConfig and FLEXSPI\_UpdateLUT to ensure bus is idle before any change to FlexSPI controller.
  - Updated interrupt API FLEXSPI\_TransferNonBlocking and interrupt handle flow FLEXSPI\_TransferHandleIRQ.
  - Updated eDMA API FLEXSPI\_TransferEDMA.

### [2.0.0]

- Initial version.
- 

## FLEXSPI DMA Driver

### [2.2.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 10.1, 10.3, 10.4, 10.8.

### [2.2.0]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 10.1, 10.3.
- New Features
  - Updated name of FLEXSPI\_TransferGetTransferCountDMA API.

**[2.1.1]**

- New Features
  - Updated driver to support feature FSL\_FEATURE\_FLEXSPI\_DMA\_MULTIPLE\_DES.

**[2.1.0]**

- Bug Fixes
  - Updated enumeration flexspi\_dma\_transfer\_nsize\_t and remove the unsupported items.
- New Features
  - Updated driver for deprecating the multiple linked descriptors inside FLEXSPI\_TransferDMA, only up to one linked descriptor is needed according to hardware update.

**[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.
- 

**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 mainin change is updating APIs with prefix of `_PinXXX()` and `_PorortXXX`

#### [2.1.0]

- New Features
  - Added GPIO initialize API.

#### [2.0.0]

- Initial version.
- 

## HASHCRYPT

#### [2.0.0]

- Initial version.

#### [2.0.1]

- Supported loading AES key from unaligned address.

**[2.0.2]**

- Supported loading AES key from unaligned address for different compiler and core variants.

**[2.0.3]**

- Remove SHA512 and AES ICB algorithm definitions

**[2.0.4]**

- Add SHA context switch support

**[2.1.0]**

- Update the register name and macro to align with new header.
- Fixed the sign-compare warning in `hashcrypt_load_data`.

**[2.1.1]**

- Fix MISRA C-2012.

**[2.1.2]**

- Support loading AES input data from unaligned address.

**[2.1.3]**

- Fix MISRA C-2012.

**[2.1.4]**

- Fix context switch cannot work when switching from AES.

**[2.1.5]**

- Add data synchronization barrier inside `hashcrypt_sha_ldm_stm_16_words()` to prevent possible optimization issue.

**[2.2.0]**

- Add AES-OFB and AES-CFB mixed IP/SW modes.

**[2.2.1]**

- Add data synchronization barrier inside `hashcrypt_sha_ldm_stm_16_words()` prevent compiler from reordering memory write when `-O2` or higher is used.

**[2.2.2]**

- Add data synchronization barrier inside `hashcrypt_sha_ldm_stm_16_words()` to fix optimization issue

#### [2.2.3]

- Added check for size in `hashcrypt_aes_one_block` to prevent overflowing COUNT field in MEMCTRL register, if its bigger than COUNT field do a multiple runs.

#### [2.2.4]

- In all `HASHCRYPT_AES_xx` functions have been added setting CTRL\_MODE bitfield to 0 after processing data, which decreases power consumption.

#### [2.2.5]

- Add data synchronization barrier and instruction synchronization barrier inside `hashcrypt_sha_process_message_data()` to fix optimization issue

#### [2.2.6]

- Add data synchronization barrier inside `HASHCRYPT_SHA_Update()` and `hashcrypt_get_data()` function to fix optimization issue on MDK and ARMGCC release targets

#### [2.2.7]

- Add data synchronization barrier inside `HASHCRYPT_SHA_Update()` to fix optimization issue on MCUX IDE release target

#### [2.2.8]

- Unify hashcrypt hashing behavior between aligned and unaligned input data

#### [2.2.9]

- Add handling of set ERROR bit in the STATUS register

#### [2.2.10]

- Fix missing error statement in `hashcrypt_save_running_hash()`

#### [2.2.11]

- Fix incorrect SHA-256 calculation for long messages with reload

#### [2.2.12]

- Fix hardfault issue on the Keil compiler due to unaligned `memcpy()` input on some optimization levels

#### [2.2.13]

- Added function `hashcrypt_seed_prng()` which loading random number into PRNG\_SEED register before AES operation for SCA protection



**[2.2.14]**

- Modify function `hashcrypt_get_data()` to prevent issue with unaligned access

**[2.2.15]**

- Add wait on DIGEST BIT inside `hashcrypt_sha_one_block()` to fix issues with some optimization flags

**[2.2.16]**

- Add DSB instruction inside `hashcrypt_sha_ldm_stm_16_words()` to fix issues with some optimization flags
- 

**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 vise 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 `MSTSCLLOW` and `MSTSCLHIGH` 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 kReceiveDataBeginState 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 execution 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) hardcode.
  - 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
- 

## I3C

### [2.14.1]

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

### [2.14.0]

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

### [2.13.1]

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

### [2.13.0]

- New features
  - Added the hot-join support for I3C bus initialization API.
- Bug Fixes
  - Set read termination with START at the same time in case unknown issue.
  - Set MCTRL[TYPE] as 0 for DDR force exit.

- Improvements
  - Added the API to reset device count assigned by ENTDAAs.
  - Provided the method to set global macro I3C\_MAX\_DEVCNT to determine how many device addresses ENTDAAs can allocate at one time.
  - Initialized target management static array based on instance number for the case that multiple instances are used at the same time.

#### [2.12.0]

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

#### [2.11.1]

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

#### [2.11.0]

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

#### [2.10.8]

- Improvements
  - Support more instances.

#### [2.10.7]

- Improvements
  - Fixed the potential compile warning.

#### [2.10.6]

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

#### [2.10.5]

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

#### [2.10.4]

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

#### [2.10.3]

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

#### [2.10.2]

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

#### [2.10.1]

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

#### [2.10.0]

- New features
  - Added I3C extended IBI data support.

#### [2.9.0]

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

#### [2.8.2]

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

#### [2.8.1]

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



**[2.8.0]**

- Improvements
  - Added API `I3C_MasterProcessDAASpecifiedBaudrate` for temporary baud rate adjustment when I3C master assigns dynamic address.

**[2.7.1]**

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

**[2.7.0]**

- Fixed the CCM problem in file `fsl_i3c.c`.
- Fixed the `FSL_FEATURE_I3C_HAS_NO_SCONFIG_IDRAND` usage issue in `I3C_GetDefaultConfig` and `I3C_Init`.

**[2.6.0]**

- Fixed the `FSL_FEATURE_I3C_HAS_NO_SCONFIG_IDRAND` usage issue in `fsl_i3c.h`.
- Changed some static functions in `fsl_i3c.c` as non-static and define the functions in `fsl_i3c.h` to make I3C DMA driver reuse:
  - `I3C_GetIBIType`
  - `I3C_GetIBIAddress`
  - `I3C_SlaveCheckAndClearError`
- Changed the handle pointer parameter in IRQ related functions to `void *` type to make it reuse in I3C DMA driver.
- Added new API `I3C_SlaveRequestIBIWithSingleData` for slave to request single data byte, this API could be used regardless slave is working in non-blocking interrupt or non-blocking dma.
- Added new API `I3C_MasterGetDeviceListAfterDAA` for master application to get the device information list built up in DAA process.

**[2.5.4]**

- Improved I3C driver to avoid setting state twice in the `SendCommandState` of `I3C_RunTransferStateMachine`.
- Fixed MISRA violation of rule 20.9.
- Fixed the issue that `I3C_MasterEmitRequest` did not use Type I3C SDR.

**[2.5.3]**

- Updated driver for new feature `FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH` and `FSL_FEATURE_I3C_HAS_NO_SCONFIG_IDRAND`.

**[2.5.2]**

- Updated driver for new feature `FSL_FEATURE_I3C_HAS_NO_MERRWARN_TERM`.
- Fixed the issue that call to `I3C_MasterTransferBlocking` API did not generate STOP signal when NAK status was returned.

### [2.5.1]

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

### [2.5.0]

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

### [2.4.0]

- Bug Fixes
  - Fixed `kI3C_SlaveMatchedFlag` interrupt is not properly handled in `I3C_SlaveTransferHandleIRQ` when it comes together with interrupt `kI3C_SlaveBusStartFlag`.
  - Fixed the inaccurate I2C baudrate calculation in `I3C_MasterSetBaudRate`.
  - Added new API `I3C_MasterGetIBIRules` to get registered IBI rules.
  - Added new variable `isReadTerm` in `struct_i3c_master_handle` for transfer state routine to check if `MCTRL.RDTERM` is configured for read transfer.
  - Changed to emit Auto IBI in transfer state routine for slave start flag assertion.
  - Fixed the slave `maxWriteLength` and `maxReadLength` does not be configured into `SMAXLIMITS` register issue.
  - Fixed incorrect state for IBI in I3C master interrupt transfer IRQ handle routine.
  - Added `isHotJoin` in `i3c_slave_config_t` to request hot-join event during slave init.

### [2.3.2]

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

### [2.3.1]

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

### [2.3.0]

- Improvements
  - Added I3C common driver APIs to initialize I3C with both master and slave configuration.

- Updated I3C master transfer callback to function set structure to include callback invoke for IBI event and slave2master event.
- Updated I3C master non-blocking transfer model and always enable the interrupts to be able to re-act to the slave start event and handle slave IBI.

#### [2.2.0]

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

#### [2.1.2]

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

#### [2.1.1]

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

#### [2.1.0]

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

#### [2.0.0]

- Initial version.
- 

### I3C\_DMA

#### [2.1.8]

- Bug Fixes
  - Updated the logic to handle Rx termination and complete event to adapt different situation.
- Improvements
  - Added the MCTRLDONE flag check after STOP request to ensure the completion of whole transfer operation.

#### [2.1.7]

- Bug Fixes
  - Fixed the issue to use subaddress to read/write data with RT500/600 DMA.

#### [2.1.6]

- Improvements
  - Added the FSL\_FEATURE\_I3C\_HAS\_NO\_MASTER\_DMA\_WDATA\_REG to select the correct register to write data based on specific Soc.

#### [2.1.5]

- New features
  - Supported I3C HDR-DDR transfer with DMA.
- Improvements
  - Added workaround for RT500/600 I3C DMA transfer.
  - Removed I3C IRQ handler calling in the Tx EDMA callback. Previously driver doesn't use the END byte which can trigger the complete interrupt for controller sending and receiving, now let I3C event handler deal with I3C events.
  - Used linked DMA to transfer all I3C subaddress and data without handling of intermediate states, simplifying code logic.
  - Prepare the Tx DMA before I3C START to ensure there's no time delay between START and transmitting data.

#### [2.1.4]

- Improvements
  - Used linked DMA transfer to reduce the latency between DMA transfers previous data and the END byte.

#### [2.1.3]

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

#### [2.1.2]

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

#### [2.1.1]

- Bug Fixes
  - Fixed MISRA issue rule 9.1.

#### [2.1.0]

- Improvements
  - Deleted legacy IBI data request code.

**[2.0.1]**

- Bug Fixes
  - Fixed issue that bus STOP occurs when Tx FIFO still takes data.
- Improvements
  - Fixed the build warning due to armgcc strict check.

**[2.0.0]**

- Initial version.
- 

**IAP****[2.1.3]**

- Bug Fixes
  - Fixed misra issue.

**[2.1.2]**

- Bug Fixes
  - Fixed some macro undefined issue.
  - Put IAP\_FlexspiNorInit API into RAM.

**[2.1.1]**

- Bug Fixes
  - Fixed misra issue.

**[2.1.0]**

- New Features
  - Added IAP\_RunBootLoader() API

**[2.0.2]**

- Bug Fixes
  - Fixed doxygen issue.

**[2.0.1]**

- Bug Fixes
  - Minor update for MISRA issue fix.

**[2.0.0]**

- Initial version.
-

## INPUTMUX

### [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.
- 

**IOPCTL****[2.0.0]**

- Initial version.
- 

**LPADC****[2.9.3]**

- Improvements
  - Add timeout for while loop code.

**[2.9.2]**

- Improvements
  - Fixed CERT-C issues.

**[2.9.1]**

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

**[2.9.0]**

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

**[2.8.4]**

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

**[2.8.3]**

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

### [2.8.2]

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

### [2.8.1]

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

### [2.8.0]

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

### [2.7.2]

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

### [2.7.1]

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

### [2.7.0]

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



### [2.6.2]

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

### [2.6.1]

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

### [2.6.0]

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

### [2.5.2]

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

### [2.5.1]

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

### [2.5.0]

- Improvements
  - Added missing items to enable trigger interrupts.

### [2.4.0]

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

### [2.3.0]

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

### [2.2.1]

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

### [2.2.0]

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

### [2.1.1]

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

### [2.1.0]

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

### [2.0.2]

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

**[2.0.1]**

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

**[2.0.0]**

- Initial version.
- 

**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.
- 

**MU****[2.2.0]**

- New Features
  - Added API MU\_GetRxStatusFlags.

#### [2.1.3]

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

#### [2.1.2]

- Bug Fixes
  - Fixed issue that MU\_GetInstance() is defined but never used.

#### [2.1.1]

- Bug Fixes
  - Fixed general interrupt comment typo.

#### [2.1.0]

- Improvements
  - Added new enum mu\_msg\_reg\_index\_t.

#### [2.0.7]

- Bug Fixes
  - Fixed MU\_GetInterruptsPending bug that can not get general interrupt status.

#### [2.0.6]

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

#### [2.0.5]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 14.4, 15.5.

#### [2.0.4]

- Improvements
  - Improved for the platforms which don't support reset assert interrupt and get the other core power mode.

#### [2.0.3]

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

**[2.0.2]**

- Improvements
  - Added support for MIMX8MQx.

**[2.0.1]**

- Improvements
  - Added support for MCIMX7Ux\_M4.

**[2.0.0]**

- Initial version.
- 

**OSTIMER****[2.2.4]**

- Bug Fixes
  - Fixed CERT INT31-C violations.

**[2.2.3]**

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

**[2.2.2]**

- Improvements
  - Support devices with different OSTIMER instance name.

**[2.2.1]**

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

**[2.2.0]**

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

**[2.1.2]**

- Bug Fixes
  - Fixed MISRA-2012 rule 10.8.

### [2.1.1]

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

### [2.1.0]

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

### [2.0.3]

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

### [2.0.2]

- Improvements
  - Added support for OSTIMER0

### [2.0.1]

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

### [2.0.0]

- Initial version.
- 

## OTFAD

### [2.1.4]

- Bug fixes
  - Fixed MISRA 2012 issue: 10.1.

**[2.1.3]**

- Bug fixes
  - Fixed the error that waiting for both FLEXSPI AHB idle and SEQ idle.

**[2.1.2]**

- Bug fixes
  - Fixed MISRA 2012 issue: 10.4.

**[2.1.1]**

- Improvements:
  - Hided some bits in CR and SR registers for selected platforms.
  - Fixed doxygen issues.

**[2.1.0]**

- Improvements:
  - Used boolean type to define 1-bit field concepts.

**[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.
- 
- 

**POWERQUAD****[2.2.0]**

- New Features
  - Added new API PQ\_Arctan2Fixed.

### [2.1.1]

- Bug Fixes
  - Remove PQ\_WaitDone from PQ\_ArctanFixed and PQ\_ArctanhFixed because it is unnecessary.

### [2.1.0]

- Improvements
  - Fixed typo issue for biquad related function name.
  - Changed operator from “%” into “&” to reduce heavy cycle for biquad functions.

### [2.0.5]

- Improvements
  - Added a note in driver for FIR that powerquad has a hardware limitation, when using it for FIR increment calculation, the address of pSrc needs to be a continuous address.

### [2.0.4]

- Improvements
  - Supported the platforms which don't have PowerQuad clock and reset control.

### [2.0.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 8.4, 10.1, 10.3, 10.4, 10.6, and so on.

### [2.0.2]

- Bug Fixes
  - Fixed array size issue in fsl\_powerquad\_data.h file.
  - Fixed vector function pipeline issue.

### [2.0.1]

- Bug Fixes
  - Fixed build error in C++ mode.

### [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.
- 
- 

## 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 SC-TIMER\_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 SC-TIMER\_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.
- 

### SEMA42

#### [2.1.0]

- New Features
  - Added SEMA42\_BUSY\_POLL\_COUNT parameter to prevent infinite polling loops in SEMA42 operations.
  - Added timeout mechanism to all polling loops in SEMA42 driver code.
- Improvements
  - Updated SEMA42\_Lock function to return status\_t instead of void for better error handling.
  - Enhanced documentation to clarify timeout behavior and return values.

#### [2.0.4]

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

#### [2.0.3]

- Improvements
  - Changed to implement SEMA42\_Lock base on SEMA42\_TryLock.

#### [2.0.2]

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

#### [2.0.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 10.4, 14.4, 18.1.

#### [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

#### [2.2.1]

- Bug Fixes
  - Fixed MISRA 2012 11.6 issue..

**[2.2.0]**

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

**TRNG****[2.0.18]**

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

**[2.0.17]**

- New features:
  - Add support for RT700.

**[2.0.16]**

- Improvements:
  - Added support for Dual oscillator mode.

**[2.0.15]**

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

**[2.0.14]**

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

**[2.0.13]**

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

**[2.0.12]**

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

#### [2.0.11]

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

#### [2.0.10]

- Bug Fix:
  - Fixed doxygen issues.

#### [2.0.9]

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

#### [2.0.8]

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

#### [2.0.7]

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

#### [2.0.6]

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

#### [2.0.5]

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

#### [2.0.4]

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

#### [2.0.3]

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

**[2.0.2]**

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

**[2.0.1]**

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

**[2.0.0]**

- Initial version.
- 

**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
- 

### USDHC

#### [2.8.5]

- Improvements
  - Enable the driver to be AARCH64 compatible.



**[2.8.4]**

- Improvements
  - Add feature macro FSL\_FEATURE\_USDHC\_HAS\_NO\_VS18.

**[2.8.3]**

- Improvements
  - Improved api USDHC\_EnableAutoTuningForCmdAndData to adapt to new bit field name for USDHC\_VEND\_SPEC2 register.

**[2.8.2]**

- Improvements
  - Added feature macro FSL\_FEATURE\_USDHC\_HAS\_NO\_VOLTAGE\_SELECT.

**[2.8.1]**

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 11.9.

**[2.8.0]**

- Improvements
  - Fixed the mmc boot transfer failed issue which is caused by the Dma complete interrupt not enabled.
  - Marked api USDHC\_AdjustDelayForManualTuning as deprecated and added new api USDHC\_SetTuningDelay/USDHC\_GetTuningDelayStatus.
  - Improved the manual tuning flow according to specification.
  - Added memory address conversion to support buffers which could only be accessed using alias address by non-core masters.
  - Fixed violations of MISRA C-2012 rule 10.4.

**[2.7.0]**

- Improvements
  - Added api USDHC\_TransferScatterGatherADMANonBlocking to support scatter gather transfer.
  - Added feature FSL\_FEATURE\_USDHC\_REGISTER\_HOST\_CTRL\_CAP\_HAS\_NO\_RETUNING\_TIME\_COUNTER for re-tuning time counter field in HOST\_CTRL\_CAP register.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 11.9, 10.1, 10.3, 10.4, 8.4.

**[2.6.0]**

- Improvements
  - Added api USDHC\_SetStandardTuningCounter to support adjust tuning counter of Standard tuning.

### [2.5.1]

- Improvements
  - Used different status code for command and data interrupt callback.
  - Added cache line invalidate for receive buffer in driver IRQ handler to fix CM7 speculative access issue.

### [2.5.0]

- Improvements
  - Added new api USDHC\_SetStrobeDllOverride for HS400 strobe dll override mode delay taps configurations.
  - Corrected the STROBE DLL configurations sequence.

### [2.4.0]

- Improvements
  - Added feature macro for read/write burst length.
    - \* Disabled redundant interrupt per different transfer request.
    - \* Disabled interrupt and reset command/data pointer in handle when transfer completes.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 11.9, 15.7, 4.7, 16.4, 10.1, 10.3, 10.4, 11.3, 14.4, 10.6, 17.7, 16.1, 16.3.
  - Fixed PA082 build warning.
  - Fixed logically dead code Coverity issue.

### [2.3.0]

- Improvements
  - Added USDHC\_SetDataConfig API to support manual tuning.
  - Removed the limitaion that source clock must be bigger than the target in function USDHC\_SetSdClock by using source clock frequency as target directly.
  - Added peripheral reset in USDHC\_Init function.
  - Added tuning reset support in function USDHC\_Reset function.

### [2.2.8]

- Bug Fixes
  - Fixed out-of bounds write in function USDHC\_ReceiveCommandResponse.

### [2.2.7]

- Improvements
  - Added API USDHC\_GetEnabledInterruptStatusFlags and used in USDHC\_TransferHandleIRQ.
  - Removed useless member interruptFlags in usdhc\_handle\_t.

### [2.2.6]

- Improvements
  - Added address align check for ADMA descriptor table address.
  - Changed USDHC\_ADMA1\_DESCRIPTOR\_MAX\_LENGTH\_PER\_ENTRY to (65536-4096) to make sure the data address is 4KB align for a transfer which need more than one ADMA1 descriptor.

### [2.2.5]

- Bug Fixes
  - Fixed MDK 66-D warning.

### [2.2.4]

- Bug Fixes
  - Fixed issue that real clock frequency wss mismatched with target clock frequency, which was caused by an incorrect prescaler calculation.
- New Features
  - Added control macro to enable/disable the CLOCK code in current driver.

### [2.2.3]

- Bug Fixes
  - Fixed issue where AMDA did not disable with DMAEN clear.
- Improvements
  - Improved set clock function to check the output frequency range.
  - Dynamic set SDCLKFS during DDR enable or disable.

### [2.2.2]

- Improvements
  - Improved read transfer cache maintain operation, combined clean, and invalidated them into one function.

### [2.2.1]

- Bug Fixes
  - Disabled the invalidate cache operation for tuning.

### [2.2.0]

- Improvements
  - Improved USDHC to support MMC boot feature.

### [2.1.3]

- Bug Fixes
  - Fixed MISRA issue.

### [2.1.2]

- Bug Fixes
  - Fixed Coverity issue.
  - Added base address and userData parameter for all callback functions.

### [2.1.1]

- Improvements
  - Added cache maintain operation.
  - Added timeout status check for the DATA transfer which ignore error.
  - Added feature macro for SDR50/SDR104 mode.
  - Removed useless IRQ handler from different platforms.

### [2.1.0]

- Improvements
  - Integrated tuning into transfer function.
  - Added strobe DLL feature.
  - Added enableAutoCommand23 in data structure.
  - Removed enable card clock function because the controller would handle the clock on/off.

### [2.0.0]

- Initial version.
- 

## UTICK

### [2.0.5]

- Improvements
  - Improved for SOC RW610.

### [2.0.4]

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

### [2.0.3]

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

### [2.0.2]

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

**[2.0.1]**

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

**[2.0.0]**

- Initial version.
- 

**WWDT****[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 WWDT 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.
- 

## 1.7 Driver API Reference Manual

This section provides a link to the Driver API RM, detailing available drivers and their usage to help you integrate hardware efficiently.

[MIMXRT685S](#)

## 1.8 Middleware Documentation

Find links to detailed middleware documentation for key components. While not all onboard middleware is covered, this serves as a useful reference for configuration and development.

### 1.8.1 Multicore

multicore

### 1.8.2 MCU Boot

mcuboot\_opensource

### 1.8.3 eIQ

eiq

## 1.8.4 FreeMASTER

*freemaster*

## 1.8.5 AWS IoT

aws\_iot

## 1.8.6 NXP Wi-Fi

wifi-bluetooth-802.15.4

## 1.8.7 FreeRTOS

*FreeRTOS*

## 1.8.8 Wireless EdgeFast Bluetooth PAL

edgefast\_bluetooth

## 1.8.9 lwIP

lwip

## 1.8.10 File systemFatfs

fatfs

## 1.8.11 DSP Audio Streamer

*Xtensa Audio Framework (XAF)*





# Chapter 2

## MIMXRT685S

### 2.1 ACMP: Analog Comparator Driver

`void ACMP_Init(CMP_Type *base, const acmp_config_t *config)`

Initializes the ACMP.

The default configuration can be got by calling `ACMP_GetDefaultConfig()`.

#### Parameters

- `base` – ACMP peripheral base address.
- `config` – Pointer to ACMP configuration structure.

`void ACMP_Deinit(CMP_Type *base)`

Deinitializes the ACMP.

#### Parameters

- `base` – ACMP peripheral base address.

`void ACMP_GetDefaultConfig(acmp_config_t *config)`

Gets the default configuration for ACMP.

This function initializes the user configuration structure to default value. The default value are:

Example:

```
config->enableHighSpeed = false;
config->enableInvertOutput = false;
config->useUnfilteredOutput = false;
config->enablePinOut = false;
config->enableHysteresisBothDirections = false;
config->hysteresisMode = kACMP_hysteresisMode0;
```

#### Parameters

- `config` – Pointer to ACMP configuration structure.

`void ACMP_Enable(CMP_Type *base, bool enable)`

Enables or disables the ACMP.

#### Parameters

- `base` – ACMP peripheral base address.
- `enable` – True to enable the ACMP.

`void ACMP_EnableLinkToDAC(CMP_Type *base, bool enable)`

Enables the link from CMP to DAC enable.

When this bit is set, the DAC enable/disable is controlled by the bit `CMP_C0[EN]` instead of `CMP_C1[DACEN]`.

#### Parameters

- `base` – ACMP peripheral base address.
- `enable` – Enable the feature or not.

`void ACMP_SetChannelConfig(CMP_Type *base, const acmp_channel_config_t *config)`

Sets the channel configuration.

Note that the plus/minus mux's setting is only valid when the positive/negative port's input isn't from DAC but from channel mux.

Example:

```
acmp_channel_config_t configStruct = {0};
configStruct.positivePortInput = kACMP_PortInputFromDAC;
configStruct.negativePortInput = kACMP_PortInputFromMux;
configStruct.minusMuxInput = 1U;
ACMP_SetChannelConfig(CMP0, &configStruct);
```

#### Parameters

- `base` – ACMP peripheral base address.
- `config` – Pointer to channel configuration structure.

`void ACMP_EnableDMA(CMP_Type *base, bool enable)`

Enables or disables DMA.

#### Parameters

- `base` – ACMP peripheral base address.
- `enable` – True to enable DMA.

`void ACMP_EnableWindowMode(CMP_Type *base, bool enable)`

Enables or disables window mode.

#### Parameters

- `base` – ACMP peripheral base address.
- `enable` – True to enable window mode.

`void ACMP_SetFilterConfig(CMP_Type *base, const acmp_filter_config_t *config)`

Configures the filter.

The filter can be enabled when the filter count is bigger than 1, the filter period is greater than 0 and the sample clock is from divided bus clock or the filter is bigger than 1 and the sample clock is from external clock. Detailed usage can be got from the reference manual.

Example:

```
acmp_filter_config_t configStruct = {0};
configStruct.filterCount = 5U;
configStruct.filterPeriod = 200U;
configStruct.enableSample = false;
ACMP_SetFilterConfig(CMP0, &configStruct);
```

#### Parameters

- `base` – ACMP peripheral base address.

- `config` – Pointer to filter configuration structure.

`void ACMP_SetDACConfig(CMP_Type *base, const acmp_dac_config_t *config)`

Configures the internal DAC.

Example:

```
acmp_dac_config_t configStruct = {0};
configStruct.referenceVoltageSource = kACMP_VrefSourceVin1;
configStruct.DACValue = 20U;
configStruct.enableOutput = false;
configStruct.workMode = kACMP_DACWorkLowSpeedMode;
ACMP_SetDACConfig(CMP0, &configStruct);
```

### Parameters

- `base` – ACMP peripheral base address.
- `config` – Pointer to DAC configuration structure. “NULL” is for disabling the feature.

`void ACMP_SetRoundRobinConfig(CMP_Type *base, const acmp_round_robin_config_t *config)`

Configures the round robin mode.

Example:

```
acmp_round_robin_config_t configStruct = {0};
configStruct.fixedPort = kACMP_FixedPlusPort;
configStruct.fixedChannelNumber = 3U;
configStruct.checkerChannelMask = 0xF7U;
configStruct.sampleClockCount = 0U;
configStruct.delayModulus = 0U;
ACMP_SetRoundRobinConfig(CMP0, &configStruct);
```

### Parameters

- `base` – ACMP peripheral base address.
- `config` – Pointer to round robin mode configuration structure. “NULL” is for disabling the feature.

`void ACMP_SetRoundRobinPreState(CMP_Type *base, uint32_t mask)`

Defines the pre-set state of channels in round robin mode.

Note: The pre-state has different circuit with get-round-robin-result in the SOC even though they are same bits. So get-round-robin-result can't return the same value as the value are set by pre-state.

### Parameters

- `base` – ACMP peripheral base address.
- `mask` – Mask of round robin channel index. Available range is channel0:0x01 to channel7:0x80.

`static inline uint32_t ACMP_GetRoundRobinStatusFlags(CMP_Type *base)`

Gets the channel input changed flags in round robin mode.

### Parameters

- `base` – ACMP peripheral base address.

### Returns

Mask of channel input changed asserted flags. Available range is channel0:0x01 to channel7:0x80.

`void ACMP_ClearRoundRobinStatusFlags(CMP_Type *base, uint32_t mask)`

Clears the channel input changed flags in round robin mode.

**Parameters**

- `base` – ACMP peripheral base address.
- `mask` – Mask of channel index. Available range is channel0:0x01 to channel7:0x80.

`static inline uint32_t ACMP_GetRoundRobinResult(CMP_Type *base)`

Gets the round robin result.

Note that the set-pre-state has different circuit with get-round-robin-result in the SOC even though they are same bits. So `ACMP_GetRoundRobinResult()` can't return the same value as the value are set by `ACMP_SetRoundRobinPreState`.

**Parameters**

- `base` – ACMP peripheral base address.

**Returns**

Mask of round robin channel result. Available range is channel0:0x01 to channel7:0x80.

`void ACMP_EnableInterrupts(CMP_Type *base, uint32_t mask)`

Enables interrupts.

**Parameters**

- `base` – ACMP peripheral base address.
- `mask` – Interrupts mask. See “`_acmp_interrupt_enable`”.

`void ACMP_DisableInterrupts(CMP_Type *base, uint32_t mask)`

Disables interrupts.

**Parameters**

- `base` – ACMP peripheral base address.
- `mask` – Interrupts mask. See “`_acmp_interrupt_enable`”.

`uint32_t ACMP_GetStatusFlags(CMP_Type *base)`

Gets status flags.

**Parameters**

- `base` – ACMP peripheral base address.

**Returns**

Status flags asserted mask. See “`_acmp_status_flags`”.

`void ACMP_ClearStatusFlags(CMP_Type *base, uint32_t mask)`

Clears status flags.

**Parameters**

- `base` – ACMP peripheral base address.
- `mask` – Status flags mask. See “`_acmp_status_flags`”.

`void ACMP_SetDiscreteModeConfig(CMP_Type *base, const acmp_discrete_mode_config_t *config)`

Configure the discrete mode.

Configure the discrete mode when supporting 3V domain with 1.8V core.

**Parameters**

- base – ACMP peripheral base address.
- config – Pointer to configuration structure. See “acmp\_discrete\_mode\_config\_t”.

void ACMP\_GetDefaultDiscreteModeConfig(*acmp\_discrete\_mode\_config\_t* \*config)

Get the default configuration for discrete mode setting.

#### Parameters

- config – Pointer to configuration structure to be restored with the setting values.

FSL\_ACMP\_DRIVER\_VERSION

ACMP driver version 2.3.0.

enum \_acmp\_interrupt\_enable

Interrupt enable/disable mask.

*Values:*

enumerator kACMP\_OutputRisingInterruptEnable

Enable the interrupt when comparator outputs rising.

enumerator kACMP\_OutputFallingInterruptEnable

Enable the interrupt when comparator outputs falling.

enumerator kACMP\_RoundRobinInterruptEnable

Enable the Round-Robin interrupt.

enum \_acmp\_status\_flags

Status flag mask.

*Values:*

enumerator kACMP\_OutputRisingEventFlag

Rising-edge on compare output has occurred.

enumerator kACMP\_OutputFallingEventFlag

Falling-edge on compare output has occurred.

enumerator kACMP\_OutputAssertEventFlag

Return the current value of the analog comparator output.

enum \_acmp\_offset\_mode

Comparator hard block offset control.

If OFFSET level is 1, then there is no hysteresis in the case of positive port input crossing negative port input in the positive direction (or negative port input crossing positive port input in the negative direction). Hysteresis still exists for positive port input crossing negative port input in the falling direction. If OFFSET level is 0, then the hysteresis selected by *acmp\_hysteresis\_mode\_t* is valid for both directions.

*Values:*

enumerator kACMP\_OffsetLevel0

The comparator hard block output has level 0 offset internally.

enumerator kACMP\_OffsetLevel1

The comparator hard block output has level 1 offset internally.

enum \_acmp\_hysteresis\_mode

Comparator hard block hysteresis control.

See chip data sheet to get the actual hysteresis value with each level.

*Values:*

enumerator kACMP\_HysteresisLevel0

Offset is level 0 and Hysteresis is level 0.

enumerator kACMP\_HysteresisLevel1

Offset is level 0 and Hysteresis is level 1.

enumerator kACMP\_HysteresisLevel2

Offset is level 0 and Hysteresis is level 2.

enumerator kACMP\_HysteresisLevel3

Offset is level 0 and Hysteresis is level 3.

enum \_acmp\_reference\_voltage\_source

CMP Voltage Reference source.

*Values:*

enumerator kACMP\_VrefSourceVin1

Vin1 is selected as resistor ladder network supply reference Vin.

enumerator kACMP\_VrefSourceVin2

Vin2 is selected as resistor ladder network supply reference Vin.

enum \_acmp\_port\_input

Port input source.

*Values:*

enumerator kACMP\_PortInputFromDAC

Port input from the 8-bit DAC output.

enumerator kACMP\_PortInputFromMux

Port input from the analog 8-1 mux.

enum \_acmp\_fixed\_port

Fixed mux port.

*Values:*

enumerator kACMP\_FixedPlusPort

Only the inputs to the Minus port are swept in each round.

enumerator kACMP\_FixedMinusPort

Only the inputs to the Plus port are swept in each round.

enum \_acmp\_dac\_work\_mode

Internal DAC's work mode.

*Values:*

enumerator kACMP\_DACWorkLowSpeedMode

DAC is selected to work in low speed and low power mode.

enumerator kACMP\_DACWorkHighSpeedMode

DAC is selected to work in high speed high power mode.

typedef enum \_acmp\_offset\_mode acmp\_offset\_mode\_t

Comparator hard block offset control.

If OFFSET level is 1, then there is no hysteresis in the case of positive port input crossing negative port input in the positive direction (or negative port input crossing positive port input in the negative direction). Hysteresis still exists for positive port input crossing negative port input in the falling direction. If OFFSET level is 0, then the hysteresis selected by acmp\_hysteresis\_mode\_t is valid for both directions.

typedef enum *\_acmp\_hysteresis\_mode* acmp\_hysteresis\_mode\_t

Comparator hard block hysteresis control.

See chip data sheet to get the actual hysteresis value with each level.

typedef enum *\_acmp\_reference\_voltage\_source* acmp\_reference\_voltage\_source\_t

CMP Voltage Reference source.

typedef enum *\_acmp\_port\_input* acmp\_port\_input\_t

Port input source.

typedef enum *\_acmp\_fixed\_port* acmp\_fixed\_port\_t

Fixed mux port.

typedef enum *\_acmp\_dac\_work\_mode* acmp\_dac\_work\_mode\_t

Internal DAC's work mode.

typedef struct *\_acmp\_config* acmp\_config\_t

Configuration for ACMP.

typedef struct *\_acmp\_channel\_config* acmp\_channel\_config\_t

Configuration for channel.

The comparator's port can be input from channel mux or DAC. If port input is from channel mux, detailed channel number for the mux should be configured.

typedef struct *\_acmp\_filter\_config* acmp\_filter\_config\_t

Configuration for filter.

typedef struct *\_acmp\_dac\_config* acmp\_dac\_config\_t

Configuration for DAC.

typedef struct *\_acmp\_round\_robin\_config* acmp\_round\_robin\_config\_t

Configuration for round robin mode.

typedef struct *\_acmp\_discrete\_mode\_config* acmp\_discrete\_mode\_config\_t

Configuration for discrete mode.

CMP\_C0\_CFX\_MASK

The mask of status flags cleared by writing 1.

CMP\_C1\_CHNn\_MASK

CMP\_C2\_CHnF\_MASK

struct *\_acmp\_config*

*#include <fsl\_acmp.h>* Configuration for ACMP.

### Public Members

*acmp\_offset\_mode\_t* offsetMode

Offset mode.

*acmp\_hysteresis\_mode\_t* hysteresisMode

Hysteresis mode.

bool enableHighSpeed

Enable High Speed (HS) comparison mode.

bool enableInvertOutput

Enable inverted comparator output.

bool useUnfilteredOutput

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

bool enablePinOut

The comparator output is available on the associated pin.

struct \_\_acmp\_channel\_config

*#include <fsl\_acmp.h>* Configuration for channel.

The comparator's port can be input from channel mux or DAC. If port input is from channel mux, detailed channel number for the mux should be configured.

### Public Members

*acmp\_port\_input\_t* positivePortInput

Input source of the comparator's positive port.

uint32\_t plusMuxInput

Plus mux input channel(0~7).

*acmp\_port\_input\_t* negativePortInput

Input source of the comparator's negative port.

uint32\_t minusMuxInput

Minus mux input channel(0~7).

struct \_\_acmp\_filter\_config

*#include <fsl\_acmp.h>* Configuration for filter.

### Public Members

bool enableSample

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

uint32\_t filterCount

Filter Sample Count. Available range is 1-7, 0 would cause the filter disabled.

uint32\_t filterPeriod

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

struct \_\_acmp\_dac\_config

*#include <fsl\_acmp.h>* Configuration for DAC.

### Public Members

*acmp\_reference\_voltage\_source\_t* referenceVoltageSource

Supply voltage reference source.

uint32\_t DACValue

Value for DAC Output Voltage. Available range is 0-255.

bool enableOutput

Enable the DAC output.

struct \_\_acmp\_round\_robin\_config

*#include <fsl\_acmp.h>* Configuration for round robin mode.



### Public Members

*acmp\_fixed\_port\_t* fixedPort

Fixed mux port.

uint32\_t fixedChannelNumber

Indicates which channel is fixed in the fixed mux port.

uint32\_t checkerChannelMask

Mask of checker channel index. Available range is channel0:0x01 to channel7:0x80 for round-robin checker.

uint32\_t sampleClockCount

Specifies how many round-robin clock cycles(0~3) later the sample takes place.

uint32\_t delayModulus

Comparator and DAC initialization delay modulus.

struct \_acmp\_discrete\_mode\_config

*#include <fsl\_acmp.h>* Configuration for discrete mode.

### Public Members

bool enablePositiveChannelDiscreteMode

Positive Channel Continuous Mode Enable. By default, the continuous mode is used.

bool enableNegativeChannelDiscreteMode

Negative Channel Continuous Mode Enable. By default, the continuous mode is used.

## 2.2 CACHE: CACHE Memory Controller

uint32\_t CACHE64\_GetInstance(CACHE64\_POLSEL\_Type \*base)

Returns an instance number given peripheral base address.

### Parameters

- base – The peripheral base address.

### Returns

CACHE64\_POLSEL instance number starting from 0.

uint32\_t CACHE64\_GetInstanceByAddr(uint32\_t address)

brief Returns an instance number given physical memory address.

param address The physical memory address.

### Returns

CACHE64\_CTRL instance number starting from 0.

status\_t CACHE64\_Init(CACHE64\_POLSEL\_Type \*base, const *cache64\_config\_t* \*config)

Initializes an CACHE64 instance with the user configuration structure.

This function configures the CACHE64 module with user-defined settings. Call the CACHE64\_GetDefaultConfig() function to configure the configuration structure and get the default configuration.

### Parameters

- base – CACHE64\_POLSEL peripheral base address.
- config – Pointer to a user-defined configuration structure.

**Return values**

kStatus\_Success – CACHE64 initialize succeed

void CACHE64\_GetDefaultConfig(*cache64\_config\_t* \*config)

Gets the default configuration structure.

This function initializes the CACHE64 configuration structure to a default value. The default values are first region covers whole cacheable area, and policy set to write back.

**Parameters**

- config – Pointer to a configuration structure.

void CACHE64\_EnableCache(CACHE64\_CTRL\_Type \*base)

Enables the cache.

**Parameters**

- base – CACHE64\_CTRL peripheral base address.

void CACHE64\_DisableCache(CACHE64\_CTRL\_Type \*base)

Disables the cache.

**Parameters**

- base – CACHE64\_CTRL peripheral base address.

void CACHE64\_InvalidateCache(CACHE64\_CTRL\_Type \*base)

Invalidates the cache.

**Parameters**

- base – CACHE64\_CTRL peripheral base address.

void CACHE64\_InvalidateCacheByRange(uint32\_t address, uint32\_t size\_byte)

Invalidates cache by range.

---

**Note:** Address and size should be aligned to “CACHE64\_LINESIZE\_BYTE”. The startAddr here will be forced to align to CACHE64\_LINESIZE\_BYTE if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

---

**Parameters**

- address – The physical address of cache.
- size\_byte – size of the memory to be invalidated, should be larger than 0.

void CACHE64\_CleanCache(CACHE64\_CTRL\_Type \*base)

Cleans the cache.

**Parameters**

- base – CACHE64\_CTRL peripheral base address.

void CACHE64\_CleanCacheByRange(uint32\_t address, uint32\_t size\_byte)

Cleans cache by range.

---

**Note:** Address and size should be aligned to “CACHE64\_LINESIZE\_BYTE”. The startAddr here will be forced to align to CACHE64\_LINESIZE\_BYTE if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

---

**Parameters**

- address – The physical address of cache.
- size\_byte – size of the memory to be cleaned, should be larger than 0.

void CACHE64\_CleanInvalidateCache(CACHE64\_CTRL\_Type \*base)

Cleans and invalidates the cache.

#### Parameters

- base – CACHE64\_CTRL peripheral base address.

void CACHE64\_CleanInvalidateCacheByRange(uint32\_t address, uint32\_t size\_byte)

Cleans and invalidate cache by range.

---

**Note:** Address and size should be aligned to “CACHE64\_LINESIZE\_BYTE”. The startAddr here will be forced to align to CACHE64\_LINESIZE\_BYTE if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

---

#### Parameters

- address – The physical address of cache.
- size\_byte – size of the memory to be Cleaned and Invalidated, should be larger than 0.

void CACHE64\_EnableWriteBuffer(CACHE64\_CTRL\_Type \*base, bool enable)

Enables/disables the write buffer.

#### Parameters

- base – CACHE64\_CTRL peripheral base address.
- enable – The enable or disable flag. true - enable the write buffer. false - disable the write buffer.

static inline void ICACHE\_InvalidateByRange(uint32\_t address, uint32\_t size\_byte)

Invalidates instruction cache by range.

---

**Note:** Address and size should be aligned to CACHE64\_LINESIZE\_BYTE due to the cache operation unit FSL\_FEATURE\_CACHE64\_CTRL\_LINESIZE\_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

---

#### Parameters

- address – The physical address.
- size\_byte – size of the memory to be invalidated, should be larger than 0.

static inline void DCACHE\_InvalidateByRange(uint32\_t address, uint32\_t size\_byte)

Invalidates data cache by range.

---

**Note:** Address and size should be aligned to CACHE64\_LINESIZE\_BYTE due to the cache operation unit FSL\_FEATURE\_CACHE64\_CTRL\_LINESIZE\_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

---

**Parameters**

- address – The physical address.
- size\_byte – size of the memory to be invalidated, should be larger than 0.

```
static inline void DCACHE_CleanByRange(uint32_t address, uint32_t size_byte)
```

Clean data cache by range.

---

**Note:** Address and size should be aligned to CACHE64\_LINESIZE\_BYTE due to the cache operation unit FSL\_FEATURE\_CACHE64\_CTRL\_LINESIZE\_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

---

**Parameters**

- address – The physical address.
- size\_byte – size of the memory to be cleaned, should be larger than 0.

```
static inline void DCACHE_CleanInvalidateByRange(uint32_t address, uint32_t size_byte)
```

Cleans and Invalidates data cache by range.

---

**Note:** Address and size should be aligned to CACHE64\_LINESIZE\_BYTE due to the cache operation unit FSL\_FEATURE\_CACHE64\_CTRL\_LINESIZE\_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

---

**Parameters**

- address – The physical address.
- size\_byte – size of the memory to be Cleaned and Invalidated, should be larger than 0.

```
FSL_CACHE_DRIVER_VERSION
```

cache driver version.

```
enum _cache64_policy
```

Level 2 cache controller way size.

*Values:*

```
enumerator kCACHE64_PolicyNonCacheable
```

Non-cacheable

```
enumerator kCACHE64_PolicyWriteThrough
```

Write through

```
enumerator kCACHE64_PolicyWriteBack
```

Write back

```
typedef enum _cache64_policy cache64_policy_t
```

Level 2 cache controller way size.

```
typedef struct _cache64_config cache64_config_t
```

CACHE64 configuration structure.

CACHE64\_LINESIZE\_BYTE

cache line size.

CACHE64\_REGION\_NUM

cache region number.

CACHE64\_REGION\_ALIGNMENT

cache region alignment.

struct \_\_cache64\_config

#include <fsl\_cache.h> CACHE64 configuration structure.

### Public Members

uint32\_t boundaryAddr[(3U) - 1]

< The cache controller can divide whole memory into 3 regions. Boundary address is the FlexSPI internal address (start from 0) instead of system address (start from FlexSPI AMBA base) to split adjacent regions and must be 1KB aligned. The boundary address itself locates in upper region. Cacheable policy for each region.

## 2.3 CASPER: The Cryptographic Accelerator and Signal Processing Engine with RAM sharing

### 2.4 casper\_driver

FSL\_CASPER\_DRIVER\_VERSION

CASPER driver version. Version 2.2.4.

Current version: 2.2.4

Change log:

- Version 2.0.0
  - Initial version
- Version 2.0.1
  - Bug fix KPSDK-24531 double\_scalar\_multiplication() result may be all zeroes for some specific input
- Version 2.0.2
  - Bug fix KPSDK-25015 CASPER\_MEMCPY hard-fault on LPC55xx when both source and destination buffers are outside of CASPER\_RAM
- Version 2.0.3
  - Bug fix KPSDK-28107 RSUB, FILL and ZERO operations not implemented in enum\_casper\_operation.
- Version 2.0.4
  - For GCC compiler, enforce O1 optimize level, specifically to remove strict-aliasing option. This driver is very specific and requires -fno-strict-aliasing.
- Version 2.0.5
  - Fix sign-compare warning.
- Version 2.0.6

- Fix IAR Pa082 warning.
- Version 2.0.7
  - Fix MISRA-C 2012 issue.
- Version 2.0.8
  - Add feature macro for CASPER\_RAM\_OFFSET.
- Version 2.0.9
  - Remove unused function Jac\_oncurve().
  - Fix ECC384 build.
- Version 2.0.10
  - Fix MISRA-C 2012 issue.
- Version 2.1.0
  - Add ECC NIST P-521 elliptic curve.
- Version 2.2.0
  - Rework driver to support multiple curves at once.
- Version 2.2.1
  - Fix MISRA-C 2012 issue.
- Version 2.2.2
  - Enable hardware interleaving to RAMX0 and RAMX1 for CASPER by feature macro FSL\_FEATURE\_CASPER\_RAM\_HW\_INTERLEAVE
- Version 2.2.3
  - Added macro into CASPER\_Init and CASPER\_Deinit to support devices without clock and reset control.
- Version 2.2.4
  - Fix MISRA-C 2012 issue.

enum \_casper\_operation  
CASPER operation.

*Values:*

enumerator kCASPER\_OpMul6464NoSum

enumerator kCASPER\_OpMul6464Sum

Walking 1 or more of J loop, doing  $r=a*b$  using  $64x64=128$

enumerator kCASPER\_OpMul6464FullSum

Walking 1 or more of J loop, doing  $c,r=r+a*b$  using  $64x64=128$ , but assume inner j loop

enumerator kCASPER\_OpMul6464Reduce

Walking 1 or more of J loop, doing  $c,r=r+a*b$  using  $64x64=128$ , but sum all of w.

enumerator kCASPER\_OpAdd64

Walking 1 or more of J loop, doing  $c,r[-1]=r+a*b$  using  $64x64=128$ , but skip 1st write

enumerator kCASPER\_OpSub64

Walking add with off\_AB, and in/out off\_RES doing  $c,r=r+a+c$  using  $64+64=65$

enumerator kCASPER\_OpDouble64

Walking subtract with off\_AB, and in/out off\_RES doing  $r=r-a$  using  $64-64=64$ , with last borrow implicit if any

enumerator kCASPER\_OpXor64

Walking add to self with off\_RES doing  $c, r = r + r + c$  using  $64 + 64 = 65$

enumerator kCASPER\_OpRSub64

Walking XOR with off\_AB, and in/out off\_RES doing  $r = r \wedge a$  using  $64 \wedge 64 = 64$

enumerator kCASPER\_OpShiftLeft32

Walking subtract with off\_AB, and in/out off\_RES using  $r = a - r$

enumerator kCASPER\_OpShiftRight32

Walking shift left doing  $r1, r = (b * D) | r1$ , where D is  $2^{\text{amt}}$  and is loaded by app (off\_CD not used)

enumerator kCASPER\_OpCopy

Walking shift right doing  $r, r1 = (b * D) | r1$ , where D is  $2^{(32 - \text{amt})}$  and is loaded by app (off\_CD not used) and off\_RES starts at MSW

enumerator kCASPER\_OpRemask

Copy from ABoff to resoff, 64b at a time

enumerator kCASPER\_OpFill

Copy and mask from ABoff to resoff, 64b at a time

enumerator kCASPER\_OpZero

Fill RESOFF using 64 bits at a time with value in A and B

enumerator kCASPER\_OpCompare

Fill RESOFF using 64 bits at a time of 0s

enumerator kCASPER\_OpCompareFast

Compare two arrays, running all the way to the end

enum \_casper\_algo\_t

Algorithm used for CASPER operation.

*Values:*

enumerator kCASPER\_ECC\_P256

ECC\_P256

enumerator kCASPER\_ECC\_P384

ECC\_P384

enumerator kCASPER\_ECC\_P521

ECC\_P521

*Values:*

enumerator kCASPER\_RamOffset\_Result

enumerator kCASPER\_RamOffset\_Base

enumerator kCASPER\_RamOffset\_TempBase

enumerator kCASPER\_RamOffset\_Modulus

enumerator kCASPER\_RamOffset\_M64

typedef enum \_casper\_operation casper\_operation\_t

CASPER operation.

typedef enum \_casper\_algo\_t casper\_algo\_t

Algorithm used for CASPER operation.

`void CASPER_Init(CASPER_Type *base)`

Enables clock and disables reset for CASPER peripheral.

Enable clock and disable reset for CASPER.

**Parameters**

- `base` – CASPER base address

`void CASPER_Deinit(CASPER_Type *base)`

Disables clock for CASPER peripheral.

Disable clock and enable reset.

**Parameters**

- `base` – CASPER base address

`CASPER_CP`

`CASPER_CP_CTRL0`

`CASPER_CP_CTRL1`

`CASPER_CP_LOADER`

`CASPER_CP_STATUS`

`CASPER_CP_INTENSET`

`CASPER_CP_INTENCLR`

`CASPER_CP_INTSTAT`

`CASPER_CP_AREG`

`CASPER_CP_BREG`

`CASPER_CP_CREG`

`CASPER_CP_DREG`

`CASPER_CP_RES0`

`CASPER_CP_RES1`

`CASPER_CP_RES2`

`CASPER_CP_RES3`

`CASPER_CP_MASK`

`CASPER_CP_REMASK`

`CASPER_CP_LOCK`

`CASPER_CP_ID`

`CASPER_Wr32b(value, off)`

`CASPER_Wr64b(value, off)`

`CASPER_Rd32b(off)`

`N_wordlen_max`



## 2.5 casper\_driver\_pkha

```
void CASPER_ModExp(CASPER_Type *base, const uint8_t *signature, const uint8_t *pubN,
                  size_t wordLen, uint32_t pubE, uint8_t *plaintext)
```

Performs modular exponentiation -  $(A^E) \bmod N$ .

This function performs modular exponentiation.

### Parameters

- base – CASPER base address
- signature – first addend (in little endian format)
- pubN – modulus (in little endian format)
- wordLen – Size of pubN in bytes
- pubE – exponent
- plaintext – **[out]** Output array to store result of operation (in little endian format)

```
void CASPER_ecc_init(casper_algo_t curve)
```

Initialize prime modulus mod in Casper memory .

Set the prime modulus mod in Casper memory and set N\_wordlen according to selected algorithm.

### Parameters

- curve – elliptic curve algorithm

```
void CASPER_ECC_SECP256R1_Mul(CASPER_Type *base, uint32_t resX[8], uint32_t resY[8],
                              uint32_t X[8], uint32_t Y[8], uint32_t scalar[8])
```

Performs ECC secp256r1 point single scalar multiplication.

This function performs ECC secp256r1 point single scalar multiplication  $[resX; resY] = scalar * [X; Y]$  Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32\_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

### Parameters

- base – CASPER base address
- resX – **[out]** Output X affine coordinate in normal form, little endian.
- resY – **[out]** Output Y affine coordinate in normal form, little endian.
- X – Input X affine coordinate in normal form, little endian.
- Y – Input Y affine coordinate in normal form, little endian.
- scalar – Input scalar integer, in normal form, little endian.

```
void CASPER_ECC_SECP256R1_MulAdd(CASPER_Type *base, uint32_t resX[8], uint32_t
                                resY[8], uint32_t X1[8], uint32_t Y1[8], uint32_t
                                scalar1[8], uint32_t X2[8], uint32_t Y2[8], uint32_t
                                scalar2[8])
```

Performs ECC secp256r1 point double scalar multiplication.

This function performs ECC secp256r1 point double scalar multiplication  $[resX; resY] = scalar1 * [X1; Y1] + scalar2 * [X2; Y2]$  Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32\_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

### Parameters

- base – CASPER base address
- resX – **[out]** Output X affine coordinate.
- resY – **[out]** Output Y affine coordinate.
- X1 – Input X1 affine coordinate.
- Y1 – Input Y1 affine coordinate.
- scalar1 – Input scalar1 integer.
- X2 – Input X2 affine coordinate.
- Y2 – Input Y2 affine coordinate.
- scalar2 – Input scalar2 integer.

```
void CASPER_ECC_SECP384R1_Mul(CASPER_Type *base, uint32_t resX[12], uint32_t resY[12],  
                             uint32_t X[12], uint32_t Y[12], uint32_t scalar[12])
```

Performs ECC secp384r1 point single scalar multiplication.

This function performs ECC secp384r1 point single scalar multiplication [resX; resY] = scalar \* [X; Y] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32\_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

#### Parameters

- base – CASPER base address
- resX – **[out]** Output X affine coordinate in normal form, little endian.
- resY – **[out]** Output Y affine coordinate in normal form, little endian.
- X – Input X affine coordinate in normal form, little endian.
- Y – Input Y affine coordinate in normal form, little endian.
- scalar – Input scalar integer, in normal form, little endian.

```
void CASPER_ECC_SECP384R1_MulAdd(CASPER_Type *base, uint32_t resX[12], uint32_t  
                                resY[12], uint32_t X1[12], uint32_t Y1[12], uint32_t  
                                scalar1[12], uint32_t X2[12], uint32_t Y2[12], uint32_t  
                                scalar2[12])
```

Performs ECC secp384r1 point double scalar multiplication.

This function performs ECC secp384r1 point double scalar multiplication [resX; resY] = scalar1 \* [X1; Y1] + scalar2 \* [X2; Y2] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32\_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

#### Parameters

- base – CASPER base address
- resX – **[out]** Output X affine coordinate.
- resY – **[out]** Output Y affine coordinate.
- X1 – Input X1 affine coordinate.
- Y1 – Input Y1 affine coordinate.
- scalar1 – Input scalar1 integer.
- X2 – Input X2 affine coordinate.
- Y2 – Input Y2 affine coordinate.
- scalar2 – Input scalar2 integer.

```
void CASPER_ECC_SECP521R1_Mul(CASPER_Type *base, uint32_t resX[18], uint32_t resY[18],
                               uint32_t X[18], uint32_t Y[18], uint32_t scalar[18])
```

Performs ECC secp521r1 point single scalar multiplication.

This function performs ECC secp521r1 point single scalar multiplication  $[resX; resY] = scalar * [X; Y]$  Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32\_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

#### Parameters

- base – CASPER base address
- resX – **[out]** Output X affine coordinate in normal form, little endian.
- resY – **[out]** Output Y affine coordinate in normal form, little endian.
- X – Input X affine coordinate in normal form, little endian.
- Y – Input Y affine coordinate in normal form, little endian.
- scalar – Input scalar integer, in normal form, little endian.

```
void CASPER_ECC_SECP521R1_MulAdd(CASPER_Type *base, uint32_t resX[18], uint32_t
                                   resY[18], uint32_t X1[18], uint32_t Y1[18], uint32_t
                                   scalar1[18], uint32_t X2[18], uint32_t Y2[18], uint32_t
                                   scalar2[18])
```

Performs ECC secp521r1 point double scalar multiplication.

This function performs ECC secp521r1 point double scalar multiplication  $[resX; resY] = scalar1 * [X1; Y1] + scalar2 * [X2; Y2]$  Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32\_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

#### Parameters

- base – CASPER base address
- resX – **[out]** Output X affine coordinate.
- resY – **[out]** Output Y affine coordinate.
- X1 – Input X1 affine coordinate.
- Y1 – Input Y1 affine coordinate.
- scalar1 – Input scalar1 integer.
- X2 – Input X2 affine coordinate.
- Y2 – Input Y2 affine coordinate.
- scalar2 – Input scalar2 integer.

```
void CASPER_ECC_equal(int *res, uint32_t *op1, uint32_t *op2)
```

```
void CASPER_ECC_equal_to_zero(int *res, uint32_t *op1)
```

## 2.6 Clock Driver

```
enum __clock_ip_name
```

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

*Values:*

enumerator kCLOCK\_IpInvalid  
Invalid Ip Name.

enumerator kCLOCK\_RomCtrlr  
Clock gate name: RomCtrlr

enumerator kCLOCK\_PowerQuad  
Clock gate name: PowerQuad

enumerator kCLOCK\_Casper  
Clock gate name: Casper

enumerator kCLOCK\_HashCrypt  
Clock gate name: HashCrypt

enumerator kCLOCK\_Puf  
Clock gate name: Puf

enumerator kCLOCK\_Rng  
Clock gate name: Rng

enumerator kCLOCK\_Flexspi  
Clock gate name: Flexspi

enumerator kCLOCK\_OtpCtrl  
Clock gate name: OtpCtrl

enumerator kCLOCK\_UsbhsPhy  
Clock gate name: UsbhsPhy

enumerator kCLOCK\_UsbhsDevice  
Clock gate name: UsbhsDevice

enumerator kCLOCK\_UsbhsHost  
Clock gate name: UsbhsHost

enumerator kCLOCK\_UsbhsSram  
Clock gate name: UsbhsSram

enumerator kCLOCK\_Sct  
Clock gate name: Sct

enumerator kCLOCK\_Sdio0  
Clock gate name: Sdio0

enumerator kCLOCK\_Sdio1  
Clock gate name: Sdio1

enumerator kCLOCK\_Acmp0  
Clock gate name: Acmp0

enumerator kCLOCK\_Adc0  
Clock gate name: Adc0

enumerator kCLOCK\_ShsGpio0  
Clock gate name: ShsGpio0

enumerator kCLOCK\_Utick0  
Clock gate name: Utick0

enumerator kCLOCK\_Wwdt0  
Clock gate name: Wwdt0

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\_Flexcomm14  
Clock gate name: Flexcomm14

enumerator kCLOCK\_Flexcomm15  
Clock gate name: Flexcomm15

enumerator kCLOCK\_Usart0  
Clock gate name: Usart0

enumerator kCLOCK\_Usart1  
Clock gate name: Usart1

enumerator kCLOCK\_Usart2  
Clock gate name: Usart2

enumerator kCLOCK\_Usart3  
Clock gate name: Usart3

enumerator kCLOCK\_Usart4  
Clock gate name: Usart4

enumerator kCLOCK\_Usart5  
Clock gate name: Usart5

enumerator kCLOCK\_Usart6  
Clock gate name: Usart6

enumerator kCLOCK\_Usart7  
Clock gate name: Usart7

enumerator kCLOCK\_I2s0  
Clock gate name: I2s0

enumerator kCLOCK\_I2s1  
Clock gate name: I2s1

enumerator kCLOCK\_I2s2  
Clock gate name: I2s2

enumerator kCLOCK\_I2s3  
Clock gate name: I2s3

enumerator kCLOCK\_I2s4  
Clock gate name: I2s4

enumerator kCLOCK\_I2s5  
Clock gate name: I2s5

enumerator kCLOCK\_I2s6  
Clock gate name: I2s6

enumerator kCLOCK\_I2s7  
Clock gate name: I2s7

enumerator kCLOCK\_I2c0  
Clock gate name: I2c0

enumerator kCLOCK\_I2c1  
Clock gate name: I2c1

enumerator kCLOCK\_I2c2  
Clock gate name: I2c2

enumerator kCLOCK\_I2c3  
Clock gate name: I2c3

enumerator kCLOCK\_I2c4  
Clock gate name: I2c4

enumerator kCLOCK\_I2c5  
Clock gate name: I2c5

enumerator kCLOCK\_I2c6  
Clock gate name: I2c6

enumerator kCLOCK\_I2c7  
Clock gate name: I2c7

enumerator kCLOCK\_I2c15  
Clock gate name: I2c15

enumerator kCLOCK\_Spi0  
Clock gate name: Spi0

enumerator kCLOCK\_Spi1  
Clock gate name: Spi1

enumerator kCLOCK\_Spi2  
Clock gate name: Spi2

enumerator kCLOCK\_Spi3  
Clock gate name: Spi3

enumerator kCLOCK\_Spi4  
Clock gate name: Spi4

enumerator kCLOCK\_Spi5  
Clock gate name: Spi5

enumerator kCLOCK\_Spi6  
Clock gate name: Spi6

enumerator kCLOCK\_Spi7  
Clock gate name: Spi7

enumerator kCLOCK\_Spi14  
Clock gate name: Spi14

enumerator kCLOCK\_Dmic0  
Clock gate name: Dmic0

enumerator kCLOCK\_OsEventTimer  
Clock gate name: OsEventTimer

enumerator kCLOCK\_HsGpio0  
Clock gate name: HsGpio0

enumerator kCLOCK\_HsGpio1  
Clock gate name: HsGpio1

enumerator kCLOCK\_HsGpio2  
Clock gate name: HsGpio2

enumerator kCLOCK\_HsGpio3  
Clock gate name: HsGpio3

enumerator kCLOCK\_HsGpio4  
Clock gate name: HsGpio4

enumerator kCLOCK\_HsGpio5  
Clock gate name: HsGpio5

enumerator kCLOCK\_HsGpio6  
Clock gate name: HsGpio6

enumerator kCLOCK\_HsGpio7  
Clock gate name: HsGpio7

enumerator kCLOCK\_Crc  
Clock gate name: Crc

enumerator kCLOCK\_Dmac0  
Clock gate name: Dmac0

enumerator kCLOCK\_Dmac1  
Clock gate name: Dmac1

enumerator kCLOCK\_Mu  
Clock gate name: Mu

enumerator kCLOCK\_Sema  
Clock gate name: Sema

enumerator kCLOCK\_Freqme  
Clock gate name: Freqme

enumerator kCLOCK\_Ct32b0  
Clock gate name: Ct32b0

enumerator kCLOCK\_Ct32b1  
Clock gate name: Ct32b1

enumerator kCLOCK\_Ct32b2  
Clock gate name: Ct32b2

enumerator kCLOCK\_Ct32b3  
Clock gate name: Ct32b3

enumerator kCLOCK\_Ct32b4  
Clock gate name: Ct32b4

enumerator kCLOCK\_Rtc  
Clock gate name: Rtc

enumerator kCLOCK\_Mrt0  
Clock gate name: Mrt0

enumerator kCLOCK\_Wwdt1  
Clock gate name: Wwdt1

enumerator kCLOCK\_I3c0  
Clock gate name: I3c0

enumerator kCLOCK\_Pint  
Clock gate name: Pint

enumerator kCLOCK\_InputMux  
Clock gate name: Input Mux.

enum \_clock\_name

Clock name used to get clock frequency.

*Values:*

enumerator kCLOCK\_CoreSysClk  
Core clock (aka HCLK)

enumerator kCLOCK\_BusClk  
Bus clock (AHB/APB clock, aka HCLK)

enumerator kCLOCK\_MclkClk  
MCLK, to MCLK pin

enumerator kCLOCK\_ClockOutClk  
CLOCKOUT

enumerator kCLOCK\_AdcClk  
ADC

enumerator kCLOCK\_FlexspiClk  
FLEXSPI

enumerator kCLOCK\_SctClk  
SCT

enumerator kCLOCK\_Wdt0Clk  
Watchdog0

enumerator kCLOCK\_Wdt1Clk  
Watchdog1

enumerator kCLOCK\_SystickClk  
Systick

enumerator kCLOCK\_Sdio0Clk  
SDIO0



enumerator kCLOCK\_Sdio1Clk  
SDIO1

enumerator kCLOCK\_I3cClk  
I3C

enumerator kCLOCK\_UsbClk  
USB

enumerator kCLOCK\_DmicClk  
Digital Mic clock

enumerator kCLOCK\_DspCpuClk  
DSP clock

enumerator kCLOCK\_AcmpClk  
Acmp clock

enumerator kCLOCK\_Flexcomm0Clk  
Flexcomm0Clock

enumerator kCLOCK\_Flexcomm1Clk  
Flexcomm1Clock

enumerator kCLOCK\_Flexcomm2Clk  
Flexcomm2Clock

enumerator kCLOCK\_Flexcomm3Clk  
Flexcomm3Clock

enumerator kCLOCK\_Flexcomm4Clk  
Flexcomm4Clock

enumerator kCLOCK\_Flexcomm5Clk  
Flexcomm5Clock

enumerator kCLOCK\_Flexcomm6Clk  
Flexcomm6Clock

enumerator kCLOCK\_Flexcomm7Clk  
Flexcomm7Clock

enumerator kCLOCK\_Flexcomm14Clk  
Flexcomm14Clock

enumerator kCLOCK\_Flexcomm15Clk  
Flexcomm15Clock

enum \_clock\_pfd  
PLL PFD clock name.

*Values:*

enumerator kCLOCK\_Pfd0  
PLL PFD0

enumerator kCLOCK\_Pfd1  
PLL PFD1

enumerator kCLOCK\_Pfd2  
PLL PFD2

enumerator kCLOCK\_Pfd3

PLL PFD3

enum \_clock\_attach\_id

The enumerator of clock attach Id.

*Values:*

enumerator kSFRO\_to\_SYS\_PLL

Attach SFRO to SYS\_PLL.

enumerator kXTALIN\_CLK\_to\_SYS\_PLL

Attach XTALIN\_CLK to SYS\_PLL.

enumerator kFFRO\_DIV2\_to\_SYS\_PLL

Attach FFRO\_DIV2 to SYS\_PLL.

enumerator kNONE\_to\_SYS\_PLL

Attach NONE to SYS\_PLL.

enumerator kSFRO\_to\_AUDIO\_PLL

Attach SFRO to AUDIO\_PLL.

enumerator kXTALIN\_CLK\_to\_AUDIO\_PLL

Attach XTALIN\_CLK to AUDIO\_PLL.

enumerator kFFRO\_DIV2\_to\_AUDIO\_PLL

Attach FFRO\_DIV2 to AUDIO\_PLL.

enumerator kNONE\_to\_AUDIO\_PLL

Attach NONE to AUDIO\_PLL.

enumerator kFFRO\_DIV4\_to\_MAIN\_CLK

Attach FFRO\_DIV4 to MAIN\_CLK.

enumerator kXTALIN\_CLK\_to\_MAIN\_CLK

Attach XTALIN\_CLK to MAIN\_CLK.

enumerator kLPOSC\_to\_MAIN\_CLK

Attach LPOSC to MAIN\_CLK.

enumerator kFFRO\_to\_MAIN\_CLK

Attach FFRO to MAIN\_CLK.

enumerator kSFRO\_to\_MAIN\_CLK

Attach SFRO to MAIN\_CLK.

enumerator kMAIN\_PLL\_to\_MAIN\_CLK

Attach MAIN\_PLL to MAIN\_CLK.

enumerator kOSC32K\_to\_MAIN\_CLK

Attach OSC32K to MAIN\_CLK.

enumerator kFFRO\_to\_DSP\_MAIN\_CLK

Attach FFRO to DSP\_MAIN\_CLK.

enumerator kXTALIN\_CLK\_to\_DSP\_MAIN\_CLK

Attach XTALIN\_CLK to DSP\_MAIN\_CLK.

enumerator kLPOSC\_to\_DSP\_MAIN\_CLK

Attach LPOSC to DSP\_MAIN\_CLK.

enumerator kSFRO\_to\_DSP\_MAIN\_CLK  
Attach SFRO to DSP\_MAIN\_CLK.

enumerator kMAIN\_PLL\_to\_DSP\_MAIN\_CLK  
Attach MAIN\_PLL to DSP\_MAIN\_CLK.

enumerator kDSP\_PLL\_to\_DSP\_MAIN\_CLK  
Attach DSP\_PLL to DSP\_MAIN\_CLK.

enumerator kOSC32K\_to\_DSP\_MAIN\_CLK  
Attach OSC32K to DSP\_MAIN\_CLK.

enumerator kSFRO\_to\_ADC\_CLK  
Attach SFRO to ADC\_CLK.

enumerator kXTALIN\_CLK\_to\_ADC\_CLK  
Attach XTALIN\_CLK to ADC\_CLK.

enumerator kLPOSC\_to\_ADC\_CLK  
Attach LPOSC to ADC\_CLK.

enumerator kFFRO\_to\_ADC\_CLK  
Attach FFRO to ADC\_CLK.

enumerator kMAIN\_PLL\_to\_ADC\_CLK  
Attach MAIN\_PLL to ADC\_CLK.

enumerator kAUX0\_PLL\_to\_ADC\_CLK  
Attach AUX0\_PLL to ADC\_CLK.

enumerator kAUX1\_PLL\_to\_ADC\_CLK  
Attach AUX1\_PLL to ADC\_CLK.

enumerator kSFRO\_to\_CLKOUT  
Attach SFRO to CLKOUT.

enumerator kXTALIN\_CLK\_to\_CLKOUT  
Attach XTALIN\_CLK to CLKOUT.

enumerator kLPOSC\_to\_CLKOUT  
Attach LPOSC to CLKOUT.

enumerator kFFRO\_to\_CLKOUT  
Attach FFRO to CLKOUT.

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

enumerator kDSP\_MAIN\_to\_CLKOUT  
Attach DSP\_MAIN to CLKOUT.

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

enumerator kAUX0\_PLL\_to\_CLKOUT  
Attach AUX0\_PLL to CLKOUT.

enumerator kDSP\_PLL\_to\_CLKOUT  
Attach DSP\_PLL to CLKOUT.

enumerator kAUX1\_PLL\_to\_CLKOUT  
Attach AUX1\_PLL to CLKOUT.

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

enumerator kOSC32K\_to\_CLKOUT  
Attach OSC32K to CLKOUT.

enumerator kNONE\_to\_CLKOUT  
Attach NONE to CLKOUT.

enumerator kMAIN\_CLK\_to\_I3C\_CLK  
Attach MAIN\_CLK to I3C\_CLK.

enumerator kFFRO\_to\_I3C\_CLK  
Attach FFRO to I3C\_CLK.

enumerator kNONE\_to\_I3C\_CLK  
Attach NONE to I3C\_CLK.

enumerator kI3C\_CLK\_to\_I3C\_TC\_CLK  
Attach I3C\_CLK to I3C\_TC\_CLK.

enumerator kLPOSC\_to\_I3C\_TC\_CLK  
Attach LPOSC to I3C\_TC\_CLK.

enumerator kNONE\_to\_I3C\_TC\_CLK  
Attach NONE to I3C\_TC\_CLK.

enumerator kLPOSC\_to\_OSTIMER\_CLK  
Attach LPOSC to OSTIMER\_CLK.

enumerator kOSC32K\_to\_OSTIMER\_CLK  
Attach OSC32K to OSTIMER\_CLK.

enumerator kHCLK\_to\_OSTIMER\_CLK  
Attach HCLK to OSTIMER\_CLK.

enumerator kNONE\_to\_OSTIMER\_CLK  
Attach NONE to OSTIMER\_CLK.

enumerator kMAIN\_CLK\_to\_FLEXSPI\_CLK  
Attach MAIN\_CLK to FLEXSPI\_CLK.

enumerator kMAIN\_PLL\_to\_FLEXSPI\_CLK  
Attach MAIN\_PLL to FLEXSPI\_CLK.

enumerator kAUX0\_PLL\_to\_FLEXSPI\_CLK  
Attach AUX0\_PLL to FLEXSPI\_CLK.

enumerator kFFRO\_to\_FLEXSPI\_CLK  
Attach FFRO to FLEXSPI\_CLK.

enumerator kAUX1\_PLL\_to\_FLEXSPI\_CLK  
Attach AUX1\_PLL to FLEXSPI\_CLK.

enumerator kNONE\_to\_FLEXSPI\_CLK  
Attach NONE to FLEXSPI\_CLK.

enumerator kMAIN\_CLK\_to\_SCT\_CLK  
Attach MAIN\_CLK to SCT\_CLK.

enumerator kMAIN\_PLL\_to\_SCT\_CLK  
Attach MAIN\_PLL to SCT\_CLK.

enumerator kAUX0\_PLL\_to\_SCT\_CLK  
Attach AUX0\_PLL to SCT\_CLK.

enumerator kFFRO\_to\_SCT\_CLK  
Attach FFRO to SCT\_CLK.

enumerator kAUX1\_PLL\_to\_SCT\_CLK  
Attach AUX1\_PLL 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 kLPOSC\_to\_UTICK\_CLK  
Attach LPOSC to UTICK\_CLK.

enumerator kNONE\_to\_UTICK\_CLK  
Attach NONE to UTICK\_CLK.

enumerator kLPOSC\_to\_WDT0\_CLK  
Attach LPOSC to WDT0\_CLK.

enumerator kNONE\_to\_WDT0\_CLK  
Attach NONE to WDT0\_CLK.

enumerator kLPOSC\_to\_WDT1\_CLK  
Attach LPOSC to WDT1\_CLK.

enumerator kNONE\_to\_WDT1\_CLK  
Attach NONE to WDT1\_CLK.

enumerator kOSC32K\_to\_32KHZWAKE\_CLK  
Attach OSC32K to 32KHZWAKE\_CLK.

enumerator kLPOSC\_DIV32\_to\_32KHZWAKE\_CLK  
Attach LPOSC\_DIV32 to 32KHZWAKE\_CLK.

enumerator kNONE\_to\_32KHZWAKE\_CLK  
Attach NONE to 32KHZWAKE\_CLK.

enumerator kMAIN\_CLK\_DIV\_to\_SYSTICK\_CLK  
Attach MAIN\_CLK\_DIV to SYSTICK\_CLK.

enumerator kLPOSC\_to\_SYSTICK\_CLK  
Attach LPOSC to SYSTICK\_CLK.

enumerator kOSC32K\_to\_SYSTICK\_CLK  
Attach OSC32K to SYSTICK\_CLK.

enumerator kSFRO\_to\_SYSTICK\_CLK  
Attach SFRO to SYSTICK\_CLK.

enumerator kNONE\_to\_SYSTICK\_CLK  
Attach NONE to SYSTICK\_CLK.

enumerator kMAIN\_CLK\_to\_SDIO0\_CLK  
Attach MAIN\_CLK to SDIO0\_CLK.

enumerator kMAIN\_PLL\_to\_SDIO0\_CLK  
Attach MAIN\_PLL to SDIO0\_CLK.

enumerator kAUX0\_PLL\_to\_SDIO0\_CLK

Attach AUX0\_PLL to SDIO0\_CLK.

enumerator kFFRO\_to\_SDIO0\_CLK

Attach FFRO to SDIO0\_CLK.

enumerator kAUX1\_PLL\_to\_SDIO0\_CLK

Attach AUX1\_PLL to SDIO0\_CLK.

enumerator kNONE\_to\_SDIO0\_CLK

Attach NONE to SDIO0\_CLK.

enumerator kMAIN\_CLK\_to\_SDIO1\_CLK

Attach MAIN\_CLK to SDIO1\_CLK.

enumerator kMAIN\_PLL\_to\_SDIO1\_CLK

Attach MAIN\_PLL to SDIO1\_CLK.

enumerator kAUX0\_PLL\_to\_SDIO1\_CLK

Attach AUX0\_PLL to SDIO1\_CLK.

enumerator kFFRO\_to\_SDIO1\_CLK

Attach FFRO to SDIO1\_CLK.

enumerator kAUX1\_PLL\_to\_SDIO1\_CLK

Attach AUX1\_PLL to SDIO1\_CLK.

enumerator kNONE\_to\_SDIO1\_CLK

Attach NONE to SDIO1\_CLK.

enumerator kFFRO\_to\_ESPI\_CLK

Attach FFRO to ESPI\_CLK.

enumerator kNONE\_to\_ESPI\_CLK

Attach NONE to ESPI\_CLK.

enumerator kXTALIN\_CLK\_to\_USB\_CLK

Attach XTALIN\_CLK to USB\_CLK.

enumerator kMAIN\_CLK\_to\_USB\_CLK

Attach MAIN\_CLK to USB\_CLK.

enumerator kNONE\_to\_USB\_CLK

Attach NONE to USB\_CLK.

enumerator kFFRO\_to\_MCLK\_CLK

Attach FFRO to MCLK\_CLK.

enumerator kAUDIO\_PLL\_to\_MCLK\_CLK

Attach AUDIO\_PLL to MCLK\_CLK.

enumerator kNONE\_to\_MCLK\_CLK

Attach NONE to MCLK\_CLK.

enumerator kSFRO\_to\_DMIC\_CLK

Attach SFRO to DMIC\_CLK.

enumerator kFFRO\_to\_DMIC\_CLK

Attach FFRO to DMIC\_CLK.

enumerator kAUDIO\_PLL\_to\_DMIC\_CLK

Attach AUDIO\_PLL to DMIC\_CLK.

enumerator kMASTER\_CLK\_to\_DMIC\_CLK  
Attach MASTER\_CLK to DMIC\_CLK.

enumerator kLPOSC\_to\_DMIC\_CLK  
Attach LPOSC to DMIC\_CLK.

enumerator k32K\_WAKE\_CLK\_to\_DMIC\_CLK  
Attach 32K\_WAKE\_CLK to DMIC\_CLK.

enumerator kNONE\_to\_DMIC\_CLK  
Attach NONE to DMIC\_CLK.

enumerator kMAIN\_CLK\_to\_ACMP\_CLK  
Attach MAIN\_CLK to ACMP\_CLK.

enumerator kSFRO\_to\_ACMP\_CLK  
Attach SFRO to ACMP\_CLK.

enumerator kFFRO\_to\_ACMP\_CLK  
Attach FFRO to ACMP\_CLK.

enumerator kAUX0\_PLL\_to\_ACMP\_CLK  
Attach AUX0\_PLL to ACMP\_CLK.

enumerator kAUX1\_PLL\_to\_ACMP\_CLK  
Attach AUX1\_PLL to ACMP\_CLK.

enumerator kNONE\_to\_ACMP\_CLK  
Attach NONE to ACMP\_CLK.

enumerator kSFRO\_to\_FLEXCOMM0  
Attach SFRO to FLEXCOMM0.

enumerator kFFRO\_to\_FLEXCOMM0  
Attach FFRO to FLEXCOMM0.

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

enumerator kMASTER\_CLK\_to\_FLEXCOMM0  
Attach MASTER\_CLK to FLEXCOMM0.

enumerator kFRG\_to\_FLEXCOMM0  
Attach FRG to FLEXCOMM0.

enumerator kNONE\_to\_FLEXCOMM0  
Attach NONE to FLEXCOMM0.

enumerator kSFRO\_to\_FLEXCOMM1  
Attach SFRO to FLEXCOMM1.

enumerator kFFRO\_to\_FLEXCOMM1  
Attach FFRO to FLEXCOMM1.

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

enumerator kMASTER\_CLK\_to\_FLEXCOMM1  
Attach MASTER\_CLK to FLEXCOMM1.

enumerator kFRG\_to\_FLEXCOMM1  
Attach FRG to FLEXCOMM1.

enumerator kNONE\_to\_FLEXCOMM1  
Attach NONE to FLEXCOMM1.

enumerator kSFRO\_to\_FLEXCOMM2  
Attach SFRO to FLEXCOMM2.

enumerator kFFRO\_to\_FLEXCOMM2  
Attach FFRO to FLEXCOMM2.

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

enumerator kMASTER\_CLK\_to\_FLEXCOMM2  
Attach MASTER\_CLK to FLEXCOMM2.

enumerator kFRG\_to\_FLEXCOMM2  
Attach FRG to FLEXCOMM2.

enumerator kNONE\_to\_FLEXCOMM2  
Attach NONE to FLEXCOMM2.

enumerator kSFRO\_to\_FLEXCOMM3  
Attach SFRO to FLEXCOMM3.

enumerator kFFRO\_to\_FLEXCOMM3  
Attach FFRO to FLEXCOMM3.

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

enumerator kMASTER\_CLK\_to\_FLEXCOMM3  
Attach MASTER\_CLK to FLEXCOMM3.

enumerator kFRG\_to\_FLEXCOMM3  
Attach FRG to FLEXCOMM3.

enumerator kNONE\_to\_FLEXCOMM3  
Attach NONE to FLEXCOMM3.

enumerator kSFRO\_to\_FLEXCOMM4  
Attach SFRO to FLEXCOMM4.

enumerator kFFRO\_to\_FLEXCOMM4  
Attach FFRO to FLEXCOMM4.

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

enumerator kMASTER\_CLK\_to\_FLEXCOMM4  
Attach MASTER\_CLK to FLEXCOMM4.

enumerator kFRG\_to\_FLEXCOMM4  
Attach FRG to FLEXCOMM4.

enumerator kNONE\_to\_FLEXCOMM4  
Attach NONE to FLEXCOMM4.

enumerator kSFRO\_to\_FLEXCOMM5  
Attach SFRO to FLEXCOMM5.

enumerator kFFRO\_to\_FLEXCOMM5  
Attach FFRO to FLEXCOMM5.



enumerator kAUDIO\_PLL\_to\_FLEXCOMM5

Attach AUDIO\_PLL to FLEXCOMM5.

enumerator kMASTER\_CLK\_to\_FLEXCOMM5

Attach MASTER\_CLK to FLEXCOMM5.

enumerator kFRG\_to\_FLEXCOMM5

Attach FRG to FLEXCOMM5.

enumerator kNONE\_to\_FLEXCOMM5

Attach NONE to FLEXCOMM5.

enumerator kSFRO\_to\_FLEXCOMM6

Attach SFRO to FLEXCOMM6.

enumerator kFFRO\_to\_FLEXCOMM6

Attach FFRO to FLEXCOMM6.

enumerator kAUDIO\_PLL\_to\_FLEXCOMM6

Attach AUDIO\_PLL to FLEXCOMM6.

enumerator kMASTER\_CLK\_to\_FLEXCOMM6

Attach MASTER\_CLK to FLEXCOMM6.

enumerator kFRG\_to\_FLEXCOMM6

Attach FRG to FLEXCOMM6.

enumerator kNONE\_to\_FLEXCOMM6

Attach NONE to FLEXCOMM6.

enumerator kSFRO\_to\_FLEXCOMM7

Attach SFRO to FLEXCOMM7.

enumerator kFFRO\_to\_FLEXCOMM7

Attach FFRO to FLEXCOMM7.

enumerator kAUDIO\_PLL\_to\_FLEXCOMM7

Attach AUDIO\_PLL to FLEXCOMM7.

enumerator kMASTER\_CLK\_to\_FLEXCOMM7

Attach MASTER\_CLK to FLEXCOMM7.

enumerator kFRG\_to\_FLEXCOMM7

Attach FRG to FLEXCOMM7.

enumerator kNONE\_to\_FLEXCOMM7

Attach NONE to FLEXCOMM7.

enumerator kSFRO\_to\_FLEXCOMM14

Attach SFRO to FLEXCOMM14.

enumerator kFFRO\_to\_FLEXCOMM14

Attach FFRO to FLEXCOMM14.

enumerator kAUDIO\_PLL\_to\_FLEXCOMM14

Attach AUDIO\_PLL to FLEXCOMM14.

enumerator kMASTER\_CLK\_to\_FLEXCOMM14

Attach MASTER\_CLK to FLEXCOMM14.

enumerator kFRG\_to\_FLEXCOMM14

Attach FRG to FLEXCOMM14.

enumerator kNONE\_to\_FLEXCOMM14  
Attach NONE to FLEXCOMM14.

enumerator kSFRO\_to\_FLEXCOMM15  
Attach SFRO to FLEXCOMM15.

enumerator kFFRO\_to\_FLEXCOMM15  
Attach FFRO to FLEXCOMM15.

enumerator kAUDIO\_PLL\_to\_FLEXCOMM15  
Attach AUDIO\_PLL to FLEXCOMM15.

enumerator kMASTER\_CLK\_to\_FLEXCOMM15  
Attach MASTER\_CLK to FLEXCOMM15.

enumerator kFRG\_to\_FLEXCOMM15  
Attach FRG to FLEXCOMM15.

enumerator kNONE\_to\_FLEXCOMM15  
Attach NONE to FLEXCOMM15.

enumerator kMAIN\_CLK\_to\_CTIMER0  
Attach MAIN\_CLK to CTIMER0.

enumerator kSFRO\_to\_CTIMER0  
Attach SFRO to CTIMER0.

enumerator kFFRO\_to\_CTIMER0  
Attach FFRO to CTIMER0.

enumerator kAUDIO\_PLL\_to\_CTIMER0  
Attach AUDIO\_PLL to CTIMER0.

enumerator kMASTER\_CLK\_to\_CTIMER0  
Attach MASTER\_CLK to CTIMER0.

enumerator kLPOSC\_to\_CTIMER0  
Attach LPOSC to CTIMER0.

enumerator kNONE\_to\_CTIMER0  
Attach NONE to CTIMER0.

enumerator kMAIN\_CLK\_to\_CTIMER1  
Attach MAIN\_CLK to CTIMER1.

enumerator kSFRO\_to\_CTIMER1  
Attach SFRO to CTIMER1.

enumerator kFFRO\_to\_CTIMER1  
Attach FFRO to CTIMER1.

enumerator kAUDIO\_PLL\_to\_CTIMER1  
Attach AUDIO\_PLL to CTIMER1.

enumerator kMASTER\_CLK\_to\_CTIMER1  
Attach MASTER\_CLK to CTIMER1.

enumerator kLPOSC\_to\_CTIMER1  
Attach LPOSC to CTIMER1.

enumerator kNONE\_to\_CTIMER1  
Attach NONE to CTIMER1.

enumerator kMAIN\_CLK\_to\_CTIMER2  
Attach MAIN\_CLK to CTIMER2.

enumerator kSFRO\_to\_CTIMER2  
Attach SFRO to CTIMER2.

enumerator kFFRO\_to\_CTIMER2  
Attach FFRO to CTIMER2.

enumerator kAUDIO\_PLL\_to\_CTIMER2  
Attach AUDIO\_PLL to CTIMER2.

enumerator kMASTER\_CLK\_to\_CTIMER2  
Attach MASTER\_CLK to CTIMER2.

enumerator kLPOSC\_to\_CTIMER2  
Attach LPOSC to CTIMER2.

enumerator kNONE\_to\_CTIMER2  
Attach NONE to CTIMER2.

enumerator kMAIN\_CLK\_to\_CTIMER3  
Attach MAIN\_CLK to CTIMER3.

enumerator kSFRO\_to\_CTIMER3  
Attach SFRO to CTIMER3.

enumerator kFFRO\_to\_CTIMER3  
Attach FFRO to CTIMER3.

enumerator kAUDIO\_PLL\_to\_CTIMER3  
Attach AUDIO\_PLL to CTIMER3.

enumerator kMASTER\_CLK\_to\_CTIMER3  
Attach MASTER\_CLK to CTIMER3.

enumerator kLPOSC\_to\_CTIMER3  
Attach LPOSC to CTIMER3.

enumerator kNONE\_to\_CTIMER3  
Attach NONE to CTIMER3.

enumerator kMAIN\_CLK\_to\_CTIMER4  
Attach MAIN\_CLK to CTIMER4.

enumerator kSFRO\_to\_CTIMER4  
Attach SFRO to CTIMER4.

enumerator kFFRO\_to\_CTIMER4  
Attach FFRO to CTIMER4.

enumerator kAUDIO\_PLL\_to\_CTIMER4  
Attach AUDIO\_PLL to CTIMER4.

enumerator kMASTER\_CLK\_to\_CTIMER4  
Attach MASTER\_CLK to CTIMER4.

enumerator kLPOSC\_to\_CTIMER4  
Attach LPOSC to CTIMER4.

enumerator kNONE\_to\_CTIMER4  
Attach NONE to CTIMER4.

enum \_clock\_div\_name

Clock dividers.

*Values:*

enumerator kCLOCK\_DivSysCpuAhbClk

Sys Cpu Ahb Clk Divider.

enumerator kCLOCK\_DivMainPllClk

Main Pll Clk Divider.

enumerator kCLOCK\_DivDspPllClk

Dsp Pll Clk Divider.

enumerator kCLOCK\_DivAux0PllClk

Aux0 Pll Clk Divider.

enumerator kCLOCK\_DivAux1PllClk

Aux1 Pll Clk Divider.

enumerator kCLOCK\_DivPfc0Clk

Pfc0 Clk Divider.

enumerator kCLOCK\_DivPfc1Clk

Pfc1 Clk Divider.

enumerator kCLOCK\_DivAdcClk

Adc Clk Divider.

enumerator kCLOCK\_DivFlexspiClk

Flexspi Clk Divider.

enumerator kCLOCK\_DivSctClk

Sct Clk Divider.

enumerator kCLOCK\_DivSdio0Clk

Sdio0 Clk Divider.

enumerator kCLOCK\_DivSdio1Clk

Sdio1 Clk Divider.

enumerator kCLOCK\_DivSystickClk

Systick Clk Divider.

enumerator kCLOCK\_DivUsbHsFclk

Usb Hs Fclk Divider.

enumerator kCLOCK\_DivAudioPllClk

Audio Pll Clk Divider.

enumerator kCLOCK\_DivAcmpClk

Acmp Clk Divider.

enumerator kCLOCK\_DivClockOut

Clock Out Divider.

enumerator kCLOCK\_DivDmicClk

Dmic Clk Divider.

enumerator kCLOCK\_DivDspCpuClk

Dsp Cpu Clk Divider.

enumerator kCLOCK\_DivDspRamClk  
Dsp Ram Clk Divider.

enumerator kCLOCK\_DivMclkClk  
Mclk Clk Divider.

enumerator kCLOCK\_DivPllFrgClk  
Pll Frg Clk Divider.

enumerator kCLOCK\_DivI3cClk  
I3c Clk Divider.

enumerator kCLOCK\_DivI3cTcClk  
I3c Tc Clk Divider.

enumerator kCLOCK\_DivI3cSlowClk  
I3c Slow Clk Divider.

enum \_clock\_ffro\_freq  
FFRO frequency configuration.

*Values:*

enumerator kCLOCK\_Ffro48M  
48MHz FFRO clock.

enumerator kCLOCK\_Ffro60M  
60MHz FFRO clock.

enum \_sys\_pll\_src  
SysPLL Reference Input Clock Source.

*Values:*

enumerator kCLOCK\_SysPllSFroClk  
16MHz FRO clock

enumerator kCLOCK\_SysPllXtalIn  
OSC clock

enumerator kCLOCK\_SysPllFFroDiv2  
FRO48/60 div2 clock

enumerator kCLOCK\_SysPllNone  
Gated to reduce power

enum \_sys\_pll\_mult  
SysPLL Multiplication Factor.

*Values:*

enumerator kCLOCK\_SysPllMult16  
Divide by 16

enumerator kCLOCK\_SysPllMult17  
Divide by 17

enumerator kCLOCK\_SysPllMult18  
Divide by 18

enumerator kCLOCK\_SysPllMult19  
Divide by 19

```
    enumerator kCLOCK_SysPllMult20
        Divide by 20
    enumerator kCLOCK_SysPllMult21
        Divide by 21
    enumerator kCLOCK_SysPllMult22
        Divide by 22
enum _audio_pll_src
    AudioPll Reference Input Clock Source.
    Values:
    enumerator kCLOCK_AudioPllSFroClk
        16MHz FRO clock
    enumerator kCLOCK_AudioPllXtalIn
        OSC clock
    enumerator kCLOCK_AudioPllFFroDiv2
        FRO48/60 div2 clock
    enumerator kCLOCK_AudioPllNone
        Gated to reduce power
enum _audio_pll_mult
    AudioPll Multiplication Factor.
    Values:
    enumerator kCLOCK_AudioPllMult16
        Divide by 16
    enumerator kCLOCK_AudioPllMult17
        Divide by 17
    enumerator kCLOCK_AudioPllMult18
        Divide by 18
    enumerator kCLOCK_AudioPllMult19
        Divide by 19
    enumerator kCLOCK_AudioPllMult20
        Divide by 20
    enumerator kCLOCK_AudioPllMult21
        Divide by 21
    enumerator kCLOCK_AudioPllMult22
        Divide by 22
typedef enum _clock_ip_name clock_ip_name_t
    Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.
typedef enum _clock_name clock_name_t
    Clock name used to get clock frequency.
typedef enum _clock_pfd clock_pfd_t
    PLL PFD clock name.
typedef enum _clock_attach_id clock_attach_id_t
    The enumerator of clock attach Id.
```

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

typedef enum *\_clock\_ffro\_freq* clock\_ffro\_freq\_t  
FFRO frequency configuration.

typedef enum *\_sys\_pll\_src* sys\_pll\_src\_t  
SysPLL Reference Input Clock Source.

typedef enum *\_sys\_pll\_mult* sys\_pll\_mult\_t  
SysPLL Multiplication Factor.

typedef struct *\_clock\_sys\_pll\_config* clock\_sys\_pll\_config\_t  
PLL configuration for SYSPLL.

typedef enum *\_audio\_pll\_src* audio\_pll\_src\_t  
AudioPll Reference Input Clock Source.

typedef enum *\_audio\_pll\_mult* audio\_pll\_mult\_t  
AudioPll Multiplication Factor.

typedef struct *\_clock\_audio\_pll\_config* clock\_audio\_pll\_config\_t  
PLL configuration for SYSPLL.

typedef struct *\_clock\_frg\_clk\_config* clock\_frg\_clk\_config\_t  
PLL configuration for FRG.

volatile uint32\_t g\_xtalFreq  
External XTAL (SYSOSC) clock frequency.

The XTAL (SYSOSC) clock frequency in Hz, when the clock is setup, use the function `CLOCK_SetXtalFreq` to set the value in to clock driver. For example, if XTAL is 16MHz,

```
CLOCK_SetXtalFreq(16000000);
```

volatile uint32\_t g\_clkinFreq  
External CLK\_IN pin clock frequency (clkin) clock frequency.

The CLK\_IN pin (clkin) clock frequency in Hz, when the clock is setup, use the function `CLOCK_SetClkinFreq` to set the value in to clock driver. For example, if CLK\_IN is 16MHz,

```
CLOCK_SetClkinFreq(16000000);
```

volatile uint32\_t g\_mclkFreq  
External XTAL (SYSOSC) clock frequency.

The MCLK in (mclk\_in) PIN clock frequency in Hz, when the clock is setup, use the function `CLOCK_SetMclkInFreq` to set the value in to clock driver. For example, if mclk\_In is 16MHz,

```
CLOCK_SetMclkInFreq(16000000);
```

static inline void `CLOCK_EnableClock(clock_ip_name_t clk)`

static inline void `CLOCK_DisableClock(clock_ip_name_t clk)`

void `CLOCK_AttachClk(clock_attach_id_t connection)`  
Configure the clock selection muxes.

#### Parameters

- connection – : Clock to be configured.

#### Returns

Nothing

void CLOCK\_SetClkDiv(*clock\_div\_name\_t* div\_name, uint32\_t divider)

Setup peripheral clock dividers.

**Parameters**

- div\_name – : Clock divider name
- divider – : Value to be divided. Divided clock frequency = Undivided clock frequency / divider.

**Returns**

Nothing

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

Return Frequency of selected clock.

**Returns**

Frequency of selected clock

uint32\_t CLOCK\_GetFRGClk(uint32\_t id)

Return Input frequency for the Fractional baud rate generator.

**Returns**

Input Frequency for FRG

void CLOCK\_SetFRGClk(const *clock\_frg\_clk\_config\_t* \*config)

Set output of the Fractional baud rate generator.

**Parameters**

- config – : Configuration to set to FRGn clock.

static inline uint32\_t CLOCK\_GetSFroFreq(void)

Return Frequency of FRO 16MHz.

**Returns**

Frequency of FRO 16MHz

uint32\_t CLOCK\_GetSysPllFreq(void)

Return Frequency of SYSPLL.

**Returns**

Frequency of SYSPLL

uint32\_t CLOCK\_GetSysPfdFreq(*clock\_pfd\_t* pfd)

Get current output frequency of specific System PLL PFD.

**Parameters**

- pfd – : pfd name to get frequency.

**Returns**

Frequency of SYSPLL PFD.

uint32\_t CLOCK\_GetAudioPllFreq(void)

Return Frequency of AUDIO PLL.

**Returns**

Frequency of AUDIO PLL

uint32\_t CLOCK\_GetAudioPfdFreq(*clock\_pfd\_t* pfd)

Get current output frequency of specific Audio PLL PFD.

**Parameters**

- pfd – : pfd name to get frequency.

**Returns**

Frequency of AUDIO PLL PFD.



uint32\_t CLOCK\_GetFFroFreq(void)

Return Frequency of High-Freq output of FRO.

**Returns**

Frequency of High-Freq output of FRO

uint32\_t CLOCK\_GetMainClkFreq(void)

Return Frequency of main clk.

**Returns**

Frequency of main clk

uint32\_t CLOCK\_GetDspMainClkFreq(void)

Return Frequency of DSP main clk.

**Returns**

Frequency of DSP main clk

uint32\_t CLOCK\_GetAcmpClkFreq(void)

Return Frequency of ACMP clk.

**Returns**

Frequency of ACMP clk

uint32\_t CLOCK\_GetDmicClkFreq(void)

Return Frequency of DMIC clk.

**Returns**

Frequency of DMIC clk

uint32\_t CLOCK\_GetUsbClkFreq(void)

Return Frequency of USB clk.

**Returns**

Frequency of USB clk

uint32\_t CLOCK\_GetSdioClkFreq(uint32\_t id)

Return Frequency of SDIO clk.

**Parameters**

- id – : SDIO index to get frequency.

**Returns**

Frequency of SDIO clk

uint32\_t CLOCK\_GetI3cClkFreq(void)

Return Frequency of I3C clk.

**Returns**

Frequency of I3C clk

uint32\_t CLOCK\_GetSystickClkFreq(void)

Return Frequency of systick clk.

**Returns**

Frequency of systick clk

uint32\_t CLOCK\_GetWdtClkFreq(uint32\_t id)

Return Frequency of WDT clk.

**Parameters**

- id – : WDT index to get frequency.

**Returns**

Frequency of WDT clk

uint32\_t CLOCK\_GetMclkClkFreq(void)

Return Frequency of mclk.

**Returns**

Frequency of mclk clk

uint32\_t CLOCK\_GetSctClkFreq(void)

Return Frequency of sct.

**Returns**

Frequency of sct clk

void CLOCK\_EnableSysOscClk(bool enable, bool enableLowPower, uint32\_t delay\_us)

Enable/Disable sys osc clock from external crystal clock.

**Parameters**

- enable – : true to enable system osc clock, false to bypass system osc.
- enableLowPower – : true to enable low power mode, false to enable high gain mode.
- delay\_us – : Delay time after OSC power up.

static inline uint32\_t CLOCK\_GetXtalInClkFreq(void)

Return Frequency of sys osc Clock.

**Returns**

Frequency of sys osc Clock. Or CLK\_IN pin frequency.

static inline uint32\_t CLOCK\_GetMclkInClkFreq(void)

Return Frequency of MCLK Input Clock.

**Returns**

Frequency of MCLK input Clock.

static inline uint32\_t CLOCK\_GetLpOscFreq(void)

Return Frequency of Lower power osc.

**Returns**

Frequency of LPOSC

static inline uint32\_t CLOCK\_GetOsc32KFreq(void)

Return Frequency of 32kHz osc.

**Returns**

Frequency of 32kHz osc

static inline void CLOCK\_EnableOsc32K(bool enable)

Enables and disables 32kHz osc.

**Parameters**

- enable – : true to enable 32k osc clock, false to disable clock

static inline uint32\_t CLOCK\_GetWakeClk32KFreq(void)

Return Frequency of 32khz wake clk.

**Returns**

Frequency of 32kHz wake clk

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

Set the XTALIN (system OSC) frequency based on board setting.

**Parameters**

- freq – : The XTAL input clock frequency in Hz.

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

Set the CLKIN (CLKIN pin) frequency based on board setting.

#### Parameters

- freq – : The CLK\_IN pin input clock frequency in Hz.

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

Set the MCLK in (mclk\_in) clock frequency based on board setting.

#### Parameters

- freq – : The MCLK input clock frequency in Hz.

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

Return Frequency of Flexcomm functional Clock.

#### Parameters

- id – : flexcomm index to get frequency.

#### Returns

Frequency of Flexcomm functional Clock

uint32\_t CLOCK\_GetCtimerClkFreq(uint32\_t id)

Return Frequency of Ctimer Clock.

#### Parameters

- id – : ctimer index to get frequency.

#### Returns

Frequency of Ctimer Clock

uint32\_t CLOCK\_GetClockOutClkFreq(void)

Return Frequency of ClockOut.

#### Returns

Frequency of ClockOut

uint32\_t CLOCK\_GetAdcClkFreq(void)

Return Frequency of Adc Clock.

#### Returns

Frequency of Adc Clock.

uint32\_t CLOCK\_GetFlexspiClkFreq(void)

Return Frequency of Flexspi Clock.

#### Returns

Frequency of Flexspi.

void CLOCK\_EnableFfroClk(*clock\_ffro\_freq\_t* ffroFreq)

brief Enable FFRO 48M/60M clock. param ffroFreq : target fro frequency. return Nothing

void CLOCK\_EnableSfroClk(void)

brief Enable SFRO clock. param Nothing return Nothing

void CLOCK\_InitSysPll(const *clock\_sys\_pll\_config\_t* \*config)

Initialize the System PLL.

#### Parameters

- config – : Configuration to set to PLL.

static inline void CLOCK\_DeinitSysPll(void)

brief Deinit the System PLL. param none.

*status\_t* CLOCK\_InitSysPfd(*clock\_pfd\_t* pfd, uint8\_t divider)  
Initialize the System PLL PFD.

---

**Note:** It is recommended that PFD settings are kept between 12-35.

---

#### Parameters

- pfd – : Which PFD clock to enable.
- divider – : The PFD divider value.

#### Returns

kStatus\_Success if successfully, kStatus\_Timeout if timeout happen.

static inline void CLOCK\_DeinitSysPfd(*clock\_pfd\_t* pfd)  
brief Disable the audio PLL PFD. param pfd : Which PFD clock to disable.

void CLOCK\_InitAudioPll(const *clock\_audio\_pll\_config\_t* \*config)  
Initialize the audio PLL.

#### Parameters

- config – : Configuration to set to PLL.

static inline void CLOCK\_DeinitAudioPll(void)  
brief Deinit the Audio PLL. param none.

*status\_t* CLOCK\_InitAudioPfd(*clock\_pfd\_t* pfd, uint8\_t divider)  
Initialize the audio PLL PFD.

---

**Note:** It is recommended that PFD settings are kept between 12-35.

---

#### Parameters

- pfd – : Which PFD clock to enable.
- divider – : The PFD divider value.

#### Returns

kStatus\_Success if successfully, kStatus\_Timeout if timeout happen.

static inline void CLOCK\_DeinitAudioPfd(uint32\_t pfd)  
brief Disable the audio PLL PFD. param pfd : Which PFD clock to disable.

void CLOCK\_EnableUsbhsDeviceClock(void)  
Enable USB HS device PLL clock.  
This function enables USB HS device PLL clock.

void CLOCK\_EnableUsbhsHostClock(void)  
Enable USB HS host PLL clock.  
This function enables USB HS host PLL clock.

bool CLOCK\_EnableUsbHs0PhyPllClock(*clock\_attach\_id\_t* src, uint32\_t freq)  
brief Enable USB hs0PhyPll clock.  
param src USB HS clock source. param freq The frequency specified by src. retval true The clock is set successfully. retval false The clock source is invalid to get proper USB HS clock.

FSL\_CLOCK\_DRIVER\_VERSION  
CLOCK driver version 2.7.4.

SDK\_DEVICE\_MAXIMUM\_CPU\_CLOCK\_FREQUENCY

CLOCK\_GetCTimerClkFreq

CMP\_CLOCKS

Clock ip name array for ACMP.

FLEXCOMM\_CLOCKS

Clock ip name array for FLEXCOMM.

USART\_CLOCKS

Clock ip name array for LPUART.

I2C\_CLOCKS

Clock ip name array for I2C.

I3C\_CLOCKS

Clock ip name array for I3C.

SPI\_CLOCKS

Clock ip name array for SPI.

FLEXI2S\_CLOCKS

Clock ip name array for FLEXI2S.

UTICK\_CLOCKS

Clock ip name array for UTICK.

DMIC\_CLOCKS

Clock ip name array for DMIC.

CTIMER\_CLOCKS

Clock ip name array for CT32B.

GPIO\_CLOCKS

Clock ip name array for GPIO.

LPADC\_CLOCKS

Clock ip name array for ADC.

MRT\_CLOCKS

Clock ip name array for MRT.

SCT\_CLOCKS

Clock ip name array for SCT.

RTC\_CLOCKS

Clock ip name array for RTC.

WWDT\_CLOCKS

Clock ip name array for WWDT.

CRC\_CLOCKS

Clock ip name array for CRC.

USB\_CLOCKS

Clock ip name array for USB.

DMA\_CLOCKS

Clock ip name array for DMA.

PINT\_CLOCKS

Clock ip name array for PINT.

FLEXSPI\_CLOCKS

Clock ip name array for FLEXSPI.

CACHE64\_CLOCKS

Clock ip name array for Cache64.

MU\_CLOCKS

Clock ip name array for MUA.

SEMA42\_CLOCKS

Clock ip name array for SEMA.

TRNG\_CLOCKS

Clock ip name array for RNG.

PUF\_CLOCKS

Clock ip name array for PUF.

HASHCRYPT\_CLOCKS

Clock ip name array for HashCrypt.

CASPER\_CLOCKS

Clock ip name array for Casper.

USDHC\_CLOCKS

Clock ip name array for uSDHC.

OSTIMER\_CLOCKS

Clock ip name array for OSTimer.

POWERQUAD\_CLOCKS

Clock ip name array for Powerquad.

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)

CLK\_CTL0\_PSCCTL0

CLK\_CTL0\_PSCCTL1

CLK\_CTL0\_PSCCTL2

CLK\_CTL1\_PSCCTL0

CLK\_CTL1\_PSCCTL1

CLK\_CTL1\_PSCCTL2

SYSPLL0CLKSEL\_\_OFFSET

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

[12 bits for reg offset, 0 means end of descriptor, 4 bits for choice] [bit 31 define CLKCTL0 or CLKCTL1]\*

MAINCLKSELA\_\_OFFSET

MAINCLKSELB\_\_OFFSET

FLEXSPIFCLKSEL\_\_OFFSET

SCTFCLKSEL\_\_OFFSET

USBHSFCLKSEL\_\_OFFSET

SDIO0FCLKSEL\_\_OFFSET

SDIO1FCLKSEL\_\_OFFSET

ESPICLKSEL\_\_OFFSET

ADC0FCLKSEL0\_\_OFFSET

ADC0FCLKSEL1\_\_OFFSET

UTICKFCLKSEL\_\_OFFSET

WDT0FCLKSEL\_\_OFFSET

WAKECLK32KHZSEL\_\_OFFSET

SYSTICKFCLKSEL\_\_OFFSET

AUDIOPLL0CLKSEL\_\_OFFSET

DSPCPUCLKSELA\_\_OFFSET

DSPCPUCLKSELB\_\_OFFSET

OSEVENTFCLKSEL\_\_OFFSET

FC0FCLKSEL\_\_OFFSET

FC1FCLKSEL\_\_OFFSET

FC2FCLKSEL\_\_OFFSET

FC3FCLKSEL\_\_OFFSET

FC4FCLKSEL\_\_OFFSET

FC5FCLKSEL\_\_OFFSET

FC6FCLKSEL\_\_OFFSET

FC7FCLKSEL\_\_OFFSET

FC14FCLKSEL\_\_OFFSET

FC15FCLKSEL\_\_OFFSET

DMIC0FCLKSEL\_\_OFFSET

CT32BIT0FCLKSEL\_OFFSET  
CT32BIT1FCLKSEL\_OFFSET  
CT32BIT2FCLKSEL\_OFFSET  
CT32BIT3FCLKSEL\_OFFSET  
CT32BIT4FCLKSEL\_OFFSET  
AUDIOMCLKSEL\_OFFSET  
CLKOUTSEL0\_OFFSET  
CLKOUTSEL1\_OFFSET  
I3C0FCLKSEL\_OFFSET  
I3C0FCLKSTCSEL\_OFFSET  
WDT1FCLKSEL\_OFFSET  
ACMP0FCLKSEL\_OFFSET  
MAINPLLCLKDIV\_OFFSET  
DSPPLLCLKDIV\_OFFSET  
AUX0PLLCLKDIV\_OFFSET  
AUX1PLLCLKDIV\_OFFSET  
SYSCPUAHBCLKDIV\_OFFSET  
PFC0CLKDIV\_OFFSET  
PFC1CLKDIV\_OFFSET  
FLEXSPIFCLKDIV\_OFFSET  
SCTFCLKDIV\_OFFSET  
USBHSFCLKDIV\_OFFSET  
SDIO0FCLKDIV\_OFFSET  
SDIO1FCLKDIV\_OFFSET  
ADC0FCLKDIV\_OFFSET  
WAKECLK32KHZDIV\_OFFSET  
SYSTICKFCLKDIV\_OFFSET  
AUDIOPLLCLKDIV\_OFFSET  
DSPCPUCLKDIV\_OFFSET  
DSPMAINRAMCLKDIV\_OFFSET  
FRGPLLCLKDIV\_OFFSET  
DMIC0FCLKDIV\_OFFSET  
AUDIOMCLKDIV\_OFFSET



CLKOUTDIV\_OFFSET  
I3C0CLKSTCDIV\_OFFSET  
I3C0CLKSDIV\_OFFSET  
I3C0CLKDIV\_OFFSET  
ACMP0CLKDIV\_OFFSET  
CLKCTL0\_TUPLE\_MUXA(reg, choice)  
CLKCTL0\_TUPLE\_MUXB(reg, choice)  
CLKCTL1\_TUPLE\_MUXA(reg, choice)  
CLKCTL1\_TUPLE\_MUXB(reg, choice)  
CLKCTL\_TUPLE\_REG(base, tuple)  
CLKCTL\_TUPLE\_SEL(tuple)

*Values:*

enumerator kCLOCK\_FrgMainClk

Main System clock

enumerator kCLOCK\_FrgPllDiv

Main pll clock divider

enumerator kCLOCK\_FrgSFro

16MHz FRO

enumerator kCLOCK\_FrgFFro

FRO48/60

*sys\_pll\_src\_t* sys\_pll\_src

Reference Input Clock Source

uint32\_t numerator

30 bit numerator of fractional loop divider.

uint32\_t denominator

30 bit numerator of fractional loop divider.

*sys\_pll\_mult\_t* sys\_pll\_mult

Multiplication Factor

*audio\_pll\_src\_t* audio\_pll\_src

Reference Input Clock Source

uint32\_t numerator

30 bit numerator of fractional loop divider.

uint32\_t denominator

30 bit numerator of fractional loop divider.

*audio\_pll\_mult\_t* audio\_pll\_mult

Multiplication Factor

uint8\_t num

FRG clock

```
enum _clock_frg_clk_config sfg_clock_src
uint8_t divider
    Denominator of the fractional divider.
uint8_t mult
    Numerator of the fractional divider.
struct _clock_sys_pll_config
    #include <fsl_clock.h> PLL configuration for SYSPLL.
struct _clock_audio_pll_config
    #include <fsl_clock.h> PLL configuration for SYSPLL.
struct _clock_frg_clk_config
    #include <fsl_clock.h> PLL configuration for FRG.
```

## 2.7 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.

## 2.8 CTIMER: Standard counter/timers

`void CTIMER_Init(CTIMER_Type *base, const ctimer_config_t *config)`

Ungates the clock and configures the peripheral for basic operation.

---

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

---

### Parameters

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

`void CTIMER_Deinit(CTIMER_Type *base)`

Gates the timer clock.

### Parameters

- `base` – Ctimer peripheral base address

`void CTIMER_GetDefaultConfig(ctimer_config_t *config)`

Fills in the timers configuration structure with the default settings.

The default values are:

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

### Parameters

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

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

Configures the PWM signal parameters.

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

---

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

---

### Parameters

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

**Returns**

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

*status\_t* CTIMER\_SetupPwm(CTIMER\_Type \*base, const *ctimer\_match\_t* pwmPeriodChannel, *ctimer\_match\_t* matchChannel, uint8\_t dutyCyclePercent, uint32\_t pwmFreq\_Hz, uint32\_t srcClock\_Hz, bool enableInt)

Configures the PWM signal parameters.

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

---

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

---

**Parameters**

- base – Ctimer peripheral base address
- pwmPeriodChannel – Specify the channel to control the PWM period
- matchChannel – Match pin to be used to output the PWM signal
- dutyCyclePercent – PWM pulse width; the value should be between 0 to 100
- pwmFreq\_Hz – PWM signal frequency in Hz
- srcClock\_Hz – Timer counter clock in Hz
- enableInt – Enable interrupt when the timer value reaches the match value of the PWM pulse, if it is 0 then no interrupt will be generated.

static inline void CTIMER\_UpdatePwmPulsePeriod(CTIMER\_Type \*base, *ctimer\_match\_t* matchChannel, uint32\_t pulsePeriod)

Updates the pulse period of an active PWM signal.

**Parameters**

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

*status\_t* CTIMER\_UpdatePwmDutycycle(CTIMER\_Type \*base, const *ctimer\_match\_t* pwmPeriodChannel, *ctimer\_match\_t* matchChannel, uint8\_t dutyCyclePercent)

Updates the duty cycle of an active PWM signal.

---

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

---

**Parameters**

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

**Returns**

kStatus\_Success on success kStatus\_Fail If PWM pulse width register value is larger than 0xFFFFFFFF.

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

Enables the selected Timer interrupts.

**Parameters**

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

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

Disables the selected Timer interrupts.

**Parameters**

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

```
static inline uint32_t CTIMER_GetEnabledInterrupts(CTIMER_Type *base)
```

Gets the enabled Timer interrupts.

**Parameters**

- base – Ctimer peripheral base address

**Returns**

The enabled interrupts. This is the logical OR of members of the enumeration ctimer\_interrupt\_enable\_t

```
static inline uint32_t CTIMER_GetStatusFlags(CTIMER_Type *base)
```

Gets the Timer status flags.

**Parameters**

- base – Ctimer peripheral base address

**Returns**

The status flags. This is the logical OR of members of the enumeration ctimer\_status\_flags\_t

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

Clears the Timer status flags.

**Parameters**

- base – Ctimer peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration ctimer\_status\_flags\_t

```
static inline void CTIMER_StartTimer(CTIMER_Type *base)
```

Starts the Timer counter.

**Parameters**

- base – Ctimer peripheral base address

```
static inline void CTIMER_StopTimer(CTIMER_Type *base)
```

Stops the Timer counter.

**Parameters**

- base – Ctimer peripheral base address

FSL\_CTIMER\_DRIVER\_VERSION

Version 2.3.3

enum \_ctimer\_capture\_channel

List of Timer capture channels.

*Values:*

enumerator kCTIMER\_Capture\_0

Timer capture channel 0

enumerator kCTIMER\_Capture\_1

Timer capture channel 1

enumerator kCTIMER\_Capture\_3

Timer capture channel 3

enum \_ctimer\_capture\_edge

List of capture edge options.

*Values:*

enumerator kCTIMER\_Capture\_RiseEdge

Capture on rising edge

enumerator kCTIMER\_Capture\_FallEdge

Capture on falling edge

enumerator kCTIMER\_Capture\_BothEdge

Capture on rising and falling edge

enum \_ctimer\_match

List of Timer match registers.

*Values:*

enumerator kCTIMER\_Match\_0

Timer match register 0

enumerator kCTIMER\_Match\_1

Timer match register 1

enumerator kCTIMER\_Match\_2

Timer match register 2

enumerator kCTIMER\_Match\_3

Timer match register 3

enum \_ctimer\_external\_match

List of external match.

*Values:*

enumerator kCTIMER\_External\_Match\_0

External match 0

enumerator kCTIMER\_External\_Match\_1

External match 1

enumerator kCTIMER\_External\_Match\_2

External match 2

enumerator kCTIMER\_External\_Match\_3

External match 3



enum \_ctimer\_match\_output\_control

List of output control options.

*Values:*

enumerator kCTIMER\_Output\_NoAction

No action is taken

enumerator kCTIMER\_Output\_Clear

Clear the EM bit/output to 0

enumerator kCTIMER\_Output\_Set

Set the EM bit/output to 1

enumerator kCTIMER\_Output\_Toggle

Toggle the EM bit/output

enum \_ctimer\_timer\_mode

List of Timer modes.

*Values:*

enumerator kCTIMER\_TimerMode

enumerator kCTIMER\_IncreaseOnRiseEdge

enumerator kCTIMER\_IncreaseOnFallEdge

enumerator kCTIMER\_IncreaseOnBothEdge

enum \_ctimer\_interrupt\_enable

List of Timer interrupts.

*Values:*

enumerator kCTIMER\_Match0InterruptEnable

Match 0 interrupt

enumerator kCTIMER\_Match1InterruptEnable

Match 1 interrupt

enumerator kCTIMER\_Match2InterruptEnable

Match 2 interrupt

enumerator kCTIMER\_Match3InterruptEnable

Match 3 interrupt

enum \_ctimer\_status\_flags

List of Timer flags.

*Values:*

enumerator kCTIMER\_Match0Flag

Match 0 interrupt flag

enumerator kCTIMER\_Match1Flag

Match 1 interrupt flag

enumerator kCTIMER\_Match2Flag

Match 2 interrupt flag

enumerator kCTIMER\_Match3Flag

Match 3 interrupt flag

enum *ctimer\_callback\_type\_t*

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

*Values:*

enumerator *kCTIMER\_SingleCallback*

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

enumerator *kCTIMER\_MultipleCallback*

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

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

List of Timer capture channels.

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

List of capture edge options.

typedef enum *\_ctimer\_match* *ctimer\_match\_t*

List of Timer match registers.

typedef enum *\_ctimer\_external\_match* *ctimer\_external\_match\_t*

List of external match.

typedef enum *\_ctimer\_match\_output\_control* *ctimer\_match\_output\_control\_t*

List of output control options.

typedef enum *\_ctimer\_timer\_mode* *ctimer\_timer\_mode\_t*

List of Timer modes.

typedef enum *\_ctimer\_interrupt\_enable* *ctimer\_interrupt\_enable\_t*

List of Timer interrupts.

typedef enum *\_ctimer\_status\_flags* *ctimer\_status\_flags\_t*

List of Timer flags.

typedef void (\**ctimer\_callback\_t*)(uint32\_t flags)

typedef struct *\_ctimer\_match\_config* *ctimer\_match\_config\_t*

Match configuration.

This structure holds the configuration settings for each match register.

typedef struct *\_ctimer\_config* *ctimer\_config\_t*

Timer configuration structure.

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

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

void *CTIMER\_SetupMatch*(*CTIMER\_Type* \*base, *ctimer\_match\_t* matchChannel, const *ctimer\_match\_config\_t* \*config)

Setup the match register.

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

#### Parameters

- base – Ctimer peripheral base address
- matchChannel – Match register to configure

- `config` – Pointer to the match configuration structure

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

Get the status of output match.

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

#### Parameters

- `base` – Ctimer peripheral base address
- `matchChannel` – External match channel, user can obtain the status of multiple match channels at the same time by using the logic of “|” enumeration `ctimer_external_match_t`

#### Returns

The mask of external match channel status flags. Users need to use the `_ctimer_external_match` type to decode the return variables.

```
void CTIMER_SetupCapture(CTIMER_Type *base, ctimer_capture_channel_t capture,
                        ctimer_capture_edge_t edge, bool enableInt)
```

Setup the capture.

#### Parameters

- `base` – Ctimer peripheral base address
- `capture` – Capture channel to configure
- `edge` – Edge on the channel that will trigger a capture
- `enableInt` – Flag to enable channel interrupts, if enabled then the registered call back is called upon capture

```
static inline uint32_t CTIMER_GetTimerCountValue(CTIMER_Type *base)
```

Get the timer count value from TC register.

#### Parameters

- `base` – Ctimer peripheral base address.

#### Returns

return the timer count value.

```
void CTIMER_RegisterCallBack(CTIMER_Type *base, ctimer_callback_t *cb_func,
                           ctimer_callback_type_t cb_type)
```

Register callback.

This function configures CTimer Callback in following modes:

- **Single Callback:** `cb_func` should be pointer to callback function pointer  
For example: `ctimer_callback_t ctimer_callback = pwm_match_callback;`  
`CTIMER_RegisterCallBack(CTIMER, &ctimer_callback, kCTIMER_SingleCallback);`
- **Multiple Callback:** `cb_func` should be pointer to array of callback function pointers Each element corresponds to Interrupt Flag in IR register.  
For example: `ctimer_callback_t ctimer_callback_table[] = { ctimer_match0_callback, NULL, NULL, ctimer_match3_callback, NULL, NULL, NULL, NULL};`  
`CTIMER_RegisterCallBack(CTIMER, &ctimer_callback_table[0], kCTIMER_MultipleCallback);`

#### Parameters

- `base` – Ctimer peripheral base address
- `cb_func` – Pointer to callback function pointer

- `cb_type` – callback function type, singular or multiple

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

Reset the counter.

The timer counter and prescale counter are reset on the next positive edge of the APB clock.

#### Parameters

- `base` – Ctimer peripheral base address

static inline void CTIMER\_SetPrescale(CTIMER\_Type \*base, uint32\_t prescale)

Setup the timer prescale value.

Specifies the maximum value for the Prescale Counter.

#### Parameters

- `base` – Ctimer peripheral base address
- `prescale` – Prescale value

static inline uint32\_t CTIMER\_GetCaptureValue(CTIMER\_Type \*base, *ctimer\_capture\_channel\_t* capture)

Get capture channel value.

Get the counter/timer value on the corresponding capture channel.

#### Parameters

- `base` – Ctimer peripheral base address
- `capture` – Select capture channel

#### Returns

The timer count capture value.

static inline void CTIMER\_EnableResetMatchChannel(CTIMER\_Type \*base, *ctimer\_match\_t* match, bool enable)

Enable reset match channel.

Set the specified match channel reset operation.

#### Parameters

- `base` – Ctimer peripheral base address
- `match` – match channel used
- `enable` – Enable match channel reset operation.

static inline void CTIMER\_EnableStopMatchChannel(CTIMER\_Type \*base, *ctimer\_match\_t* match, bool enable)

Enable stop match channel.

Set the specified match channel stop operation.

#### Parameters

- `base` – Ctimer peripheral base address.
- `match` – match channel used.
- `enable` – Enable match channel stop operation.

static inline void CTIMER\_EnableMatchChannelReload(CTIMER\_Type \*base, *ctimer\_match\_t* match, bool enable)

Enable reload channel falling edge.

Enable the specified match channel reload match shadow value.

**Parameters**

- base – Ctimer peripheral base address.
- match – match channel used.
- enable – Enable .

```
static inline void CTIMER__EnableRisingEdgeCapture(CTIMER_Type *base,  
                                                    ctimer_capture_channel_t capture, bool  
                                                    enable)
```

Enable capture channel rising edge.

Sets the specified capture channel for rising edge capture.

**Parameters**

- base – Ctimer peripheral base address.
- capture – capture channel used.
- enable – Enable rising edge capture.

```
static inline void CTIMER__EnableFallingEdgeCapture(CTIMER_Type *base,  
                                                    ctimer_capture_channel_t capture, bool  
                                                    enable)
```

Enable capture channel falling edge.

Sets the specified capture channel for falling edge capture.

**Parameters**

- base – Ctimer peripheral base address.
- capture – capture channel used.
- enable – Enable falling edge capture.

```
static inline void CTIMER__SetShadowValue(CTIMER_Type *base, ctimer_match_t match,  
                                          uint32_t matchvalue)
```

Set the specified match shadow channel.

**Parameters**

- base – Ctimer peripheral base address.
- match – match channel used.
- matchvalue – Reload the value of the corresponding match register.

```
struct __ctimer_match_config  
#include <fsl_ctimer.h> Match configuration.
```

This structure holds the configuration settings for each match register.

**Public Members**

uint32\_t matchValue

This is stored in the match register

bool enableCounterReset

true: Match will reset the counter false: Match will not reset the counter

bool enableCounterStop

true: Match will stop the counter false: Match will not stop the counter

*ctimer\_match\_output\_control\_t* outControl

Action to be taken on a match on the EM bit/output

bool outPinInitState

Initial value of the EM bit/output

bool enableInterrupt

true: Generate interrupt upon match false: Do not generate interrupt on match

struct *\_ctimer\_config*

*#include <fsl\_ctimer.h>* Timer configuration structure.

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

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

### Public Members

*ctimer\_timer\_mode\_t* mode

Timer mode

*ctimer\_capture\_channel\_t* input

Input channel to increment the timer, used only in timer modes that rely on this input signal to increment TC

uint32\_t prescale

Prescale value

## 2.9 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 superceded 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 desscriptor 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 un-processed 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:

- 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)
```

- 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:

- a. 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\_int* *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\_LINK\_DESCRIPTOR\_ALIGN\_SIZE*  
DMA channel numbers.

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

## 2.10 DMIC: Digital Microphone

### 2.11 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 2.4.1.

```
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
```

## 2.12 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\_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.

static inline void DMIC\_EnableChannelSignExtend(DMIC\_Type \*base, dmic\_channel\_t channel, bool enable)

Enbale channel sign extend which allows processing of 24bit audio data on 32bit machines.

**Parameters**

- base – : The base address of DMIC interface
- channel – : DMIC channel

- enable – : true is enable sign extend, false is disable sign extend

```
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__ResetChannelDecimator(DMIC_Type *base, uint32_t channelMask, bool reset)`  
DMIC channel Decimator reset.

**Parameters**

- `base` – : The base address of DMIC interface
- `channelMask` – : DMIC channel mask, reference `_dmic_channel_mask`
- `reset` – : `true` is reset decimator, `false` is release decimator.

`static inline void DMIC__EnableChannelGlobalSync(DMIC_Type *base, uint32_t channelMask, uint32_t syncCounter)`

Enable DMIC channel global sync function.

**Parameters**

- `base` – : The base address of DMIC interface
- `channelMask` – : DMIC channel mask, reference `_dmic_channel_mask`
- `syncCounter` – : sync counter will trigger a pulse whenever count reaches `CCOUNTVAL`. If `CCOUNTVAL` is set to 0, there will be a pulse on every cycle

`static inline void DMIC__DisableChannelGlobalSync(DMIC_Type *base, uint32_t channelMask)`  
Disbale DMIC channel global sync function.

**Parameters**

- `base` – : The base address of DMIC interface
- `channelMask` – : DMIC channel mask, reference `_dmic_channel_mask`

`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\_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_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` divhfk  
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.

`bool` enableSignExtend  
sign extend feature which allows processing of 24bit audio data on 32bit machine

## 2.13 DSP Driver

```
typedef struct _dsp_copy_image dsp_copy_image_t
```

Structure for DSP copy image to destination address.

Defines start and destination address for copying image with given size.

```
void DSP_Init(void)
```

Initializing DSP core.

Power up DSP TCM Enable DSP clock Reset DSP peripheral

```
void DSP_Deinit(void)
```

Deinit DSP core.

Power down DSP TCM Disable DSP clock Set DSP peripheral reset

```
void DSP_CopyImage(dsp_copy_image_t *dspCopyImage)
```

Copy DSP image to destination address.

Copy DSP image from source address to destination address with given size.

### Parameters

- `dspCopyImage` – Structure contains information for DSP copy image to destination address.

```
static inline void DSP_Start(void)
```

Start DSP core.

```
static inline void DSP_Stop(void)
```

Stop DSP core.

```
FSL_DSP_DRIVER_VERSION
```

reset driver version 2.1.1.

```
uint32_t *srcAddr
```

```
uint32_t *destAddr
```

```
uint32_t size
```

```
struct _dsp_copy_image
```

`#include <fsl_dsp.h>` Structure for DSP copy image to destination address.

Defines start and destination address for copying image with given size.

## 2.14 FLEXCOMM: FLEXCOMM Driver

## 2.15 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.

## 2.16 FLEXSPI: Flexible Serial Peripheral Interface Driver

uint32\_t FLEXSPI\_GetInstance(FLEXSPI\_Type \*base)

Get the instance number for FLEXSPI.

### Parameters

- base – FLEXSPI base pointer.

status\_t FLEXSPI\_CheckAndClearError(FLEXSPI\_Type \*base, uint32\_t status)

Check and clear IP command execution errors.

### Parameters

- base – FLEXSPI base pointer.
- status – interrupt status.

void FLEXSPI\_Init(FLEXSPI\_Type \*base, const flexspi\_config\_t \*config)

Initializes the FLEXSPI module and internal state.

This function enables the clock for FLEXSPI and also configures the FLEXSPI with the input configure parameters. Users should call this function before any FLEXSPI operations.

### Parameters

- base – FLEXSPI peripheral base address.
- config – FLEXSPI configure structure.

void FLEXSPI\_GetDefaultConfig(*flexspi\_config\_t* \*config)

Gets default settings for FLEXSPI.

**Parameters**

- config – FLEXSPI configuration structure.

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

Deinitializes the FLEXSPI module.

Clears the FLEXSPI state and FLEXSPI module registers.

**Parameters**

- base – FLEXSPI peripheral base address.

void FLEXSPI\_UpdateDllValue(FLEXSPI\_Type \*base, *flexspi\_device\_config\_t* \*config,  
*flexspi\_port\_t* port)

Update FLEXSPI DLL value depending on currently flexspi root clock.

**Parameters**

- base – FLEXSPI peripheral base address.
- config – Flash configuration parameters.
- port – FLEXSPI Operation port.

void FLEXSPI\_SetFlashConfig(FLEXSPI\_Type \*base, *flexspi\_device\_config\_t* \*config,  
*flexspi\_port\_t* port)

Configures the connected device parameter.

This function configures the connected device relevant parameters, such as the size, command, and so on. The flash configuration value cannot have a default value. The user needs to configure it according to the connected device.

**Parameters**

- base – FLEXSPI peripheral base address.
- config – Flash configuration parameters.
- port – FLEXSPI Operation port.

void FLEXSPI\_SoftwareReset(FLEXSPI\_Type \*base)

Software reset for the FLEXSPI logic.

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

**Parameters**

- base – FLEXSPI peripheral base address.

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

Enables or disables the FLEXSPI module.

**Parameters**

- base – FLEXSPI peripheral base address.
- enable – True means enable FLEXSPI, false means disable.

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

Enables the FLEXSPI interrupts.

**Parameters**

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

```
static inline void FLEXSPI_DisableInterrupts(FLEXSPI_Type *base, uint32_t mask)
```

Disable the FLEXSPI interrupts.

**Parameters**

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

```
static inline void FLEXSPI_EnableTxDMA(FLEXSPI_Type *base, bool enable)
```

Enables or disables FLEXSPI IP Tx FIFO DMA requests.

**Parameters**

- base – FLEXSPI peripheral base address.
- enable – Enable flag for transmit DMA request. Pass true for enable, false for disable.

```
static inline void FLEXSPI_EnableRxDMA(FLEXSPI_Type *base, bool enable)
```

Enables or disables FLEXSPI IP Rx FIFO DMA requests.

**Parameters**

- base – FLEXSPI peripheral base address.
- enable – Enable flag for receive DMA request. Pass true for enable, false for disable.

```
static inline uint32_t FLEXSPI_GetTxFifoAddress(FLEXSPI_Type *base)
```

Gets FLEXSPI IP tx fifo address for DMA transfer.

**Parameters**

- base – FLEXSPI peripheral base address.

**Return values**

The – tx fifo address.

```
static inline uint32_t FLEXSPI_GetRxFifoAddress(FLEXSPI_Type *base)
```

Gets FLEXSPI IP rx fifo address for DMA transfer.

**Parameters**

- base – FLEXSPI peripheral base address.

**Return values**

The – rx fifo address.

```
static inline void FLEXSPI_ResetFifos(FLEXSPI_Type *base, bool txFifo, bool rxFifo)
```

Clears the FLEXSPI IP FIFO logic.

**Parameters**

- base – FLEXSPI peripheral base address.
- txFifo – Pass true to reset TX FIFO.
- rxFifo – Pass true to reset RX FIFO.

```
static inline void FLEXSPI_GetFifoCounts(FLEXSPI_Type *base, size_t *txCount, size_t  
                                         *rxCount)
```

Gets the valid data entries in the FLEXSPI FIFOs.

**Parameters**

- base – FLEXSPI peripheral base address.
- txCount – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.

- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

static inline uint32\_t FLEXSPI\_GetInterruptStatusFlags(FLEXSPI\_Type \*base)

Get the FLEXSPI interrupt status flags.

#### Parameters

- `base` – FLEXSPI peripheral base address.

#### Return values

`interrupt` – status flag, use status flag to AND `flexspi_flags_t` could get the related status.

static inline void FLEXSPI\_ClearInterruptStatusFlags(FLEXSPI\_Type \*base, uint32\_t mask)

Get the FLEXSPI interrupt status flags.

#### Parameters

- `base` – FLEXSPI peripheral base address.
- `mask` – FLEXSPI interrupt source.

static inline void FLEXSPI\_GetDataLearningPhase(FLEXSPI\_Type \*base, uint8\_t \*portAPhase, uint8\_t \*portBPhase)

Gets the sampling clock phase selection after Data Learning.

#### Parameters

- `base` – FLEXSPI peripheral base address.
- `portAPhase` – Pointer to a `uint8_t` type variable to receive the selected clock phase on PORTA.
- `portBPhase` – Pointer to a `uint8_t` type variable to receive the selected clock phase on PORTB.

static inline *flexspi\_arb\_command\_source\_t* FLEXSPI\_GetArbitratorCommandSource(FLEXSPI\_Type \*base)

Gets the trigger source of current command sequence granted by arbitrator.

#### Parameters

- `base` – FLEXSPI peripheral base address.

#### Return values

`trigger` – source of current command sequence.

static inline *flexspi\_ip\_error\_code\_t* FLEXSPI\_GetIPCommandErrorCode(FLEXSPI\_Type \*base, uint8\_t \*index)

Gets the error code when IP command error detected.

#### Parameters

- `base` – FLEXSPI peripheral base address.
- `index` – Pointer to a `uint8_t` type variable to receive the sequence index when error detected.

#### Return values

`error` – code when IP command error detected.

static inline *flexspi\_ahb\_error\_code\_t* FLEXSPI\_GetAHBCommandErrorCode(FLEXSPI\_Type \*base, uint8\_t \*index)

Gets the error code when AHB command error detected.

#### Parameters

- `base` – FLEXSPI peripheral base address.
- `index` – Pointer to a `uint8_t` type variable to receive the sequence index when error detected.

**Return values**

`error` – code when AHB command error detected.

```
static inline bool FLEXSPI_GetBusIdleStatus(FLEXSPI_Type *base)
```

Returns whether the bus is idle.

**Parameters**

- `base` – FLEXSPI peripheral base address.

**Return values**

- `true` – Bus is idle.
- `false` – Bus is busy.

```
void FLEXSPI_UpdateRxSampleClock(FLEXSPI_Type *base, flexspi_read_sample_clock_t  
                                clockSource)
```

Update read sample clock source.

**Parameters**

- `base` – FLEXSPI peripheral base address.
- `clockSource` – clockSource of type `flexspi_read_sample_clock_t`

```
void FLEXSPI_UpdateLUT(FLEXSPI_Type *base, uint32_t index, const uint32_t *cmd, uint32_t  
                      count)
```

Updates the LUT table.

**Parameters**

- `base` – FLEXSPI peripheral base address.
- `index` – From which index start to update. It could be any index of the LUT table, which also allows user to update command content inside a command. Each command consists of up to 8 instructions and occupy 4\*32-bit memory.
- `cmd` – Command sequence array.
- `count` – Number of sequences.

```
static inline void FLEXSPI_WriteData(FLEXSPI_Type *base, uint32_t data, uint8_t fifoIndex)
```

Writes data into FIFO.

**Parameters**

- `base` – FLEXSPI peripheral base address
- `data` – The data bytes to send
- `fifoIndex` – Destination fifo index.

```
static inline uint32_t FLEXSPI_ReadData(FLEXSPI_Type *base, uint8_t fifoIndex)
```

Receives data from data FIFO.

**Parameters**

- `base` – FLEXSPI peripheral base address
- `fifoIndex` – Source fifo index.

**Returns**

The data in the FIFO.



*status\_t* FLEXSPI\_WriteBlocking(FLEXSPI\_Type \*base, uint8\_t \*buffer, size\_t size)

Sends a buffer of data bytes using blocking method.

---

**Note:** This function blocks via polling until all bytes have been sent.

---

#### Parameters

- base – FLEXSPI peripheral base address
- buffer – The data bytes to send
- size – The number of data bytes to send

#### Return values

- kStatus\_Success – write success without error
- kStatus\_FLEXSPI\_SequenceExecutionTimeout – sequence execution timeout
- kStatus\_FLEXSPI\_IpCommandSequenceError – IP command sequence error detected
- kStatus\_FLEXSPI\_IpCommandGrantTimeout – IP command grant timeout detected

*status\_t* FLEXSPI\_ReadBlocking(FLEXSPI\_Type \*base, uint8\_t \*buffer, size\_t size)

Receives a buffer of data bytes using a blocking method.

---

**Note:** This function blocks via polling until all bytes have been sent.

---

#### Parameters

- base – FLEXSPI peripheral base address
- buffer – The data bytes to send
- size – The number of data bytes to receive

#### Return values

- kStatus\_Success – read success without error
- kStatus\_FLEXSPI\_SequenceExecutionTimeout – sequence execution timeout
- kStatus\_FLEXSPI\_IpCommandSequenceError – IP command sequence error detected
- kStatus\_FLEXSPI\_IpCommandGrantTimeout – IP command grant timeout detected

*status\_t* FLEXSPI\_TransferBlocking(FLEXSPI\_Type \*base, *flexspi\_transfer\_t* \*xfer)

Execute command to transfer a buffer data bytes using a blocking method.

#### Parameters

- base – FLEXSPI peripheral base address
- xfer – pointer to the transfer structure.

#### Return values

- kStatus\_Success – command transfer success without error

- `kStatus_FLEXSPI_SequenceExecutionTimeout` – sequence execution timeout
- `kStatus_FLEXSPI_IpCommandSequenceError` – IP command sequence error detected
- `kStatus_FLEXSPI_IpCommandGrantTimeout` – IP command grant timeout detected

`void FLEXSPI_TransferCreateHandle(FLEXSPI_Type *base, flexspi_handle_t *handle, flexspi_transfer_callback_t callback, void *userData)`

Initializes the FLEXSPI handle which is used in transactional functions.

#### Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – pointer to `flexspi_handle_t` structure to store the transfer state.
- `callback` – pointer to user callback function.
- `userData` – user parameter passed to the callback function.

`status_t FLEXSPI_TransferNonBlocking(FLEXSPI_Type *base, flexspi_handle_t *handle, flexspi_transfer_t *xfer)`

Performs a interrupt non-blocking transfer on the FLEXSPI bus.

---

**Note:** Calling the API returns immediately after transfer initiates. The user needs to call `FLEXSPI_GetTransferCount` to poll the transfer status to check whether the transfer is finished. If the return status is not `kStatus_FLEXSPI_Busy`, the transfer is finished. For `FLEXSPI_Read`, the `dataSize` should be multiple of rx watermark level, or FLEXSPI could not read data properly.

---

#### Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – pointer to `flexspi_handle_t` structure which stores the transfer state.
- `xfer` – pointer to `flexspi_transfer_t` structure.

#### Return values

- `kStatus_Success` – Successfully start the data transmission.
- `kStatus_FLEXSPI_Busy` – Previous transmission still not finished.

`status_t FLEXSPI_TransferGetCount(FLEXSPI_Type *base, flexspi_handle_t *handle, size_t *count)`

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

#### Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – pointer to `flexspi_handle_t` structure which stores the transfer state.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

#### Return values

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

`void FLEXSPI_TransferAbort(FLEXSPI_Type *base, flexspi_handle_t *handle)`

Aborts an interrupt non-blocking transfer early.

---

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

---

#### Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – pointer to `flexspi_handle_t` structure which stores the transfer state

`void FLEXSPI_TransferHandleIRQ(FLEXSPI_Type *base, flexspi_handle_t *handle)`

Master interrupt handler.

#### Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – pointer to `flexspi_handle_t` structure.

`FSL_FLEXSPI_DRIVER_VERSION`

FLEXSPI driver version.

Status structure of FLEXSPI.

*Values:*

enumerator `kStatus_FLEXSPI_Busy`

FLEXSPI is busy

enumerator `kStatus_FLEXSPI_SequenceExecutionTimeout`

Sequence execution timeout error occurred during FLEXSPI transfer.

enumerator `kStatus_FLEXSPI_IpCommandSequenceError`

IP command Sequence execution timeout error occurred during FLEXSPI transfer.

enumerator `kStatus_FLEXSPI_IpCommandGrantTimeout`

IP command grant timeout error occurred during FLEXSPI transfer.

CMD definition of FLEXSPI, use to form LUT instruction, `_flexspi_command`.

*Values:*

enumerator `kFLEXSPI_Command_STOP`

Stop execution, deassert CS.

enumerator `kFLEXSPI_Command_SDR`

Transmit Command code to Flash, using SDR mode.

enumerator `kFLEXSPI_Command_RADDR_SDR`

Transmit Row Address to Flash, using SDR mode.

enumerator `kFLEXSPI_Command_CADDR_SDR`

Transmit Column Address to Flash, using SDR mode.

enumerator `kFLEXSPI_Command_MODE1_SDR`

Transmit 1-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI\_Command\_MODE2\_SDR  
Transmit 2-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI\_Command\_MODE4\_SDR  
Transmit 4-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI\_Command\_MODE8\_SDR  
Transmit 8-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI\_Command\_WRITE\_SDR  
Transmit Programming Data to Flash, using SDR mode.

enumerator kFLEXSPI\_Command\_READ\_SDR  
Receive Read Data from Flash, using SDR mode.

enumerator kFLEXSPI\_Command\_LEARN\_SDR  
Receive Read Data or Preamble bit from Flash, SDR mode.

enumerator kFLEXSPI\_Command\_DATSZ\_SDR  
Transmit Read/Program Data size (byte) to Flash, SDR mode.

enumerator kFLEXSPI\_Command\_DUMMY\_SDR  
Leave data lines undriven by FlexSPI controller.

enumerator kFLEXSPI\_Command\_DUMMY\_RWDS\_SDR  
Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

enumerator kFLEXSPI\_Command\_DDR  
Transmit Command code to Flash, using DDR mode.

enumerator kFLEXSPI\_Command\_RADDR\_DDR  
Transmit Row Address to Flash, using DDR mode.

enumerator kFLEXSPI\_Command\_CADDR\_DDR  
Transmit Column Address to Flash, using DDR mode.

enumerator kFLEXSPI\_Command\_MODE1\_DDR  
Transmit 1-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI\_Command\_MODE2\_DDR  
Transmit 2-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI\_Command\_MODE4\_DDR  
Transmit 4-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI\_Command\_MODE8\_DDR  
Transmit 8-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI\_Command\_WRITE\_DDR  
Transmit Programming Data to Flash, using DDR mode.

enumerator kFLEXSPI\_Command\_READ\_DDR  
Receive Read Data from Flash, using DDR mode.

enumerator kFLEXSPI\_Command\_LEARN\_DDR  
Receive Read Data or Preamble bit from Flash, DDR mode.

enumerator kFLEXSPI\_Command\_DATSZ\_DDR  
Transmit Read/Program Data size (byte) to Flash, DDR mode.

enumerator kFLEXSPI\_Command\_DUMMY\_DDR  
Leave data lines undriven by FlexSPI controller.

enumerator kFLEXSPI\_Command\_DUMMY\_RWDS\_DDR

Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

enumerator kFLEXSPI\_Command\_JUMP\_ON\_CS

Stop execution, deassert CS and save operand[7:0] as the instruction start pointer for next sequence

enum \_flexspi\_pad

pad definition of FLEXSPI, use to form LUT instruction.

*Values:*

enumerator kFLEXSPI\_1PAD

Transmit command/address and transmit/receive data only through DATA0/DATA1.

enumerator kFLEXSPI\_2PAD

Transmit command/address and transmit/receive data only through DATA[1:0].

enumerator kFLEXSPI\_4PAD

Transmit command/address and transmit/receive data only through DATA[3:0].

enumerator kFLEXSPI\_8PAD

Transmit command/address and transmit/receive data only through DATA[7:0].

enum \_flexspi\_flags

FLEXSPI interrupt status flags.

*Values:*

enumerator kFLEXSPI\_SequenceExecutionTimeoutFlag

Sequence execution timeout.

enumerator kFLEXSPI\_AhbBusTimeoutFlag

AHB Bus timeout.

enumerator kFLEXSPI\_SckStoppedBecauseTxEmptyFlag

SCK is stopped during command sequence because Async TX FIFO empty.

enumerator kFLEXSPI\_SckStoppedBecauseRxFullFlag

SCK is stopped during command sequence because Async RX FIFO full.

enumerator kFLEXSPI\_DataLearningFailedFlag

Data learning failed.

enumerator kFLEXSPI\_IpTxFifoWatermarkEmptyFlag

IP TX FIFO WaterMark empty.

enumerator kFLEXSPI\_IpRxFifoWatermarkAvailableFlag

IP RX FIFO WaterMark available.

enumerator kFLEXSPI\_AhbCommandSequenceErrorFlag

AHB triggered Command Sequences Error.

enumerator kFLEXSPI\_IpCommandSequenceErrorFlag

IP triggered Command Sequences Error.

enumerator kFLEXSPI\_AhbCommandGrantTimeoutFlag

AHB triggered Command Sequences Grant Timeout.

enumerator kFLEXSPI\_IpCommandGrantTimeoutFlag

IP triggered Command Sequences Grant Timeout.

enumerator kFLEXSPI\_IpCommandExecutionDoneFlag

IP triggered Command Sequences Execution finished.

enumerator kFLEXSPI\_AllInterruptFlags

All flags.

enum \_flexspi\_read\_sample\_clock

FLEXSPI sample clock source selection for Flash Reading.

*Values:*

enumerator kFLEXSPI\_ReadSampleClkLoopbackInternally

Dummy Read strobe generated by FlexSPI Controller and loopback internally.

enumerator kFLEXSPI\_ReadSampleClkLoopbackFromDqsPad

Dummy Read strobe generated by FlexSPI Controller and loopback from DQS pad.

enumerator kFLEXSPI\_ReadSampleClkLoopbackFromSckPad

SCK output clock and loopback from SCK pad.

enumerator kFLEXSPI\_ReadSampleClkExternalInputFromDqsPad

Flash provided Read strobe and input from DQS pad.

enum \_flexspi\_cs\_interval\_cycle\_unit

FLEXSPI interval unit for flash device select.

*Values:*

enumerator kFLEXSPI\_CsIntervalUnit1SckCycle

Chip selection interval: CSINTERVAL \* 1 serial clock cycle.

enumerator kFLEXSPI\_CsIntervalUnit256SckCycle

Chip selection interval: CSINTERVAL \* 256 serial clock cycle.

enum \_flexspi\_ahb\_write\_wait\_unit

FLEXSPI AHB wait interval unit for writing.

*Values:*

enumerator kFLEXSPI\_AhbWriteWaitUnit2AhbCycle

AWRWAIT unit is 2 ahb clock cycle.

enumerator kFLEXSPI\_AhbWriteWaitUnit8AhbCycle

AWRWAIT unit is 8 ahb clock cycle.

enumerator kFLEXSPI\_AhbWriteWaitUnit32AhbCycle

AWRWAIT unit is 32 ahb clock cycle.

enumerator kFLEXSPI\_AhbWriteWaitUnit128AhbCycle

AWRWAIT unit is 128 ahb clock cycle.

enumerator kFLEXSPI\_AhbWriteWaitUnit512AhbCycle

AWRWAIT unit is 512 ahb clock cycle.

enumerator kFLEXSPI\_AhbWriteWaitUnit2048AhbCycle

AWRWAIT unit is 2048 ahb clock cycle.

enumerator kFLEXSPI\_AhbWriteWaitUnit8192AhbCycle

AWRWAIT unit is 8192 ahb clock cycle.

enumerator kFLEXSPI\_AhbWriteWaitUnit32768AhbCycle

AWRWAIT unit is 32768 ahb clock cycle.

enum \_flexspi\_ip\_error\_code

Error Code when IP command Error detected.

*Values:*

enumerator kFLEXSPI\_IpCmdErrorNoError

No error.

enumerator kFLEXSPI\_IpCmdErrorJumpOnCsInIpCmd

IP command with JMP\_ON\_CS instruction used.

enumerator kFLEXSPI\_IpCmdErrorUnknownOpCode

Unknown instruction opcode in the sequence.

enumerator kFLEXSPI\_IpCmdErrorSdrDummyInDdrSequence

Instruction DUMMY\_SDR/DUMMY\_RWDS\_SDR used in DDR sequence.

enumerator kFLEXSPI\_IpCmdErrorDdrDummyInSdrSequence

Instruction DUMMY\_DDR/DUMMY\_RWDS\_DDR used in SDR sequence.

enumerator kFLEXSPI\_IpCmdErrorInvalidAddress

Flash access start address exceed the whole flash address range (A1/A2/B1/B2).

enumerator kFLEXSPI\_IpCmdErrorSequenceExecutionTimeout

Sequence execution timeout.

enumerator kFLEXSPI\_IpCmdErrorFlashBoundaryAcrosss

Flash boundary crossed.

enum \_flexspi\_ahb\_error\_code

Error Code when AHB command Error detected.

*Values:*

enumerator kFLEXSPI\_AhbCmdErrorNoError

No error.

enumerator kFLEXSPI\_AhbCmdErrorJumpOnCsInWriteCmd

AHB Write command with JMP\_ON\_CS instruction used in the sequence.

enumerator kFLEXSPI\_AhbCmdErrorUnknownOpCode

Unknown instruction opcode in the sequence.

enumerator kFLEXSPI\_AhbCmdErrorSdrDummyInDdrSequence

Instruction DUMMY\_SDR/DUMMY\_RWDS\_SDR used in DDR sequence.

enumerator kFLEXSPI\_AhbCmdErrorDdrDummyInSdrSequence

Instruction DUMMY\_DDR/DUMMY\_RWDS\_DDR used in SDR sequence.

enumerator kFLEXSPI\_AhbCmdSequenceExecutionTimeout

Sequence execution timeout.

enum \_flexspi\_port

FLEXSPI operation port select.

*Values:*

enumerator kFLEXSPI\_PortA1

Access flash on A1 port.

enumerator kFLEXSPI\_PortA2

Access flash on A2 port.

enumerator kFLEXSPI\_PortB1

Access flash on B1 port.

enumerator kFLEXSPI\_PortB2

Access flash on B2 port.

enumerator kFLEXSPI\_PortCount

enum \_flexspi\_arb\_command\_source

Trigger source of current command sequence granted by arbitrator.

*Values:*

enumerator kFLEXSPI\_AhbReadCommand

enumerator kFLEXSPI\_AhbWriteCommand

enumerator kFLEXSPI\_IpCommand

enumerator kFLEXSPI\_SuspendedCommand

enum \_flexspi\_command\_type

Command type.

*Values:*

enumerator kFLEXSPI\_Command

FlexSPI operation: Only command, both TX and Rx buffer are ignored.

enumerator kFLEXSPI\_Config

FlexSPI operation: Configure device mode, the TX fifo size is fixed in LUT.

enumerator kFLEXSPI\_Read

enumerator kFLEXSPI\_Write

typedef enum \_flexspi\_pad flexspi\_pad\_t

pad definition of FLEXSPI, use to form LUT instruction.

typedef enum \_flexspi\_flags flexspi\_flags\_t

FLEXSPI interrupt status flags.

typedef enum \_flexspi\_read\_sample\_clock flexspi\_read\_sample\_clock\_t

FLEXSPI sample clock source selection for Flash Reading.

typedef enum \_flexspi\_cs\_interval\_cycle\_unit flexspi\_cs\_interval\_cycle\_unit\_t

FLEXSPI interval unit for flash device select.

typedef enum \_flexspi\_ahb\_write\_wait\_unit flexspi\_ahb\_write\_wait\_unit\_t

FLEXSPI AHB wait interval unit for writing.

typedef enum \_flexspi\_ip\_error\_code flexspi\_ip\_error\_code\_t

Error Code when IP command Error detected.

typedef enum \_flexspi\_ahb\_error\_code flexspi\_ahb\_error\_code\_t

Error Code when AHB command Error detected.

typedef enum \_flexspi\_port flexspi\_port\_t

FLEXSPI operation port select.

typedef enum \_flexspi\_arb\_command\_source flexspi\_arb\_command\_source\_t

Trigger source of current command sequence granted by arbitrator.

typedef enum \_flexspi\_command\_type flexspi\_command\_type\_t

Command type.

typedef struct \_flexspi\_ahbBuffer\_config flexspi\_ahbBuffer\_config\_t

typedef struct \_flexspi\_config flexspi\_config\_t

FLEXSPI configuration structure.



```
typedef struct _flexspi_device_config flexspi_device_config_t
    External device configuration items.

typedef struct _flexspi_transfer flexspi_transfer_t
    Transfer structure for FLEXSPI.

typedef struct _flexspi_handle flexspi_handle_t

typedef void (*flexspi_transfer_callback_t)(FLEXSPI_Type *base, flexspi_handle_t *handle,
status_t status, void *userData)
    FLEXSPI transfer callback function.

typedef struct _flexspi_addr_map_config flexspi_addr_map_config_t
    Address mapping configuration structure.

FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNT

FLEXSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)
    Formula to form FLEXSPI instructions in LUT table.

struct _flexspi_ahbBuffer_config
    #include <fsl_flexspi.h>
```

### Public Members

```
uint8_t priority
    This priority for AHB Master Read which this AHB RX Buffer is assigned.

uint8_t masterIndex
    AHB Master ID the AHB RX Buffer is assigned.

uint16_t bufferSize
    AHB buffer size in byte.

bool enablePrefetch
    AHB Read Prefetch Enable for current AHB RX Buffer corresponding Master, allows
    prefetch disable/enable separately for each master.

struct _flexspi_config
    #include <fsl_flexspi.h> FLEXSPI configuration structure.
```

### Public Members

```
flexspi_read_sample_clock_t rxSampleClock
    Sample Clock source selection for Flash Reading.

bool enableSckFreeRunning
    Enable/disable SCK output free-running.

bool enableDoze
    Enable/disable doze mode support.

bool enableHalfSpeedAccess
    Enable/disable divide by 2 of the clock for half speed commands.

bool enableSckBDiffOpt
    Enable/disable SCKB pad use as SCKA differential clock output, when enable, Port B
    flash access is not available.
```

bool enableSameConfigForAll

Enable/disable same configuration for all connected devices when enabled, same configuration in FLASHA1CRx is applied to all.

uint16\_t seqTimeoutCycle

Timeout wait cycle for command sequence execution, timeout after ahbGrantTimeoutCycle\*1024 serial root clock cycles.

uint8\_t ipGrantTimeoutCycle

Timeout wait cycle for IP command grant, timeout after ipGrantTimeoutCycle\*1024 AHB clock cycles.

uint8\_t txWatermark

FLEXSPI IP transmit watermark value.

uint8\_t rxWatermark

FLEXSPI receive watermark value.

struct \_\_flexspi\_device\_config

*#include <fsl\_flexspi.h>* External device configuration items.

### Public Members

uint32\_t flexspiRootClk

FLEXSPI serial root clock.

bool isSck2Enabled

FLEXSPI use SCK2.

uint32\_t flashSize

Flash size in KByte.

bool addressShift

Address shift.

*flexspi\_cs\_interval\_cycle\_unit\_t* CSIntervalUnit

CS interval unit, 1 or 256 cycle.

uint16\_t CSInterval

CS line assert interval, multiply CS interval unit to get the CS line assert interval cycles.

uint8\_t CSHoldTime

CS line hold time.

uint8\_t CSSetupTime

CS line setup time.

uint8\_t dataValidTime

Data valid time for external device.

uint8\_t columnSpace

Column space size.

bool enableWordAddress

If enable word address.

uint8\_t AWRSeqIndex

Sequence ID for AHB write command.

uint8\_t AWRSeqNumber

Sequence number for AHB write command.

uint8\_t ARDSeqIndex

Sequence ID for AHB read command.

uint8\_t ARDSeqNumber

Sequence number for AHB read command.

*flexspi\_ahb\_write\_wait\_unit\_t* AHBWriteWaitUnit

AHB write wait unit.

uint16\_t AHBWriteWaitInterval

AHB write wait interval, multiply AHB write interval unit to get the AHB write wait cycles.

bool enableWriteMask

Enable/Disable FLEXSPI drive DQS pin as write mask when writing to external device.

struct *\_flexspi\_transfer*

*#include <fsl\_flexspi.h>* Transfer structure for FLEXSPI.

### Public Members

uint32\_t deviceAddress

Operation device address.

*flexspi\_port\_t* port

Operation port.

*flexspi\_command\_type\_t* cmdType

Execution command type.

uint8\_t seqIndex

Sequence ID for command.

uint8\_t SeqNumber

Sequence number for command.

uint32\_t \*data

Data buffer.

size\_t dataSize

Data size in bytes.

struct *\_flexspi\_handle*

*#include <fsl\_flexspi.h>* Transfer handle structure for FLEXSPI.

### Public Members

uint32\_t state

Internal state for FLEXSPI transfer

uint8\_t \*data

Data buffer.

size\_t dataSize

Remaining Data size in bytes.

size\_t transferTotalSize

Total Data size in bytes.

*flexspi\_transfer\_callback\_t* completionCallback

Callback for users while transfer finish or error occurred

void \*userData

FLEXSPI callback function parameter.

struct *\_flexspi\_addr\_map\_config*

*#include <fsl\_flexspi.h>* Address mapping configuration structure.

### Public Members

uint32\_t addrStart

Remapping start address.

uint32\_t addrEnd

Remapping end address.

uint32\_t addrOffset

Address offset.

bool remapEnable

Enable address remapping.

struct *ahbConfig*

### Public Members

uint8\_t ahbGrantTimeoutCycle

Timeout wait cycle for AHB command grant, timeout after ahbGrantTimeoutCycle\*1024 AHB clock cycles.

uint16\_t ahbBusTimeoutCycle

Timeout wait cycle for AHB read/write access, timeout after ahbBusTimeoutCycle\*1024 AHB clock cycles.

uint8\_t resumeWaitCycle

Wait cycle for idle state before suspended command sequence resume, timeout after ahbBusTimeoutCycle AHB clock cycles.

*flexspi\_ahbBuffer\_config\_t* buffer[FSL\_FEATURE\_FLEXSPI\_AHB\_BUFFER\_COUNTn(0)]

AHB buffer size.

bool enableClearAHBBufferOpt

Enable/disable automatically clean AHB RX Buffer and TX Buffer when FLEXSPI returns STOP mode ACK.

bool enableReadAddressOpt

Enable/disable remove AHB read burst start address alignment limitation. when enable, there is no AHB read burst start address alignment limitation.

bool enableAHBPrefetch

Enable/disable AHB read prefetch feature, when enabled, FLEXSPI will fetch more data than current AHB burst.

bool enableAHBBufferable

Enable/disable AHB bufferable write access support, when enabled, FLEXSPI return before waiting for command execution finished.

bool enableAHBCachable

Enable AHB bus cachable read access support.

## 2.17 FLEXSPI DMA Driver

```
void FLEXSPI_TransferCreateHandleDMA(FLEXSPI_Type *base, flexspi_dma_handle_t *handle,
                                     flexspi_dma_callback_t callback, void *userData,
                                     dma_handle_t *txDmaHandle, dma_handle_t
                                     *rxDmaHandle)
```

Initializes the FLEXSPI handle for transfer which is used in transactional functions and set the callback.

### Parameters

- base – FLEXSPI peripheral base address
- handle – Pointer to flexspi\_dma\_handle\_t structure
- callback – FLEXSPI callback, NULL means no callback.
- userData – User callback function data.
- txDmaHandle – User requested DMA handle for TX DMA transfer.
- rxDmaHandle – User requested DMA handle for RX DMA transfer.

```
void FLEXSPI_TransferUpdateSizeDMA(FLEXSPI_Type *base, flexspi_dma_handle_t *handle,
                                   flexspi_dma_transfer_nsize_t nsize)
```

Update FLEXSPI DMA transfer source data transfer size(SSIZE) and destination data transfer size(DSIZE).

### See also:

flexspi\_dma\_transfer\_nsize\_t.

### Parameters

- base – FLEXSPI peripheral base address
- handle – Pointer to flexspi\_dma\_handle\_t structure
- nsize – FLEXSPI DMA transfer data transfer size(SSIZE/DSIZE), by default the size is kFLEXPSI\_DMAnSize1Bytes(one byte).

```
status_t FLEXSPI_TransferDMA(FLEXSPI_Type *base, flexspi_dma_handle_t *handle,
                             flexspi_transfer_t *xfer)
```

Transfers FLEXSPI data using an dma non-blocking method.

This function writes/receives data to/from the FLEXSPI transmit/receive FIFO. This function is non-blocking.

### Parameters

- base – FLEXSPI peripheral base address.
- handle – Pointer to flexspi\_dma\_handle\_t structure
- xfer – FLEXSPI transfer structure.

### Return values

- kStatus\_FLEXSPI\_Busy – FLEXSPI is busy transfer.
- kStatus\_InvalidArgument – The watermark configuration is invalid, the watermark should be power of 2 to do successfully DMA transfer.
- kStatus\_Success – FLEXSPI successfully start dma transfer.

`void FLEXSPI__TransferAbortDMA(FLEXSPI_Type *base, flexspi_dma_handle_t *handle)`

Aborts the transfer data using dma.

This function aborts the transfer data using dma.

#### Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – Pointer to `flexspi_dma_handle_t` structure

`status_t FLEXSPI__TransferGetTransferCountDMA(FLEXSPI_Type *base, flexspi_dma_handle_t *handle, size_t *count)`

Gets the transferred counts of transfer.

#### Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – Pointer to `flexspi_dma_handle_t` structure.
- `count` – Bytes transfer.

#### Return values

- `kStatus_Success` – Succeed get the transfer count.
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

`FSL_FLEXSPI_DMA_DRIVER_VERSION`

FLEXSPI DMA driver version 2.2.1.

`enum _flexspi_dma_ntransfer_size`

dma transfer configuration

*Values:*

enumerator `kFLEXPSI_DMAnSize1Bytes`

Source/Destination data transfer size is 1 byte every time

enumerator `kFLEXPSI_DMAnSize2Bytes`

Source/Destination data transfer size is 2 bytes every time

enumerator `kFLEXPSI_DMAnSize4Bytes`

Source/Destination data transfer size is 4 bytes every time

`typedef struct _flexspi_dma_handle flexspi_dma_handle_t`

`typedef void (*flexspi_dma_callback_t)(FLEXSPI_Type *base, flexspi_dma_handle_t *handle, status_t status, void *userData)`

FLEXSPI dma transfer callback function for finish and error.

`typedef enum _flexspi_dma_ntransfer_size flexspi_dma_transfer_nsize_t`

dma transfer configuration

`struct _flexspi_dma_handle`

`#include <fsl_flexspi_dma.h>` FLEXSPI DMA transfer handle, users should not touch the content of the handle.

#### Public Members

`dma_handle_t *txDmaHandle`

dma handler for FLEXSPI Tx.

*dma\_handle\_t* \*rxDmaHandle  
dma handler for FLEXSPI Rx.

size\_t transferSize  
Bytes need to transfer.

*flexspi\_dma\_transfer\_nsize\_t* nsize  
dma SSIZE/DSIZE in each transfer.

uint8\_t nbytes  
dma minor byte transfer count initially configured.

uint8\_t count  
The transfer data count in a DMA request.

uint32\_t state  
Internal state for FLEXSPI dma transfer.

*flexspi\_dma\_callback\_t* completionCallback  
A callback function called after the dma transfer is finished.

void \*userData  
User callback parameter

## 2.18 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 FREQME\_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

## 2.19 Hashcrypt: The Cryptographic Accelerator

### 2.20 Hashcrypt Background HASH

```
void HASHCRYPT_SHA_SetCallback(HASHCRYPT_Type *base, hashcrypt_hash_ctx_t *ctx,  
                              hashcrypt_callback_t callback, void *userData)
```

Initializes the HASHCRYPT 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 – HASHCRYPT 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 HASHCRYPT_SHA_UpdateNonBlocking(HASHCRYPT_Type *base, hashcrypt_hash_ctx_t  
                                         *ctx, const uint8_t *input, size_t inputSize)
```

Create running hash on given data.

Configures the HASHCRYPT to compute new running hash as AHB master and returns immediately. HASHCRYPT 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 HASHCRYPT\_SHA\_Init() and finished with HASHCRYPT\_SHA\_Finish(). Once callback function is invoked by HASHCRYPT isr, it should set a flag for the main application to finalize the hashing (padding) and to read out the final digest by calling HASHCRYPT\_SHA\_Finish().

#### Parameters

- base – HASHCRYPT 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.



## 2.21 Hashcrypt common functions

FSL\_HASHCRYPT\_DRIVER\_VERSION

HASHCRYPT driver version. Version 2.2.16.

Current version: 2.2.16

Change log:

- Version 2.0.0
  - Initial version
- Version 2.0.1
  - Support loading AES key from unaligned address
- Version 2.0.2
  - Support loading AES key from unaligned address for different compiler and core variants
- Version 2.0.3
  - Remove SHA512 and AES ICB algorithm definitions
- Version 2.0.4
  - Add SHA context switch support
- Version 2.1.0
  - Update the register name and macro to align with new header.
- Version 2.1.1
  - Fix MISRA C-2012.
- Version 2.1.2
  - Support loading AES input data from unaligned address.
- Version 2.1.3
  - Fix MISRA C-2012.
- Version 2.1.4
  - Fix context switch cannot work when switching from AES.
- Version 2.1.5
  - Add data synchronization barrier inside hashcrypt\_sha\_ldm\_stm\_16\_words() to prevent possible optimization issue.
- Version 2.2.0
  - Add AES-OFB and AES-CFB mixed IP/SW modes.
- Version 2.2.1
  - Add data synchronization barrier inside hashcrypt\_sha\_ldm\_stm\_16\_words() prevent compiler from reordering memory write when -O2 or higher is used.
- Version 2.2.2
  - Add data synchronization barrier inside hashcrypt\_sha\_ldm\_stm\_16\_words() to fix optimization issue
- Version 2.2.3
  - Added check for size in hashcrypt\_aes\_one\_block to prevent overflowing COUNT field in MEMCTRL register, if its bigger than COUNT field do a multiple runs.

- Version 2.2.4
  - In all HASHCRYPT\_AES\_xx functions have been added setting CTRL\_MODE bitfield to 0 after processing data, which decreases power consumption.
- Version 2.2.5
  - Add data synchronization barrier and instruction synchronization barrier inside hashcrypt\_sha\_process\_message\_data() to fix optimization issue
- Version 2.2.6
  - Add data synchronization barrier inside HASHCRYPT\_SHA\_Update() and hashcrypt\_get\_data() function to fix optimization issue on MDK and ARMGCC release targets
- Version 2.2.7
  - Add data synchronization barrier inside HASHCRYPT\_SHA\_Update() to fix optimization issue on MCUX IDE release target
- Version 2.2.8
  - Unify hashcrypt hashing behavior between aligned and unaligned input data
- Version 2.2.9
  - Add handling of set ERROR bit in the STATUS register
- Version 2.2.10
  - Fix missing error statement in hashcrypt\_save\_running\_hash()
- Version 2.2.11
  - Fix incorrect SHA-256 calculation for long messages with reload
- Version 2.2.12
  - Fix hardfault issue on the Keil compiler due to unaligned memcpy() input on some optimization levels
- Version 2.2.13
  - Added function hashcrypt\_seed\_prng() which loading random number into PRNG\_SEED register before AES operation for SCA protection
- Version 2.2.14
  - Modify function hashcrypt\_get\_data() to prevent issue with unaligned access
- Version 2.2.15
  - Add wait on DIGEST BIT inside hashcrypt\_sha\_one\_block() to fix issues with some optimization flags
- Version 2.2.16
  - Add DSB instruction inside hashcrypt\_sha\_ldm\_stm\_16\_words() to fix issues with some optimization flags

enum \_hashcrypt\_algo\_t

Algorithm used for Hashcrypt operation.

*Values:*

enumerator kHASHCRYPT\_Sha1  
SHA\_1

enumerator kHASHCRYPT\_Sha256  
SHA\_256

enumerator kHASHCRYPT\_\_Aes  
AES

typedef enum *\_hashcrypt\_algo\_t* hashcrypt\_algo\_t  
Algorithm used for Hashcrypt operation.

void HASHCRYPT\_\_Init(HASHCRYPT\_Type \*base)  
Enables clock and disables reset for HASHCRYPT peripheral.  
Enable clock and disable reset for HASHCRYPT.

#### Parameters

- base – HASHCRYPT base address

void HASHCRYPT\_\_Deinit(HASHCRYPT\_Type \*base)  
Disables clock for HASHCRYPT peripheral.  
Disable clock and enable reset.

#### Parameters

- base – HASHCRYPT base address

HASHCRYPT\_\_MODE\_SHA1

Algorithm definitions correspond with the values for Mode field in Control register !

HASHCRYPT\_\_MODE\_SHA256

HASHCRYPT\_\_MODE\_AES

## 2.22 Hashcrypt AES

enum *\_hashcrypt\_aes\_mode\_t*  
AES mode.

*Values:*

enumerator kHASHCRYPT\_\_AesEcb  
AES ECB mode

enumerator kHASHCRYPT\_\_AesCbc  
AES CBC mode

enumerator kHASHCRYPT\_\_AesCtr  
AES CTR mode

enum *\_hashcrypt\_aes\_keysize\_t*  
Size of AES key.

*Values:*

enumerator kHASHCRYPT\_\_Aes128  
AES 128 bit key

enumerator kHASHCRYPT\_\_Aes192  
AES 192 bit key

enumerator kHASHCRYPT\_\_Aes256  
AES 256 bit key

enumerator kHASHCRYPT\_\_InvalidKey  
AES invalid key

enum *\_hashcrypt\_key*

HASHCRYPT key source selection.

*Values:*

enumerator *kHASHCRYPT\_UserKey*

HASHCRYPT user key

enumerator *kHASHCRYPT\_SecretKey*

HASHCRYPT secret key (dedicated hw bus from PUF)

typedef enum *\_hashcrypt\_aes\_mode\_t* *hashcrypt\_aes\_mode\_t*

AES mode.

typedef enum *\_hashcrypt\_aes\_keysize\_t* *hashcrypt\_aes\_keysize\_t*

Size of AES key.

typedef enum *\_hashcrypt\_key* *hashcrypt\_key\_t*

HASHCRYPT key source selection.

typedef struct *\_hashcrypt\_handle* *hashcrypt\_handle\_t*

struct *\_hashcrypt\_handle* \_\_attribute\_\_((aligned))

*status\_t* HASHCRYPT\_AES\_SetKey(HASHCRYPT\_Type \*base, *hashcrypt\_handle\_t* \*handle,  
const uint8\_t \*key, size\_t keySize)

Set AES key to *hashcrypt\_handle\_t* struct and optionally to HASHCRYPT.

Sets the AES key for encryption/decryption with the *hashcrypt\_handle\_t* structure. The *hashcrypt\_handle\_t* input argument specifies key source.

#### Parameters

- base – HASHCRYPT peripheral base address.
- handle – Handle used for the request.
- key – 0-mod-4 aligned pointer to AES key.
- keySize – AES key size in bytes. Shall equal 16, 24 or 32.

#### Returns

status from set key operation

*status\_t* HASHCRYPT\_AES\_EncryptEcb(HASHCRYPT\_Type \*base, *hashcrypt\_handle\_t* \*handle,  
const uint8\_t \*plaintext, uint8\_t \*ciphertext, size\_t  
size)

Encrypts AES on one or multiple 128-bit block(s).

Encrypts AES. The source plaintext and destination ciphertext can overlap in system memory.

#### Parameters

- base – HASHCRYPT peripheral base address
- handle – Handle used for this request.
- 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 HASHCRYPT_AES_DecryptEcb(HASHCRYPT_Type *base, hashcrypt_handle_t *handle,  
                                const uint8_t *ciphertext, uint8_t *plaintext, size_t  
                                size)
```

Decrypts AES on one or multiple 128-bit block(s).

Decrypts AES. The source ciphertext and destination plaintext can overlap in system memory.

#### Parameters

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

#### Returns

Status from decrypt operation

```
status_t HASHCRYPT_AES_EncryptCbc(HASHCRYPT_Type *base, hashcrypt_handle_t *handle,  
                                const uint8_t *plaintext, uint8_t *ciphertext, size_t  
                                size, const uint8_t iv[16])
```

Encrypts AES using CBC block mode.

#### Parameters

- base – HASHCRYPT peripheral base address
- handle – Handle used for this request.
- 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 HASHCRYPT_AES_DecryptCbc(HASHCRYPT_Type *base, hashcrypt_handle_t *handle,  
                                const uint8_t *ciphertext, uint8_t *plaintext, size_t  
                                size, const uint8_t iv[16])
```

Decrypts AES using CBC block mode.

#### Parameters

- base – HASHCRYPT peripheral base address
- handle – Handle used for this request.
- 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 HASHCRYPT_AES_CryptCtr(HASHCRYPT_Type *base, hashcrypt_handle_t *handle,  
                                const uint8_t *input, uint8_t *output, size_t size, uint8_t  
                                counter[16U], uint8_t counterlast[16U], 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 – HASHCRYPT peripheral base address
- handle – Handle used for this request.
- 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 (statefull encryption). 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 encrypt operation

```
status_t HASHCRYPT_AES_CryptOfb(HASHCRYPT_Type *base, hashcrypt_handle_t *handle,  
                                const uint8_t *input, uint8_t *output, size_t size, const  
                                uint8_t iv[16U])
```

Encrypts or decrypts AES using OFB block mode.

Encrypts or decrypts AES using OFB block mode. AES OFB 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 – HASHCRYPT peripheral base address
- handle – Handle used for this request.
- input – Input data for OFB block mode
- output – **[out]** Output data for OFB block mode
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first input block.

#### Returns

Status from encrypt operation

```
status_t HASHCRYPT_AES_EncryptCfb(HASHCRYPT_Type *base, hashcrypt_handle_t *handle,  
                                   const uint8_t *plaintext, uint8_t *ciphertext, size_t size,  
                                   const uint8_t iv[16])
```

Encrypts AES using CFB block mode.

#### Parameters

- `base` – HASHCRYPT peripheral base address
- `handle` – Handle used for this request.
- `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 HASHCRYPT_AES_DecryptCfb(HASHCRYPT_Type *base, hashcrypt_handle_t *handle,
                                   const uint8_t *ciphertext, uint8_t *plaintext, size_t size,
                                   const uint8_t iv[16])
```

Decrypts AES using CFB block mode.

**Parameters**

- `base` – HASHCRYPT peripheral base address
- `handle` – Handle used for this request.
- `ciphertext` – Input cipher text to decrypt
- `plaintext` – **[out]** Output plaintext 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

```
HASHCRYPT_AES_BLOCK_SIZE
```

AES block size in bytes

```
AES_ENCRYPT
```

```
AES_DECRYPT
```

```
struct _hashcrypt_handle
```

*#include <fsl\_hashcrypt.h>* Specify HASHCRYPT's key resource.

**Public Members**

```
uint32_t keyWord[8]
```

Copy of user key (set by HASHCRYPT\_AES\_SetKey()).

```
hashcrypt_key_t keyType
```

For operations with key (such as AES encryption/decryption), specify key type.

## 2.23 Hashcrypt HASH

```
typedef struct _hashcrypt_hash_ctx_t hashcrypt_hash_ctx_t
```

Storage type used to save hash context.

```
typedef void (*hashcrypt_callback_t)(HASHCRYPT_Type *base, hashcrypt_hash_ctx_t *ctx,
                                     status_t status, void *userData)
```

HASHCRYPT background hash callback function.

```
status_t HASHCRYPT_SHA(HASHCRYPT_Type *base, hashcrypt_algo_t algo, const uint8_t
    *input, size_t inputSize, uint8_t *output, size_t *outputSize)
```

Create HASH on given data.

Perform the full SHA in one function call. The function is blocking.

#### Parameters

- base – HASHCRYPT peripheral base address
- algo – Underlying algorithm to use for hash computation.
- input – Input data
- inputSize – Size of input data in bytes
- output – **[out]** Output hash data
- outputSize – **[out]** Output parameter storing the size of the output hash in bytes

#### Returns

Status of the one call hash operation.

```
status_t HASHCRYPT_SHA_Init(HASHCRYPT_Type *base, hashcrypt_hash_ctx_t *ctx,
    hashcrypt_algo_t algo)
```

Initialize HASH context.

This function initializes the HASH.

#### Parameters

- base – HASHCRYPT peripheral base address
- ctx – **[out]** Output hash context
- algo – Underlying algorithm to use for hash computation.

#### Returns

Status of initialization

```
status_t HASHCRYPT_SHA_Update(HASHCRYPT_Type *base, hashcrypt_hash_ctx_t *ctx, const
    uint8_t *input, size_t inputSize)
```

Add data to current HASH.

Add data to current HASH. This can be called repeatedly with an arbitrary amount of data to be hashed. The functions blocks. If it returns kStatus\_Success, the running hash has been updated (HASHCRYPT has processed the input data), so the memory at *input* pointer can be released back to system. The HASHCRYPT context buffer is updated with the running hash and with all necessary information to support possible context switch.

#### Parameters

- base – HASHCRYPT peripheral base address
- ctx – **[inout]** HASH context
- input – Input data
- inputSize – Size of input data in bytes

#### Returns

Status of the hash update operation

```
status_t HASHCRYPT_SHA_Finish(HASHCRYPT_Type *base, hashcrypt_hash_ctx_t *ctx, uint8_t
    *output, size_t *outputSize)
```

Finalize hashing.

Outputs the final hash (computed by HASHCRYPT\_HASH\_Update()) and erases the context.



**Parameters**

- base – HASHCRYPT peripheral base address
- ctx – **[inout]** Input hash context
- output – **[out]** Output hash data
- outputSize – **[inout]** Optional parameter (can be passed as NULL). On function entry, it specifies the size of output[] buffer. On function return, it stores the number of updated output bytes.

**Returns**

Status of the hash finish operation

HASHCRYPT\_HASH\_CTX\_SIZE

HASHCRYPT HASH Context size.

struct \_hashcrypt\_hash\_ctx\_t

*#include <fsl\_hashcrypt.h>* Storage type used to save hash context.

**Public Members**

uint32\_t x[30]

storage

## 2.24 I2C: Inter-Integrated Circuit Driver

### 2.25 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.

## 2.26 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`

## 2.27 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.

## 2.28 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 call-back.

**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.

## 2.29 I2S: I2S Driver

## 2.30 I2S\_BRIDGE: I2S bridging and signal sharing configuration

`enum _i2s_bridge_share_set_index`  
I2S Bridge share set.

*Values:*

`enumerator kI2S_BRIDGE_OriginalSignal`  
Original FLEXCOMM I2S signals

`enumerator kI2S_BRIDGE_ShareSet0`  
share set 0 signals

`enumerator kI2S_BRIDGE_ShareSet1`  
share set 1 signals

`enum _i2s_bridge_signal`  
I2S signal.

*Values:*

`enumerator kI2S_BRIDGE_SignalSCK`  
SCK signal

`enumerator kI2S_BRIDGE_SignalWS`  
WS signal

`enumerator kI2S_BRIDGE_SignalDataIn`  
Data in signal

`enumerator kI2S_BRIDGE_SignalDataOut`  
Data out signal

`enum _i2s_bridge_share_src`  
I2S signal source.

*Values:*

`enumerator kI2S_BRIDGE_Flexcomm0`  
Shared signal comes from FLEXCOMM0

`enumerator kI2S_BRIDGE_Flexcomm1`  
Shared signal comes from FLEXCOMM1

enumerator kI2S\_BRIDGE\_Flexcomm2

Shared signal comes from FLEXCOMM2

enumerator kI2S\_BRIDGE\_Flexcomm3

Shared signal comes from FLEXCOMM3

enumerator kI2S\_BRIDGE\_Flexcomm4

Shared signal comes from FLEXCOMM4

enumerator kI2S\_BRIDGE\_Flexcomm5

Shared signal comes from FLEXCOMM5

enumerator kI2S\_BRIDGE\_Flexcomm6

Shared signal comes from FLEXCOMM6

enumerator kI2S\_BRIDGE\_Flexcomm7

Shared signal comes from FLEXCOMM7

enum \_i2s\_bridge\_dataout\_mask

I2S Bridge shared data out mask.

*Values:*

enumerator kI2S\_BRIDGE\_Flexcomm0DataOut

FLEXCOMM0 DATAOUT Output Enable

enumerator kI2S\_BRIDGE\_Flexcomm1DataOut

FLEXCOMM1 DATAOUT Output Enable

enumerator kI2S\_BRIDGE\_Flexcomm2DataOut

FLEXCOMM2 DATAOUT Output Enable

enumerator kI2S\_BRIDGE\_Flexcomm3DataOut

FLEXCOMM3 DATAOUT Output Enable

enumerator kI2S\_BRIDGE\_Flexcomm4DataOut

FLEXCOMM4 DATAOUT Output Enable

enumerator kI2S\_BRIDGE\_Flexcomm5DataOut

FLEXCOMM5 DATAOUT Output Enable

enumerator kI2S\_BRIDGE\_Flexcomm6DataOut

FLEXCOMM6 DATAOUT Output Enable

enumerator kI2S\_BRIDGE\_Flexcomm7DataOut

FLEXCOMM7 DATAOUT Output Enable

typedef enum \_i2s\_bridge\_signal i2s\_bridge\_signal\_t

I2S signal.

FSL\_I2S\_BRIDGE\_DRIVER\_VERSION

Group I2S Bridge driver version for SDK.

Version 2.0.0.

void I2S\_BRIDGE\_SetFlexcommShareSet(uint32\_t flexCommIndex, uint32\_t sckSet, uint32\_t wsSet, uint32\_t dataInSet, uint32\_t dataOutSet)

I2S Bridge share set selection for flexcomm instance.

#### Parameters

- flexCommIndex – index of flexcomm, refer to RM for supported FLEXCOMM instances.
- sckSet – share set for sck, refer to \_i2s\_bridge\_share\_set\_index

- wsSet – share set for ws, refer to `_i2s_bridge_share_set_index`
- dataInSet – share set for data in, refer to `_i2s_bridge_share_set_index`
- dataOutSet – share set for data out, refer to `_i2s_bridge_share_set_index`

```
void I2S__BRIDGE_SetFlexcommSignalShareSet(uint32_t flexCommIndex, i2s_bridge_signal_t
                                          signal, uint32_t set)
```

I2S Bridge share set selection for a separate signal.

#### Parameters

- flexCommIndex – index of flexcomm, refer to RM for supported FLEXCOMM instances.
- signal – The signal need to be configured.
- set – share set for the signal, refer to `_i2s_bridge_share_set_index`

```
void I2S__BRIDGE_SetShareSetSrc(uint32_t setIndex, uint32_t sckShareSrc, uint32_t wsShareSrc,
                                uint32_t dataInShareSrc, uint32_t dataOutShareSrc)
```

I2S Bridge share set source configure.

#### Parameters

- setIndex – index of share set, refer `_i2s_bridge_share_set_index`
- sckShareSrc – sck source for this share set, refer to `_i2s_bridge_share_src`
- wsShareSrc – ws source for this share set, refer to `_i2s_bridge_share_src`
- dataInShareSrc – data in source for this share set, refer to `_i2s_bridge_share_src`
- dataOutShareSrc – data out source for this share set, refer to `_i2s_bridge_dataout_mask`

```
void I2S__BRIDGE_SetShareSignalSrc(uint32_t setIndex, i2s_bridge_signal_t signal, uint32_t
                                   shareSrc)
```

I2S Bridge shared signal source selection for a share set.

#### Parameters

- setIndex – index of share set, refer to `_i2s_bridge_share_set_index`
- signal – the shared signal to be configured
- shareSrc – the signal selection, refer to `_i2s_bridge_share_src`.

## 2.31 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 tcdds)

Invoked from DMA interrupt handler.

#### Parameters

- handle – pointer to DMA handle structure.
- userData – argument for user callback.

- transferDone – if transfer was done.
- teds –

```
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

## 2.32 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;
```

(continues on next page)

(continued from previous page)

```
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 &lt;fsl\_i2s.h&gt; 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

## 2.33 I3C: I3C Driver

FSL\_I3C\_DRIVER\_VERSION

I3C driver version.

I3C status return codes.

*Values:*

enumerator kStatus\_I3C\_Busy

The master is already performing a transfer.

enumerator kStatus\_I3C\_Idle

The slave driver is idle.

enumerator kStatus\_I3C\_Nak

The slave device sent a NAK in response to an address.

enumerator kStatus\_I3C\_WriteAbort

The slave device sent a NAK in response to a write.

enumerator kStatus\_I3C\_Term

The master terminates slave read.

enumerator kStatus\_I3C\_HdrParityError

Parity error from DDR read.

enumerator kStatus\_I3C\_CrcError

CRC error from DDR read.

enumerator kStatus\_I3C\_ReadFifoError

Read from M/SRDATAB register when FIFO empty.

enumerator kStatus\_I3C\_WriteFifoError

Write to M/SWDATAB register when FIFO full.

enumerator kStatus\_I3C\_MsgError

Message SDR/DDR mismatch or read/write message in wrong state

enumerator kStatus\_I3C\_InvalidReq

Invalid use of request.

enumerator kStatus\_I3C\_Timeout

The module has stalled too long in a frame.

enumerator kStatus\_I3C\_SlaveCountExceed

The I3C slave count has exceed the definition in I3C\_MAX\_DEVCNT.

enumerator kStatus\_I3C\_IBIWon

The I3C slave event IBI or MR or HJ won the arbitration on a header address.

enumerator kStatus\_I3C\_OverflowError

Slave internal from-bus buffer/FIFO overrun.

enumerator kStatus\_I3C\_UnderrunError

Slave internal to-bus buffer/FIFO underrun

enumerator kStatus\_I3C\_UnderrunNak

Slave internal from-bus buffer/FIFO underrun and NACK error



```
enumerator kStatus_I3C_InvalidStart
    Slave invalid start flag
enumerator kStatus_I3C_SdrParityError
    SDR parity error
enumerator kStatus_I3C_S0S1Error
    S0 or S1 error
enum _i3c_hdr_mode
    I3C HDR modes.
    Values:
enumerator kI3C_HDRModeNone
enumerator kI3C_HDRModeDDR
enumerator kI3C_HDRModeTSP
enumerator kI3C_HDRModeTSL
typedef enum _i3c_hdr_mode i3c_hdr_mode_t
    I3C HDR modes.
typedef struct _i3c_device_info i3c_device_info_t
    I3C device information.
I3C_RETRY_TIMES
    Timeout times for waiting flag.
I3C_MAX_DEVCNT
I3C_IBI_BUFF_SIZE
struct _i3c_device_info
    #include <fsl_i3c.h> I3C device information.
```

### Public Members

```
uint8_t dynamicAddr
    Device dynamic address.
uint8_t staticAddr
    Static address.
uint8_t dcr
    Device characteristics register information.
uint8_t bcr
    Bus characteristics register information.
uint16_t vendorID
    Device vendor ID(manufacture ID).
uint32_t partNumber
    Device part number info
uint16_t maxReadLength
    Maximum read length.
```

uint16\_t maxWriteLength

Maximum write length.

uint8\_t hdrMode

Support hdr mode, could be OR logic in i3c\_hdr\_mode.

## 2.34 I3C Common Driver

typedef struct *i3c\_config* i3c\_config\_t

Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C\_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

uint32\_t I3C\_GetInstance(I3C\_Type \*base)

Get which instance current I3C is used.

### Parameters

- base – The I3C peripheral base address.

void I3C\_GetDefaultConfig(*i3c\_config\_t* \*config)

Provides a default configuration for the I3C peripheral, the configuration covers both master functionality and slave functionality.

This function provides the following default configuration for I3C:

```
config->enableMaster      = kI3C_MasterCapable;
config->disableTimeout    = false;
config->hKeep              = kI3C_MasterHighKeeperNone;
config->enableOpenDrainStop = true;
config->enableOpenDrainHigh = true;
config->baudRate_Hz.i2cBaud = 400000U;
config->baudRate_Hz.i3cPushPullBaud = 12500000U;
config->baudRate_Hz.i3cOpenDrainBaud = 2500000U;
config->masterDynamicAddress = 0x0AU;
config->slowClock_Hz       = 1000000U;
config->enableSlave        = true;
config->vendorID           = 0x11BU;
config->enableRandomPart   = false;
config->partNumber         = 0;
config->dcr                 = 0;
config->bcr = 0;
config->hdrMode             = (uint8_t)kI3C_HDRModeDDR;
config->nakAllRequest       = false;
config->ignoreS0S1Error    = false;
config->offline             = false;
config->matchSlaveStartStop = false;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the common I3C driver with I3C\_Init().

### Parameters

- config – **[out]** User provided configuration structure for default values. Refer to i3c\_config\_t.

```
void I3C_Init(I3C_Type *base, const i3c_config_t *config, uint32_t sourceClock_Hz)
```

Initializes the I3C peripheral. This function enables the peripheral clock and initializes the I3C peripheral as described by the user provided configuration. This will initialize both the master peripheral and slave peripheral so that I3C module could work as pure master, pure slave or secondary master, etc. A software reset is performed prior to configuration.

#### Parameters

- `base` – The I3C peripheral base address.
- `config` – User provided peripheral configuration. Use `I3C_GetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

```
struct _i3c_config
```

*#include <fsl\_i3c.h>* Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_GetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

#### Public Members

```
i3c_master_enable_t enableMaster
```

Enable master mode.

```
bool disableTimeout
```

Whether to disable timeout to prevent the ERRWARN.

```
i3c_master_hkeep_t hKeep
```

High keeper mode setting.

```
bool enableOpenDrainStop
```

Whether to emit open-drain speed STOP.

```
bool enableOpenDrainHigh
```

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

```
i3c_baudrate_hz_t baudRate_Hz
```

Desired baud rate settings.

```
uint8_t masterDynamicAddress
```

Main master dynamic address configuration.

```
uint32_t maxWriteLength
```

Maximum write length.

```
uint32_t maxReadLength
```

Maximum read length.

```
bool enableSlave
```

Whether to enable slave.

```
uint8_t staticAddr
```

Static address.

```
uint16_t vendorID
```

Device vendor ID(manufacture ID).

uint32\_t partNumber

Device part number info

uint8\_t dcr

Device characteristics register information.

uint8\_t bcr

Bus characteristics register information.

uint8\_t hdrMode

Support hdr mode, could be OR logic in enumeration: `i3c_hdr_mode_t`.

bool nakAllRequest

Whether to reply NAK to all requests except broadcast CCC.

bool ignoreS0S1Error

Whether to ignore S0/S1 error in SDR mode.

bool offline

Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

bool matchSlaveStartStop

Whether to assert start/stop status only the time slave is addressed.

## 2.35 I3C Master DMA Driver

```
typedef struct i3c_master_dma_handle i3c_master_dma_handle_t
```

```
typedef struct i3c_master_dma_callback i3c_master_dma_callback_t  
i3c master callback functions.
```

```
void I3C_MasterTransferCreateHandleDMA(I3C_Type *base, i3c_master_dma_handle_t *handle,  
                                       const i3c_master_dma_callback_t *callback, void  
                                       *userData, dma_handle_t *rxDmaHandle,  
                                       dma_handle_t *txDmaHandle)
```

Create a new handle for the I3C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_MasterTransferAbortDMA()` API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

### Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C master driver handle.
- *callback* – User provided pointer to the asynchronous callback function.
- *userData* – User provided pointer to the application callback data.
- *rxDmaHandle* – Handle for the DMA receive channel. Created by the user prior to calling this function.
- *txDmaHandle* – Handle for the DMA transmit channel. Created by the user prior to calling this function.

```
status_t I3C_MasterTransferDMA(I3C_Type *base, i3c_master_dma_handle_t *handle,  
                               i3c_master_transfer_t *transfer)
```

Performs a non-blocking DMA-based transaction on the I3C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

#### Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C master driver handle.
- *transfer* – The pointer to the transfer descriptor.

#### Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

```
status_t I3C_MasterTransferGetCountDMA(I3C_Type *base, i3c_master_dma_handle_t *handle,  
                                       size_t *count)
```

Returns number of bytes transferred so far.

#### Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C master driver handle.
- *count* – **[out]** Number of bytes transferred so far by the non-blocking transaction.

#### Return values

- `kStatus_Success` –
- `kStatus_NoTransferInProgress` – There is not a DMA transaction currently in progress.

```
void I3C_MasterTransferAbortDMA(I3C_Type *base, i3c_master_dma_handle_t *handle)
```

Terminates a non-blocking I3C master transmission early.

---

**Note:** It is not safe to call this function from an IRQ handler that has a higher priority than the DMA peripheral's IRQ priority.

---

#### Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C master driver handle.

```
void I3C_MasterTransferDMAHandleIRQ(I3C_Type *base, void *i3cHandle)
```

Reusable routine to handle master interrupts.

---

**Note:** This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

---

#### Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C master DMA driver handle.

`void (*slave2Master)(I3C_Type *base, void *userData)`

Transfer complete callback

`void (*ibiCallback)(I3C_Type *base, i3c_master_dma_handle_t *handle, i3c_ibi_type_t ibiType, i3c_ibi_state_t ibiState)`

IBI event callback

`void (*transferComplete)(I3C_Type *base, i3c_master_dma_handle_t *handle, status_t status, void *userData)`

Transfer complete callback

`I3C_Type *base`

I3C base pointer.

`uint8_t state`

Transfer state machine current state.

`uint32_t transferCount`

Indicates progress of the transfer

`uint8_t subaddressBuffer[4]`

Saving subaddress command.

`uint8_t subaddressCount`

Saving command count.

`i3c_master_transfer_t transfer`

Copy of the current transfer info.

`i3c_master_dma_callback_t callback`

Callback function pointer.

`void *userData`

Application data passed to callback.

`dma_handle_t *rxDmaHandle`

Handle for receive DMA channel.

`dma_handle_t *txDmaHandle`

Handle for transmit DMA channel.

`uint8_t ibiAddress`

Slave address which request IBI.

`uint8_t *ibiBuff`

Pointer to IBI buffer to keep ibi bytes.

`size_t ibiPayloadSize`

IBI payload size.

`i3c_ibi_type_t ibiType`

IBI type.

`uint32_t transDataSize`

Transferred data size.

`uint8_t workaroundBuff[16]`

Workaround buffer to store temporary data.

`uint32_t event`

Reason the callback is being invoked.

`uint8_t *txData`  
Transfer buffer

`size_t txDataSize`  
Transfer size

`uint8_t *rxData`  
Transfer buffer

`size_t rxDataSize`  
Transfer size

`status_t completionStatus`  
Success or error code describing how the transfer completed. Only applies for `kI3C_SlaveCompletionEvent`.

`I3C_Type *base`  
I3C base pointer.

`i3c_slave_dma_transfer_t transfer`  
I3C slave transfer copy.

`bool isBusy`  
Whether transfer is busy.

`bool wasTransmit`  
Whether the last transfer was a transmit.

`uint32_t eventMask`  
Mask of enabled events.

`i3c_slave_dma_callback_t callback`  
Callback function called at transfer event.

`dma_handle_t *rxDmaHandle`  
Handle for receive DMA channel.

`dma_handle_t *txDmaHandle`  
Handle for transmit DMA channel.

`void *userData`  
Callback parameter passed to callback.

`struct _i3c_master_dma_callback`  
`#include <fsl_i3c_dma.h>` i3c master callback functions.

`struct _i3c_master_dma_handle`  
`#include <fsl_i3c_dma.h>` Driver handle for master DMA APIs.

---

**Note:** The contents of this structure are private and subject to change.

---

## 2.36 I3C Master Driver

`void I3C_MasterGetDefaultConfig(i3c_master_config_t *masterConfig)`

Provides a default configuration for the I3C master peripheral.

This function provides the following default configuration for the I3C master peripheral:

```

masterConfig->enableMaster      = kI3C_MasterOn;
masterConfig->disableTimeout    = false;
masterConfig->hKeep             = kI3C_MasterHighKeeperNone;
masterConfig->enableOpenDrainStop = true;
masterConfig->enableOpenDrainHigh = true;
masterConfig->baudRate_Hz       = 100000U;
masterConfig->busType           = kI3C_TypeI2C;

```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with `I3C_MasterInit()`.

#### Parameters

- `masterConfig` – **[out]** User provided configuration structure for default values. Refer to `i3c_master_config_t`.

```
void I3C_MasterInit(I3C_Type *base, const i3c_master_config_t *masterConfig, uint32_t
sourceClock_Hz)
```

Initializes the I3C master peripheral.

This function enables the peripheral clock and initializes the I3C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

#### Parameters

- `base` – The I3C peripheral base address.
- `masterConfig` – User provided peripheral configuration. Use `I3C_MasterGetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

```
void I3C_MasterDeinit(I3C_Type *base)
```

Deinitializes the I3C master peripheral.

This function disables the I3C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

#### Parameters

- `base` – The I3C peripheral base address.

```
status_t I3C_MasterCheckAndClearError(I3C_Type *base, uint32_t status)
```

```
status_t I3C_MasterWaitForCtrlDone(I3C_Type *base, bool waitIdle)
```

```
status_t I3C_CheckForBusyBus(I3C_Type *base)
```

```
static inline void I3C_MasterEnable(I3C_Type *base, i3c_master_enable_t enable)
```

Set I3C module master mode.

#### Parameters

- `base` – The I3C peripheral base address.
- `enable` – Enable master mode.

```
void I3C_SlaveGetDefaultConfig(i3c_slave_config_t *slaveConfig)
```

Provides a default configuration for the I3C slave peripheral.

This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableslave      = true;
```



After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with `I3C_SlaveInit()`.

#### Parameters

- `slaveConfig` – **[out]** User provided configuration structure for default values. Refer to `i3c_slave_config_t`.

```
void I3C_SlaveInit(I3C_Type *base, const i3c_slave_config_t *slaveConfig, uint32_t
    slowClock_Hz)
```

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

#### Parameters

- `base` – The I3C peripheral base address.
- `slaveConfig` – User provided peripheral configuration. Use `I3C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- `slowClock_Hz` – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If `FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH` defines as 1, this parameter is useless.

```
void I3C_SlaveDeinit(I3C_Type *base)
```

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

#### Parameters

- `base` – The I3C peripheral base address.

```
static inline void I3C_SlaveEnable(I3C_Type *base, bool isEnabled)
```

Enable/Disable Slave.

#### Parameters

- `base` – The I3C peripheral base address.
- `isEnabled` – Enable or disable.

```
static inline uint32_t I3C_MasterGetStatusFlags(I3C_Type *base)
```

Gets the I3C master status flags.

A bit mask with the state of all I3C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

#### See also:

`_i3c_master_flags`

#### Parameters

- `base` – The I3C peripheral base address.

#### Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_MasterClearStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C master status flag state.

The following status register flags can be cleared:

- kI3C\_MasterSlaveStartFlag
- kI3C\_MasterControlDoneFlag
- kI3C\_MasterCompleteFlag
- kI3C\_MasterArbitrationWonFlag
- kI3C\_MasterSlave2MasterFlag

Attempts to clear other flags has no effect.

**See also:**

`_i3c_master_flags`.

**Parameters**

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_master_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_MasterGetStatusFlags()`.

```
static inline uint32_t I3C_MasterGetErrorStatusFlags(I3C_Type *base)
```

Gets the I3C master error status flags.

A bit mask with the state of all I3C master error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

**See also:**

`_i3c_master_error_flags`

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_MasterClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C master error status flag state.

**See also:**

`_i3c_master_error_flags`.

**Parameters**

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_master_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_MasterGetStatusFlags()`.

*i3c\_master\_state\_t* I3C\_MasterGetState(I3C\_Type \*base)

Gets the I3C master state.

#### Parameters

- base – The I3C peripheral base address.

#### Returns

I3C master state.

static inline uint32\_t I3C\_SlaveGetStatusFlags(I3C\_Type \*base)

Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

#### See also:

`_i3c_slave_flags`

#### Parameters

- base – The I3C peripheral base address.

#### Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C\_SlaveClearStatusFlags(I3C\_Type \*base, uint32\_t statusMask)

Clears the I3C slave status flag state.

The following status register flags can be cleared:

- kI3C\_SlaveBusStartFlag
- kI3C\_SlaveMatchedFlag
- kI3C\_SlaveBusStopFlag

Attempts to clear other flags has no effect.

#### See also:

`_i3c_slave_flags`.

#### Parameters

- base – The I3C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetStatusFlags()`.

static inline uint32\_t I3C\_SlaveGetErrorStatusFlags(I3C\_Type \*base)

Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

#### See also:

`_i3c_slave_error_flags`

**Parameters**

- base – The I3C peripheral base address.

**Returns**

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_SlaveClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C slave error status flag state.

**See also:**

`_i3c_slave_error_flags`.

**Parameters**

- base – The I3C peripheral base address.
- statusMask – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_slave_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetErrorStatusFlags()`.

```
i3c_slave_activity_state_t I3C_SlaveGetActivityState(I3C_Type *base)
```

Gets the I3C slave state.

**Parameters**

- base – The I3C peripheral base address.

**Returns**

I3C slave activity state, refer `i3c_slave_activity_state_t`.

```
status_t I3C_SlaveCheckAndClearError(I3C_Type *base, uint32_t status)
```

```
static inline void I3C_MasterEnableInterrupts(I3C_Type *base, uint32_t interruptMask)
```

Enables the I3C master interrupt requests.

All flags except `kI3C_MasterBetweenFlag` and `kI3C_MasterNackDetectFlag` can be enabled as interrupts.

**Parameters**

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See `_i3c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline void I3C_MasterDisableInterrupts(I3C_Type *base, uint32_t interruptMask)
```

Disables the I3C master interrupt requests.

All flags except `kI3C_MasterBetweenFlag` and `kI3C_MasterNackDetectFlag` can be enabled as interrupts.

**Parameters**

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See `_i3c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline uint32_t I3C_MasterGetEnabledInterrupts(I3C_Type *base)
```

Returns the set of currently enabled I3C master interrupt requests.

**Parameters**

- base – The I3C peripheral base address.

**Returns**

A bitmask composed of `_i3c_master_flags` enumerators OR'd together to indicate the set of enabled interrupts.

```
static inline uint32_t I3C_MasterGetPendingInterrupts(I3C_Type *base)
```

Returns the set of pending I3C master interrupt requests.

**Parameters**

- base – The I3C peripheral base address.

**Returns**

A bitmask composed of `_i3c_master_flags` enumerators OR'd together to indicate the set of pending interrupts.

```
static inline void I3C_SlaveEnableInterrupts(I3C_Type *base, uint32_t interruptMask)
```

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`
- `kI3C_SlaveErrorFlag`
- `kI3C_SlaveHDRCommandMatchFlag`
- `kI3C_SlaveCCCHandledFlag`
- `kI3C_SlaveEventSentFlag`

**Parameters**

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline void I3C_SlaveDisableInterrupts(I3C_Type *base, uint32_t interruptMask)
```

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`
- `kI3C_SlaveErrorFlag`
- `kI3C_SlaveHDRCommandMatchFlag`

- kI3C\_SlaveCCCHandledFlag
- kI3C\_SlaveEventSentFlag

**Parameters**

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline uint32\_t I3C\_SlaveGetEnabledInterrupts(I3C\_Type \*base)

Returns the set of currently enabled I3C slave interrupt requests.

**Parameters**

- base – The I3C peripheral base address.

**Returns**

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

static inline uint32\_t I3C\_SlaveGetPendingInterrupts(I3C\_Type \*base)

Returns the set of pending I3C slave interrupt requests.

**Parameters**

- base – The I3C peripheral base address.

**Returns**

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of pending interrupts.

static inline void I3C\_MasterEnableDMA(I3C\_Type \*base, bool enableTx, bool enableRx, uint32\_t width)

Enables or disables I3C master DMA requests.

**Parameters**

- base – The I3C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.
- width – DMA read/write unit in bytes.

static inline uint32\_t I3C\_MasterGetTxFifoAddress(I3C\_Type \*base, uint32\_t width)

Gets I3C master transmit data register address for DMA transfer.

**Parameters**

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

**Returns**

The I3C Master Transmit Data Register address.

static inline uint32\_t I3C\_MasterGetRxFifoAddress(I3C\_Type \*base, uint32\_t width)

Gets I3C master receive data register address for DMA transfer.

**Parameters**

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

**Returns**

The I3C Master Receive Data Register address.

```
static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)
```

Enables or disables I3C slave DMA requests.

**Parameters**

- *base* – The I3C peripheral base address.
- *enableTx* – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- *enableRx* – Enable flag for receive DMA request. Pass true for enable, false for disable.
- *width* – DMA read/write unit in bytes.

```
static inline uint32_t I3C_SlaveGetTxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave transmit data register address for DMA transfer.

**Parameters**

- *base* – The I3C peripheral base address.
- *width* – DMA read/write unit in bytes.

**Returns**

The I3C Slave Transmit Data Register address.

```
static inline uint32_t I3C_SlaveGetRxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave receive data register address for DMA transfer.

**Parameters**

- *base* – The I3C peripheral base address.
- *width* – DMA read/write unit in bytes.

**Returns**

The I3C Slave Receive Data Register address.

```
static inline void I3C_MasterSetWatermarks(I3C_Type *base, i3c_tx_trigger_level_t txLvl, i3c_rx_trigger_level_t rxLvl, bool flushTx, bool flushRx)
```

Sets the watermarks for I3C master FIFOs.

**Parameters**

- *base* – The I3C peripheral base address.
- *txLvl* – Transmit FIFO watermark level. The `kI3C_MasterTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
- *rxLvl* – Receive FIFO watermark level. The `kI3C_MasterRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
- *flushTx* – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- *flushRx* – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

```
static inline void I3C_MasterGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of bytes in the I3C master FIFOs.

**Parameters**

- *base* – The I3C peripheral base address.

- `txCount` – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

```
static inline void I3C_SlaveSetWatermarks(I3C_Type *base, i3c_tx_trigger_level_t txLvl,  
                                         i3c_rx_trigger_level_t rxLvl, bool flushTx, bool  
                                         flushRx)
```

Sets the watermarks for I3C slave FIFOs.

#### Parameters

- `base` – The I3C peripheral base address.
- `txLvl` – Transmit FIFO watermark level. The `kI3C_SlaveTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches `txLvl`.
- `rxLvl` – Receive FIFO watermark level. The `kI3C_SlaveRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches `rxLvl`.
- `flushTx` – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- `flushRx` – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

```
static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of bytes in the I3C slave FIFOs.

#### Parameters

- `base` – The I3C peripheral base address.
- `txCount` – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

```
void I3C_MasterSetBaudRate(I3C_Type *base, const i3c_baudrate_hz_t *baudRate_Hz, uint32_t  
                           sourceClock_Hz)
```

Sets the I3C bus frequency for master transactions.

The I3C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

#### Parameters

- `base` – The I3C peripheral base address.
- `baudRate_Hz` – Pointer to structure of requested bus frequency in Hertz.
- `sourceClock_Hz` – I3C functional clock frequency in Hertz.

```
static inline bool I3C_MasterGetBusIdleState(I3C_Type *base)
```

Returns whether the bus is idle.

Requires the master mode to be enabled.

#### Parameters

- `base` – The I3C peripheral base address.

#### Return values

- `true` – Bus is busy.
- `false` – Bus is idle.



*status\_t* I3C\_MasterStartWithRxSize(I3C\_Type \*base, *i3c\_bus\_type\_t* type, uint8\_t address, *i3c\_direction\_t* dir, uint8\_t rxSize)

Sends a START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

#### Parameters

- *base* – The I3C peripheral base address.
- *type* – The bus type to use in this transaction.
- *address* – 7-bit slave device address, in bits [6:0].
- *dir* – Master transfer direction, either *kI3C\_Read* or *kI3C\_Write*. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- *rxSize* – Read terminate size for the followed read transfer, limit to 255 bytes.

#### Return values

- *kStatus\_Success* – START signal and address were successfully enqueued in the transmit FIFO.
- *kStatus\_I3C\_Busy* – Another master is currently utilizing the bus.

*status\_t* I3C\_MasterStart(I3C\_Type \*base, *i3c\_bus\_type\_t* type, uint8\_t address, *i3c\_direction\_t* dir)

Sends a START signal and slave address on the I2C/I3C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

#### Parameters

- *base* – The I3C peripheral base address.
- *type* – The bus type to use in this transaction.
- *address* – 7-bit slave device address, in bits [6:0].
- *dir* – Master transfer direction, either *kI3C\_Read* or *kI3C\_Write*. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

#### Return values

- *kStatus\_Success* – START signal and address were successfully enqueued in the transmit FIFO.
- *kStatus\_I3C\_Busy* – Another master is currently utilizing the bus.

*status\_t* I3C\_MasterRepeatedStartWithRxSize(I3C\_Type \*base, *i3c\_bus\_type\_t* type, uint8\_t address, *i3c\_direction\_t* dir, uint8\_t rxSize)

Sends a repeated START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to send a Repeated START signal when a transfer is already in progress. Like *I3C\_MasterStart()*, it also sends the specified 7-bit address. Call this API also configures the read terminate size for the following read transfer. For example, set the *rxSize* = 2, the

following read transfer will be terminated after two bytes of data received. Write transfer will not be affected by the rxSize configuration.

---

**Note:** This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

---

#### Parameters

- `base` – The I3C peripheral base address.
- `type` – The bus type to use in this transaction.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- `rxSize` – Read terminate size for the followed read transfer, limit to 255 bytes.

#### Return values

`kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.

```
static inline status_t I3C_MasterRepeatedStart(I3C_Type *base, i3c_bus_type_t type, uint8_t address, i3c_direction_t dir)
```

Sends a repeated START signal and slave address on the I2C/I3C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like `I3C_MasterStart()`, it also sends the specified 7-bit address.

---

**Note:** This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

---

#### Parameters

- `base` – The I3C peripheral base address.
- `type` – The bus type to use in this transaction.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

#### Return values

`kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.

```
status_t I3C_MasterSend(I3C_Type *base, const void *txBuff, size_t txSize, uint32_t flags)
```

Performs a polling send transfer on the I2C/I3C bus.

Sends up to `txSize` number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns `kStatus_I3C_Nak`.

#### Parameters

- `base` – The I3C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.

- `flags` – Bit mask of options for the transfer. See enumeration `_i3c_master_transfer_flags` for available options.

**Return values**

- `kStatus_Success` – Data was sent successfully.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.
- `kStatus_I3C_Nak` – The slave device sent a NAK in response to an address.
- `kStatus_I3C_WriteAbort` – The slave device sent a NAK in response to a write.
- `kStatus_I3C_MsgError` – Message SDR/DDR mismatch or read/write message in wrong state.
- `kStatus_I3C_WriteFifoError` – Write to M/SWDATAB register when FIFO full.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

*status\_t* I3C\_MasterReceive(I3C\_Type \*base, void \*rxBuff, size\_t rxSize, uint32\_t flags)

Performs a polling receive transfer on the I2C/I3C bus.

**Parameters**

- `base` – The I3C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.
- `flags` – Bit mask of options for the transfer. See enumeration `_i3c_master_transfer_flags` for available options.

**Return values**

- `kStatus_Success` – Data was received successfully.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.
- `kStatus_I3C_Term` – The master terminates slave read.
- `kStatus_I3C_HdrParityError` – Parity error from DDR read.
- `kStatus_I3C_CrcError` – CRC error from DDR read.
- `kStatus_I3C_MsgError` – Message SDR/DDR mismatch or read/write message in wrong state.
- `kStatus_I3C_ReadFifoError` – Read from M/SRDATAB register when FIFO empty.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

*status\_t* I3C\_MasterStop(I3C\_Type \*base)

Sends a STOP signal on the I2C/I3C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

**Parameters**

- `base` – The I3C peripheral base address.

**Return values**

- `kStatus_Success` – The STOP signal was successfully sent on the bus and the transaction terminated.

- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

`void I3C_MasterEmitRequest(I3C_Type *base, i3c_bus_request_t masterReq)`  
I3C master emit request.

**Parameters**

- `base` – The I3C peripheral base address.
- `masterReq` – I3C master request of type `i3c_bus_request_t`

`static inline void I3C_MasterEmitIBIResponse(I3C_Type *base, i3c_ibi_response_t ibiResponse)`  
I3C master emit request.

**Parameters**

- `base` – The I3C peripheral base address.
- `ibiResponse` – I3C master emit IBI response of type `i3c_ibi_response_t`

`void I3C_MasterRegisterIBI(I3C_Type *base, i3c_register_ibi_addr_t *ibiRule)`  
I3C master register IBI rule.

**Parameters**

- `base` – The I3C peripheral base address.
- `ibiRule` – Pointer to ibi rule description of type `i3c_register_ibi_addr_t`

`void I3C_MasterGetIBIRules(I3C_Type *base, i3c_register_ibi_addr_t *ibiRule)`  
I3C master get IBI rule.

**Parameters**

- `base` – The I3C peripheral base address.
- `ibiRule` – Pointer to store the read out ibi rule description.

`i3c_ibi_type_t I3C_GetIBIType(I3C_Type *base)`  
I3C master get IBI Type.

**Parameters**

- `base` – The I3C peripheral base address.

**Return values**

`i3c_ibi_type_t` – Type of `i3c_ibi_type_t`.

`static inline uint8_t I3C_GetIBIAddress(I3C_Type *base)`  
I3C master get IBI Address.

**Parameters**

- `base` – The I3C peripheral base address.

**Return values**

The – 8-bit IBI address.

`status_t I3C_MasterProcessDAASpecifiedBaudrate(I3C_Type *base, uint8_t *addressList, uint32_t count, i3c_master_daa_baudrate_t *daaBaudRate)`

Performs a DAA in the i3c bus with specified temporary baud rate.

**Parameters**

- `base` – The I3C peripheral base address.

- `addressList` – The pointer for address list which is used to do DAA.
- `count` – The address count in the address list.
- `daaBaudRate` – The temporary baud rate in DAA process, NULL for using initial setting. The initial setting is set back between the completion of the DAA and the return of this function.

**Return values**

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- `kStatus_I3C_SlaveCountExceed` – The I3C slave count has exceed the definition in `I3C_MAX_DEVCNT`.

```
static inline status_t I3C_MasterProcessDAA(I3C_Type *base, uint8_t *addressList, uint32_t count)
```

Performs a DAA in the i3c bus.

**Parameters**

- `base` – The I3C peripheral base address.
- `addressList` – The pointer for address list which is used to do DAA.
- `count` – The address count in the address list. The initial setting is set back between the completion of the DAA and the return of this function.

**Return values**

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- `kStatus_I3C_SlaveCountExceed` – The I3C slave count has exceed the definition in `I3C_MAX_DEVCNT`.

```
i3c_device_info_t *I3C_MasterGetDeviceListAfterDAA(I3C_Type *base, uint8_t *count)
```

Get device information list after DAA process is done.

**Parameters**

- `base` – The I3C peripheral base address.
- `count` – **[out]** The pointer to store the available device count.

**Returns**

Pointer to the `i3c_device_info_t` array.

```
void I3C_MasterClearDeviceCount(I3C_Type *base)
```

Clear the global device count which represents current devices number on the bus. When user resets all dynamic addresses on the bus, should call this API.

**Parameters**

- `base` – The I3C peripheral base address.

```
status_t I3C_MasterTransferBlocking(I3C_Type *base, i3c_master_transfer_t *transfer)
```

Performs a master polling transfer on the I2C/I3C bus.

---

**Note:** The API does not return until the transfer succeeds or fails due to error happens during transfer.

---

**Parameters**

- `base` – The I3C peripheral base address.
- `transfer` – Pointer to the transfer structure.

**Return values**

- `kStatus_I3C_Success` – Data was received successfully.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_IBIWon` – The I3C slave event IBI or MR or HJ won the arbitration on a header address.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.
- `kStatus_I3C_Nak` – The slave device sent a NAK in response to an address.
- `kStatus_I3C_WriteAbort` – The slave device sent a NAK in response to a write.
- `kStatus_I3C_Term` – The master terminates slave read.
- `kStatus_I3C_HdrParityError` – Parity error from DDR read.
- `kStatus_I3C_CrcError` – CRC error from DDR read.
- `kStatus_I3C_MsgError` – Message SDR/DDR mismatch or read/write message in wrong state.
- `kStatus_I3C_ReadFifoError` – Read from M/SRDATA register when FIFO empty.
- `kStatus_I3C_WriteFifoError` – Write to M/SWDATA register when FIFO full.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

*status\_t* I3C\_SlaveSend(I3C\_Type \*base, const void \*txBuff, size\_t txSize)

Performs a polling send transfer on the I3C bus.

**Parameters**

- `base` – The I3C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.

**Returns**

Error or success status returned by API.

*status\_t* I3C\_SlaveReceive(I3C\_Type \*base, void \*rxBuff, size\_t rxSize)

Performs a polling receive transfer on the I3C bus.

**Parameters**

- `base` – The I3C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.

**Returns**

Error or success status returned by API.

void I3C\_MasterTransferCreateHandle(I3C\_Type \*base, *i3c\_master\_handle\_t* \*handle, const *i3c\_master\_transfer\_callback\_t* \*callback, void \*userData)

Creates a new handle for the I3C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_MasterTransferAbort()` API shall be called.

---

**Note:** The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

---

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – **[out]** Pointer to the I3C master driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

`status_t I3C_MasterTransferNonBlocking(I3C_Type *base, i3c_master_handle_t *handle, i3c_master_transfer_t *transfer)`

Performs a non-blocking transaction on the I2C/I3C bus.

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.
- `transfer` – The pointer to the transfer descriptor.

#### Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

`status_t I3C_MasterTransferGetCount(I3C_Type *base, i3c_master_handle_t *handle, size_t *count)`

Returns number of bytes transferred so far.

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.
- `count` – **[out]** Number of bytes transferred so far by the non-blocking transaction.

#### Return values

- `kStatus_Success` –
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

`void I3C_MasterTransferAbort(I3C_Type *base, i3c_master_handle_t *handle)`

Terminates a non-blocking I3C master transmission early.

---

**Note:** It is not safe to call this function from an IRQ handler that has a higher priority than the I3C peripheral's IRQ priority.

---

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.

#### Return values

- `kStatus_Success` – A transaction was successfully aborted.
- `kStatus_I3C_Idle` – There is not a non-blocking transaction currently in progress.

`void I3C_MasterTransferHandleIRQ(I3C_Type *base, void *intHandle)`

Reusable routine to handle master interrupts.

---

**Note:** This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

---

### Parameters

- `base` – The I3C peripheral base address.
- `intHandle` – Pointer to the I3C master driver handle.

`enum _i3c_master_flags`

I3C master peripheral flags.

The following status register flags can be cleared:

- `kI3C_MasterSlaveStartFlag`
- `kI3C_MasterControlDoneFlag`
- `kI3C_MasterCompleteFlag`
- `kI3C_MasterArbitrationWonFlag`
- `kI3C_MasterSlave2MasterFlag`

All flags except `kI3C_MasterBetweenFlag` and `kI3C_MasterNackDetectFlag` can be enabled as interrupts.

---

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

---

### Values:

enumerator `kI3C_MasterBetweenFlag`

Between messages/DAAs flag

enumerator `kI3C_MasterNackDetectFlag`

NACK detected flag

enumerator `kI3C_MasterSlaveStartFlag`

Slave request start flag

enumerator `kI3C_MasterControlDoneFlag`

Master request complete flag

enumerator `kI3C_MasterCompleteFlag`

Transfer complete flag

enumerator `kI3C_MasterRxReadyFlag`

Rx data ready in Rx buffer flag

enumerator `kI3C_MasterTxReadyFlag`

Tx buffer ready for Tx data flag

enumerator `kI3C_MasterArbitrationWonFlag`

Header address won arbitration flag



enumerator kI3C\_MasterErrorFlag

Error occurred flag

enumerator kI3C\_MasterSlave2MasterFlag

Switch from slave to master flag

enumerator kI3C\_MasterClearFlags

enum \_i3c\_master\_error\_flags

I3C master error flags to indicate the causes.

---

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

---

*Values:*

enumerator kI3C\_MasterErrorNackFlag

Slave NACKed the last address

enumerator kI3C\_MasterErrorWriteAbortFlag

Slave NACKed the write data

enumerator kI3C\_MasterErrorParityFlag

Parity error from DDR read

enumerator kI3C\_MasterErrorCrcFlag

CRC error from DDR read

enumerator kI3C\_MasterErrorReadFlag

Read from MRDATAB register when FIFO empty

enumerator kI3C\_MasterErrorWriteFlag

Write to MWDATAB register when FIFO full

enumerator kI3C\_MasterErrorMsgFlag

Message SDR/DDR mismatch or read/write message in wrong state

enumerator kI3C\_MasterErrorInvalidReqFlag

Invalid use of request

enumerator kI3C\_MasterErrorTimeoutFlag

The module has stalled too long in a frame

enumerator kI3C\_MasterAllErrorFlags

All error flags

enum \_i3c\_master\_state

I3C working master state.

*Values:*

enumerator kI3C\_MasterStateIdle

Bus stopped.

enumerator kI3C\_MasterStateSlvReq

Bus stopped but slave holding SDA low.

enumerator kI3C\_MasterStateMsgSdr

In SDR Message mode from using MWMSG\_SDR.

enumerator kI3C\_MasterStateNormAct

In normal active SDR mode.

enumerator kI3C\_MasterStateDdr

In DDR Message mode.

enumerator kI3C\_MasterStateDaa

In ENTDAAs mode.

enumerator kI3C\_MasterStateIbiAck

Waiting on IBI ACK/NACK decision.

enumerator kI3C\_MasterStateIbiRcv

Receiving IBI.

enum \_i3c\_master\_enable

I3C master enable configuration.

*Values:*

enumerator kI3C\_MasterOff

Master off.

enumerator kI3C\_MasterOn

Master on.

enumerator kI3C\_MasterCapable

Master capable.

enum \_i3c\_master\_hkeep

I3C high keeper configuration.

*Values:*

enumerator kI3C\_MasterHighKeeperNone

Use PUR to hold SCL high.

enumerator kI3C\_MasterHighKeeperWiredIn

Use pin\_HK controls.

enumerator kI3C\_MasterPassiveSDA

Hi-Z for Bus Free and hold SDA.

enumerator kI3C\_MasterPassiveSDASCL

Hi-Z both for Bus Free, and can Hi-Z SDA for hold.

enum \_i3c\_bus\_request

Emits the requested operation when doing in pieces vs. by message.

*Values:*

enumerator kI3C\_RequestNone

No request.

enumerator kI3C\_RequestEmitStartAddr

Request to emit start and address on bus.

enumerator kI3C\_RequestEmitStop

Request to emit stop on bus.

enumerator kI3C\_RequestIbiAckNack

Manual IBI ACK or NACK.

enumerator kI3C\_RequestProcessDAA

Process DAA.

enumerator kI3C\_RequestForceExit  
Request to force exit.

enumerator kI3C\_RequestAutoIbi  
Hold in stopped state, but Auto-emit START,7E.

enum \_\_i3c\_\_bus\_\_type  
Bus type with EmitStartAddr.

*Values:*

enumerator kI3C\_TypeI3CSdr  
SDR mode of I3C.

enumerator kI3C\_TypeI2C  
Standard i2c protocol.

enumerator kI3C\_TypeI3CDdr  
HDR-DDR mode of I3C.

enum \_\_i3c\_\_ibi\_\_response  
IBI response.

*Values:*

enumerator kI3C\_IbiRespAck  
ACK with no mandatory byte.

enumerator kI3C\_IbiRespNack  
NACK.

enumerator kI3C\_IbiRespAckMandatory  
ACK with mandatory byte.

enumerator kI3C\_IbiRespManual  
Reserved.

enum \_\_i3c\_\_ibi\_\_type  
IBI type.

*Values:*

enumerator kI3C\_IbiNormal  
In-band interrupt.

enumerator kI3C\_IbiHotJoin  
slave hot join.

enumerator kI3C\_IbiMasterRequest  
slave master ship request.

enum \_\_i3c\_\_ibi\_\_state  
IBI state.

*Values:*

enumerator kI3C\_IbiReady  
In-band interrupt ready state, ready for user to handle.

enumerator kI3C\_IbiDataBuffNeed  
In-band interrupt need data buffer for data receive.

enumerator kI3C\_IbiAckNackPending  
In-band interrupt Ack/Nack pending for decision.

enum `_i3c_direction`

Direction of master and slave transfers.

*Values:*

enumerator `kI3C_Write`

Master transmit.

enumerator `kI3C_Read`

Master receive.

enum `_i3c_tx_trigger_level`

Watermark of TX int/dma trigger level.

*Values:*

enumerator `kI3C_TxTriggerOnEmpty`

Trigger on empty.

enumerator `kI3C_TxTriggerUntilOneQuarterOrLess`

Trigger on 1/4 full or less.

enumerator `kI3C_TxTriggerUntilOneHalfOrLess`

Trigger on 1/2 full or less.

enumerator `kI3C_TxTriggerUntilOneLessThanFull`

Trigger on 1 less than full or less.

enum `_i3c_rx_trigger_level`

Watermark of RX int/dma trigger level.

*Values:*

enumerator `kI3C_RxTriggerOnNotEmpty`

Trigger on not empty.

enumerator `kI3C_RxTriggerUntilOneQuarterOrMore`

Trigger on 1/4 full or more.

enumerator `kI3C_RxTriggerUntilOneHalfOrMore`

Trigger on 1/2 full or more.

enumerator `kI3C_RxTriggerUntilThreeQuarterOrMore`

Trigger on 3/4 full or more.

enum `_i3c_rx_term_ops`

I3C master read termination operations.

*Values:*

enumerator `kI3C_RxTermDisable`

Master doesn't terminate read, used for CCC transfer.

enumerator `kI3C_RxAutoTerm`

Master auto terminate read after receiving specified bytes(<=255).

enumerator `kI3C_RxTermLastByte`

Master terminates read at any time after START, no length limitation.

enum `_i3c_start_scl_delay`

I3C start SCL delay options.

*Values:*

enumerator kI3C\_NoDelay

No delay.

enumerator kI3C\_IncreaseSclHalfPeriod

Increases SCL clock period by 1/2.

enumerator kI3C\_IncreaseSclOnePeriod

Increases SCL clock period by 1.

enumerator kI3C\_IncreaseSclOneAndHalfPeriod

Increases SCL clock period by 1 1/2

enum \_i3c\_master\_transfer\_flags

Transfer option flags.

---

**Note:** These enumerations are intended to be OR'd together to form a bit mask of options for the `_i3c_master_transfer::flags` field.

---

*Values:*

enumerator kI3C\_TransferDefaultFlag

Transfer starts with a start signal, stops with a stop signal.

enumerator kI3C\_TransferNoStartFlag

Don't send a start condition, address, and sub address

enumerator kI3C\_TransferRepeatedStartFlag

Send a repeated start condition

enumerator kI3C\_TransferNoStopFlag

Don't send a stop condition.

enumerator kI3C\_TransferWordsFlag

Transfer in words, else transfer in bytes.

enumerator kI3C\_TransferDisableRxTermFlag

Disable Rx termination. Note: It's for I3C CCC transfer.

enumerator kI3C\_TransferRxAutoTermFlag

Set Rx auto-termination. Note: It's adaptive based on Rx size(<=255 bytes) except in I3C\_MasterReceive.

enumerator kI3C\_TransferStartWithBroadcastAddr

Start transfer with 0x7E, then read/write data with device address.

typedef enum \_i3c\_master\_state i3c\_master\_state\_t

I3C working master state.

typedef enum \_i3c\_master\_enable i3c\_master\_enable\_t

I3C master enable configuration.

typedef enum \_i3c\_master\_hkeep i3c\_master\_hkeep\_t

I3C high keeper configuration.

typedef enum \_i3c\_bus\_request i3c\_bus\_request\_t

Emits the requested operation when doing in pieces vs. by message.

typedef enum \_i3c\_bus\_type i3c\_bus\_type\_t

Bus type with EmitStartAddr.

```
typedef enum _i3c_ibi_response i3c_ibi_response_t
    IBI response.
```

```
typedef enum _i3c_ibi_type i3c_ibi_type_t
    IBI type.
```

```
typedef enum _i3c_ibi_state i3c_ibi_state_t
    IBI state.
```

```
typedef enum _i3c_direction i3c_direction_t
    Direction of master and slave transfers.
```

```
typedef enum _i3c_tx_trigger_level i3c_tx_trigger_level_t
    Watermark of TX int/dma trigger level.
```

```
typedef enum _i3c_rx_trigger_level i3c_rx_trigger_level_t
    Watermark of RX int/dma trigger level.
```

```
typedef enum _i3c_rx_term_ops i3c_rx_term_ops_t
    I3C master read termination operations.
```

```
typedef enum _i3c_start_scl_delay i3c_start_scl_delay_t
    I3C start SCL delay options.
```

```
typedef struct _i3c_register_ibi_addr i3c_register_ibi_addr_t
    Structure with setting master IBI rules and slave registry.
```

```
typedef struct _i3c_baudrate i3c_baudrate_hz_t
    Structure with I3C baudrate settings.
```

```
typedef struct _i3c_master_daa_baudrate i3c_master_daa_baudrate_t
    I3C DAA baud rate configuration.
```

```
typedef struct _i3c_master_config i3c_master_config_t
    Structure with settings to initialize the I3C master module.
```

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_MasterGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

```
typedef struct _i3c_master_transfer i3c_master_transfer_t
```

```
typedef struct _i3c_master_handle i3c_master_handle_t
```

```
typedef struct _i3c_master_transfer_callback i3c_master_transfer_callback_t
    i3c master callback functions.
```

```
typedef void (*i3c_master_isr_t)(I3C_Type *base, void *handle)
    Typedef for master interrupt handler.
```

```
struct _i3c_register_ibi_addr
    #include <fsl_i3c.h> Structure with setting master IBI rules and slave registry.
```

### Public Members

```
uint8_t address[5]
    Address array for registry.
```

```
bool i3cFastStart
    Allow the START header to run as push-pull speed if all dynamic addresses take MSB 0.
```

bool ibiHasPayload

Whether the address array has mandatory IBI byte.

struct \_\_i3c\_\_baudrate

*#include <fsl\_i3c.h>* Structure with I3C baudrate settings.

### Public Members

uint32\_t i2cBaud

Desired I2C baud rate in Hertz.

uint32\_t i3cPushPullBaud

Desired I3C push-pull baud rate in Hertz.

uint32\_t i3cOpenDrainBaud

Desired I3C open-drain baud rate in Hertz.

struct \_\_i3c\_\_master\_\_daa\_\_baudrate

*#include <fsl\_i3c.h>* I3C DAA baud rate configuration.

### Public Members

uint32\_t sourceClock\_Hz

FCLK, function clock in Hertz.

uint32\_t i3cPushPullBaud

Desired I3C push-pull baud rate in Hertz.

uint32\_t i3cOpenDrainBaud

Desired I3C open-drain baud rate in Hertz.

struct \_\_i3c\_\_master\_\_config

*#include <fsl\_i3c.h>* Structure with settings to initialize the I3C master module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C\_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

### Public Members

i3c\_master\_enable\_t enableMaster

Enable master mode.

bool disableTimeout

Whether to disable timeout to prevent the ERRWARN.

i3c\_master\_hkeep\_t hKeep

High keeper mode setting.

bool enableOpenDrainStop

Whether to emit open-drain speed STOP.

bool enableOpenDrainHigh

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

i3c\_baudrate\_hz\_t baudRate\_Hz

Desired baud rate settings.

```
struct _i3c_master_transfer_callback
    #include <fsl_i3c.h> i3c master callback functions.
```

### Public Members

```
void (*slave2Master)(I3C_Type *base, void *userData)
```

Transfer complete callback

```
void (*ibiCallback)(I3C_Type *base, i3c_master_handle_t *handle, i3c_ibi_type_t ibiType,
i3c_ibi_state_t ibiState)
```

IBI event callback

```
void (*transferComplete)(I3C_Type *base, i3c_master_handle_t *handle, status_t
completionStatus, void *userData)
```

Transfer complete callback

```
struct _i3c_master_transfer
    #include <fsl_i3c.h> Non-blocking transfer descriptor structure.
```

This structure is used to pass transaction parameters to the I3C\_MasterTransferNonBlocking() API.

### Public Members

```
uint32_t flags
```

Bit mask of options for the transfer. See enumeration `_i3c_master_transfer_flags` for available options. Set to 0 or `kI3C_TransferDefaultFlag` for normal transfers.

```
uint8_t slaveAddress
```

The 7-bit slave address.

```
i3c_direction_t direction
```

Either `kI3C_Read` or `kI3C_Write`.

```
uint32_t subaddress
```

Sub address. Transferred MSB first.

```
size_t subaddressSize
```

Length of sub address to send in bytes. Maximum size is 4 bytes.

```
void *data
```

Pointer to data to transfer.

```
size_t dataSize
```

Number of bytes to transfer.

```
i3c_bus_type_t busType
```

bus type.

```
i3c_ibi_response_t ibiResponse
```

ibi response during transfer.

```
struct _i3c_master_handle
    #include <fsl_i3c.h> Driver handle for master non-blocking APIs.
```

---

**Note:** The contents of this structure are private and subject to change.

---



**Public Members**

`uint8_t` state  
Transfer state machine current state.

`uint32_t` remainingBytes  
Remaining byte count in current state.

`i3c_rx_term_ops_t` rxTermOps  
Read termination operation.

`i3c_master_transfer_t` transfer  
Copy of the current transfer info.

`uint8_t` ibiAddress  
Slave address which request IBI.

`uint8_t` \*ibiBuff  
Pointer to IBI buffer to keep ibi bytes.

`size_t` ibiPayloadSize  
IBI payload size.

`i3c_ibi_type_t` ibiType  
IBI type.

`i3c_master_transfer_callback_t` callback  
Callback functions pointer.

`void` \*userData  
Application data passed to callback.

## 2.37 I3C Slave DMA Driver

```
void I3C_SlaveTransferCreateHandleDMA(I3C_Type *base, i3c_slave_dma_handle_t *handle,
                                     i3c_slave_dma_callback_t callback, void *userData,
                                     dma_handle_t *rxDmaHandle, dma_handle_t
                                     *txDmaHandle)
```

Create a new handle for the I3C slave DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_SlaveTransferAbortDMA()` API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

**Parameters**

- base – The I3C peripheral base address.
- handle – Pointer to the I3C slave driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.
- rxDmaHandle – Handle for the DMA receive channel. Created by the user prior to calling this function.
- txDmaHandle – Handle for the DMA transmit channel. Created by the user prior to calling this function.

```
status_t I3C_SlaveTransferDMA(I3C_Type *base, i3c_slave_dma_handle_t *handle,  
                             i3c_slave_dma_transfer_t *transfer, uint32_t eventMask)
```

Prepares for a non-blocking DMA-based transaction on the I3C bus.

The API will do DMA configuration according to the input transfer descriptor, and the data will be transferred when there's bus master requesting transfer from/to this slave. So the timing of call to this API need be aligned with master application to ensure the transfer is executed as expected. Callback specified when the *handle* was created is invoked when the transaction has completed.

#### Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C slave driver handle.
- *transfer* – The pointer to the transfer descriptor.
- *eventMask* – Bit mask formed by OR'ing together *i3c\_slave\_transfer\_event\_t* enumerators to specify which events to send to the callback. The transmit and receive events is not allowed to be enabled.

#### Return values

- *kStatus\_Success* – The transaction was started successfully.
- *kStatus\_I3C\_Busy* – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

```
void I3C_SlaveTransferAbortDMA(I3C_Type *base, i3c_slave_dma_handle_t *handle)
```

Abort a slave dma non-blocking transfer in a early time.

#### Parameters

- *base* – I3C peripheral base address
- *handle* – pointer to *i3c\_slave\_dma\_handle\_t* structure

```
void I3C_SlaveTransferDMAHandleIRQ(I3C_Type *base, void *i3CHandle)
```

Reusable routine to handle slave interrupts.

---

**Note:** This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

---

#### Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C slave DMA driver handle.

```
typedef struct _i3c_slave_dma_handle i3c_slave_dma_handle_t
```

```
typedef struct _i3c_slave_dma_transfer i3c_slave_dma_transfer_t
```

I3C slave transfer structure.

```
typedef void (*i3c_slave_dma_callback_t)(I3C_Type *base, i3c_slave_dma_transfer_t *transfer,  
void *userData)
```

Slave event callback function pointer type.

This callback is used only for the slave DMA transfer API.

#### Param base

Base address for the I3C instance on which the event occurred.

#### Param handle

Pointer to slave DMA transfer handle.

**Param transfer**

Pointer to transfer descriptor containing values passed to and/or from the call-back.

**Param userData**

Arbitrary pointer-sized value passed from the application.

```
struct __i3c_slave_dma_transfer
```

```
#include <fsl_i3c_dma.h> I3C slave transfer structure.
```

```
struct __i3c_slave_dma_handle
```

```
#include <fsl_i3c_dma.h> I3C slave dma handle structure.
```

---

**Note:** The contents of this structure are private and subject to change.

---

## 2.38 I3C Slave Driver

```
void I3C_SlaveGetDefaultConfig(i3c_slave_config_t *slaveConfig)
```

Provides a default configuration for the I3C slave peripheral.

This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableSlave = true;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with `I3C_SlaveInit()`.

**Parameters**

- `slaveConfig` – **[out]** User provided configuration structure for default values. Refer to `i3c_slave_config_t`.

```
void I3C_SlaveInit(I3C_Type *base, const i3c_slave_config_t *slaveConfig, uint32_t  
slowClock_Hz)
```

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

**Parameters**

- `base` – The I3C peripheral base address.
- `slaveConfig` – User provided peripheral configuration. Use `I3C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- `slowClock_Hz` – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If `FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH` defines as 1, this parameter is useless.

```
void I3C_SlaveDeinit(I3C_Type *base)
```

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

**Parameters**

- `base` – The I3C peripheral base address.

static inline void I3C\_SlaveEnable(I3C\_Type \*base, bool isEnabled)  
    Enable/Disable Slave.

**Parameters**

- base – The I3C peripheral base address.
- isEnabled – Enable or disable.

static inline uint32\_t I3C\_SlaveGetStatusFlags(I3C\_Type \*base)  
    Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

**See also:**

`_i3c_slave_flags`

**Parameters**

- base – The I3C peripheral base address.

**Returns**

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C\_SlaveClearStatusFlags(I3C\_Type \*base, uint32\_t statusMask)  
    Clears the I3C slave status flag state.

The following status register flags can be cleared:

- kI3C\_SlaveBusStartFlag
- kI3C\_SlaveMatchedFlag
- kI3C\_SlaveBusStopFlag

Attempts to clear other flags has no effect.

**See also:**

`_i3c_slave_flags`.

**Parameters**

- base – The I3C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetStatusFlags()`.

static inline uint32\_t I3C\_SlaveGetErrorStatusFlags(I3C\_Type \*base)  
    Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

**See also:**

`_i3c_slave_error_flags`

**Parameters**

- base – The I3C peripheral base address.

**Returns**

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_SlaveClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C slave error status flag state.

**See also:**

`_i3c_slave_error_flags`.

**Parameters**

- base – The I3C peripheral base address.
- statusMask – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_slave_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetErrorStatusFlags()`.

```
i3c_slave_activity_state_t I3C_SlaveGetActivityState(I3C_Type *base)
```

Gets the I3C slave state.

**Parameters**

- base – The I3C peripheral base address.

**Returns**

I3C slave activity state, refer `i3c_slave_activity_state_t`.

```
status_t I3C_SlaveCheckAndClearError(I3C_Type *base, uint32_t status)
```

```
static inline void I3C_SlaveEnableInterrupts(I3C_Type *base, uint32_t interruptMask)
```

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`
- `kI3C_SlaveErrorFlag`
- `kI3C_SlaveHDRCommandMatchFlag`
- `kI3C_SlaveCCCHandledFlag`
- `kI3C_SlaveEventSentFlag`

**Parameters**

- base – The I3C peripheral base address.

- `interruptMask` – Bit mask of interrupts to enable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline void I3C_SlaveDisableInterrupts(I3C_Type *base, uint32_t interruptMask)`

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`
- `kI3C_SlaveErrorFlag`
- `kI3C_SlaveHDRCommandMatchFlag`
- `kI3C_SlaveCCCHandledFlag`
- `kI3C_SlaveEventSentFlag`

#### Parameters

- `base` – The I3C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline uint32_t I3C_SlaveGetEnabledInterrupts(I3C_Type *base)`

Returns the set of currently enabled I3C slave interrupt requests.

#### Parameters

- `base` – The I3C peripheral base address.

#### Returns

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

`static inline uint32_t I3C_SlaveGetPendingInterrupts(I3C_Type *base)`

Returns the set of pending I3C slave interrupt requests.

#### Parameters

- `base` – The I3C peripheral base address.

#### Returns

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of pending interrupts.

`static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)`

Enables or disables I3C slave DMA requests.

#### Parameters

- `base` – The I3C peripheral base address.
- `enableTx` – Enable flag for transmit DMA request. Pass true for enable, false for disable.

- `enableRx` – Enable flag for receive DMA request. Pass true for enable, false for disable.
- `width` – DMA read/write unit in bytes.

`static inline uint32_t I3C_SlaveGetTxFifoAddress(I3C_Type *base, uint32_t width)`

Gets I3C slave transmit data register address for DMA transfer.

#### Parameters

- `base` – The I3C peripheral base address.
- `width` – DMA read/write unit in bytes.

#### Returns

The I3C Slave Transmit Data Register address.

`static inline uint32_t I3C_SlaveGetRxFifoAddress(I3C_Type *base, uint32_t width)`

Gets I3C slave receive data register address for DMA transfer.

#### Parameters

- `base` – The I3C peripheral base address.
- `width` – DMA read/write unit in bytes.

#### Returns

The I3C Slave Receive Data Register address.

`static inline void I3C_SlaveSetWatermarks(I3C_Type *base, i3c_tx_trigger_level_t txLvl, i3c_rx_trigger_level_t rxLvl, bool flushTx, bool flushRx)`

Sets the watermarks for I3C slave FIFOs.

#### Parameters

- `base` – The I3C peripheral base address.
- `txLvl` – Transmit FIFO watermark level. The `kI3C_SlaveTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches `txLvl`.
- `rxLvl` – Receive FIFO watermark level. The `kI3C_SlaveRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches `rxLvl`.
- `flushTx` – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- `flushRx` – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

`static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)`

Gets the current number of bytes in the I3C slave FIFOs.

#### Parameters

- `base` – The I3C peripheral base address.
- `txCount` – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

`status_t I3C_SlaveSend(I3C_Type *base, const void *txBuff, size_t txSize)`

Performs a polling send transfer on the I3C bus.

#### Parameters

- `base` – The I3C peripheral base address.

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

**Returns**

Error or success status returned by API.

`status_t I3C_SlaveReceive(I3C_Type *base, void *rxBuff, size_t rxSize)`

Performs a polling receive transfer on the I3C bus.

**Parameters**

- `base` – The I3C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.

**Returns**

Error or success status returned by API.

`void I3C_SlaveTransferCreateHandle(I3C_Type *base, i3c_slave_handle_t *handle, i3c_slave_transfer_callback_t callback, void *userData)`

Creates a new handle for the I3C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_SlaveTransferAbort()` API shall be called.

---

**Note:** The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

---

**Parameters**

- `base` – The I3C peripheral base address.
- `handle` – **[out]** Pointer to the I3C slave driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

`status_t I3C_SlaveTransferNonBlocking(I3C_Type *base, i3c_slave_handle_t *handle, uint32_t eventMask)`

Starts accepting slave transfers.

Call this API after calling `I2C_SlaveInit()` and `I3C_SlaveTransferCreateHandle()` to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to `I3C_SlaveTransferCreateHandle()`. The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the `eventMask` parameter to the OR'd combination of `i3c_slave_transfer_event_t` enumerators for the events you wish to receive. The `kI3C_SlaveTransmitEvent` and `kI3C_SlaveReceiveEvent` events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the `kI3C_SlaveAllEvents` constant is provided as a convenient way to enable all events.

**Parameters**

- `base` – The I3C peripheral base address.
- `handle` – Pointer to struct: `_i3c_slave_handle` structure which stores the transfer state.



- `eventMask` – Bit mask formed by OR'ing together `i3c_slave_transfer_event_t` enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and `kI3C_SlaveAllEvents` to enable all events.

**Return values**

- `kStatus__Success` – Slave transfers were successfully started.
- `kStatus__I3C_Busy` – Slave transfers have already been started on this handle.

`status_t I3C_SlaveTransferGetCount(I3C_Type *base, i3c_slave_handle_t *handle, size_t *count)`

Gets the slave transfer status during a non-blocking transfer.

**Parameters**

- `base` – The I3C peripheral base address.
- `handle` – Pointer to `i3c_slave_handle_t` structure.
- `count` – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

**Return values**

- `kStatus__Success` –
- `kStatus__NoTransferInProgress` –

`void I3C_SlaveTransferAbort(I3C_Type *base, i3c_slave_handle_t *handle)`

Aborts the slave non-blocking transfers.

---

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

---

**Parameters**

- `base` – The I3C peripheral base address.
- `handle` – Pointer to struct: `_i3c_slave_handle` structure which stores the transfer state.

**Return values**

- `kStatus__Success` –
- `kStatus__I3C_Idle` –

`void I3C_SlaveTransferHandleIRQ(I3C_Type *base, void *intHandle)`

Reusable routine to handle slave interrupts.

---

**Note:** This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

---

**Parameters**

- `base` – The I3C peripheral base address.
- `intHandle` – Pointer to struct: `_i3c_slave_handle` structure which stores the transfer state.

`enum _i3c_slave_flags`

I3C slave peripheral flags.

The following status register flags can be cleared:

- kI3C\_SlaveBusStartFlag
- kI3C\_SlaveMatchedFlag
- kI3C\_SlaveBusStopFlag

Only below flags can be enabled as interrupts.

- kI3C\_SlaveBusStartFlag
- kI3C\_SlaveMatchedFlag
- kI3C\_SlaveBusStopFlag
- kI3C\_SlaveRxReadyFlag
- kI3C\_SlaveTxReadyFlag
- kI3C\_SlaveDynamicAddrChangedFlag
- kI3C\_SlaveReceivedCCCFlag
- kI3C\_SlaveErrorFlag
- kI3C\_SlaveHDRCommandMatchFlag
- kI3C\_SlaveCCCHandledFlag
- kI3C\_SlaveEventSentFlag

---

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

---

*Values:*

enumerator kI3C\_SlaveNotStopFlag

Slave status not stop flag

enumerator kI3C\_SlaveMessageFlag

Slave status message, indicating slave is listening to the bus traffic or responding

enumerator kI3C\_SlaveRequiredReadFlag

Slave status required, either is master doing SDR read from slave, or is IBI pushing out.

enumerator kI3C\_SlaveRequiredWriteFlag

Slave status request write, master is doing SDR write to slave, except slave in ENTDAAMode

enumerator kI3C\_SlaveBusDAAModeFlag

I3C bus is in ENTDAAMode

enumerator kI3C\_SlaveBusHDRModeFlag

I3C bus is in HDR mode

enumerator kI3C\_SlaveBusStartFlag

Start/Re-start event is seen since the bus was last cleared

enumerator kI3C\_SlaveMatchedFlag

Slave address(dynamic/static) matched since last cleared

enumerator kI3C\_SlaveBusStopFlag

Stop event is seen since the bus was last cleared

enumerator kI3C\_SlaveRxReadyFlag

Rx data ready in rx buffer flag

enumerator kI3C\_SlaveTxReadyFlag  
Tx buffer ready for Tx data flag

enumerator kI3C\_SlaveDynamicAddrChangedFlag  
Slave dynamic address has been assigned, re-assigned, or lost

enumerator kI3C\_SlaveReceivedCCCFlag  
Slave received Common command code

enumerator kI3C\_SlaveErrorFlag  
Error occurred flag

enumerator kI3C\_SlaveHDRCommandMatchFlag  
High data rate command match

enumerator kI3C\_SlaveCCCHandledFlag  
Slave received Common command code is handled by I3C module

enumerator kI3C\_SlaveEventSentFlag  
Slave IBI/P2P/MR/HJ event has been sent

enumerator kI3C\_SlaveIbiDisableFlag  
Slave in band interrupt is disabled.

enumerator kI3C\_SlaveMasterRequestDisabledFlag  
Slave master request is disabled.

enumerator kI3C\_SlaveHotJoinDisabledFlag  
Slave Hot-Join is disabled.

enumerator kI3C\_SlaveClearFlags  
All flags which are cleared by the driver upon starting a transfer.

enumerator kI3C\_SlaveAllIrqFlags

enum \_i3c\_slave\_error\_flags  
I3C slave error flags to indicate the causes.

---

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

---

*Values:*

enumerator kI3C\_SlaveErrorOvrerrunFlag  
Slave internal from-bus buffer/FIFO overrun.

enumerator kI3C\_SlaveErrorUnderrunFlag  
Slave internal to-bus buffer/FIFO underrun

enumerator kI3C\_SlaveErrorUnderrunNakFlag  
Slave internal from-bus buffer/FIFO underrun and NACK error

enumerator kI3C\_SlaveErrorTermFlag  
Terminate error from master

enumerator kI3C\_SlaveErrorInvalidStartFlag  
Slave invalid start flag

enumerator kI3C\_SlaveErrorSdrParityFlag  
SDR parity error

enumerator kI3C\_SlaveErrorHdrParityFlag  
HDR parity error

enumerator kI3C\_SlaveErrorHdrCRCFlag  
HDR-DDR CRC error

enumerator kI3C\_SlaveErrorS0S1Flag  
S0 or S1 error

enumerator kI3C\_SlaveErrorOverreadFlag  
Over-read error

enumerator kI3C\_SlaveErrorOverwriteFlag  
Over-write error

enum \_i3c\_slave\_event  
I3C slave.event.

*Values:*

enumerator kI3C\_SlaveEventNormal  
Normal mode.

enumerator kI3C\_SlaveEventIBI  
In band interrupt event.

enumerator kI3C\_SlaveEventMasterReq  
Master request event.

enumerator kI3C\_SlaveEventHotJoinReq  
Hot-join event.

enum \_i3c\_slave\_activity\_state  
I3C slave.activity state.

*Values:*

enumerator kI3C\_SlaveNoLatency  
Normal bus operation

enumerator kI3C\_SlaveLatency1Ms  
1ms of latency.

enumerator kI3C\_SlaveLatency100Ms  
100ms of latency.

enumerator kI3C\_SlaveLatency10S  
10s latency.

enum \_i3c\_slave\_transfer\_event

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to `I3C_SlaveTransferNonBlocking()` in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

---

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

---

*Values:*

enumerator kI3C\_SlaveAddressMatchEvent  
Received the slave address after a start or repeated start.

enumerator `kI3C_SlaveTransmitEvent`

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator `kI3C_SlaveReceiveEvent`

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

enumerator `kI3C_SlaveRequiredTransmitEvent`

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

enumerator `kI3C_SlaveStartEvent`

A start/repeated start was detected.

enumerator `kI3C_SlaveHDRCommandMatchEvent`

Slave Match HDR Command.

enumerator `kI3C_SlaveCompletionEvent`

A stop was detected, completing the transfer.

enumerator `kI3C_SlaveRequestSentEvent`

Slave request event sent.

enumerator `kI3C_SlaveReceivedCCCEvent`

Slave received CCC event, need to handle by application.

enumerator `kI3C_SlaveAllEvents`

Bit mask of all available events.

typedef enum `_i3c_slave_event` `i3c_slave_event_t`

I3C slave.event.

typedef enum `_i3c_slave_activity_state` `i3c_slave_activity_state_t`

I3C slave.activity state.

typedef struct `_i3c_slave_config` `i3c_slave_config_t`

Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_SlaveGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum `_i3c_slave_transfer_event` `i3c_slave_transfer_event_t`

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to `I3C_SlaveTransferNonBlocking()` in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

---

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

---

typedef struct `_i3c_slave_transfer` `i3c_slave_transfer_t`

I3C slave transfer structure.

typedef struct `_i3c_slave_handle` `i3c_slave_handle_t`

```
typedef void (*i3c_slave_transfer_callback_t)(I3C_Type *base, i3c_slave_transfer_t *transfer, void *userData)
```

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the I3C\_SlaveSetCallback() function after you have created a handle.

**Param base**

Base address for the I3C instance on which the event occurred.

**Param transfer**

Pointer to transfer descriptor containing values passed to and/or from the callback.

**Param userData**

Arbitrary pointer-sized value passed from the application.

```
typedef void (*i3c_slave_isr_t)(I3C_Type *base, void *handle)
```

Typedef for slave interrupt handler.

```
struct _i3c_slave_config
```

*#include <fsl\_i3c.h>* Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C\_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

## Public Members

bool enableSlave

Whether to enable slave.

uint8\_t staticAddr

Static address.

uint16\_t vendorID

Device vendor ID(manufacture ID).

uint32\_t partNumber

Device part number info

uint8\_t dcr

Device characteristics register information.

uint8\_t bcr

Bus characteristics register information.

uint8\_t hdrMode

Support hdr mode, could be OR logic in enumeration:i3c\_hdr\_mode\_t.

bool nakAllRequest

Whether to reply NAK to all requests except broadcast CCC.

bool ignoreS0S1Error

Whether to ignore S0/S1 error in SDR mode.

bool offline

Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

bool matchSlaveStartStop

Whether to assert start/stop status only the time slave is addressed.

uint32\_t maxWriteLength

Maximum write length.

uint32\_t maxReadLength

Maximum read length.

struct \_\_i3c\_slave\_transfer

*#include <fsl\_i3c.h>* I3C slave transfer structure.

### Public Members

uint32\_t event

Reason the callback is being invoked.

uint8\_t \*txData

Transfer buffer

size\_t txDataSize

Transfer size

uint8\_t \*rxData

Transfer buffer

size\_t rxDataSize

Transfer size

status\_t completionStatus

Success or error code describing how the transfer completed. Only applies for kI3C\_SlaveCompletionEvent.

size\_t transferredCount

Number of bytes actually transferred since start or last repeated start.

struct \_\_i3c\_slave\_handle

*#include <fsl\_i3c.h>* I3C slave handle structure.

---

**Note:** The contents of this structure are private and subject to change.

---

### Public Members

i3c\_slave\_transfer\_t transfer

I3C slave transfer copy.

bool isBusy

Whether transfer is busy.

bool wasTransmit

Whether the last transfer was a transmit.

uint32\_t eventMask

Mask of enabled events.

uint32\_t transferredCount

Count of bytes transferred.

*i3c\_slave\_transfer\_callback\_t* callback  
Callback function called at transfer event.

void \*userData  
Callback parameter passed to callback.

uint8\_t txFifoSize  
Tx Fifo size

## 2.39 IAP Boot Driver

typedef struct *iap\_boot\_option* iap\_boot\_option\_t  
IAP boot option.

void IAP\_RunBootLoader(*iap\_boot\_option\_t* \*option)  
Invoke into ROM with specified boot parameters.

### Parameters

- option – Boot parameters. Refer to *iap\_boot\_option\_t*.

IAP\_BOOT\_OPTION\_TAG  
IAP boot option tag

IAP\_BOOT\_OPTION\_MODE\_MASTER  
IAP boot option mode

IAP\_BOOT\_OPTION\_MODE\_ISP

struct *iap\_boot\_option*  
*#include <fsl\_iap.h>* IAP boot option.

union option

### Public Members

struct *iap\_boot\_option* B

uint32\_t U

struct B

### Public Members

uint32\_t bootImageIndex  
reserved field.

uint32\_t instance  
FlexSPI boot image index for FlexSPI NOR flash.

uint32\_t bootInterface  
Only used when boot interface is FlexSPI/SD/MMC.

uint32\_t mode  
RT500: 0: USART 2: SPI 3: USB HID 4:FlexSPI 6:SD 7:MMC. RT600: 0: USART 1: I2C 2: SPI 3: USB HID 4:FlexSPI 7:SD 8:MMC



## 2.40 IAP: In Application Programming Driver

FSL\_IAP\_DRIVER\_VERSION

IAP driver version.

## 2.41 IAP FlexSPI Driver

FlexSPI Driver status group.

*Values:*

enumerator kStatusGroup\_FlexSPI

enumerator kStatusGroup\_FlexSPINOR

enum \_flexspi\_status

FlexSPI Driver status.

*Values:*

enumerator kStatus\_FLEXSPI\_Success

API is executed successfully

enumerator kStatus\_FLEXSPI\_Fail

API is executed fails

enumerator kStatus\_FLEXSPI\_InvalidArgument

Invalid argument

enumerator kStatus\_FLEXSPI\_SequenceExecutionTimeout

The FlexSPI Sequence Execution timeout

enumerator kStatus\_FLEXSPI\_InvalidSequence

The FlexSPI LUT sequence invalid

enumerator kStatus\_FLEXSPI\_DeviceTimeout

The FlexSPI device timeout

enumerator kStatus\_FLEXSPINOR\_ProgramFail

Status for Page programming failure

enumerator kStatus\_FLEXSPINOR\_EraseSectorFail

Status for Sector Erase failure

enumerator kStatus\_FLEXSPINOR\_EraseAllFail

Status for Chip Erase failure

enumerator kStatus\_FLEXSPINOR\_WaitTimeout

Status for timeout

enumerator kStatus\_FLEXSPINOR\_NotSupported

enumerator kStatus\_FLEXSPINOR\_WriteAlignmentError

Status for Alignment error

enumerator kStatus\_FLEXSPINOR\_CommandFailure

Status for Erase/Program Verify Error

enumerator kStatus\_FLEXSPINOR\_SFDP\_NotFound

Status for SFDP read failure

enumerator kStatus\_FLEXSPINOR\_Unsupported\_SFDP\_Version

Status for Unrecognized SFDP version

enumerator kStatus\_FLEXSPINOR\_Flash\_NotFound

Status for Flash detection failure

enumerator kStatus\_FLEXSPINOR\_DTRRead\_DummyProbeFailed

Status for DDR Read dummy probe failure

Flash Configuration Option0 device\_type.

*Values:*

enumerator kSerialNorCfgOption\_Tag

enumerator kSerialNorCfgOption\_DeviceType\_ReadSFDP\_SDR

enumerator kSerialNorCfgOption\_DeviceType\_ReadSFDP\_DDR

enumerator kSerialNorCfgOption\_DeviceType\_HyperFLASH1V8

enumerator kSerialNorCfgOption\_DeviceType\_HyperFLASH3V0

enumerator kSerialNorCfgOption\_DeviceType\_MacronixOctalDDR

enumerator kSerialNorCfgOption\_DeviceType\_MacronixOctalSDR

enumerator kSerialNorCfgOption\_DeviceType\_MicronOctalDDR

enumerator kSerialNorCfgOption\_DeviceType\_MicronOctalSDR

enumerator kSerialNorCfgOption\_DeviceType\_AdestoOctalDDR

enumerator kSerialNorCfgOption\_DeviceType\_AdestoOctalSDR

Flash Configuration Option0 quad\_mode\_setting.

*Values:*

enumerator kSerialNorQuadMode\_NotConfig

enumerator kSerialNorQuadMode\_StatusReg1\_Bit6

enumerator kSerialNorQuadMode\_StatusReg2\_Bit1

enumerator kSerialNorQuadMode\_StatusReg2\_Bit7

enumerator kSerialNorQuadMode\_StatusReg2\_Bit1\_0x31

Flash Configuration Option0 misc\_mode.

*Values:*

enumerator kSerialNorEnhanceMode\_Disabled

enumerator kSerialNorEnhanceMode\_0\_4\_4\_Mode

enumerator kSerialNorEnhanceMode\_0\_8\_8\_Mode

enumerator kSerialNorEnhanceMode\_DataOrderSwapped

enumerator kSerialNorEnhanceMode\_2ndPinMux

FLEXSPI\_RESET\_PIN boot configurations in OTP.

*Values:*

enumerator kFlashResetLogic\_Disabled

enumerator kFlashResetLogic\_ResetPin

enumerator kFlashResetLogic\_JedecHwReset

Flash Configuration Option1 flash\_connection.

*Values:*

enumerator kSerialNorConnection\_SinglePortA

enumerator kSerialNorConnection\_Parallel

enumerator kSerialNorConnection\_SinglePortB

enumerator kSerialNorConnection\_BothPorts

Flash Device Mode Configuration Sequence.

*Values:*

enumerator kRestoreSequence\_None

enumerator kRestoreSequence\_HW\_Reset

enumerator kRestoreSequence\_4QPI\_FF

enumerator kRestoreSequence\_5QPI\_FF

enumerator kRestoreSequence\_8QPI\_FF

enumerator kRestoreSequence\_Send\_F0

enumerator kRestoreSequence\_Send\_66\_99

enumerator kRestoreSequence\_Send\_6699\_9966

enumerator kRestoreSequence\_Send\_06\_FF

Flash Config Mode Definition.

*Values:*

enumerator kFlashInstMode\_ExtendedSpi

enumerator kFlashInstMode\_0\_4\_4\_SDR

enumerator kFlashInstMode\_0\_4\_4\_DDR

enumerator kFlashInstMode\_QPI\_SDR

enumerator kFlashInstMode\_QPI\_DDR

enumerator kFlashInstMode\_OPI\_SDR

enumerator kFlashInstMode\_OPI\_DDR

#### Flash Device Type Definition.

##### *Values:*

enumerator kFlexSpiDeviceType\_SerialNOR

Flash devices are Serial NOR

enumerator kFlexSpiDeviceType\_SerialNAND

Flash devices are Serial NAND

enumerator kFlexSpiDeviceType\_SerialRAM

Flash devices are Serial RAM/HyperFLASH

enumerator kFlexSpiDeviceType\_MCP\_NOR\_NAND

Flash device is MCP device, A1 is Serial NOR, A2 is Serial NAND

enumerator kFlexSpiDeviceType\_MCP\_NOR\_RAM

Flash device is MCP device, A1 is Serial NOR, A2 is Serial RAMs

#### Flash Pad Definitions.

##### *Values:*

enumerator kSerialFlash\_1Pad

enumerator kSerialFlash\_2Pads

enumerator kSerialFlash\_4Pads

enumerator kSerialFlash\_8Pads

#### Flash Configuration Command Type.

##### *Values:*

enumerator kDeviceConfigCmdType\_Generic

Generic command, for example: configure dummy cycles, drive strength, etc

enumerator kDeviceConfigCmdType\_QuadEnable

Quad Enable command

enumerator kDeviceConfigCmdType\_Spi2Xpi

Switch from SPI to DPI/QPI/OPI mode

enumerator kDeviceConfigCmdType\_Xpi2Spi

Switch from DPI/QPI/OPI to SPI mode

enumerator kDeviceConfigCmdType\_Spi2NoCmd

Switch to 0-4-4/0-8-8 mode

enumerator kDeviceConfigCmdType\_Reset

Reset device command

enum \_FlexSPIOperationType

FlexSPI Operation Type.

##### *Values:*

enumerator kFlexSpiOperation\_Command

FlexSPI operation: Only command, both TX and

enumerator `kFlexSpiOperation_Config`

RX buffer are ignored. FlexSPI operation: Configure device mode, the

enumerator `kFlexSpiOperation_Write`

TX FIFO size is fixed in LUT. FlexSPI operation: Write, only TX buffer is

enumerator `kFlexSpiOperation_Read`

effective FlexSPI operation: Read, only Rx Buffer is

enumerator `kFlexSpiOperation_End`

effective.

typedef struct *\_serial\_nor\_config\_option* serial\_nor\_config\_option\_t

Serial NOR Configuration Option.

typedef struct *\_lut\_sequence* flexspi\_lut\_seq\_t

FlexSPI LUT Sequence structure.

typedef struct *\_FlexSPISConfig* flexspi\_mem\_config\_block\_t

FlexSPI Memory Configuration Block.

typedef enum *\_FlexSPIOperationType* flexspi\_operation\_t

FlexSPI Operation Type.

typedef struct *\_FlexSpiXfer* flexspi\_xfer\_t

FlexSPI Transfer Context.

typedef struct *\_flexspi\_nor\_config* flexspi\_nor\_config\_t

Serial NOR configuration block.

AT\_QUICKACCESS\_SECTION\_CODE (status\_t IAP\_FlexspiNorInit(uint32\_t instance, flexspi\_nor\_config\_t \*config))

Initialize Serial NOR devices via FlexSPI.

This function configures the FlexSPI controller with the arguments pointed by param config.

#### Parameters

- instance – FlexSPI controller instance, only support 0.
- config – The Flash configuration block. Refer to flexspi\_nor\_config\_t.

#### Returns

The status flags. This is a member of the enumeration flexspi\_status

status\_t IAP\_FlexspiNorPageProgram(uint32\_t instance, flexspi\_nor\_config\_t \*config, uint32\_t dstAddr, const uint32\_t \*src)

Program data to Serial NOR via FlexSPI.

This function Program data to specified destination address.

#### Parameters

- instance – FlexSPI controller instance, only support 0.
- config – The Flash configuration block. Refer to flexspi\_nor\_config\_t.
- dstAddr – The destination address to be programmed.
- src – Points to the buffer which hold the data to be programmed.

#### Returns

The status flags. This is a member of the enumeration flexspi\_status

*status\_t* IAP\_FlexspiNorEraseAll(uint32\_t instance, *flexspi\_nor\_config\_t* \*config)

Erase all the Serial NOR devices connected on FlexSPI.

#### Parameters

- instance – FlexSPI controller instance, only support 0.
- config – The Flash configuration block. Refer to *flexspi\_nor\_config\_t*.

#### Returns

The status flags. This is a member of the enumeration *\_flexspi\_status*

*status\_t* IAP\_FlexspiNorErase(uint32\_t instance, *flexspi\_nor\_config\_t* \*config, uint32\_t start, uint32\_t length)

Erase Flash Region specified by address and length.

#### Parameters

- instance – FlexSPI controller instance, only support 0.
- config – The Flash configuration block. Refer to *flexspi\_nor\_config\_t*.
- start – The start address to be erased.
- length – The length to be erased.

#### Returns

The status flags. This is a member of the enumeration *\_flexspi\_status*

*status\_t* IAP\_FlexspiNorEraseSector(uint32\_t instance, *flexspi\_nor\_config\_t* \*config, uint32\_t address)

Erase one sector specified by address.

#### Parameters

- instance – FlexSPI controller instance, only support 0.
- config – The Flash configuration block. Refer to *flexspi\_nor\_config\_t*.
- address – The address of the sector to be erased.

#### Returns

The status flags. This is a member of the enumeration *\_flexspi\_status*

*status\_t* IAP\_FlexspiNorEraseBlock(uint32\_t instance, *flexspi\_nor\_config\_t* \*config, uint32\_t address)

Erase one block specified by address.

#### Parameters

- instance – FlexSPI controller instance, only support 0.
- config – The Flash configuration block. Refer to *flexspi\_nor\_config\_t*.
- address – The address of the block to be erased.

#### Returns

The status flags. This is a member of the enumeration *\_flexspi\_status*

*status\_t* IAP\_FlexspiNorGetConfig(uint32\_t instance, *flexspi\_nor\_config\_t* \*config, *serial\_nor\_config\_option\_t* \*option)

Get FlexSPI NOR Configuration Block based on specified option.

#### Parameters

- instance – FlexSPI controller instance, only support 0.
- config – The Flash configuration block. Refer to *flexspi\_nor\_config\_t*.
- option – The Flash Configuration Option block. Refer to *serial\_nor\_config\_option\_t*.

**Returns**

The status flags. This is a member of the enumeration `_flexspi_status`

`status_t IAP_FlexspiNorRead(uint32_t instance, flexspi_nor_config_t *config, uint32_t *dst, uint32_t start, uint32_t bytes)`

Read data from Flexspi NOR Flash.

**Parameters**

- `instance` – FlexSPI controller instance, only support 0.
- `config` – The Flash configuration block. Refer to `flexspi_nor_config_t`.
- `dst` – Buffer address used to store the read data.
- `start` – The Read address.
- `bytes` – The Read size

**Returns**

The status flags. This is a member of the enumeration `_flexspi_status`

`status_t IAP_FlexspiXfer(uint32_t instance, flexspi_xfer_t *xfer)`

Get FlexSPI Xfer data.

**Parameters**

- `instance` – FlexSPI controller instance, only support 0.
- `xfer` – The FlexSPI Transfer Context block. Refer to `flexspi_xfer_t`.

**Returns**

The status flags. This is a member of the enumeration `_flexspi_status`

`status_t IAP_FlexspiUpdateLut(uint32_t instance, uint32_t seqIndex, const uint32_t *lutBase, uint32_t numberOfSeq)`

Update FlexSPI Lookup table.

**Parameters**

- `instance` – FlexSPI controller instance, only support 0.
- `seqIndex` – The index of FlexSPI LUT to be updated.
- `lutBase` – Points to the buffer which hold the LUT data to be programmed.
- `numberOfSeq` – The number of LUT seq that need to be updated.

**Returns**

The status flags. This is a member of the enumeration `_flexspi_status`

`status_t IAP_FlexspiSetClockSource(uint32_t clockSrc)`

Set the clock source for FlexSPI.

**Parameters**

- `clockSrc` – Clock source for flexspi interface.

**Returns**

The status flags. This is a member of the enumeration `_flexspi_status`

`void IAP_FlexspiConfigClock(uint32_t instance, uint32_t freqOption, uint32_t sampleClkMode)`

Configure the flexspi interface clock frequency and data sample mode.

**Parameters**

- `instance` – FlexSPI controller instance, only support 0.
- `freqOption` – FlexSPI interface clock frequency selection.
- `sampleClkMode` – FlexSPI controller data sample mode.

**Returns**

The status flags. This is a member of the enumeration `_flexspi_status`

`AT_QUICKACCESS_SECTION_CODE (status_t IAP_FlexspiNorAutoConfig(uint32_t instance, flexspi_nor_config_t *config, serial_nor_config_option_t *option))`

Configure flexspi nor automatically.

**Parameters**

- `instance` – FlexSPI controller instance, only support 0.
- `config` – The Flash configuration block. Refer to `flexspi_nor_config_t`.
- `option` – The Flash Configuration Option block. Refer to `serial_nor_config_option_t`.

**Returns**

The status flags. This is a member of the enumeration `_flexspi_status`

`NOR_CMD_INDEX_READ`

FlexSPI LUT command.

0

`NOR_CMD_INDEX_READSTATUS`

1

`NOR_CMD_INDEX_WRITEENABLE`

2

`NOR_CMD_INDEX_ERASESECTOR`

3

`NOR_CMD_INDEX_PAGEPROGRAM`

4

`NOR_CMD_INDEX_CHIPERASE`

5

`NOR_CMD_INDEX_DUMMY`

6

`NOR_CMD_INDEX_ERASEBLOCK`

7

`NOR_CMD_LUT_SEQ_IDX_READ`

0 READ LUT sequence id in lookupTable stored in config block

`NOR_CMD_LUT_SEQ_IDX_READSTATUS`

1 Read Status LUT sequence id in lookupTable stored in config block

`NOR_CMD_LUT_SEQ_IDX_READSTATUS_XPI`

2 Read status DPI/QPI/OPI sequence id in lookupTable stored in config block

`NOR_CMD_LUT_SEQ_IDX_WRITEENABLE`

3 Write Enable sequence id in lookupTable stored in config block

`NOR_CMD_LUT_SEQ_IDX_WRITEENABLE_XPI`

4 Write Enable DPI/QPI/OPI sequence id in lookupTable stored in config block

`NOR_CMD_LUT_SEQ_IDX_ERASESECTOR`

5 Erase Sector sequence id in lookupTable stored in config block

`NOR_CMD_LUT_SEQ_IDX_ERASEBLOCK`

8 Erase Block sequence id in lookupTable stored in config block



NOR\_CMD\_LUT\_SEQ\_IDX\_PAGEPROGRAM

9 Program sequence id in lookupTable stored in config block

NOR\_CMD\_LUT\_SEQ\_IDX\_CHIPERASE

11 Chip Erase sequence in lookupTable id stored in config block

NOR\_CMD\_LUT\_SEQ\_IDX\_READ\_SFDP

13 Read SFDP sequence in lookupTable id stored in config block

NOR\_CMD\_LUT\_SEQ\_IDX\_RESTORE\_NOCMD

14 Restore 0-4-4/0-8-8 mode sequence id in lookupTable stored in config block

NOR\_CMD\_LUT\_SEQ\_IDX\_EXIT\_NOCMD

15 Exit 0-4-4/0-8-8 mode sequence id in lookupTable stored in config block

struct \_serial\_nor\_config\_option

*#include <fsl\_iap.h>* Serial NOR Configuration Option.

union flash\_run\_context\_t

*#include <fsl\_iap.h>* Flash Run Context.

### Public Members

struct *flash\_run\_context\_t* B

uint32\_t U

struct \_lut\_sequence

*#include <fsl\_iap.h>* FlexSPI LUT Sequence structure.

### Public Members

uint8\_t seqNum

Sequence Number, valid number: 1-16

uint8\_t seqId

Sequence Index, valid number: 0-15

struct flexspi\_dll\_time\_t

*#include <fsl\_iap.h>* FlexSPI Dll Time Block.

struct \_FlexSPIConfig

*#include <fsl\_iap.h>* FlexSPI Memory Configuration Block.

### Public Members

uint32\_t tag

[0x000-0x003] Tag, fixed value 0x42464346UL

uint32\_t version

[0x004-0x007] Version,[31:24] -'V', [23:16] - Major, [15:8] - Minor, [7:0] - bugfix

uint32\_t reserved0

[0x008-0x00b] Reserved for future use

uint8\_t readSampleClkSrc

[0x00c-0x00c] Read Sample Clock Source, valid value: 0/1/3

`uint8_t csHoldTime`  
[0x00d-0x00d] CS hold time, default value: 3

`uint8_t csSetupTime`  
[0x00e-0x00e] CS setup time, default value: 3

`uint8_t columnAddressWidth`  
[0x00f-0x00f] Column Address with, for HyperBus protocol, it is fixed to 3, For

`uint8_t deviceModeCfgEnable`  
Serial NAND, need to refer to datasheet [0x010-0x010] Device Mode Configure enable flag, 1 - Enable, 0 - Disable

`uint8_t deviceModeType`  
[0x011-0x011] Specify the configuration command type:Quad Enable, DPI/QPI/OPI switch,

`uint16_t waitTimeCfgCommands`  
Generic configuration, etc. [0x012-0x013] Wait time for all configuration commands, unit: 100us, Used for

*flexspi\_lut\_seq\_t* `deviceModeSeq`  
DPI/QPI/OPI switch or reset command [0x014-0x017] Device mode sequence info, [7:0] - LUT sequence id, [15:8] - LUT

`uint32_t deviceModeArg`  
sequence number, [31:16] Reserved [0x018-0x01b] Argument/Parameter for device configuration

`uint8_t configCmdEnable`  
[0x01c-0x01c] Configure command Enable Flag, 1 - Enable, 0 - Disable

`uint8_t configModeType[3]`  
[0x01d-0x01f] Configure Mode Type, similar as deviceModeType

*flexspi\_lut\_seq\_t* `configCmdSeqs[3]`  
[0x020-0x02b] Sequence info for Device Configuration command, similar as device-ModeSeq

`uint32_t reserved1`  
[0x02c-0x02f] Reserved for future use

`uint32_t configCmdArgs[3]`  
[0x030-0x03b] Arguments/Parameters for device Configuration commands

`uint32_t reserved2`  
[0x03c-0x03f] Reserved for future use

`uint32_t controllerMiscOption`  
[0x040-0x043] Controller Misc Options, see Misc feature bit definitions for more

`uint8_t deviceType`  
details [0x044-0x044] Device Type: See Flash Type Definition for more details

`uint8_t sflashPadType`  
[0x045-0x045] Serial Flash Pad Type: 1 - Single, 2 - Dual, 4 - Quad, 8 - Octal

`uint8_t serialClkFreq`  
[0x046-0x046] Serial Flash Frequency, device specific definitions, See System Boot

`uint8_t lutCustomSeqEnable`  
Chapter for more details [0x047-0x047] LUT customization Enable, it is required if the program/erase cannot

```

uint32_t reserved3[2]
    be done using 1 LUT sequence, currently, only applicable to HyperFLASH [0x048-
    0x04f] Reserved for future use

uint32_t sflashA1Size
    [0x050-0x053] Size of Flash connected to A1

uint32_t sflashA2Size
    [0x054-0x057] Size of Flash connected to A2

uint32_t sflashB1Size
    [0x058-0x05b] Size of Flash connected to B1

uint32_t sflashB2Size
    [0x05c-0x05f] Size of Flash connected to B2

uint32_t csPadSettingOverride
    [0x060-0x063] CS pad setting override value

uint32_t sclkPadSettingOverride
    [0x064-0x067] SCK pad setting override value

uint32_t dataPadSettingOverride
    [0x068-0x06b] data pad setting override value

uint32_t dqsPadSettingOverride
    [0x06c-0x06f] DQS pad setting override value

uint32_t timeoutInMs
    [0x070-0x073] Timeout threshold for read status command

uint32_t commandInterval
    [0x074-0x077] CS deselect interval between two commands

flexspi_dll_time_t dataValidTime[2]
    [0x078-0x07b] CLK edge to data valid time for PORT A and PORT B

uint16_t busyOffset
    [0x07c-0x07d] Busy offset, valid value: 0-31

uint16_t busyBitPolarity
    [0x07e-0x07f] Busy flag polarity, 0 - busy flag is 1 when flash device is busy, 1 -

uint32_t lookupTable[64]
    busy flag is 0 when flash device is busy [0x080-0x17f] Lookup table holds Flash com-
    mand sequences

flexspi_lut_seq_t lutCustomSeq[12]
    [0x180-0x1af] Customizable LUT Sequences

uint32_t reserved4[4]
    [0x1b0-0x1bf] Reserved for future use

struct __FlexSpiXfer
    #include <fsl_iap.h> FlexSPI Transfer Context.

```

### Public Members

```

flexspi_operation_t operation
    FlexSPI operation

```

uint32\_t baseAddress  
FlexSPI operation base address

uint32\_t seqId  
Sequence Id

uint32\_t seqNum  
Sequence Number

bool isParallelModeEnable  
Is a parallel transfer

uint32\_t \*txBuffer  
Tx buffer

uint32\_t txSize  
Tx size in bytes

uint32\_t \*rxBuffer  
Rx buffer

uint32\_t rxSize  
Rx size in bytes

struct \_\_flexspi\_nor\_config  
#include <fsl\_iap.h> Serial NOR configuration block.

### Public Members

*flexspi\_mem\_config\_block\_t* memConfig  
Common memory configuration info via FlexSPI

uint32\_t pageSize  
Page size of Serial NOR

uint32\_t sectorSize  
Sector size of Serial NOR

uint8\_t ipcmdSerialClkFreq  
Clock frequency for IP command

uint8\_t isUniformBlockSize  
Sector/Block size is the same

uint8\_t isDataOrderSwapped  
Data order (D0, D1, D2, D3) is swapped (D1,D0, D3, D2)

uint8\_t reserved0[1]  
Reserved for future use

uint8\_t serialNorType  
Serial NOR Flash type: 0/1/2/3

uint8\_t needExitNoCmdMode  
Need to exit NoCmd mode before other IP command

uint8\_t halfClkForNonReadCmd  
Half the Serial Clock for non-read command: true/false

uint8\_t needRestoreNoCmdMode  
Need to Restore NoCmd mode after IP commmand execution

```
uint32_t blockSize
    Block size
uint32_t flashStateCtx
    Flash State Context
uint32_t reserve2[10]
    Reserved for future use
union option0
```

### Public Members

```
struct _serial_nor_config_option B
uint32_t U
struct B
```

### Public Members

```
uint32_t max_freq
    Maximum supported Frequency
uint32_t misc_mode
    miscellaneous mode
uint32_t quad_mode_setting
    Quad mode setting
uint32_t cmd_pads
    Command pads
uint32_t query_pads
    SFDP read pads
uint32_t device_type
    Device type
uint32_t option_size
    Option size, in terms of uint32_t, size = (option_size + 1) * 4
uint32_t tag
    Tag, must be 0x0C
union option1
```

### Public Members

```
struct _serial_nor_config_option B
uint32_t U
struct B
```

### Public Members

uint32\_t dummy\_\_cycles

Dummy cycles before read

uint32\_t status\_\_override

Override status register value during device mode configuration

uint32\_t pinmux\_\_group

The pinmux group selection

uint32\_t dqs\_\_pinmux\_\_group

The DQS Pinmux Group Selection

uint32\_t drive\_\_strength

The Drive Strength of FlexSPI Pads

uint32\_t flash\_\_connection

Flash connection option: 0 - Single Flash connected to port A, 1 -

struct B

## 2.42 IAP OTP Driver

OTP Status Group.

*Values:*

enumerator kStatusGroup\_\_OtpGroup

OTP Error Status definitions.

*Values:*

enumerator kStatus\_\_OTP\_\_InvalidAddress

Invalid OTP address

enumerator kStatus\_\_OTP\_\_ProgramFail

Program Fail

enumerator kStatus\_\_OTP\_\_CrcFail

CrcCheck Fail

enumerator kStatus\_\_OTP\_\_Error

Errors happened during OTP operation

enumerator kStatus\_\_OTP\_\_EccCheckFail

Ecc Check failed during OTP operation

enumerator kStatus\_\_OTP\_\_Locked

OTP Fuse field has been locked

enumerator kStatus\_\_OTP\_\_Timeout

OTP operation time out

enumerator kStatus\_\_OTP\_\_CrcCheckPass

OTP CRC Check Pass

*status\_t* IAP\_OtpInit(uint32\_t src\_clk\_freq)

Initialize OTP controller.

This function enables OTP Controller clock.

**Parameters**

- src\_clk\_freq – The Frequency of the source clock of OTP controller

**Returns**

kStatus\_Success

*status\_t* IAP\_OtpDeinit(void)

De-Initialize OTP controller.

This function disables OTP Controller Clock.

**Returns**

kStatus\_Success

*status\_t* IAP\_OtpFuseRead(uint32\_t addr, uint32\_t \*data)

Read Fuse value from OTP Fuse Block.

This function read fuse data from OTP Fuse block to specified data buffer.

**Parameters**

- addr – Fuse address
- data – Buffer to hold the data read from OTP Fuse block

**Returns**

kStatus\_Success - Data read from OTP Fuse block successfully  
kStatus\_InvalidArgument - data pointer is invalid  
kStatus\_OTP\_EccCheckFail - Ecc Check Failed  
kStatus\_OTP\_Error - Other Errors

*status\_t* IAP\_OtpFuseProgram(uint32\_t addr, uint32\_t data, bool lock)

Program value to OTP Fuse block.

This function program data to specified OTP Fuse address.

**Parameters**

- addr – Fuse address
- data – data to be programmed into OTP Fuse block
- lock – lock the fuse field or not

**Returns**

kStatus\_Success - Data has been programmed into OTP Fuse block successfully  
kStatus\_OTP\_ProgramFail - Fuse programming failed  
kStatus\_OTP\_Locked - The address to be programmed into is locked  
kStatus\_OTP\_Error - Other Errors

*status\_t* IAP\_OtpShadowRegisterReload(void)

Reload all shadow registers from OTP fuse block.

This function reloads all the shadow registers from OTP Fuse block

**Returns**

kStatus\_Success - Shadow registers' reloading succeeded.  
kStatus\_OTP\_EccCheckFail - Ecc Check Failed  
kStatus\_OTP\_Error - Other Errors

*status\_t* IAP\_OtpCrcCheck(uint32\_t start\_addr, uint32\_t end\_addr, uint32\_t crc\_addr)

Do CRC Check via OTP controller.

This function checks whether data in specified fuse address ranges match the crc value in the specified CRC address and return the actual crc value as needed.

**Parameters**

- `start_addr` – Start address of selected Fuse address range
- `end_addr` – End address of selected Fuse address range
- `crc_addr` – Address that hold CRC data

**Returns**

`kStatus_Success` CRC check succeeded, CRC value matched. `kStatus_InvalidArgument` - Invalid Argument `kStatus_OTP_EccCheckFail` Ecc Check Failed `kStatus_OTP_CrcFail` CRC Check Failed

`status_t` IAP\_OtpCrcCalc(uint32\_t \*src, uint32\_t numberOfWords, uint32\_t \*crcChecksum)

Calculate the CRC checksum for specified data for OTP.

This function calculates the CRC checksum for specified data for OTP

**Parameters**

- `src` – the source address of data
- `numberOfWords` – number of Fuse words
- `crcChecksum` – Buffer to store the CRC checksum

**Returns**

`kStatus_Success` CRC checksum is computed successfully. `kStatus_InvalidArgument` - Invalid Argument

## 2.43 INPUTMUX: Input Multiplexing Driver

FSL\_INPUTMUX\_DRIVER\_VERSION

Group interrupt driver version for SDK.

enum \_\_inputmux\_connection\_t

INPUTMUX connections type.

*Values:*

enumerator `kINPUTMUX_Sct0PinInp0ToSct0`  
SCT INMUX.

enumerator `kINPUTMUX_Sct0PinInp1ToSct0`

enumerator `kINPUTMUX_Sct0PinInp2ToSct0`

enumerator `kINPUTMUX_Sct0PinInp3ToSct0`

enumerator `kINPUTMUX_Sct0PinInp4ToSct0`

enumerator `kINPUTMUX_Sct0PinInp5ToSct0`

enumerator `kINPUTMUX_Sct0PinInp6ToSct0`

enumerator `kINPUTMUX_Sct0PinInp7ToSct0`

enumerator `kINPUTMUX_Ctimer0Mat0ToSct0`

enumerator `kINPUTMUX_Ctimer1Mat0ToSct0`

enumerator `kINPUTMUX_Ctimer2Mat0ToSct0`

enumerator `kINPUTMUX_Ctimer3Mat0ToSct0`



enumerator kINPUTMUX\_\_Ctimer4Mat0ToSct0  
enumerator kINPUTMUX\_\_AdcIrqToSct0  
enumerator kINPUTMUX\_\_GpioIntBmatchToSct0  
enumerator kINPUTMUX\_\_Usb1FrameToggleToSct0  
enumerator kINPUTMUX\_\_Cmp0OutToSct0  
enumerator kINPUTMUX\_\_SharedI2s0SclkToSct0  
enumerator kINPUTMUX\_\_SharedI2s1SclkToSct0  
enumerator kINPUTMUX\_\_SharedI2s0WsToSct0  
enumerator kINPUTMUX\_\_SharedI2s1WsToSct0  
enumerator kINPUTMUX\_\_MclkToSct0  
enumerator kINPUTMUX\_\_ArmTxevToSct0  
enumerator kINPUTMUX\_\_DebugHaltedToSct0  
    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

DSP Interrupt.

enumerator kINPUTMUX\_Flexcomm0ToDspInterrupt

enumerator kINPUTMUX\_Flexcomm1ToDspInterrupt

enumerator kINPUTMUX\_Flexcomm2ToDspInterrupt

enumerator kINPUTMUX\_Flexcomm3ToDspInterrupt

enumerator kINPUTMUX\_Flexcomm4ToDspInterrupt

enumerator kINPUTMUX\_Flexcomm5ToDspInterrupt

enumerator kINPUTMUX\_Flexcomm6ToDspInterrupt

enumerator kINPUTMUX\_Flexcomm7ToDspInterrupt

enumerator kINPUTMUX\_GpioInt0ToDspInterrupt

enumerator kINPUTMUX\_GpioInt1ToDspInterrupt

enumerator kINPUTMUX\_GpioInt2ToDspInterrupt

enumerator kINPUTMUX\_GpioInt3ToDspInterrupt

enumerator kINPUTMUX\_GpioInt4ToDspInterrupt

enumerator kINPUTMUX\_GpioInt5ToDspInterrupt

enumerator kINPUTMUX\_GpioInt6ToDspInterrupt

enumerator kINPUTMUX\_GpioInt7ToDspInterrupt

enumerator kINPUTMUX\_NsHsGpioInt0ToDspInterrupt

enumerator kINPUTMUX\_NsHsGpioInt1ToDspInterrupt

enumerator kINPUTMUX\_Wdt1ToDspInterrupt

enumerator kINPUTMUX\_Dmac0ToDspInterrupt

enumerator kINPUTMUX\_Dmac1ToDspInterrupt

enumerator kINPUTMUX\_\_MuBToDspInterrupt  
enumerator kINPUTMUX\_\_Utick0ToDspInterrupt  
enumerator kINPUTMUX\_\_Mrt0ToDspInterrupt  
enumerator kINPUTMUX\_\_OsEventTimerToDspInterrupt  
enumerator kINPUTMUX\_\_Ctimer0ToDspInterrupt  
enumerator kINPUTMUX\_\_Ctimer1ToDspInterrupt  
enumerator kINPUTMUX\_\_Ctimer2ToDspInterrupt  
enumerator kINPUTMUX\_\_Ctimer3ToDspInterrupt  
enumerator kINPUTMUX\_\_Ctimer4ToDspInterrupt  
enumerator kINPUTMUX\_\_RtcToDspInterrupt  
enumerator kINPUTMUX\_\_I3c0ToDspInterrupt  
enumerator kINPUTMUX\_\_Dmic0ToDspInterrupt  
enumerator kINPUTMUX\_\_Hwvad0ToDspInterrupt  
enumerator kINPUTMUX\_\_FlexspiToDspInterrupt

Frequency measure.

enumerator kINPUTMUX\_\_XtalinToFreqmeas  
enumerator kINPUTMUX\_\_SfroToFreqmeas  
enumerator kINPUTMUX\_\_FfroToFreqmeas  
enumerator kINPUTMUX\_\_LposcToFreqmeas  
enumerator kINPUTMUX\_\_Rtc32KhzOscToFreqmeas  
enumerator kINPUTMUX\_\_MainSysClkToFreqmeas  
enumerator kINPUTMUX\_\_FreqmeGpioClkToFreqmeas

CTmier0 capture input mux.

enumerator kINPUTMUX\_\_CtInp0ToTimer0CaptureChannels  
enumerator kINPUTMUX\_\_CtInp1ToTimer0CaptureChannels  
enumerator kINPUTMUX\_\_CtInp2ToTimer0CaptureChannels  
enumerator kINPUTMUX\_\_CtInp3ToTimer0CaptureChannels  
enumerator kINPUTMUX\_\_CtInp4ToTimer0CaptureChannels  
enumerator kINPUTMUX\_\_CtInp5ToTimer0CaptureChannels  
enumerator kINPUTMUX\_\_CtInp6ToTimer0CaptureChannels  
enumerator kINPUTMUX\_\_CtInp7ToTimer0CaptureChannels  
enumerator kINPUTMUX\_\_CtInp8ToTimer0CaptureChannels  
enumerator kINPUTMUX\_\_CtInp9ToTimer0CaptureChannels  
enumerator kINPUTMUX\_\_CtInp10ToTimer0CaptureChannels

enumerator kINPUTMUX\_\_CtInp11ToTimer0CaptureChannels  
enumerator kINPUTMUX\_\_CtInp12ToTimer0CaptureChannels  
enumerator kINPUTMUX\_\_CtInp13ToTimer0CaptureChannels  
enumerator kINPUTMUX\_\_CtInp14ToTimer0CaptureChannels  
enumerator kINPUTMUX\_\_CtInp15ToTimer0CaptureChannels  
enumerator kINPUTMUX\_\_SharedI2s0WsToTimer0CaptureChannels  
enumerator kINPUTMUX\_\_SharedI2s1WsToTimer0CaptureChannels  
enumerator kINPUTMUX\_\_Usb1FrameToggleToTimer0CaptureChannels

CTmier1 capture input mux.

enumerator kINPUTMUX\_\_CtInp0ToTimer1CaptureChannels  
enumerator kINPUTMUX\_\_CtInp1ToTimer1CaptureChannels  
enumerator kINPUTMUX\_\_CtInp2ToTimer1CaptureChannels  
enumerator kINPUTMUX\_\_CtInp3ToTimer1CaptureChannels  
enumerator kINPUTMUX\_\_CtInp4ToTimer1CaptureChannels  
enumerator kINPUTMUX\_\_CtInp5ToTimer1CaptureChannels  
enumerator kINPUTMUX\_\_CtInp6ToTimer1CaptureChannels  
enumerator kINPUTMUX\_\_CtInp7ToTimer1CaptureChannels  
enumerator kINPUTMUX\_\_CtInp8ToTimer1CaptureChannels  
enumerator kINPUTMUX\_\_CtInp9ToTimer1CaptureChannels  
enumerator kINPUTMUX\_\_CtInp10ToTimer1CaptureChannels  
enumerator kINPUTMUX\_\_CtInp11ToTimer1CaptureChannels  
enumerator kINPUTMUX\_\_CtInp12ToTimer1CaptureChannels  
enumerator kINPUTMUX\_\_CtInp13ToTimer1CaptureChannels  
enumerator kINPUTMUX\_\_CtInp14ToTimer1CaptureChannels  
enumerator kINPUTMUX\_\_CtInp15ToTimer1CaptureChannels  
enumerator kINPUTMUX\_\_SharedI2s0WsToTimer1CaptureChannels  
enumerator kINPUTMUX\_\_SharedI2s1WsToTimer1CaptureChannels  
enumerator kINPUTMUX\_\_Usb1FrameToggleToTimer1CaptureChannels

CTmier2 capture input mux.

enumerator kINPUTMUX\_\_CtInp0ToTimer2CaptureChannels  
enumerator kINPUTMUX\_\_CtInp1ToTimer2CaptureChannels  
enumerator kINPUTMUX\_\_CtInp2ToTimer2CaptureChannels  
enumerator kINPUTMUX\_\_CtInp3ToTimer2CaptureChannels  
enumerator kINPUTMUX\_\_CtInp4ToTimer2CaptureChannels

enumerator kINPUTMUX\_CtInp5ToTimer2CaptureChannels  
enumerator kINPUTMUX\_CtInp6ToTimer2CaptureChannels  
enumerator kINPUTMUX\_CtInp7ToTimer2CaptureChannels  
enumerator kINPUTMUX\_CtInp8ToTimer2CaptureChannels  
enumerator kINPUTMUX\_CtInp9ToTimer2CaptureChannels  
enumerator kINPUTMUX\_CtInp10ToTimer2CaptureChannels  
enumerator kINPUTMUX\_CtInp11ToTimer2CaptureChannels  
enumerator kINPUTMUX\_CtInp12ToTimer2CaptureChannels  
enumerator kINPUTMUX\_CtInp13ToTimer2CaptureChannels  
enumerator kINPUTMUX\_CtInp14ToTimer2CaptureChannels  
enumerator kINPUTMUX\_CtInp15ToTimer2CaptureChannels  
enumerator kINPUTMUX\_SharedI2s0WsToTimer2CaptureChannels  
enumerator kINPUTMUX\_SharedI2s1WsToTimer2CaptureChannels  
enumerator kINPUTMUX\_Usb1FrameToggleToTimer2CaptureChannels  
CTmier3 capture input mux.  
enumerator kINPUTMUX\_CtInp0ToTimer3CaptureChannels  
enumerator kINPUTMUX\_CtInp1ToTimer3CaptureChannels  
enumerator kINPUTMUX\_CtInp2ToTimer3CaptureChannels  
enumerator kINPUTMUX\_CtInp3ToTimer3CaptureChannels  
enumerator kINPUTMUX\_CtInp4ToTimer3CaptureChannels  
enumerator kINPUTMUX\_CtInp5ToTimer3CaptureChannels  
enumerator kINPUTMUX\_CtInp6ToTimer3CaptureChannels  
enumerator kINPUTMUX\_CtInp7ToTimer3CaptureChannels  
enumerator kINPUTMUX\_CtInp8ToTimer3CaptureChannels  
enumerator kINPUTMUX\_CtInp9ToTimer3CaptureChannels  
enumerator kINPUTMUX\_CtInp10ToTimer3CaptureChannels  
enumerator kINPUTMUX\_CtInp11ToTimer3CaptureChannels  
enumerator kINPUTMUX\_CtInp12ToTimer3CaptureChannels  
enumerator kINPUTMUX\_CtInp13ToTimer3CaptureChannels  
enumerator kINPUTMUX\_CtInp14ToTimer3CaptureChannels  
enumerator kINPUTMUX\_CtInp15ToTimer3CaptureChannels  
enumerator kINPUTMUX\_SharedI2s0WsToTimer3CaptureChannels  
enumerator kINPUTMUX\_SharedI2s1WsToTimer3CaptureChannels

enumerator kINPUTMUX\_Usb1FrameToggleToTimer3CaptureChannels  
CTmier4 capture input mux.

enumerator kINPUTMUX\_CtInp0ToTimer4CaptureChannels

enumerator kINPUTMUX\_CtInp1ToTimer4CaptureChannels

enumerator kINPUTMUX\_CtInp2ToTimer4CaptureChannels

enumerator kINPUTMUX\_CtInp3ToTimer4CaptureChannels

enumerator kINPUTMUX\_CtInp4ToTimer4CaptureChannels

enumerator kINPUTMUX\_CtInp5ToTimer4CaptureChannels

enumerator kINPUTMUX\_CtInp6ToTimer4CaptureChannels

enumerator kINPUTMUX\_CtInp7ToTimer4CaptureChannels

enumerator kINPUTMUX\_CtInp8ToTimer4CaptureChannels

enumerator kINPUTMUX\_CtInp9ToTimer4CaptureChannels

enumerator kINPUTMUX\_CtInp10ToTimer4CaptureChannels

enumerator kINPUTMUX\_CtInp11ToTimer4CaptureChannels

enumerator kINPUTMUX\_CtInp12ToTimer4CaptureChannels

enumerator kINPUTMUX\_CtInp13ToTimer4CaptureChannels

enumerator kINPUTMUX\_CtInp14ToTimer4CaptureChannels

enumerator kINPUTMUX\_CtInp15ToTimer4CaptureChannels

enumerator kINPUTMUX\_SharedI2s0WsToTimer4CaptureChannels

enumerator kINPUTMUX\_SharedI2s1WsToTimer4CaptureChannels

enumerator kINPUTMUX\_Usb1FrameToggleToTimer4CaptureChannels  
DMA0 ITRIG.

enumerator kINPUTMUX\_NsGpioPint0Int0ToDma0

enumerator kINPUTMUX\_NsGpioPint0Int1ToDma0

enumerator kINPUTMUX\_NsGpioPint0Int2ToDma0

enumerator kINPUTMUX\_NsGpioPint0Int3ToDma0

enumerator kINPUTMUX\_Ctimer0M0ToDma0

enumerator kINPUTMUX\_Ctimer0M1ToDma0

enumerator kINPUTMUX\_Ctimer1M0ToDma0

enumerator kINPUTMUX\_Ctimer1M1ToDma0

enumerator kINPUTMUX\_Ctimer2M0ToDma0

enumerator kINPUTMUX\_Ctimer2M1ToDma0

enumerator kINPUTMUX\_Ctimer3M0ToDma0

enumerator kINPUTMUX\_Ctimer3M1ToDma0

enumerator kINPUTMUX\_\_Ctimer4M0ToDma0  
enumerator kINPUTMUX\_\_Ctimer4M1ToDma0  
enumerator kINPUTMUX\_\_Dma0TrigOutAToDma0  
enumerator kINPUTMUX\_\_Dma0TrigOutBToDma0  
enumerator kINPUTMUX\_\_Dma0TrigOutCToDma0  
enumerator kINPUTMUX\_\_Dma0TrigOutDToDma0  
enumerator kINPUTMUX\_\_Sct0Dmac0ToDma0  
enumerator kINPUTMUX\_\_Sct0Dmac1ToDma0  
enumerator kINPUTMUX\_\_HashCryptOutToDma0  
enumerator kINPUTMUX\_\_AcmpToDma0  
enumerator kINPUTMUX\_\_AdcToDma0  
enumerator kINPUTMUX\_\_FlexspiRxToDma0  
enumerator kINPUTMUX\_\_FlexspiTxToDma0  
DMA1 ITRIG.  
enumerator kINPUTMUX\_\_NsGpioPint0Int0ToDma1  
enumerator kINPUTMUX\_\_NsGpioPint0Int1ToDma1  
enumerator kINPUTMUX\_\_NsGpioPint0Int2ToDma1  
enumerator kINPUTMUX\_\_NsGpioPint0Int3ToDma1  
enumerator kINPUTMUX\_\_Ctimer0M0ToDma1  
enumerator kINPUTMUX\_\_Ctimer0M1ToDma1  
enumerator kINPUTMUX\_\_Ctimer1M0ToDma1  
enumerator kINPUTMUX\_\_Ctimer1M1ToDma1  
enumerator kINPUTMUX\_\_Ctimer2M0ToDma1  
enumerator kINPUTMUX\_\_Ctimer2M1ToDma1  
enumerator kINPUTMUX\_\_Ctimer3M0ToDma1  
enumerator kINPUTMUX\_\_Ctimer3M1ToDma1  
enumerator kINPUTMUX\_\_Ctimer4M0ToDma1  
enumerator kINPUTMUX\_\_Ctimer4M1ToDma1  
enumerator kINPUTMUX\_\_Dma1TrigOutAToDma1  
enumerator kINPUTMUX\_\_Dma1TrigOutBToDma1  
enumerator kINPUTMUX\_\_Dma1TrigOutCToDma1  
enumerator kINPUTMUX\_\_Dma1TrigOutDToDma1  
enumerator kINPUTMUX\_\_Sct0Dmac0ToDma1



enumerator kINPUTMUX\_\_Sct0Dmac1ToDma1

enumerator kINPUTMUX\_\_HashCryptOutToDma1

enumerator kINPUTMUX\_\_AcmpToDma1

enumerator kINPUTMUX\_\_AdcToDma1

enumerator kINPUTMUX\_\_FlexspiRxToDma1

enumerator kINPUTMUX\_\_FlexspiTxToDma1

DMA0 OTRIG.

enumerator kINPUTMUX\_\_Dma0OtrigChannel0ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel1ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel2ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel3ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel4ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel5ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel6ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel7ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel8ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel9ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel10ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel11ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel12ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel13ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel14ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel15ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel16ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel17ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel18ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel19ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel20ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel21ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel22ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel23ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel24ToTriginChannels

enumerator kINPUTMUX\_\_Dma0OtrigChannel25ToTriginChannels

enumerator kINPUTMUX\_Dma0OtrigChannel26ToTriginChannels  
enumerator kINPUTMUX\_Dma0OtrigChannel27ToTriginChannels  
enumerator kINPUTMUX\_Dma0OtrigChannel28ToTriginChannels  
enumerator kINPUTMUX\_Dma0OtrigChannel29ToTriginChannels  
enumerator kINPUTMUX\_Dma0OtrigChannel30ToTriginChannels  
enumerator kINPUTMUX\_Dma0OtrigChannel31ToTriginChannels  
enumerator kINPUTMUX\_Dma0OtrigChannel32ToTriginChannels  
DMA1 OTRIG.

enumerator kINPUTMUX\_Dma1OtrigChannel0ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel1ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel2ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel3ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel4ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel5ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel6ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel7ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel8ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel9ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel10ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel11ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel12ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel13ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel14ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel15ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel16ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel17ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel18ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel19ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel20ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel21ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel22ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel23ToTriginChannels  
enumerator kINPUTMUX\_Dma1OtrigChannel24ToTriginChannels

enumerator kINPUTMUX\_\_Dma1OtrigChannel25ToTriginChannels  
enumerator kINPUTMUX\_\_Dma1OtrigChannel26ToTriginChannels  
enumerator kINPUTMUX\_\_Dma1OtrigChannel27ToTriginChannels  
enumerator kINPUTMUX\_\_Dma1OtrigChannel28ToTriginChannels  
enumerator kINPUTMUX\_\_Dma1OtrigChannel29ToTriginChannels  
enumerator kINPUTMUX\_\_Dma1OtrigChannel30ToTriginChannels  
enumerator kINPUTMUX\_\_Dma1OtrigChannel31ToTriginChannels  
enumerator kINPUTMUX\_\_Dma1OtrigChannel32ToTriginChannels

enum \_\_inputmux\_signal\_t

INPUTMUX signal enable/disable type.

*Values:*

enumerator kINPUTMUX\_\_Dmac0InputTriggerPint0Ena  
DMA0 input trigger source enable.  
enumerator kINPUTMUX\_\_Dmac0InputTriggerPint1Ena  
enumerator kINPUTMUX\_\_Dmac0InputTriggerPint2Ena  
enumerator kINPUTMUX\_\_Dmac0InputTriggerPint3Ena  
enumerator kINPUTMUX\_\_Dmac0InputTriggerCtimer0M0Ena  
enumerator kINPUTMUX\_\_Dmac0InputTriggerCtimer0M1Ena  
enumerator kINPUTMUX\_\_Dmac0InputTriggerCtimer1M0Ena  
enumerator kINPUTMUX\_\_Dmac0InputTriggerCtimer1M1Ena  
enumerator kINPUTMUX\_\_Dmac0InputTriggerCtimer2M0Ena  
enumerator kINPUTMUX\_\_Dmac0InputTriggerCtimer2M1Ena  
enumerator kINPUTMUX\_\_Dmac0InputTriggerCtimer3M0Ena  
enumerator kINPUTMUX\_\_Dmac0InputTriggerCtimer3M1Ena  
enumerator kINPUTMUX\_\_Dmac0InputTriggerCtimer4M0Ena  
enumerator kINPUTMUX\_\_Dmac0InputTriggerCtimer4M1Ena  
enumerator kINPUTMUX\_\_Dmac0InputTriggerDma0OutAEna  
enumerator kINPUTMUX\_\_Dmac0InputTriggerDma0OutBEna  
enumerator kINPUTMUX\_\_Dmac0InputTriggerDma0OutCEna  
enumerator kINPUTMUX\_\_Dmac0InputTriggerDma0OutDEna  
enumerator kINPUTMUX\_\_Dmac0InputTriggerSctDmac0Ena  
enumerator kINPUTMUX\_\_Dmac0InputTriggerSctDmac1Ena  
enumerator kINPUTMUX\_\_Dmac0InputTriggerHashOutEna  
enumerator kINPUTMUX\_\_Dmac0InputTriggerAcmpEna

enumerator kINPUTMUX\_\_Dmac0InputTriggerAdcEna  
enumerator kINPUTMUX\_\_Dmac0InputTriggerFlexspiRxEna  
enumerator kINPUTMUX\_\_Dmac0InputTriggerFlexspiTxEna  
DMA1 input trigger source enable.  
enumerator kINPUTMUX\_\_Dmac1InputTriggerPint0Ena  
enumerator kINPUTMUX\_\_Dmac1InputTriggerPint1Ena  
enumerator kINPUTMUX\_\_Dmac1InputTriggerPint2Ena  
enumerator kINPUTMUX\_\_Dmac1InputTriggerPint3Ena  
enumerator kINPUTMUX\_\_Dmac1InputTriggerCtimer0M0Ena  
enumerator kINPUTMUX\_\_Dmac1InputTriggerCtimer0M1Ena  
enumerator kINPUTMUX\_\_Dmac1InputTriggerCtimer1M0Ena  
enumerator kINPUTMUX\_\_Dmac1InputTriggerCtimer1M1Ena  
enumerator kINPUTMUX\_\_Dmac1InputTriggerCtimer2M0Ena  
enumerator kINPUTMUX\_\_Dmac1InputTriggerCtimer2M1Ena  
enumerator kINPUTMUX\_\_Dmac1InputTriggerCtimer3M0Ena  
enumerator kINPUTMUX\_\_Dmac1InputTriggerCtimer3M1Ena  
enumerator kINPUTMUX\_\_Dmac1InputTriggerCtimer4M0Ena  
enumerator kINPUTMUX\_\_Dmac1InputTriggerCtimer4M1Ena  
enumerator kINPUTMUX\_\_Dmac1InputTriggerDma1OutAEna  
enumerator kINPUTMUX\_\_Dmac1InputTriggerDma1OutBEna  
enumerator kINPUTMUX\_\_Dmac1InputTriggerDma1OutCEna  
enumerator kINPUTMUX\_\_Dmac1InputTriggerDma1OutDEna  
enumerator kINPUTMUX\_\_Dmac1InputTriggerSctDmac0Ena  
enumerator kINPUTMUX\_\_Dmac1InputTriggerSctDmac1Ena  
enumerator kINPUTMUX\_\_Dmac1InputTriggerHashOutEna  
enumerator kINPUTMUX\_\_Dmac1InputTriggerAcmpEna  
enumerator kINPUTMUX\_\_Dmac1InputTriggerAdcEna  
enumerator kINPUTMUX\_\_Dmac1InputTriggerFlexspiRxEna  
enumerator kINPUTMUX\_\_Dmac1InputTriggerFlexspiTxEna  
DMA0 REQ signal.  
enumerator kINPUTMUX\_\_Flexcomm0RxToDmac0Ch0RequestEna  
enumerator kINPUTMUX\_\_Flexcomm0TxToDmac0Ch1RequestEna  
enumerator kINPUTMUX\_\_Flexcomm1RxToDmac0Ch2RequestEna  
enumerator kINPUTMUX\_\_Flexcomm1TxToDmac0Ch3RequestEna

enumerator kINPUTMUX\_Flexcomm2RxToDmac0Ch4RequestEna  
enumerator kINPUTMUX\_Flexcomm2TxToDmac0Ch5RequestEna  
enumerator kINPUTMUX\_Flexcomm3RxToDmac0Ch6RequestEna  
enumerator kINPUTMUX\_Flexcomm3TxToDmac0Ch7RequestEna  
enumerator kINPUTMUX\_Flexcomm4RxToDmac0Ch8RequestEna  
enumerator kINPUTMUX\_Flexcomm4TxToDmac0Ch9RequestEna  
enumerator kINPUTMUX\_Flexcomm5RxToDmac0Ch10RequestEna  
enumerator kINPUTMUX\_Flexcomm5TxToDmac0Ch11RequestEna  
enumerator kINPUTMUX\_Flexcomm6RxToDmac0Ch12RequestEna  
enumerator kINPUTMUX\_Flexcomm6TxToDmac0Ch13RequestEna  
enumerator kINPUTMUX\_Flexcomm7RxToDmac0Ch14RequestEna  
enumerator kINPUTMUX\_Flexcomm7TxToDmac0Ch15RequestEna  
enumerator kINPUTMUX\_Dmic0Ch0ToDmac0Ch16RequestEna  
enumerator kINPUTMUX\_Dmic0Ch1ToDmac0Ch17RequestEna  
enumerator kINPUTMUX\_Dmic0Ch2ToDmac0Ch18RequestEna  
enumerator kINPUTMUX\_Dmic0Ch3ToDmac0Ch19RequestEna  
enumerator kINPUTMUX\_Dmic0Ch4ToDmac0Ch20RequestEna  
enumerator kINPUTMUX\_Dmic0Ch5ToDmac0Ch21RequestEna  
enumerator kINPUTMUX\_Dmic0Ch6ToDmac0Ch22RequestEna  
enumerator kINPUTMUX\_Dmic0Ch7ToDmac0Ch23RequestEna  
enumerator kINPUTMUX\_I3c0RxToDmac0Ch24RequestEna  
enumerator kINPUTMUX\_I3c0TxToDmac0Ch25RequestEna  
enumerator kINPUTMUX\_Flexcomm14RxToDmac0Ch26RequestEna  
enumerator kINPUTMUX\_Flexcomm14TxToDmac0Ch27RequestEna  
enumerator kINPUTMUX\_HashCryptToDmac0Ch30RequestEna  
DMA1 REQ signal.  
enumerator kINPUTMUX\_Flexcomm0RxToDmac1Ch0RequestEna  
enumerator kINPUTMUX\_Flexcomm0TxToDmac1Ch1RequestEna  
enumerator kINPUTMUX\_Flexcomm1RxToDmac1Ch2RequestEna  
enumerator kINPUTMUX\_Flexcomm1TxToDmac1Ch3RequestEna  
enumerator kINPUTMUX\_Flexcomm2RxToDmac1Ch4RequestEna  
enumerator kINPUTMUX\_Flexcomm2TxToDmac1Ch5RequestEna  
enumerator kINPUTMUX\_Flexcomm3RxToDmac1Ch6RequestEna

enumerator kINPUTMUX\_Flexcomm3TxToDmac1Ch7RequestEna  
enumerator kINPUTMUX\_Flexcomm4RxToDmac1Ch8RequestEna  
enumerator kINPUTMUX\_Flexcomm4TxToDmac1Ch9RequestEna  
enumerator kINPUTMUX\_Flexcomm5RxToDmac1Ch10RequestEna  
enumerator kINPUTMUX\_Flexcomm5TxToDmac1Ch11RequestEna  
enumerator kINPUTMUX\_Flexcomm6RxToDmac1Ch12RequestEna  
enumerator kINPUTMUX\_Flexcomm6TxToDmac1Ch13RequestEna  
enumerator kINPUTMUX\_Flexcomm7RxToDmac1Ch14RequestEna  
enumerator kINPUTMUX\_Flexcomm7TxToDmac1Ch15RequestEna  
enumerator kINPUTMUX\_Dmic0Ch0ToDmac1Ch16RequestEna  
enumerator kINPUTMUX\_Dmic0Ch1ToDmac1Ch17RequestEna  
enumerator kINPUTMUX\_Dmic0Ch2ToDmac1Ch18RequestEna  
enumerator kINPUTMUX\_Dmic0Ch3ToDmac1Ch19RequestEna  
enumerator kINPUTMUX\_Dmic0Ch4ToDmac1Ch20RequestEna  
enumerator kINPUTMUX\_Dmic0Ch5ToDmac1Ch21RequestEna  
enumerator kINPUTMUX\_Dmic0Ch6ToDmac1Ch22RequestEna  
enumerator kINPUTMUX\_Dmic0Ch7ToDmac1Ch23RequestEna  
enumerator kINPUTMUX\_I3c0RxToDmac1Ch24RequestEna  
enumerator kINPUTMUX\_I3c0TxToDmac1Ch25RequestEna  
enumerator kINPUTMUX\_Flexcomm14RxToDmac1Ch26RequestEna  
enumerator kINPUTMUX\_Flexcomm14TxToDmac1Ch27RequestEna  
enumerator kINPUTMUX\_HashCryptToDmac1Ch30RequestEna

typedef enum *\_inputmux\_connection\_t* inputmux\_connection\_t  
INPUTMUX connections type.

typedef enum *\_inputmux\_signal\_t* inputmux\_signal\_t  
INPUTMUX signal enable/disable type.

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, uint32\_t index, *inputmux\_connection\_t* connection)**  
 Attaches a signal.

This function attaches multiplexed signals from INPUTMUX to target signals. For example, to attach GPIO PORT0 Pin 5 to PINT peripheral, do the following:

```
INPUTMUX_AttachSignal(INPUTMUX, 2, kINPUTMUX_GpioPort0Pin5ToPintsel);
```

In this example, INTMUX has 8 registers for PINT, PINT\_SEL0~PINT\_SEL7. With parameter index specified as 2, this function configures register PINT\_SEL2.

#### Parameters

- base – Base address of the INPUTMUX peripheral.
- index – The serial number of destination register in the group of INPUTMUX registers with same name.
- connection – Applies signal from source signals collection to target signal.

#### Return values

None. –

**void INPUTMUX\_EnableSignal(void \*base, *inputmux\_signal\_t* signal, bool enable)**  
 Enable/disable a signal.

This function gates the INPUTMUX clock.

#### Parameters

- base – Base address of the INPUTMUX peripheral.
- signal – Enable signal register id and bit offset.
- enable – Selects enable or disable.

#### 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. –

SCT0\_PMUX\_ID

Periphinmux IDs.

SHSGPIO\_PMUX\_ID

PINTSEL\_PMUX\_ID

DSP\_INT\_PMUX\_ID

DMA0\_ITRIG\_PMUX\_ID

DMA0\_OTRIG\_PMUX\_ID

DMA1\_ITRIG\_PMUX\_ID

DMA1\_OTRIG\_PMUX\_ID

CT32BIT0\_CAP\_PMUX\_ID  
CT32BIT1\_CAP\_PMUX\_ID  
CT32BIT2\_CAP\_PMUX\_ID  
CT32BIT3\_CAP\_PMUX\_ID  
CT32BIT4\_CAP\_PMUX\_ID  
FREQMEAS\_PMUX\_ID  
DMA0\_REQ\_ENA0\_ID  
DMA1\_REQ\_ENA0\_ID  
DMA0\_ITRIG\_EN0\_ID  
DMA1\_ITRIG\_EN0\_ID  
ENA\_SHIFT  
PMUX\_SHIFT

## 2.44 IOPCTL: Input/Output Pad Controller

LPC\_IOPCTL\_DRIVER\_VERSION

IOPCTL driver version 2.0.0.

typedef struct *iopctl\_group* iopctl\_group\_t

Array of IOPCTL pin definitions passed to IOPCTL\_SetPinMuxing() must be in this format.

\_\_STATIC\_INLINE void IOPCTL\_PinMuxSet (IOPCTL\_Type \*base, uint8\_t port, uint8\_t pin, uint32\_t modefunc)

Sets I/O Pad Control pin mux.

### Parameters

- base – : The base of IOPCTL peripheral on the chip
- port – : Port to mux
- pin – : Pin to mux
- modefunc – : OR'ed values of type IOPCTL\_\*

### Returns

Nothing

\_\_STATIC\_INLINE void IOPCTL\_SetPinMuxing (IOPCTL\_Type \*base, const iopctl\_group\_t \*pinArray, uint32\_t arrayLength)

Set all I/O Control pin muxing.

### Parameters

- base – : The base of IOPCTL peripheral on the chip
- pinArray – : Pointer to array of pin mux selections
- arrayLength – : Number of entries in pinArray

### Returns

Nothing



FSL\_COMPONENT\_ID

IOPCTL\_FUNC0

IOPCTL function and mode selection definitions.

---

**Note:** See the User Manual for specific modes and functions supported by the various pins.  
Selects pin function 0

---

IOPCTL\_FUNC1

Selects pin function 1

IOPCTL\_FUNC2

Selects pin function 2

IOPCTL\_FUNC3

Selects pin function 3

IOPCTL\_FUNC4

Selects pin function 4

IOPCTL\_FUNC5

Selects pin function 5

IOPCTL\_FUNC6

Selects pin function 6

IOPCTL\_FUNC7

Selects pin function 7

IOPCTL\_FUNC8

Selects pin function 8

IOPCTL\_FUNC9

Selects pin function 9

IOPCTL\_FUNC10

Selects pin function 10

IOPCTL\_FUNC11

Selects pin function 11

IOPCTL\_FUNC12

Selects pin function 12

IOPCTL\_FUNC13

Selects pin function 13

IOPCTL\_FUNC14

Selects pin function 14

IOPCTL\_FUNC15

Selects pin function 15

IOPCTL\_PUPD\_EN

Enables Pullup / Pulldown

IOPCTL\_PULLDOWN\_EN

Selects pull-down function

IOPCTL\_PULLUP\_EN

Selects pull-up function

IOPCTL\_INBUF\_EN

Enables buffer function on input

IOPCTL\_SLEW\_RATE

Slew Rate Control

IOPCTL\_FULLDRIVE\_EN

Selects full drive

IOPCTL\_ANAMUX\_EN

Enables analog mux function by setting 0 to bit 7

IOPCTL\_PSEDRAIN\_EN

Enables pseudo output drain function

IOPCTL\_INV\_EN

Enables invert function on input

struct `_iopctl_group`

*#include <fsl\_iopctl.h>* Array of IOPCTL pin definitions passed to IOPCTL\_SetPinMuxing() must be in this format.

## 2.45 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 specifiled by *bits* to the variable at address *address*.

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

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

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

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

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

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

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

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

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

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

USEC\_TO\_COUNT(*us*, *clockFreqInHz*)

Macro to convert a microsecond period to raw count value

COUNT\_TO\_USEC(*count*, *clockFreqInHz*)

Macro to convert a raw count value to microsecond

MSEC\_TO\_COUNT(*ms*, *clockFreqInHz*)

Macro to convert a millisecond period to raw count value

COUNT\_TO\_MSEC(*count*, *clockFreqInHz*)

Macro to convert a raw count value to millisecond

SDK\_ISR\_EXIT\_BARRIER

SDK\_SIZEALIGN(*var*, *alignbytes*)

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

Macro to define a variable with L2 cache line size alignment

Macro to change a value to a given size aligned value

AT\_NONCACHEABLE\_SECTION(*var*)

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

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

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

AT\_NONCACHEABLE\_SECTION\_INIT(*var*)

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

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

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

enum \_\_status\_groups

Status group numbers.

*Values:*

enumerator kStatusGroup\_Generic

Group number for generic status codes.

enumerator kStatusGroup\_FLASH

Group number for FLASH status codes.

enumerator kStatusGroup\_LPSPI

Group number for LPSPI status codes.

enumerator kStatusGroup\_FLEXIO\_SPI

Group number for FLEXIO SPI status codes.

enumerator kStatusGroup\_DSPI

Group number for DSPI status codes.

enumerator kStatusGroup\_FLEXIO\_UART

Group number for FLEXIO UART status codes.

enumerator kStatusGroup\_FLEXIO\_I2C

Group number for FLEXIO I2C status codes.

enumerator kStatusGroup\_LPI2C

Group number for LPI2C status codes.

enumerator kStatusGroup\_UART

Group number for UART status codes.

enumerator kStatusGroup\_I2C

Group number for 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\_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().

static inline bool \_SDK\_AtomicLocalCompareAndSet(uint32\_t \*addr, uint32\_t expected, uint32\_t newValue)

static inline uint32\_t \_SDK\_AtomicTestAndSet(uint32\_t \*addr, uint32\_t newValue)

FSL\_DRIVER\_TRANSFER\_DOUBLE\_WEAK\_IRQ

Macro to use the default weak IRQ handler in drivers.

MAKE\_STATUS(group, code)

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

MAKE\_VERSION(major, minor, bugfix)

Construct the version number for drivers.

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

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

ARRAY\_SIZE(x)

Computes the number of elements in an array.

UINT64\_H(X)

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

UINT64\_L(X)

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

SUPPRESS\_FALL\_THROUGH\_WARNING()

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

MSDK\_REG\_SECURE\_ADDR(x)

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

MSDK\_REG\_NONSECURE\_ADDR(x)

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

MSDK\_INVALID\_IRQ\_HANDLER

Invalid IRQ handler address.

## 2.46 LPADC: 12-bit SAR Analog-to-Digital Converter Driver

enum \_lpadc\_status\_flags

Define hardware flags of the module.

*Values:*

enumerator kLPADC\_ResultFIFO0OverflowFlag

Indicates that more data has been written to the Result FIFO 0 than it can hold.

enumerator kLPADC\_ResultFIFO0ReadyFlag

Indicates when the number of valid datawords in the result FIFO 0 is greater than the setting watermark level.

enumerator kLPADC\_TriggerExceptionFlag

Indicates that a trigger exception event has occurred.

enumerator kLPADC\_TriggerCompletionFlag

Indicates that a trigger completion event has occurred.

enumerator kLPADC\_CalibrationReadyFlag

Indicates that the calibration process is done.

enumerator kLPADC\_ActiveFlag

Indicates that the ADC is in active state.

enumerator kLPADC\_ResultFIFOOverflowFlag

To compilitable with old version, do not recommend using this, please use kLPADC\_ResultFIFO0OverflowFlag as instead.

enumerator kLPADC\_ResultFIFOReadyFlag

To compilitable with old version, do not recommend using this, please use kLPADC\_ResultFIFO0ReadyFlag as instead.

enum \_lpadc\_interrupt\_enable

Define interrupt switchers of the module.

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

*Values:*

enumerator kLPADC\_ResultFIFO0OverflowInterruptEnable

Configures ADC to generate overflow interrupt requests when FOF0 flag is asserted.

enumerator kLPADC\_FIFO0WatermarkInterruptEnable

Configures ADC to generate watermark interrupt requests when RDY0 flag is asserted.

enumerator kLPADC\_ResultFIFOOverflowInterruptEnable

To compilitable with old version, do not recommend using this, please use kLPADC\_ResultFIFO0OverflowInterruptEnable as instead.

enumerator kLPADC\_FIFOWatermarkInterruptEnable

To compilitable with old version, do not recommend using this, please use kLPADC\_FIFO0WatermarkInterruptEnable as instead.

enumerator kLPADC\_TriggerExceptionInterruptEnable

Configures ADC to generate trigger exception interrupt.

enumerator kLPADC\_Trigger0CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 0 completion.

enumerator kLPADC\_Trigger1CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 1 completion.

enumerator kLPADC\_Trigger2CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 2 completion.

enumerator kLPADC\_Trigger3CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 3 completion.

enumerator kLPADC\_Trigger4CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 4 completion.

enumerator kLPADC\_Trigger5CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 5 completion.

enumerator kLPADC\_Trigger6CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 6 completion.

enumerator kLPADC\_Trigger7CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 7 completion.

enumerator kLPADC\_Trigger8CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 8 completion.

enumerator kLPADC\_Trigger9CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 9 completion.

enumerator kLPADC\_Trigger10CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 10 completion.

enumerator kLPADC\_Trigger11CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 11 completion.

enumerator kLPADC\_Trigger12CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 12 completion.

enumerator kLPADC\_Trigger13CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 13 completion.

enumerator kLPADC\_Trigger14CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 14 completion.

enumerator kLPADC\_Trigger15CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 15 completion.

enum \_lpadc\_trigger\_status\_flags

The enumerator of lpadc trigger status flags, including interrupted flags and completed flags.

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

*Values:*

enumerator kLPADC\_Trigger0InterruptedFlag

Trigger 0 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger1InterruptedFlag

Trigger 1 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger2InterruptedFlag

Trigger 2 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger3InterruptedFlag

Trigger 3 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger4InterruptedFlag

Trigger 4 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger5InterruptedFlag

Trigger 5 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger6InterruptedFlag

Trigger 6 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger7InterruptedFlag

Trigger 7 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger8InterruptedFlag

Trigger 8 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger9InterruptedFlag

Trigger 9 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger10InterruptedFlag

Trigger 10 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger11InterruptedFlag

Trigger 11 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger12InterruptedFlag

Trigger 12 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger13InterruptedFlag

Trigger 13 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger14InterruptedFlag

Trigger 14 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger15InterruptedFlag

Trigger 15 is interrupted by a high priority exception.

enumerator kLPADC\_Trigger0CompletedFlag

Trigger 0 is completed and trigger 0 has enabled completion interrupts.

enumerator kLPADC\_Trigger1CompletedFlag

Trigger 1 is completed and trigger 1 has enabled completion interrupts.

enumerator kLPADC\_Trigger2CompletedFlag

Trigger 2 is completed and trigger 2 has enabled completion interrupts.

enumerator kLPADC\_Trigger3CompletedFlag

Trigger 3 is completed and trigger 3 has enabled completion interrupts.

enumerator kLPADC\_Trigger4CompletedFlag

Trigger 4 is completed and trigger 4 has enabled completion interrupts.



enumerator kLPADC\_Trigger5CompletedFlag

Trigger 5 is completed and trigger 5 has enabled completion interrupts.

enumerator kLPADC\_Trigger6CompletedFlag

Trigger 6 is completed and trigger 6 has enabled completion interrupts.

enumerator kLPADC\_Trigger7CompletedFlag

Trigger 7 is completed and trigger 7 has enabled completion interrupts.

enumerator kLPADC\_Trigger8CompletedFlag

Trigger 8 is completed and trigger 8 has enabled completion interrupts.

enumerator kLPADC\_Trigger9CompletedFlag

Trigger 9 is completed and trigger 9 has enabled completion interrupts.

enumerator kLPADC\_Trigger10CompletedFlag

Trigger 10 is completed and trigger 10 has enabled completion interrupts.

enumerator kLPADC\_Trigger11CompletedFlag

Trigger 11 is completed and trigger 11 has enabled completion interrupts.

enumerator kLPADC\_Trigger12CompletedFlag

Trigger 12 is completed and trigger 12 has enabled completion interrupts.

enumerator kLPADC\_Trigger13CompletedFlag

Trigger 13 is completed and trigger 13 has enabled completion interrupts.

enumerator kLPADC\_Trigger14CompletedFlag

Trigger 14 is completed and trigger 14 has enabled completion interrupts.

enumerator kLPADC\_Trigger15CompletedFlag

Trigger 15 is completed and trigger 15 has enabled completion interrupts.

enum \_lpadc\_sample\_scale\_mode

Define enumeration of sample scale mode.

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

*Values:*

enumerator kLPADC\_SamplePartScale

Use divided input voltage signal. (For scale select, please refer to the reference manual).

enumerator kLPADC\_SampleFullScale

Full scale (Factor of 1).

enum \_lpadc\_sample\_channel\_mode

Define enumeration of channel sample mode.

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

*Values:*

enumerator kLPADC\_SampleChannelSingleEndSideA

Single-end mode, only A-side channel is converted.

enumerator kLPADC\_SampleChannelSingleEndSideB

Single-end mode, only B-side channel is converted.

enumerator kLPADC\_SampleChannelDiffBothSideAB  
Differential mode, the ADC result is (CHnA-CHnB).

enumerator kLPADC\_SampleChannelDiffBothSideBA  
Differential mode, the ADC result is (CHnB-CHnA).

enumerator kLPADC\_SampleChannelDiffBothSide  
Differential mode, the ADC result is (CHnA-CHnB).

enumerator kLPADC\_SampleChannelDualSingleEndBothSide  
Dual-Single-Ended Mode. Both A side and B side channels are converted independently.

enum \_lpadc\_hardware\_average\_mode  
Define enumeration of hardware average selection.

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

---

**Note:** Some enumerator values are not available on some devices, mainly depends on the size of AVGS field in CMDH register.

---

*Values:*

enumerator kLPADC\_HardwareAverageCount1  
Single conversion.

enumerator kLPADC\_HardwareAverageCount2  
2 conversions averaged.

enumerator kLPADC\_HardwareAverageCount4  
4 conversions averaged.

enumerator kLPADC\_HardwareAverageCount8  
8 conversions averaged.

enumerator kLPADC\_HardwareAverageCount16  
16 conversions averaged.

enumerator kLPADC\_HardwareAverageCount32  
32 conversions averaged.

enumerator kLPADC\_HardwareAverageCount64  
64 conversions averaged.

enumerator kLPADC\_HardwareAverageCount128  
128 conversions averaged.

enum \_lpadc\_sample\_time\_mode  
Define enumeration of sample time selection.

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

*Values:*

enumerator kLPADC\_SampleTimeADCK3  
3 ADCK cycles total sample time.

enumerator kLPADC\_SampleTimeADCK5  
5 ADCK cycles total sample time.

enumerator kLPADC\_SampleTimeADCK7  
7 ADCK cycles total sample time.

enumerator kLPADC\_SampleTimeADCK11  
11 ADCK cycles total sample time.

enumerator kLPADC\_SampleTimeADCK19  
19 ADCK cycles total sample time.

enumerator kLPADC\_SampleTimeADCK35  
35 ADCK cycles total sample time.

enumerator kLPADC\_SampleTimeADCK67  
69 ADCK cycles total sample time.

enumerator kLPADC\_SampleTimeADCK131  
131 ADCK cycles total sample time.

enum \_lpadc\_hardware\_compare\_mode

Define enumeration of hardware compare mode.

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

*Values:*

enumerator kLPADC\_HardwareCompareDisabled  
Compare disabled.

enumerator kLPADC\_HardwareCompareStoreOnTrue  
Compare enabled. Store on true.

enumerator kLPADC\_HardwareCompareRepeatUntilTrue  
Compare enabled. Repeat channel acquisition until true.

enum \_lpadc\_conversion\_resolution\_mode

Define enumeration of conversion resolution mode.

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to lpadc\_sample\_channel\_mode\_t

*Values:*

enumerator kLPADC\_ConversionResolutionStandard  
Standard resolution. Single-ended 12-bit conversion, Differential 13-bit conversion with 2's complement output.

enumerator kLPADC\_ConversionResolutionHigh  
High resolution. Single-ended 16-bit conversion; Differential 16-bit conversion with 2's complement output.

enum \_lpadc\_conversion\_average\_mode

Define enumeration of conversion averages mode.

Configure the conversion average number for auto-calibration.

---

**Note:** Some enumerator values are not available on some devices, mainly depends on the size of CAL\_AVGS field in CTRL register.

---

*Values:*

enumerator kLPADC\_ConversionAverage1  
Single conversion.

enumerator kLPADC\_ConversionAverage2  
2 conversions averaged.

enumerator kLPADC\_ConversionAverage4  
4 conversions averaged.

enumerator kLPADC\_ConversionAverage8  
8 conversions averaged.

enumerator kLPADC\_ConversionAverage16  
16 conversions averaged.

enumerator kLPADC\_ConversionAverage32  
32 conversions averaged.

enumerator kLPADC\_ConversionAverage64  
64 conversions averaged.

enumerator kLPADC\_ConversionAverage128  
128 conversions averaged.

enum \_lpadc\_reference\_voltage\_mode

Define enumeration of reference voltage source.

For detail information, need to check the SoC's specification.

*Values:*

enumerator kLPADC\_ReferenceVoltageAlt1  
Option 1 setting.

enumerator kLPADC\_ReferenceVoltageAlt2  
Option 2 setting.

enumerator kLPADC\_ReferenceVoltageAlt3  
Option 3 setting.

enum \_lpadc\_power\_level\_mode

Define enumeration of power configuration.

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

*Values:*

enumerator kLPADC\_PowerLevelAlt1  
Lowest power setting.

enumerator kLPADC\_PowerLevelAlt2  
Next lowest power setting.

enumerator kLPADC\_PowerLevelAlt3  
...

enumerator kLPADC\_PowerLevelAlt4  
Highest power setting.

enum `_lpadc_offset_calibration_mode`

Define enumeration of offset calibration mode.

*Values:*

enumerator `kLPADC_OffsetCalibration12bitMode`

12 bit offset calibration mode.

enumerator `kLPADC_OffsetCalibration16bitMode`

16 bit offset calibration mode.

enum `_lpadc_trigger_priority_policy`

Define enumeration of trigger priority policy.

This selection controls how higher priority triggers are handled.

---

**Note:** `kLPADC_TriggerPriorityPreemptSubsequently` is not available on some devices, mainly depends on the size of `TPRCTRL` field in `CFG` register.

---

*Values:*

enumerator `kLPADC_ConvPreemptImmediatelyNotAutoResumed`

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion is not automatically resumed or restarted.

enumerator `kLPADC_ConvPreemptSoftlyNotAutoResumed`

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion is not resumed or restarted.

enumerator `kLPADC_ConvPreemptImmediatelyAutoRestarted`

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

enumerator `kLPADC_ConvPreemptSoftlyAutoRestarted`

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

enumerator `kLPADC_ConvPreemptImmediatelyAutoResumed`

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be resumed.

enumerator `kLPADC_ConvPreemptSoftlyAutoResumed`

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will be automatically be resumed.

enumerator `kLPADC_TriggerPriorityPreemptImmediately`

Legacy support is not recommended as it only ensures compatibility with older versions.

enumerator `kLPADC_TriggerPriorityPreemptSoftly`

Legacy support is not recommended as it only ensures compatibility with older versions.

enumerator `kLPADC_TriggerPriorityExceptionDisabled`

High priority trigger exception disabled.

typedef enum `_lpadc_sample_scale_mode` `lpadc_sample_scale_mode_t`

Define enumeration of sample scale mode.

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

typedef enum `_lpadc_sample_channel_mode` `lpadc_sample_channel_mode_t`

Define enumeration of channel sample mode.

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

typedef enum `_lpadc_hardware_average_mode` `lpadc_hardware_average_mode_t`

Define enumeration of hardware average selection.

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

---

**Note:** Some enumerator values are not available on some devices, mainly depends on the size of AVGS field in CMDH register.

---

typedef enum `_lpadc_sample_time_mode` `lpadc_sample_time_mode_t`

Define enumeration of sample time selection.

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

typedef enum `_lpadc_hardware_compare_mode` `lpadc_hardware_compare_mode_t`

Define enumeration of hardware compare mode.

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

typedef enum `_lpadc_conversion_resolution_mode` `lpadc_conversion_resolution_mode_t`

Define enumeration of conversion resolution mode.

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to `lpadc_sample_channel_mode_t`

```
typedef enum _lpadc_conversion_average_mode lpadc_conversion_average_mode_t
```

Define enumeration of conversion averages mode.

Configure the conversion average number for auto-calibration.

---

**Note:** Some enumerator values are not available on some devices, mainly depends on the size of CAL\_AVGS field in CTRL register.

---

```
typedef enum _lpadc_reference_voltage_mode lpadc_reference_voltage_source_t
```

Define enumeration of reference voltage source.

For detail information, need to check the SoC's specification.

```
typedef enum _lpadc_power_level_mode lpadc_power_level_mode_t
```

Define enumeration of power configuration.

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

```
typedef enum _lpadc_offset_calibration_mode lpadc_offset_calibration_mode_t
```

Define enumeration of offset calibration mode.

```
typedef enum _lpadc_trigger_priority_policy lpadc_trigger_priority_policy_t
```

Define enumeration of trigger priority policy.

This selection controls how higher priority triggers are handled.

---

**Note:** **kLPADC\_TriggerPriorityPreemptSubsequently** is not available on some devices, mainly depends on the size of TPRICTRL field in CFG register.

---

```
typedef struct _lpadc_calibration_value lpadc_calibration_value_t
```

A structure of calibration value.

LPADC\_CONVERSION\_COMPLETE\_TIMEOUT

Max loops to wait for LPADC conversion complete.

When doing calibration, driver will wait for the completion of conversion. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

LPADC\_CALIBRATION\_READY\_TIMEOUT

Max loops to wait for LPADC calibration ready.

Before doing calibration, driver will wait for the calibration ready. This parameter defines how many loops to check the calibration ready. If defined as 0, driver will wait forever until ready.

LPADC\_GAIN\_CAL\_READY\_TIMEOUT

Max loops to wait for LPADC gain calibration GAIN\_CAL ready.

Before doing calibration, driver will wait for the gain calibration GAIN\_CAL ready. This parameter defines how many loops to check the gain calibration GAIN\_CAL ready. If defined as 0, driver will wait forever until ready.

LPADC\_GET\_ACTIVE\_COMMAND\_STATUS(statusVal)

Define the MACRO function to get command status from status value.

The statusVal is the return value from LPADC\_GetStatusFlags().

LPADC\_GET\_ACTIVE\_TRIGGER\_STATUE(statusVal)

Define the MACRO function to get trigger status from status value.

The statusVal is the return value from LPADC\_GetStatusFlags().

void LPADC\_Init(ADC\_Type \*base, const *lpadc\_config\_t* \*config)

Initializes the LPADC module.

#### Parameters

- base – LPADC peripheral base address.
- config – Pointer to configuration structure. See “*lpadc\_config\_t*”.

void LPADC\_GetDefaultConfig(*lpadc\_config\_t* \*config)

Gets an available pre-defined settings for initial configuration.

This function initializes the converter configuration structure with an available settings. The default values are:

```
config->enableInDozeMode      = true;
config->enableAnalogPreliminary = false;
config->powerUpDelay           = 0x80;
config->referenceVoltageSource = kLPADC_ReferenceVoltageAlt1;
config->powerLevelMode         = kLPADC_PowerLevelAlt1;
config->triggerPriorityPolicy   = kLPADC_TriggerPriorityPreemptImmediately;
config->enableConvPause        = false;
config->convPauseDelay          = 0U;
config->FIFOWatermark           = 0U;
```

#### Parameters

- config – Pointer to configuration structure.

void LPADC\_Deinit(ADC\_Type \*base)

De-initializes the LPADC module.

#### Parameters

- base – LPADC peripheral base address.

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

Switch on/off the LPADC module.

#### Parameters

- base – LPADC peripheral base address.
- enable – switcher to the module.

static inline void LPADC\_DoResetFIFO(ADC\_Type \*base)

Do reset the conversion FIFO.

#### Parameters

- base – LPADC peripheral base address.

static inline void LPADC\_DoResetConfig(ADC\_Type \*base)

Do reset the module's configuration.

Reset all ADC internal logic and registers, except the Control Register (ADCx\_CTRL).

#### Parameters

- base – LPADC peripheral base address.



static inline uint32\_t LPADC\_GetStatusFlags(ADC\_Type \*base)

Get status flags.

**Parameters**

- base – LPADC peripheral base address.

**Returns**

status flags' mask. See to `_lpadc_status_flags`.

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

Clear status flags.

Only the flags can be cleared by writing ADCx\_STATUS register would be cleared by this API.

**Parameters**

- base – LPADC peripheral base address.
- mask – Mask value for flags to be cleared. See to `_lpadc_status_flags`.

static inline uint32\_t LPADC\_GetTriggerStatusFlags(ADC\_Type \*base)

Get trigger status flags to indicate which trigger sequences have been completed or interrupted by a high priority trigger exception.

**Parameters**

- base – LPADC peripheral base address.

**Returns**

The OR'ed value of `_lpadc_trigger_status_flags`.

static inline void LPADC\_ClearTriggerStatusFlags(ADC\_Type \*base, uint32\_t mask)

Clear trigger status flags.

**Parameters**

- base – LPADC peripheral base address.
- mask – The mask of trigger status flags to be cleared, should be the OR'ed value of `_lpadc_trigger_status_flags`.

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

Enable interrupts.

**Parameters**

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to `_lpadc_interrupt_enable`.

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

Disable interrupts.

**Parameters**

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to `_lpadc_interrupt_enable`.

static inline void LPADC\_EnableFIFOWatermarkDMA(ADC\_Type \*base, bool enable)

Switch on/off the DMA trigger for FIFO watermark event.

**Parameters**

- base – LPADC peripheral base address.
- enable – Switcher to the event.

```
static inline uint32_t LPADC_GetConvResultCount(ADC_Type *base)
```

Get the count of result kept in conversion FIFO.

**Parameters**

- base – LPADC peripheral base address.

**Returns**

The count of result kept in conversion FIFO.

```
bool LPADC_GetConvResult(ADC_Type *base, lpadc_conv_result_t *result)
```

Get the result in conversion FIFO.

**Parameters**

- base – LPADC peripheral base address.
- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

**Returns**

Status whether FIFO entry is valid.

```
void LPADC_GetConvResultBlocking(ADC_Type *base, lpadc_conv_result_t *result)
```

Get the result in conversion FIFO using blocking method.

**Parameters**

- base – LPADC peripheral base address.
- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

```
void LPADC_SetConvTriggerConfig(ADC_Type *base, uint32_t triggerId, const  
                                lpadc_conv_trigger_config_t *config)
```

Configure the conversion trigger source.

Each programmable trigger can launch the conversion command in command buffer.

**Parameters**

- base – LPADC peripheral base address.
- triggerId – ID for each trigger. Typically, the available value range is from 0.
- config – Pointer to configuration structure. See to `lpadc_conv_trigger_config_t`.

```
void LPADC_GetDefaultConvTriggerConfig(lpadc_conv_trigger_config_t *config)
```

Gets an available pre-defined settings for trigger's configuration.

This function initializes the trigger's configuration structure with an available settings. The default values are:

```
config->targetCommandId    = 0U;  
config->delayPower          = 0U;  
config->priority            = 0U;  
config->channelAFIFOSelect  = 0U;  
config->channelBFIFOSelect  = 0U;  
config->enableHardwareTrigger = false;
```

**Parameters**

- config – Pointer to configuration structure.

static inline void LPADC\_DoSoftwareTrigger(ADC\_Type \*base, uint32\_t triggerIdMask)  
Do software trigger to conversion command.

#### Parameters

- base – LPADC peripheral base address.
- triggerIdMask – Mask value for software trigger indexes, which count from zero.

void LPADC\_SetConvCommandConfig(ADC\_Type \*base, uint32\_t commandId, const  
*lpadc\_conv\_command\_config\_t* \*config)

Configure conversion command.

---

**Note:** The number of compare value register on different chips is different, that is mean in some chips, some command buffers do not have the compare functionality.

---

#### Parameters

- base – LPADC peripheral base address.
- commandId – ID for command in command buffer. Typically, the available value range is 1 - 15.
- config – Pointer to configuration structure. See to *lpadc\_conv\_command\_config\_t*.

void LPADC\_GetDefaultConvCommandConfig(*lpadc\_conv\_command\_config\_t* \*config)

Gets an available pre-defined settings for conversion command's configuration.

This function initializes the conversion command's configuration structure with an available settings. The default values are:

```
config->sampleScaleMode      = kLPADC_SampleFullScale;
config->channelBScaleMode    = kLPADC_SampleFullScale;
config->sampleChannelMode    = kLPADC_SampleChannelSingleEndSideA;
config->channelNumber        = 0U;
config->channelBNumber       = 0U;
config->chainedNextCommandNumber = 0U;
config->enableAutoChannelIncrement = false;
config->loopCount            = 0U;
config->hardwareAverageMode    = kLPADC_HardwareAverageCount1;
config->sampleTimeMode        = kLPADC_SampleTimeADCK3;
config->hardwareCompareMode    = kLPADC_HardwareCompareDisabled;
config->hardwareCompareValueHigh = 0U;
config->hardwareCompareValueLow  = 0U;
config->conversionResolutionMode = kLPADC_ConversionResolutionStandard;
config->enableWaitTrigger      = false;
config->enableChannelB        = false;
```

#### Parameters

- config – Pointer to configuration structure.

void LPADC\_EnableCalibration(ADC\_Type \*base, bool enable)

Enable the calibration function.

When CALOFS is set, the ADC is configured to perform a calibration function anytime the ADC executes a conversion. Any channel selected is ignored and the value returned in the RESFIFO is a signed value between -31 and 31. -32 is not a valid and is never a returned value. Software should copy the lower 6-bits of the conversion result stored in the RESFIFO after a completed calibration conversion to the OFSTRIM field. The OFSTRIM field is used in normal operation for offset correction.

**Parameters**

- base – LPADC peripheral base address.
- enable – switcher to the calibration function.

static inline void LPADC\_SetOffsetValue(ADC\_Type \*base, uint32\_t value)

Set proper offset value to trim ADC.

To minimize the offset during normal operation, software should read the conversion result from the RESFIFO calibration operation and write the lower 6 bits to the OFSTRIM register.

**Parameters**

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

status\_t LPADC\_DoAutoCalibration(ADC\_Type \*base)

Do auto calibration.

Calibration function should be executed before using converter in application. It used the software trigger and a dummy conversion, get the offset and write them into the OFSTRIM register. It called some of functional API including: -LPADC\_EnableCalibration(...) -LPADC\_LPADC\_SetOffsetValue(...) -LPADC\_SetConvCommandConfig(...) -LPADC\_SetConvTriggerConfig(...)

**Parameters**

- base – LPADC peripheral base address.
- base – LPADC peripheral base address.

**Return values**

- kStatus\_Success – Successfully configured.
- kStatus\_Timeout – Timeout occurs while waiting completion.

static inline void LPADC\_EnableOffsetCalibration(ADC\_Type \*base, bool enable)

Enable the offset calibration function.

**Parameters**

- base – LPADC peripheral base address.
- enable – switcher to the calibration function.

static inline void LPADC\_SetOffsetCalibrationMode(ADC\_Type \*base,  
lpadc\_offset\_calibration\_mode\_t mode)

Set offset calibration mode.

**Parameters**

- base – LPADC peripheral base address.
- mode – set offset calibration mode.see to lpadc\_offset\_calibration\_mode\_t.

status\_t LPADC\_DoOffsetCalibration(ADC\_Type \*base)

Do offset calibration.

**Parameters**

- base – LPADC peripheral base address.

**Return values**

- kStatus\_Success – Successfully configured.
- kStatus\_Timeout – Timeout occurs while waiting completion.

void LPADC\_PrepAutoCalibration(ADC\_Type \*base)

Prepare auto calibration, LPADC\_FinishAutoCalibration has to be called before using the LPADC. LPADC\_DoAutoCalibration has been split in two API to avoid to be stuck too long in the function.

#### Parameters

- base – LPADC peripheral base address.

status\_t LPADC\_FinishAutoCalibration(ADC\_Type \*base)

Finish auto calibration start with LPADC\_PrepAutoCalibration.

---

**Note:** This feature is used for LPADC with CTRL[CALOFSMODE].

---

#### Parameters

- base – LPADC peripheral base address.

#### Return values

- kStatus\_Success – Successfully configured.
- kStatus\_Timeout – Timeout occurs while waiting completion.

void LPADC\_GetCalibrationValue(ADC\_Type \*base, lpadc\_calibration\_value\_t \*ptrCalibrationValue)

Get calibration value into the memory which is defined by invoker.

---

**Note:** Please note the ADC will be disabled temporary.

---

---

**Note:** This function should be used after finish calibration.

---

#### Parameters

- base – LPADC peripheral base address.
- ptrCalibrationValue – Pointer to lpadc\_calibration\_value\_t structure, this memory block should be always powered on even in low power modes.

status\_t LPADC\_SetCalibrationValue(ADC\_Type \*base, const lpadc\_calibration\_value\_t \*ptrCalibrationValue)

Set calibration value into ADC calibration registers.

---

**Note:** Please note the ADC will be disabled temporary.

---

#### Parameters

- base – LPADC peripheral base address.
- ptrCalibrationValue – Pointer to lpadc\_calibration\_value\_t structure which contains ADC's calibration value.

#### Return values

- kStatus\_Success – Successfully configured.
- kStatus\_Timeout – Timeout occurs while waiting completion.

FSL\_LPADC\_DRIVER\_VERSION

LPADC driver version 2.9.3.

struct lpadc\_config\_t

*#include <fsl\_lpadc.h>* LPADC global configuration.

This structure would used to keep the settings for initialization.

### Public Members

bool enableInternalClock

Enables the internally generated clock source. The clock source is used in clock selection logic at the chip level and is optionally used for the ADC clock source.

bool enableVref1LowVoltage

If voltage reference option1 input is below 1.8V, it should be “true”. If voltage reference option1 input is above 1.8V, it should be “false”.

bool enableInDozeMode

Control system transition to Stop and Wait power modes while ADC is converting. When enabled in Doze mode, immediate entries to Wait or Stop are allowed. When disabled, the ADC will wait for the current averaging iteration/FIFO storage to complete before acknowledging stop or wait mode entry.

*lpadc\_conversion\_average\_mode\_t* conversionAverageMode

Auto-Calibration Averages.

bool enableAnalogPreliminary

ADC analog circuits are pre-enabled and ready to execute conversions without startup delays(at the cost of higher DC current consumption).

uint32\_t powerUpDelay

When the analog circuits are not pre-enabled, the ADC analog circuits are only powered while the ADC is active and there is a counted delay defined by this field after an initial trigger transitions the ADC from its Idle state to allow time for the analog circuits to stabilize. The startup delay count of (powerUpDelay \* 4) ADCK cycles must result in a longer delay than the analog startup time.

*lpadc\_reference\_voltage\_source\_t* referenceVoltageSource

Selects the voltage reference high used for conversions.

*lpadc\_power\_level\_mode\_t* powerLevelMode

Power Configuration Selection.

*lpadc\_trigger\_priority\_policy\_t* triggerPriorityPolicy

Control how higher priority triggers are handled, see to *lpadc\_trigger\_priority\_policy\_t*.

bool enableConvPause

Enables the ADC pausing function. When enabled, a programmable delay is inserted during command execution sequencing between LOOP iterations, between commands in a sequence, and between conversions when command is executing in “Compare Until True” configuration.

uint32\_t convPauseDelay

Controls the duration of pausing during command execution sequencing. The pause delay is a count of (convPauseDelay\*4) ADCK cycles. Only available when ADC pausing function is enabled. The available value range is in 9-bit.

uint32\_t FIFOWatermark

FIFOWatermark is a programmable threshold setting. When the number of datawords stored in the ADC Result FIFO is greater than the value in this field, the ready flag would be asserted to indicate stored data has reached the programmable threshold.

struct lpadc\_conv\_command\_config\_t

#include <fsl\_lpadc.h> Define structure to keep the configuration for conversion command.

### Public Members

*lpadc\_sample\_scale\_mode\_t* sampleScaleMode

Sample scale mode.

*lpadc\_sample\_scale\_mode\_t* channelBScaleMode

Alternate channel B Scale mode.

*lpadc\_sample\_channel\_mode\_t* sampleChannelMode

Channel sample mode.

uint32\_t channelNumber

Channel number, select the channel or channel pair.

uint32\_t channelBNumber

Alternate Channel B number, select the channel.

uint32\_t chainedNextCommandNumber

Selects the next command to be executed after this command completes. 1-15 is available, 0 is to terminate the chain after this command.

bool enableAutoChannelIncrement

Loop with increment: when disabled, the “loopCount” field selects the number of times the selected channel is converted consecutively; when enabled, the “loopCount” field defines how many consecutive channels are converted as part of the command execution.

uint32\_t loopCount

Selects how many times this command executes before finish and transition to the next command or Idle state. Command executes LOOP+1 times. 0-15 is available.

*lpadc\_hardware\_average\_mode\_t* hardwareAverageMode

Hardware average selection.

*lpadc\_sample\_time\_mode\_t* sampleTimeMode

Sample time selection.

*lpadc\_hardware\_compare\_mode\_t* hardwareCompareMode

Hardware compare selection.

uint32\_t hardwareCompareValueHigh

Compare Value High. The available value range is in 16-bit.

uint32\_t hardwareCompareValueLow

Compare Value Low. The available value range is in 16-bit.

*lpadc\_conversion\_resolution\_mode\_t* conversionResolutionMode

Conversion resolution mode.

bool enableWaitTrigger

Wait for trigger assertion before execution: when disabled, this command will be automatically executed; when enabled, the active trigger must be asserted again before executing this command.

```
struct lpadc_conv_trigger_config_t
```

*#include <fsl\_lpadc.h>* Define structure to keep the configuration for conversion trigger.

### Public Members

uint32\_t targetCommandId

Select the command from command buffer to execute upon detect of the associated trigger event.

uint32\_t delayPower

Select the trigger delay duration to wait at the start of servicing a trigger event. When this field is clear, then no delay is incurred. When this field is set to a non-zero value, the duration for the delay is  $2^{\text{delayPower}}$  ADCK cycles. The available value range is 4-bit.

uint32\_t priority

Sets the priority of the associated trigger source. If two or more triggers have the same priority level setting, the lower order trigger event has the higher priority. The lower value for this field is for the higher priority, the available value range is 1-bit.

bool enableHardwareTrigger

Enable hardware trigger source to initiate conversion on the rising edge of the input trigger source or not. The software trigger is always available.

```
struct lpadc_conv_result_t
```

*#include <fsl\_lpadc.h>* Define the structure to keep the conversion result.

### Public Members

uint32\_t commandIdSource

Indicate the command buffer being executed that generated this result.

uint32\_t loopCountIndex

Indicate the loop count value during command execution that generated this result.

uint32\_t triggerIdSource

Indicate the trigger source that initiated a conversion and generated this result.

uint16\_t convValue

Data result.

```
struct __lpadc_calibration_value
```

*#include <fsl\_lpadc.h>* A structure of calibration value.

## 2.47 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

enum `_gpio_pin_enable_mode`

GPIO Pin Interrupt enable mode.

*Values:*

enumerator `kGPIO_PinIntEnableLevel`

Generate Pin Interrupt on level mode

enumerator `kGPIO_PinIntEnableEdge`

Generate Pin Interrupt on edge mode

enum `_gpio_pin_enable_polarity`

GPIO Pin Interrupt enable polarity.

*Values:*

enumerator `kGPIO_PinIntEnableHighOrRise`

Generate Pin Interrupt on high level or rising edge

enumerator `kGPIO_PinIntEnableLowOrFall`

Generate Pin Interrupt on low level or falling edge

enum `_gpio_interrupt_index`

LPC GPIO interrupt index definition.

*Values:*

enumerator `kGPIO_InterruptA`

Set current pin as interrupt A

enumerator `kGPIO_InterruptB`

Set current pin as interrupt B

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.

typedef enum `_gpio_pin_enable_mode` `gpio_pin_enable_mode_t`

GPIO Pin Interrupt enable mode.

typedef enum `_gpio_pin_enable_polarity` `gpio_pin_enable_polarity_t`

GPIO Pin Interrupt enable polarity.

typedef enum `_gpio_interrupt_index` `gpio_interrupt_index_t`

LPC GPIO interrupt index definition.

typedef struct `_gpio_interrupt_config` `gpio_interrupt_config_t`

Configures the interrupt generation condition.

```
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

GPIO\_PIN\_INT\_LEVEL

GPIO\_PIN\_INT\_EDGE

PINT\_PIN\_INT\_HIGH\_OR\_RISE\_TRIGGER

PINT\_PIN\_INT\_LOW\_OR\_FALL\_TRIGGER

```
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

```
struct _gpio_interrupt_config
```

*#include <fsl\_gpio.h>* Configures the interrupt generation condition.

## 2.48 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

## 2.49 MU: Messaging Unit

void `MU_Init(MU_Type *base)`

Initializes the MU module.

This function enables the MU clock only.

### Parameters

- `base` – MU peripheral base address.

void `MU_Deinit(MU_Type *base)`

De-initializes the MU module.

This function disables the MU clock only.

### Parameters

- `base` – MU peripheral base address.

static inline void `MU_SendMsgNonBlocking(MU_Type *base, uint32_t regIndex, uint32_t msg)`

Writes a message to the TX register.

This function writes a message to the specific TX register. It does not check whether the TX register is empty or not. The upper layer should make sure the TX register is empty before calling this function. This function can be used in ISR for better performance.

```
while (!(kMU_Tx0EmptyFlag & MU_GetStatusFlags(base))) { } Wait for TX0 register empty.  
MU_SendMsgNonBlocking(base, kMU_MsgReg0, MSG_VAL); Write message to the TX0 register.
```

### Parameters

- `base` – MU peripheral base address.
- `regIndex` – TX register index, see `mu_msg_reg_index_t`.
- `msg` – Message to send.

void `MU_SendMsg(MU_Type *base, uint32_t regIndex, uint32_t msg)`

Blocks to send a message.

This function waits until the TX register is empty and sends the message.

### Parameters

- `base` – MU peripheral base address.
- `regIndex` – MU message register, see `mu_msg_reg_index_t`.
- `msg` – Message to send.



static inline uint32\_t MU\_ReceiveMsgNonBlocking(MU\_Type \*base, uint32\_t regIndex)

Reads a message from the RX register.

This function reads a message from the specific RX register. It does not check whether the RX register is full or not. The upper layer should make sure the RX register is full before calling this function. This function can be used in ISR for better performance.

```
uint32_t msg;
while (!(kMU_Rx0FullFlag & MU_GetStatusFlags(base)))
{
    } Wait for the RX0 register full.

msg = MU_ReceiveMsgNonBlocking(base, kMU_MsgReg0); Read message from RX0 register.
```

### Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see mu\_msg\_reg\_index\_t.

### Returns

The received message.

uint32\_t MU\_ReceiveMsg(MU\_Type \*base, uint32\_t regIndex)

Blocks to receive a message.

This function waits until the RX register is full and receives the message.

### Parameters

- base – MU peripheral base address.
- regIndex – MU message register, see mu\_msg\_reg\_index\_t

### Returns

The received message.

static inline void MU\_SetFlagsNonBlocking(MU\_Type \*base, uint32\_t flags)

Sets the 3-bit MU flags reflect on the other MU side.

This function sets the 3-bit MU flags directly. Every time the 3-bit MU flags are changed, the status flag kMU\_FlagsUpdatingFlag asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag kMU\_FlagsUpdatingFlag is cleared by hardware. During the flags updating period, the flags cannot be changed. The upper layer should make sure the status flag kMU\_FlagsUpdatingFlag is cleared before calling this function.

```
while (kMU_FlagsUpdatingFlag & MU_GetStatusFlags(base))
{
    } Wait for previous MU flags updating.

MU_SetFlagsNonBlocking(base, 0U); Set the mU flags.
```

### Parameters

- base – MU peripheral base address.
- flags – The 3-bit MU flags to set.

void MU\_SetFlags(MU\_Type \*base, uint32\_t flags)

Blocks setting the 3-bit MU flags reflect on the other MU side.

This function blocks setting the 3-bit MU flags. Every time the 3-bit MU flags are changed, the status flag kMU\_FlagsUpdatingFlag asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag kMU\_FlagsUpdatingFlag is cleared by hardware. During the flags updating period, the flags cannot be changed. This

function waits for the MU status flag `kMU_FlagsUpdatingFlag` cleared and sets the 3-bit MU flags.

#### Parameters

- `base` – MU peripheral base address.
- `flags` – The 3-bit MU flags to set.

static inline uint32\_t MU\_GetFlags(MU\_Type \*base)

Gets the current value of the 3-bit MU flags set by the other side.

This function gets the current 3-bit MU flags on the current side.

#### Parameters

- `base` – MU peripheral base address.

#### Returns

flags Current value of the 3-bit flags.

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

Gets the MU status flags.

This function returns the bit mask of the MU status flags. See `_mu_status_flags`.

```
uint32_t flags;
flags = MU_GetStatusFlags(base); Get all status flags.
if (kMU_Tx0EmptyFlag & flags)
{
    The TX0 register is empty. Message can be sent.
    MU_SendMsgNonBlocking(base, kMU_MsgReg0, MSG0_VAL);
}
if (kMU_Tx1EmptyFlag & flags)
{
    The TX1 register is empty. Message can be sent.
    MU_SendMsgNonBlocking(base, kMU_MsgReg1, MSG1_VAL);
}
```

#### Parameters

- `base` – MU peripheral base address.

#### Returns

Bit mask of the MU status flags, see `_mu_status_flags`.

static inline uint32\_t MU\_GetRxStatusFlags(MU\_Type \*base)

Return the RX status flags.

This function return the RX status flags. Note: RFn bits of SR[27-24](mu status register) are mapped in reverse numerical order: RF0 -> SR[27] RF1 -> SR[26] RF2 -> SR[25] RF3 -> SR[24]

```
status_reg = MU_GetRxStatusFlags(base);
```

#### Parameters

- `base` – MU peripheral base address.

#### Returns

MU RX status

static inline uint32\_t MU\_GetInterruptsPending(MU\_Type \*base)

Gets the MU IRQ pending status of enabled interrupts.

This function returns the bit mask of the pending MU IRQs of enabled interrupts. Only these flags are checked. `kMU_Tx0EmptyFlag` `kMU_Tx1EmptyFlag` `kMU_Tx2EmptyFlag` `kMU_Tx3EmptyFlag` `kMU_Rx0FullFlag` `kMU_Rx1FullFlag`

kMU\_Rx2FullFlag      kMU\_Rx3FullFlag      kMU\_GenInt0Flag      kMU\_GenInt1Flag  
 kMU\_GenInt2Flag kMU\_GenInt3Flag

### Parameters

- base – MU peripheral base address.

### Returns

Bit mask of the MU IRQs pending.

```
static inline void MU_ClearStatusFlags(MU_Type *base, uint32_t mask)
```

Clears the specific MU status flags.

This function clears the specific MU status flags. The flags to clear should be passed in as bit mask. See `_mu_status_flags`.

Clear general interrupt 0 and general interrupt 1 pending flags.  
 MU\_ClearStatusFlags(base, kMU\_GenInt0Flag | kMU\_GenInt1Flag);

### Parameters

- base – MU peripheral base address.
- mask – Bit mask of the MU status flags. See `_mu_status_flags`. The following flags are cleared by hardware, this function could not clear them.
  - kMU\_Tx0EmptyFlag
  - kMU\_Tx1EmptyFlag
  - kMU\_Tx2EmptyFlag
  - kMU\_Tx3EmptyFlag
  - kMU\_Rx0FullFlag
  - kMU\_Rx1FullFlag
  - kMU\_Rx2FullFlag
  - kMU\_Rx3FullFlag
  - kMU\_EventPendingFlag
  - kMU\_FlagsUpdatingFlag
  - kMU\_OtherSideInResetFlag

```
static inline void MU_EnableInterrupts(MU_Type *base, uint32_t mask)
```

Enables the specific MU interrupts.

This function enables the specific MU interrupts. The interrupts to enable should be passed in as bit mask. See `_mu_interrupt_enable`.

Enable general interrupt 0 and TX0 empty interrupt.  
 MU\_EnableInterrupts(base, kMU\_GenInt0InterruptEnable | kMU\_Tx0EmptyInterruptEnable);

### Parameters

- base – MU peripheral base address.
- mask – Bit mask of the MU interrupts. See `_mu_interrupt_enable`.

```
static inline void MU_DisableInterrupts(MU_Type *base, uint32_t mask)
```

Disables the specific MU interrupts.

This function disables the specific MU interrupts. The interrupts to disable should be passed in as bit mask. See `_mu_interrupt_enable`.

```
Disable general interrupt 0 and TX0 empty interrupt.
MU_DisableInterrupts(base, kMU_GenInt0InterruptEnable | kMU_Tx0EmptyInterruptEnable);
```

### Parameters

- base – MU peripheral base address.
- mask – Bit mask of the MU interrupts. See `_mu_interrupt_enable`.

*status\_t* MU\_TriggerInterrupts(MU\_Type \*base, uint32\_t mask)

Triggers interrupts to the other core.

This function triggers the specific interrupts to the other core. The interrupts to trigger are passed in as bit mask. See `_mu_interrupt_trigger`. The MU should not trigger an interrupt to the other core when the previous interrupt has not been processed by the other core. This function checks whether the previous interrupts have been processed. If not, it returns an error.

```
if (kStatus_Success != MU_TriggerInterrupts(base, kMU_GenInt0InterruptTrigger | kMU_
↪ GenInt2InterruptTrigger))
{
    Previous general purpose interrupt 0 or general purpose interrupt 2
    has not been processed by the other core.
}
```

### Parameters

- base – MU peripheral base address.
- mask – Bit mask of the interrupts to trigger. See `_mu_interrupt_trigger`.

### Return values

- kStatus\_Success – Interrupts have been triggered successfully.
- kStatus\_Fail – Previous interrupts have not been accepted.

static inline void MU\_MaskHardwareReset(MU\_Type \*base, bool mask)

Mask hardware reset by the other core.

The other core could call `MU_HardwareResetOtherCore()` to reset current core. To mask the reset, call this function and pass in true.

### Parameters

- base – MU peripheral base address.
- mask – Pass true to mask the hardware reset, pass false to unmask it.

static inline mu\_power\_mode\_t MU\_GetOtherCorePowerMode(MU\_Type \*base)

Gets the power mode of the other core.

This function gets the power mode of the other core.

### Parameters

- base – MU peripheral base address.

### Returns

Power mode of the other core.

FSL\_MU\_DRIVER\_VERSION

MU driver version.

enum \_mu\_status\_flags

MU status flags.

*Values:*

enumerator kMU\_\_Tx0EmptyFlag

TX0 empty.

enumerator kMU\_\_Tx1EmptyFlag

TX1 empty.

enumerator kMU\_\_Tx2EmptyFlag

TX2 empty.

enumerator kMU\_\_Tx3EmptyFlag

TX3 empty.

enumerator kMU\_\_Rx0FullFlag

RX0 full.

enumerator kMU\_\_Rx1FullFlag

RX1 full.

enumerator kMU\_\_Rx2FullFlag

RX2 full.

enumerator kMU\_\_Rx3FullFlag

RX3 full.

enumerator kMU\_\_GenInt0Flag

General purpose interrupt 0 pending.

enumerator kMU\_\_GenInt1Flag

General purpose interrupt 1 pending.

enumerator kMU\_\_GenInt2Flag

General purpose interrupt 2 pending.

enumerator kMU\_\_GenInt3Flag

General purpose interrupt 3 pending.

enumerator kMU\_\_EventPendingFlag

MU event pending.

enumerator kMU\_\_FlagsUpdatingFlag

MU flags update is on-going.

enumerator kMU\_\_ResetAssertInterruptFlag

The other core reset assert interrupt pending.

enumerator kMU\_\_ResetDeassertInterruptFlag

The other core reset de-assert interrupt pending.

enumerator kMU\_\_OtherSideInResetFlag

The other side is in reset.

enumerator kMU\_\_MuResetInterruptFlag

The other side initializes MU reset.

enumerator kMU\_\_HardwareResetInterruptFlag

Current side has been hardware reset by the other side.

enum `_mu_interrupt_enable`

MU interrupt source to enable.

*Values:*

enumerator `kMU_Tx0EmptyInterruptEnable`

TX0 empty.

enumerator `kMU_Tx1EmptyInterruptEnable`

TX1 empty.

enumerator `kMU_Tx2EmptyInterruptEnable`

TX2 empty.

enumerator `kMU_Tx3EmptyInterruptEnable`

TX3 empty.

enumerator `kMU_Rx0FullInterruptEnable`

RX0 full.

enumerator `kMU_Rx1FullInterruptEnable`

RX1 full.

enumerator `kMU_Rx2FullInterruptEnable`

RX2 full.

enumerator `kMU_Rx3FullInterruptEnable`

RX3 full.

enumerator `kMU_GenInt0InterruptEnable`

General purpose interrupt 0.

enumerator `kMU_GenInt1InterruptEnable`

General purpose interrupt 1.

enumerator `kMU_GenInt2InterruptEnable`

General purpose interrupt 2.

enumerator `kMU_GenInt3InterruptEnable`

General purpose interrupt 3.

enumerator `kMU_ResetAssertInterruptEnable`

The other core reset assert interrupt.

enumerator `kMU_ResetDeassertInterruptEnable`

The other core reset de-assert interrupt.

enumerator `kMU_MuResetInterruptEnable`

The other side initializes MU reset. The interrupt is ORed with the general purpose interrupt 3. The general purpose interrupt 3 is issued when the other side set the MU reset and this interrupt is enabled.

enumerator `kMU_HardwareResetInterruptEnable`

Current side has been hardware reset by the other side.

enum `_mu_interrupt_trigger`

MU interrupt that could be triggered to the other core.

*Values:*

enumerator `kMU_GenInt0InterruptTrigger`

General purpose interrupt 0.

enumerator kMU\_GenInt1InterruptTrigger  
General purpose interrupt 1.

enumerator kMU\_GenInt2InterruptTrigger  
General purpose interrupt 2.

enumerator kMU\_GenInt3InterruptTrigger  
General purpose interrupt 3.

enum \_mu\_msg\_reg\_index  
MU message register.

*Values:*

enumerator kMU\_MsgReg0

enumerator kMU\_MsgReg1

enumerator kMU\_MsgReg2

enumerator kMU\_MsgReg3

typedef enum \_mu\_msg\_reg\_index mu\_msg\_reg\_index\_t  
MU message register.

MU\_CR\_NMI\_MASK

FSL\_FEATURE\_MU\_HAS\_RESET\_ASSERT\_INT

FSL\_FEATURE\_MU\_HAS\_RESET\_DEASSERT\_INT

MU\_GET\_CORE\_FLAG(flags)

MU\_GET\_STAT\_FLAG(flags)

MU\_GET\_TX\_FLAG(flags)

MU\_GET\_RX\_FLAG(flags)

MU\_GET\_GI\_FLAG(flags)

## 2.50 OSTIMER: OS Event Timer Driver

void OSTIMER\_Init(OSTIMER\_Type \*base)  
Initializes an OSTIMER by turning its bus clock on.

void OSTIMER\_Deinit(OSTIMER\_Type \*base)  
Deinitializes a OSTIMER instance.  
This function shuts down OSTIMER bus clock

### Parameters

- base – OSTIMER peripheral base address.

uint64\_t OSTIMER\_GrayToDecimal(uint64\_t gray)  
Translate the value from gray-code to decimal.

### Parameters

- gray – The gray value input.

### Returns

The decimal value.

```
static inline uint64_t OSTIMER_DecimalToGray(uint64_t dec)
```

Translate the value from decimal to gray-code.

**Parameters**

- `dec` – The decimal value.

**Returns**

The gray code of the input value.

```
uint32_t OSTIMER_GetStatusFlags(OSTIMER_Type *base)
```

Get OSTIMER status Flags.

This returns the status flag. Currently, only match interrupt flag can be got.

**Parameters**

- `base` – OSTIMER peripheral base address.

**Returns**

status register value

```
void OSTIMER_ClearStatusFlags(OSTIMER_Type *base, uint32_t mask)
```

Clear Status Interrupt Flags.

This clears intrrupt status flag. Currently, only match interrupt flag can be cleared.

**Parameters**

- `base` – OSTIMER peripheral base address.
- `mask` – Clear bit mask.

**Returns**

none

```
status_t OSTIMER_SetMatchRawValue(OSTIMER_Type *base, uint64_t count, ostimer_callback_t cb)
```

Set the match raw value for OSTIMER.

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central EVTIMER. Please note that, the data format is gray-code, if decimal data was desired, please using `OSTIMER_SetMatchValue()`.

**Parameters**

- `base` – OSTIMER peripheral base address.
- `count` – OSTIMER timer match value.(Value is gray-code format)
- `cb` – OSTIMER callback (can be left as NULL if none, otherwise should be a void func(void)).

**Return values**

- `kStatus_Success` – - Set match raw value and enable interrupt Successfully.
- `kStatus_Fail` – - Set match raw value fail.

```
status_t OSTIMER_SetMatchValue(OSTIMER_Type *base, uint64_t count, ostimer_callback_t cb)
```

Set the match value for OSTIMER.

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central OS TIMER.

**Parameters**

- `base` – OSTIMER peripheral base address.



- `count` – OSTIMER timer match value.(Value is decimal format, and this value will be translate to Gray code internally.)
- `cb` – OSTIMER callback (can be left as NULL if none, otherwise should be a void func(void)).

#### Return values

- `kStatus__Success` – - Set match value and enable interrupt Successfully.
- `kStatus__Fail` – - Set match value fail.

static inline void OSTIMER\_SetMatchRegister(OSTIMER\_Type \*base, uint64\_t value)

Set value to OSTIMER MATCH register directly.

This function writes the input value to OSTIMER MATCH register directly, it does not touch any other registers. Note that, the data format is gray-code. The function OSTIMER\_DecimalToGray could convert decimal value to gray code.

#### Parameters

- `base` – OSTIMER peripheral base address.
- `value` – OSTIMER timer match value (Value is gray-code format).

static inline void OSTIMER\_EnableMatchInterrupt(OSTIMER\_Type \*base)

Enable the OSTIMER counter match interrupt.

Enable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

#### Parameters

- `base` – OSTIMER peripheral base address.

static inline void OSTIMER\_DisableMatchInterrupt(OSTIMER\_Type \*base)

Disable the OSTIMER counter match interrupt.

Disable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

#### Parameters

- `base` – OSTIMER peripheral base address.

static inline uint64\_t OSTIMER\_GetCurrentTimerRawValue(OSTIMER\_Type \*base)

Get current timer raw count value from OSTIMER.

This function will get a gray code type timer count value from OS timer register. The raw value of timer count is gray code format.

#### Parameters

- `base` – OSTIMER peripheral base address.

#### Returns

Raw value of OSTIMER, gray code format.

uint64\_t OSTIMER\_GetCurrentTimerValue(OSTIMER\_Type \*base)

Get current timer count value from OSTIMER.

This function will get a decimal timer count value. The RAW value of timer count is gray code format, will be translated to decimal data internally.

#### Parameters

- `base` – OSTIMER peripheral base address.

#### Returns

Value of OSTIMER which will be formatted to decimal value.

static inline uint64\_t OSTIMER\_GetCaptureRawValue(OSTIMER\_Type \*base)

Get the capture value from OSTIMER.

This function will get a captured gray-code value from OSTIMER. The Raw value of timer capture is gray code format.

**Parameters**

- base – OSTIMER peripheral base address.

**Returns**

Raw value of capture register, data format is gray code.

uint64\_t OSTIMER\_GetCaptureValue(OSTIMER\_Type \*base)

Get the capture value from OSTIMER.

This function will get a capture decimal-value from OSTIMER. The RAW value of timer capture is gray code format, will be translated to decimal data internally.

**Parameters**

- base – OSTIMER peripheral base address.

**Returns**

Value of capture register, data format is decimal.

void OSTIMER\_HandleIRQ(OSTIMER\_Type \*base, *ostimer\_callback\_t* cb)

OS timer interrupt Service Handler.

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in OSTIMER\_SetMatchValue()). if no user callback is scheduled, the interrupt will simply be cleared.

**Parameters**

- base – OS timer peripheral base address.
- cb – callback scheduled for this instance of OS timer

**Returns**

none

FSL\_OSTIMER\_DRIVER\_VERSION

OSTIMER driver version.

enum *\_ostimer\_flags*

OSTIMER status flags.

*Values:*

enumerator kOSTIMER\_MatchInterruptFlag

Match interrupt flag bit, sets if the match value was reached.

typedef void (\**ostimer\_callback\_t*)(void)

ostimer callback function.

## 2.51 OTFAD: On The Fly AES-128 Decryption Driver

void OTFAD\_GetDefaultConfig(*otfad\_config\_t* \*config)

OTFAD module initialization function.

**Parameters**

- config – OTFAD configuration.

AT\_QUICKACCESS\_SECTION\_CODE (void OTFAD\_Init(OTFAD\_Type \*base,  
const *otfad\_config\_t* \*config))

OTFAD module initialization function.

#### Parameters

- base – OTFAD base address.
- config – OTFAD configuration.

AT\_QUICKACCESS\_SECTION\_CODE (void OTFAD\_Deinit(OTFAD\_Type \*base))

Deinitializes the OTFAD.

static inline uint32\_t OTFAD\_GetOperateMode(OTFAD\_Type \*base)

OTFAD module get operate mode.

#### Parameters

- base – OTFAD base address.

static inline uint32\_t OTFAD\_GetStatus(OTFAD\_Type \*base)

OTFAD module get status.

#### Parameters

- base – OTFAD base address.

*status\_t* OTFAD\_SetEncryptionConfig(OTFAD\_Type \*base, const *otfad\_encryption\_config\_t*  
\*config)

OTFAD module set encryption configuration.

Note: if enable keyblob process, the first 256 bytes external memory is use for keyblob data, so this region shouldn't be in OTFAD region.

#### Parameters

- base – OTFAD base address.
- config – encryption configuration.

*status\_t* OTFAD\_GetEncryptionConfig(OTFAD\_Type \*base, *otfad\_encryption\_config\_t* \*config)

OTFAD module get encryption configuration.

Note: if enable keyblob process, the first 256 bytes external memory is use for keyblob data, so this region shouldn't be in OTFAD region.

#### Parameters

- base – OTFAD base address.
- config – encryption configuration.

*status\_t* OTFAD\_HitDetermination(OTFAD\_Type \*base, uint32\_t address, uint8\_t \*contextIndex)

OTFAD module do hit determination.

#### Parameters

- base – OTFAD base address.
- address – the physical address space assigned to the QuadSPI(FlexSPI) module.
- contextIndex – hitted context region index.

#### Returns

status, such as kStatus\_Success or kStatus\_OTFAD\_ResRegAccessMode.

FSL\_OTFAD\_DRIVER\_VERSION

Driver version.

Status codes for the OTFAD driver.

*Values:*

enumerator kStatus\_OTFAD\_ResRegAccessMode

Restricted register mode

enumerator kStatus\_OTFAD\_AddressError

End address less than start address

enumerator kStatus\_OTFAD\_RegionOverlap

the OTFAD does not support any form of memory region overlap, for system accesses that hit in multiple contexts or no contexts, the fetched data is simply bypassed

enumerator kStatus\_OTFAD\_RegionMiss

For accesses that hit in a single context, but not the selected one

OTFAD context type.

*Values:*

enumerator kOTFAD\_Context\_0

context 0

enumerator kOTFAD\_Context\_1

context 1

enumerator kOTFAD\_Context\_2

context 2

enumerator kOTFAD\_Context\_3

context 3

OTFAD operate mode.

*Values:*

enumerator kOTFAD\_NRM

Normal Mode

enumerator kOTFAD\_SVM

Security Violation Mode

enumerator kOTFAD\_LDM

Logically Disabled Mode

typedef struct *\_otfad\_encryption\_config* otfad\_encryption\_config\_t

OTFAD encryption configuration structure.

typedef struct *\_otfad\_config* otfad\_config\_t

OTFAD configuration structure.

struct *\_otfad\_encryption\_config*

*#include <fsl\_otfad.h>* OTFAD encryption configuration structure.

**Public Members**

bool valid

The context is valid or not

bool AESdecryption

AES decryption enable

uint8\_t readOnly

read write attribute for the entire set of context registers

uint8\_t contextIndex

OTFAD context index

uint32\_t startAddr

Start address

uint32\_t endAddr

End address

uint32\_t key[4]

Encryption key

uint32\_t counter[2]

Encryption counter

struct \_otfad\_config

*#include <fsl\_otfad.h>* OTFAD configuration structure.

**Public Members**

bool enableIntRequest

Interrupt request enable

bool forceError

Forces the OTFAD's key blob error flag (SR[KBERR]) to be asserted

bool forceSVM

Force entry into SVM after a write

bool forceLDM

Force entry into LDM after a write

bool keyBlobScramble

Key blob KEK scrambling

bool keyBlobProcess

Key blob processing

bool startKeyBlobProcessing

key blob processing is initiated

bool restrictedRegAccess

Restricted register access enable

bool enableOTFAD

OTFAD has decryption enabled

## 2.52 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, 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, 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, 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, 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, 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>*

## 2.53 Power Driver

enum \_pmc\_interrupt

PMC event flags.

---

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

---

*Values:*

enumerator kPMC\_INT\_LVDCORE

Vddcore Low-Voltage Detector Interrupt Enable.

enumerator kPMC\_INT\_HVDCORE

Vddcore High-Voltage Detector Interrupt Enable.

enumerator kPMC\_INT\_HVD1V8

Vdd1v8 High-Voltage Detector Interrupt Enable.

enumerator kPMC\_INT\_AUTOWK

PMC automatic wakeup enable and interrupt enable.

enumerator kPMC\_INT\_INTRPAD

Interrupt pad deep powerdown and deep sleep wake up & interrupt enable.

enum \_pmc\_event\_flags

PMC event flags.

---

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

---

*Values:*

enumerator kPMC\_FLAGS\_PORCORE

POR triggered by the vddcore POR monitor (0 = no, 1 = yes).

enumerator kPMC\_FLAGS\_POR1V8

vdd1v8 power on event detected since last cleared(0 = no, 1 = yes).

enumerator kPMC\_FLAGS\_PORAO18

vdd\_ao18 power on event detected since last cleared (0 = no, 1 = yes).

enumerator kPMC\_FLAGS\_LVDCORE

LVD tripped since last time this bit was cleared (0 = no, 1 = yes).

enumerator kPMC\_FLAGS\_HVDCORE

HVD tripped since last time this bit was cleared (0 = no, 1 = yes).

enumerator kPMC\_FLAGS\_HVD1V8

vdd1v8 HVD tripped since last time this bit was cleared (0 = no, 1 = yes).

enumerator kPMC\_FLAGS\_RTC

RTC wakeup detected since last time flag was cleared (0 = no, 1 = yes).

enumerator kPMC\_FLAGS\_AUTOWK

PMC Auto wakeup caused a deep sleep wakeup and interrupt (0 = no, 1 = yes).

enumerator kPMC\_FLAGS\_INTNPADF

Pad interrupt caused a wakeup or interrupt event since the last time this flag was cleared (0 = no, 1 = yes).

enumerator kPMC\_FLAGS\_RESETPAD

Reset pad wakeup caused a wakeup or reset event since the last time this bit was cleared. (0 = no, 1 = yes).

enumerator kPMC\_FLAGS\_DEEPPD

Deep powerdown was entered since the last time this flag was cleared (0 = no, 1 = yes).

enum pd\_bits

*Values:*

enumerator kPDRUNCFG\_PMC\_MODE0

enumerator kPDRUNCFG\_PMC\_MODE1

enumerator kPDRUNCFG\_LP\_VDD\_COREREG

enumerator kPDRUNCFG\_LP\_PMCREF

enumerator kPDRUNCFG\_PD\_HVD1V8

enumerator kPDRUNCFG\_LP\_PORCORE

enumerator kPDRUNCFG\_LP\_LVDCORE

enumerator kPDRUNCFG\_PD\_HVDCORE

enumerator kPDRUNCFG\_PD\_SYSXTAL

enumerator kPDRUNCFG\_PD\_LPOSC

enumerator kPDRUNCFG\_PD\_SFRO

enumerator kPDRUNCFG\_PD\_FFRO

enumerator kPDRUNCFG\_PD\_SYSPLL\_LDO

enumerator kPDRUNCFG\_PD\_SYSPLL\_ANA

enumerator kPDRUNCFG\_PD\_AUDPLL\_LDO

enumerator kPDRUNCFG\_PD\_AUDPLL\_ANA

enumerator kPDRUNCFG\_PD\_ADC  
enumerator kPDRUNCFG\_LP\_ADC  
enumerator kPDRUNCFG\_PD\_ADC\_TEMPSNS  
enumerator kPDRUNCFG\_PD\_ACMP  
enumerator kPDRUNCFG\_LP\_HSPAD\_VDET  
enumerator kPDRUNCFG\_PD\_HSPAD\_REF  
enumerator kPDRUNCFG\_APD\_PQ\_SRAM  
enumerator kPDRUNCFG\_PPD\_PQ\_SRAM  
enumerator kPDRUNCFG\_APD\_FLEXSPI\_SRAM  
enumerator kPDRUNCFG\_PPD\_FLEXSPI\_SRAM  
enumerator kPDRUNCFG\_APD\_USBHS\_SRAM  
enumerator kPDRUNCFG\_PPD\_USBHS\_SRAM  
enumerator kPDRUNCFG\_APD\_USDHC0\_SRAM  
enumerator kPDRUNCFG\_PPD\_USDHC0\_SRAM  
enumerator kPDRUNCFG\_APD\_USDHC1\_SRAM  
enumerator kPDRUNCFG\_PPD\_USDHC1\_SRAM  
enumerator kPDRUNCFG\_APD\_CASPER\_SRAM  
enumerator kPDRUNCFG\_PPD\_CASPER\_SRAM  
enumerator kPDRUNCFG\_APD\_DSP\_CACHE\_REGF  
enumerator kPDRUNCFG\_PPD\_DSP\_CACHE\_REGF  
enumerator kPDRUNCFG\_APD\_DSP\_TCM\_REGF  
enumerator kPDRUNCFG\_PPD\_DSP\_TCM\_REGF  
enumerator kPDRUNCFG\_PD\_ROM  
enumerator kPDRUNCFG\_SRAM\_SLEEP  
enumerator kPDRUNCFG\_APD\_SRAM\_IF0  
enumerator kPDRUNCFG\_APD\_SRAM\_IF1  
enumerator kPDRUNCFG\_APD\_SRAM\_IF2  
enumerator kPDRUNCFG\_APD\_SRAM\_IF3  
enumerator kPDRUNCFG\_APD\_SRAM\_IF4  
enumerator kPDRUNCFG\_APD\_SRAM\_IF5  
enumerator kPDRUNCFG\_APD\_SRAM\_IF6  
enumerator kPDRUNCFG\_APD\_SRAM\_IF7  
enumerator kPDRUNCFG\_APD\_SRAM\_IF8



enumerator kPDRUNCFG\_APD\_SRAM\_IF9  
enumerator kPDRUNCFG\_APD\_SRAM\_IF10  
enumerator kPDRUNCFG\_APD\_SRAM\_IF11  
enumerator kPDRUNCFG\_APD\_SRAM\_IF12  
enumerator kPDRUNCFG\_APD\_SRAM\_IF13  
enumerator kPDRUNCFG\_APD\_SRAM\_IF14  
enumerator kPDRUNCFG\_APD\_SRAM\_IF15  
enumerator kPDRUNCFG\_APD\_SRAM\_IF16  
enumerator kPDRUNCFG\_APD\_SRAM\_IF17  
enumerator kPDRUNCFG\_APD\_SRAM\_IF18  
enumerator kPDRUNCFG\_APD\_SRAM\_IF19  
enumerator kPDRUNCFG\_APD\_SRAM\_IF20  
enumerator kPDRUNCFG\_APD\_SRAM\_IF21  
enumerator kPDRUNCFG\_APD\_SRAM\_IF22  
enumerator kPDRUNCFG\_APD\_SRAM\_IF23  
enumerator kPDRUNCFG\_APD\_SRAM\_IF24  
enumerator kPDRUNCFG\_APD\_SRAM\_IF25  
enumerator kPDRUNCFG\_APD\_SRAM\_IF26  
enumerator kPDRUNCFG\_APD\_SRAM\_IF27  
enumerator kPDRUNCFG\_APD\_SRAM\_IF28  
enumerator kPDRUNCFG\_APD\_SRAM\_IF29  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF0  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF1  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF2  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF3  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF4  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF5  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF6  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF7  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF8  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF9  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF10  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF11

enumerator kPDRUNCFG\_PPD\_SRAM\_IF12  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF13  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF14  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF15  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF16  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF17  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF18  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF19  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF20  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF21  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF22  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF23  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF24  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF25  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF26  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF27  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF28  
enumerator kPDRUNCFG\_PPD\_SRAM\_IF29  
enumerator kPDRUNCFG\_ForceUnsigned

enum \_\_power\_mode\_config

Power mode configuration API parameter.

*Values:*

enumerator kPmu\_Sleep  
enumerator kPmu\_Deep\_Sleep  
enumerator kPmu\_Deep\_PowerDown  
enumerator kPmu\_Full\_Deep\_PowerDown

enum \_\_body\_bias\_mode

Body Bias mode definition.

*Values:*

enumerator kPmu\_Fbb  
enumerator kPmu\_Rbb  
enumerator kPmu\_Nbb

enum \_\_pmic\_mode\_reg

*Values:*

enumerator kCfg\_Run

enumerator kCfg\_Sleep

enum \_\_power\_\_pad\_vrange\_val  
pad voltage range value.

*Values:*

enumerator kPadVol\_171\_360  
Voltage from 1.71V to 3.60V.

enumerator kPadVol\_171\_198  
Voltage from 1.71V to 1.98V.

enumerator kPadVol\_300\_360  
Voltage from 3.00V to 3.60V.

enum \_\_power\_\_lvd\_falling\_trip\_vol\_val  
LVD falling trip voltage value.

*Values:*

enumerator kLvdFallingTripVol\_720  
Voltage 720mV.

enumerator kLvdFallingTripVol\_735  
Voltage 735mV.

enumerator kLvdFallingTripVol\_750  
Voltage 750mV.

enumerator kLvdFallingTripVol\_765  
Voltage 765mV.

enumerator kLvdFallingTripVol\_780  
Voltage 780mV.

enumerator kLvdFallingTripVol\_795  
Voltage 795mV.

enumerator kLvdFallingTripVol\_810  
Voltage 810mV.

enumerator kLvdFallingTripVol\_825  
Voltage 825mV.

enumerator kLvdFallingTripVol\_840  
Voltage 840mV.

enumerator kLvdFallingTripVol\_855  
Voltage 855mV.

enumerator kLvdFallingTripVol\_870  
Voltage 870mV.

enumerator kLvdFallingTripVol\_885  
Voltage 885mV.

enumerator kLvdFallingTripVol\_900  
Voltage 900mV.

enumerator kLvdFallingTripVol\_915  
Voltage 915mV.

enumerator kLvdFallingTripVol\_930  
Voltage 930mV.

enumerator kLvdFallingTripVol\_945  
Voltage 945mV.

enum \_\_power\_\_part\_\_temp\_\_range  
Part temperature range.

*Values:*

enumerator kPartTemp\_0C\_P85C  
Part temp range 0C - 85C.

enumerator kPartTemp\_N20C\_P85C  
Part temp range -20C - 85C.

enum \_\_power\_\_volt\_\_op\_\_range  
Voltage operation range.

*Values:*

enumerator kVoltOpLowRange  
Voltage operation range is (0.7V, 0.8V, 0.9V). Maximum supported CM33 frequency is 220MHz for 0C-85C part and 215MHz for -20C-85C part. Maximum supported DSP frequency is 375MHz for 0C-85C part and 355MHz for -20C-85C part.

enumerator kVoltOpFullRange  
Voltage operation range is (0.7V, 0.8V, 0.9V, 1.0V, 1.13V). This range can support full CM33/DSP speed clarified in Data Sheet.

enum \_\_power\_\_vddcore\_\_src  
VDDCORE supply source.

*Values:*

enumerator kVddCoreSrc\_LDO  
VDDCORE is supplied by onchip regulator.

enumerator kVddCoreSrc\_PMIC  
VDDCORE is supplied by external PMIC.

enum \_\_power\_\_control\_\_for\_\_pmic\_\_mode  
vddcore or vdd1v8 power on selection for different PMIC mode. vddcore and vdd1v8 are always on in mode0. Refer to PMC->PMICCFG.

*Values:*

enumerator kVddCoreOnMode1  
VDDCORE is powered in PMIC mode1.

enumerator kVddCoreOnMode2  
VDDCORE is powered in PMIC mode2.

enumerator kVddCoreOnMode3  
VDDCORE is powered in PMIC mode3.

enumerator kVdd1v8OnMode1  
VDD1V8 is powered in PMIC mode1.

enumerator kVdd1v8OnMode2  
VDD1V8 is powered in PMIC mode2.

enumerator `kVdd1v8OnMode3`

VDD1V8 is powered in PMIC mode3.

typedef enum `pd_bits` `pd_bit_t`

typedef enum `_power_mode_config` `power_mode_cfg_t`

Power mode configuration API parameter.

typedef enum `_body_bias_mode` `body_bias_mode_t`

Body Bias mode definition.

typedef enum `_pmic_mode_reg` `pmic_mode_reg_t`

typedef enum `_power_pad_vrange_val` `power_pad_vrange_val_t`

pad voltage range value.

typedef struct `_power_pad_vrange` `power_pad_vrange_t`

pad voltage range configuration.

typedef enum `_power_lvd_falling_trip_vol_val` `power_lvd_falling_trip_vol_val_t`

LVD falling trip voltage value.

typedef enum `_power_part_temp_range` `power_part_temp_range_t`

Part temperature range.

typedef enum `_power_volt_op_range` `power_volt_op_range_t`

Voltage operation range.

typedef enum `_power_vddcore_src` `power_vddcore_src_t`

VDDCORE supply source.

typedef enum `_power_control_for_pmic_mode` `power_control_for_pmic_mode`

vddcore or vdd1v8 power on selection for different PMIC mode. vddcore and vdd1v8 are always on in mode0. Refer to PMC->PMICCFG.

typedef void (\*`power_vddcore_set_func_t`)(uint32\_t millivolt)

Callback function used to change VDDCORE when the VDDCORE is supplied by external PMIC. Refer to `POWER_SetVddCoreSupplySrc()`

void `POWER_PmicPowerModeSelectControl`(uint32\_t `vddSelect`)

API to set vddcore or vdd1v8 power on for PMIC modes which is responded to `PDRUNCFG0[PMIC_MODE]` or `PDSLEEPCFG0[PMIC_MODE]` select pin values. If not set, the driver will use default configurations for different PMIC mode. The default configuration is as below. `PMIC_MODE`: power mode select 0b00 run mode, all supplies on. 0b01 deep sleep mode, all supplies on. 0b10 deep powerdown mode, vddcore off. 0b11 full deep powerdown mode vdd1v8 and vddcore off.

Note, be cautious to modify the VDD state in different PMIC mode. When the default configuration is changed, use `exclude_from_pd[0]` to configure the PMIC mode for deep sleep and power down mode.

#### Parameters

- `vddSelect` – the ORd value of `power_control_for_pmic_mode`. Defines run (all supplies on), deep power-down (VDDCORE off), and true deep power-down (VDD1V8 and VDDCORE off).

static inline void `POWER_EnablePD`(`pd_bit_t` `en`)

API to enable `PDRUNCFG` bit in the `Sysctl0`. Note that enabling the bit powers down the peripheral.

#### Parameters

- `en` – peripheral for which to enable the `PDRUNCFG` bit

static inline void POWER\_DisablePD(*pd\_bit\_t* en)

API to disable PDRUNCFG bit in the Sysctl0. Note that disabling the bit powers up the peripheral.

**Parameters**

- en – peripheral for which to disable the PDRUNCFG bit

static inline void POWER\_EnableDeepSleep(void)

API to enable deep sleep bit in the ARM Core.

static inline void POWER\_DisableDeepSleep(void)

API to disable deep sleep bit in the ARM Core.

void POWER\_UpdateOscSettlingTime(uint32\_t osc\_delay)

API to update XTAL oscillator settling time .

**Parameters**

- osc\_delay – : OSC stabilization time in unit of microsecond

void POWER\_UpdatePmicRecoveryTime(uint32\_t pmic\_delay)

API to update on-board PMIC vddcore recovery time.

NOTE: If LDO is used instead of PMIC, don't call it. Otherwise it must be called to allow power library to well handle the deep sleep process.

**Parameters**

- pmic\_delay – : PMIC stabilization time in unit of microsecond, or PMIC\_VDDCORE\_RECOVERY\_TIME\_IGNORE if not care.

void POWER\_ApplyPD(void)

API to apply updated PMC PDRUNCFG bits in the Sysctl0.

void POWER\_ClearEventFlags(uint32\_t statusMask)

Clears the PMC event flags state.

**Parameters**

- statusMask – : A bitmask of event flags that are to be cleared.

uint32\_t POWER\_GetEventFlags(void)

Get the PMC event flags state.

**Returns**

PMC FLAGS register value

void POWER\_EnableInterrupts(uint32\_t interruptMask)

Enable the PMC interrupt requests.

**Parameters**

- interruptMask – : A bitmask of of interrupts to enable.

void POWER\_DisableInterrupts(uint32\_t interruptMask)

Disable the PMC interrupt requests.

**Parameters**

- interruptMask – : A bitmask of of interrupts to disable.

void POWER\_SetAnalogBuffer(bool enable)

Set the PMC analog buffer for references or ATX2.

**Parameters**

- enable – : Set to true to enable analog buffer for references or ATX2, false to disable.

static inline uint32\_t POWER\_GetPmicMode(*pmic\_mode\_reg\_t* reg)

Get PMIC\_MODE pins configure value.

#### Parameters

- reg – : PDSLEEPCFG0 or PDRUNCFG0 register offset

#### Returns

PMIC\_MODE pins value in PDSLEEPCFG0

static inline *body\_bias\_mode\_t* POWER\_GetBodyBiasMode(*pmic\_mode\_reg\_t* reg)

Get RBB/FBB bit value.

#### Parameters

- reg – : PDSLEEPCFG0 or PDRUNCFG0 register offset

#### Returns

Current body bias mode

void POWER\_SetPadVolRange(const *power\_pad\_vrange\_t* \*config)

Configure pad voltage level. Wide voltage range cost more power due to enabled voltage detector.

NOTE: BE CAUTIOUS TO CALL THIS API. IF THE PAD SUPPLY IS BEYOND THE SET RANGE, SILICON MIGHT BE DAMAGED.

#### Parameters

- config – pad voltage range configuration.

AT\_QUICKACCESS\_SECTION\_CODE (void POWER\_EnterRbb(void))

PMC Enter Rbb mode function call.

PMC exit Rbb & Fbb mode function call.

PMC Enter Fbb mode function call.

bool POWER\_SetLdoVoltageForFreq(*power\_part\_temp\_range\_t* tempRange,  
*power\_volt\_op\_range\_t* voltOpRange, uint32\_t cm33Freq,  
uint32\_t dspFreq)

Deprecated and replaced by POWER\_SetVoltageForFreq()! PMC Set Ldo volatage for particular frequency. NOTE: The API is only valid when MAINPLLCLKDIV[7:0] and DSPPLLCLKDIV[7:0] are 0. If LVD falling trip voltage is higher than the required core voltage for particular frequency, LVD voltage will be decreased to safe level to avoid unexpected LVD reset or interrupt event.

#### Parameters

- tempRange – : part temperature range
- voltOpRange – : voltage operation range.
- cm33Freq – : CM33 CPU clock frequency value
- dspFreq – : DSP CPU clock frequency value

#### Returns

true for success and false for CPU frequency out of specified voltOpRange.

void POWER\_SetVddCoreSupplySrc(*power\_vddcore\_src\_t* src)

Set VDDCORE supply source, PMIC or on-chip regulator.

#### Parameters

- src – : power\_vddcore\_src\_t, VDDCore supply source

void POWER\_SetPmicCoreSupplyFunc(*power\_vddcore\_set\_func\_t* func)

Set the core supply setting function if PMIC is used. The function is not needed and ignored when using the onchip regulator to supply VDDCORE.

#### Parameters

- func – : *power\_vddcore\_set\_func\_t*, the PMIC core supply voltage set function.

bool POWER\_SetVoltageForFreq(*power\_part\_temp\_range\_t* tempRange, *power\_volt\_op\_range\_t* voltOpRange, *uint32\_t* cm33Freq, *uint32\_t* dspFreq, *uint32\_t* mini\_volt)

PMC Set volatage for particular frequency with given minimum value. POWER\_SetVddCoreSupplySrc should be called in advance to tell power driver the supply source. If PMIC is used, the VDDCORE setting function should be set by POWER\_SetPmicCoreSupplyFunc before this API is called. NOTE: The API is only valid when MAINPLLCLKDIV[7:0] and DSPPLLCLKDIV[7:0] are 0. If LVD falling trip voltage is higher than therequired core voltage for particular frequency, LVD voltage will be decreased to safe level to avoid unexpected LVD reset or interrupt event.

#### Parameters

- tempRange – : part temperature range
- voltOpRange – : voltage operation range.
- cm33Freq – : CM33 CPU clock frequency value
- dspFreq – : DSP CPU clock frequency value
- mini\_volt – : minimum voltage in millivolt(mV) for VDDCORE. Should <= 1130mV, 0 means use the core frequency to calculate voltage.

#### Returns

true for success and false for CPU frequency out of specified voltOpRange or failed to set voltage.

void POWER\_SetLvdFallingTripVoltage(*power\_lvd\_falling\_trip\_vol\_val\_t* volt)

Set vddcore low voltage detection falling trip voltage.

#### Parameters

- volt – target LVD voltage to set.

*power\_lvd\_falling\_trip\_vol\_val\_t* POWER\_GetLvdFallingTripVoltage(void)

Get current vddcore low voltage detection falling trip voltage.

#### Returns

Current LVD voltage.

void POWER\_DisableLVD(void)

Disable low voltage detection, no reset or interrupt is triggered when vddcore voltage drops below threshold. NOTE: This API is for internal use only. Application should not touch it.

void POWER\_RestoreLVD(void)

Restore low voltage detection setting. NOTE: This API is for internal use only. Application should not touch it.

void POWER\_SetPmicMode(*uint32\_t* mode, *pmic\_mode\_reg\_t* reg)

Set PMIC\_MODE pins configure value.

#### Parameters

- mode – : PMIC MODE pin value
- reg – : PDSLEEPCFG0 or PDRUNCFG0 register offset



**Returns**

PMIC\_MODE pins value in PDSLEEPCFG0

void POWER\_EnterSleep(void)

Configures and enters in SLEEP low power mode.

AT\_QUICKACCESS\_SECTION\_CODE (void POWER\_EnterDeepSleep(const uint32\_t exclude\_from\_pd[4]))

PMC Deep Sleep function call.

**Parameters**

- exclude\_from\_pd – Bit mask of the PDRUNCFG0 ~ PDRUNCFG3 that needs to be powered on during Deep Sleep mode selected.

void POWER\_EnterDeepPowerDown(const uint32\_t exclude\_from\_pd[4])

PMC Deep Power Down function call.

**Parameters**

- exclude\_from\_pd – Bit mask of the PDRUNCFG0 ~ PDRUNCFG3 that needs to be powered on during Deep Power Down mode selected.

void POWER\_EnterFullDeepPowerDown(const uint32\_t exclude\_from\_pd[4])

PMC Full Deep Power Down function call.

**Parameters**

- exclude\_from\_pd – Bit mask of the PDRUNCFG0 ~ PDRUNCFG3 that needs to be powered on during Full Deep Power Down mode selected.

void POWER\_EnterPowerMode(*power\_mode\_cfg\_t* mode, const uint32\_t exclude\_from\_pd[4])

Power Library API to enter different power mode.

**Parameters**

- mode – Power mode to enter.
- exclude\_from\_pd – Bit mask of the PDRUNCFG0 ~ PDRUNCFG3 that needs to be powered on during power mode selected.

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.

uint32\_t POWER\_GetLibVersion(void)

Power Library API to return the library version.

### Returns

version number of the power library

FSL\_POWER\_DRIVER\_VERSION

power driver version 2.5.0.

MAKE\_PD\_BITS(reg, slot)

SYSCTL0\_PDRCFGSET\_REG(x)

SYSCTL0\_PDRCFGCLR\_REG(x)

PDRCFG0

PDRCFG1

PDRCFG2

PDRCFG3

PMIC\_VDDCORE\_RECOVERY\_TIME\_IGNORE

PMIC is used but vddcore supply is always above LVD threshold.

SYSCTL0\_TUPLE\_REG(reg)

PMIC mode pin configuration API parameter.

uint32\_t Vdde0Range

VDDE0 voltage range for VDDIO\_0. power\_pad\_vrange\_val\_t

uint32\_t Vdde1Range

VDDE1 voltage range for VDDIO\_1. power\_pad\_vrange\_val\_t

uint32\_t Vdde2Range

VDDE2 voltage range for VDDIO\_2. power\_pad\_vrange\_val\_t

uint32\_t \_\_pad0\_\_

Reserved.

struct \_\_power\_pad\_vrange

#include <fsl\_power.h> pad voltage range configuration.

## 2.54 POWERQUAD: PowerQuad hardware accelerator

void PQ\_GetDefaultConfig(pq\_config\_t \*config)

Get default configuration.

This function initializes the POWERQUAD configuration structure to a default value. FORMAT register field definitions Bits[15:8] scaler (for scaled 'q31' formats) Bits[5:4] external format. 00b=q15, 01b=q31, 10b=float Bits[1:0] internal format. 00b=q15, 01b=q31, 10b=float

```
POWERQUAD->INAFORMAT = (config->inputAPrescale « 8U) | (config->inputAFormat « 4U)
| config->machineFormat
```

For all Powerquad operations internal format must be float (with the only exception being the FFT related functions, ie FFT/IFFT/DCT/IDCT which must be set to q31). The default values are: config->inputAFormat = kPQ\_Float; config->inputAPrescale = 0; config->inputBFormat = kPQ\_Float; config->inputBPrescale = 0; config->outputFormat = kPQ\_Float; config->outputPrescale = 0; config->tmpFormat = kPQ\_Float; config->tmpPrescale = 0; config->machineFormat = kPQ\_Float; config->tmpBase = 0xE0000000;

#### Parameters

- config – Pointer to “pq\_config\_t” structure.

```
void PQ_SetConfig(POWERQUAD_Type *base, const pq_config_t *config)
```

Set configuration with format/prescale.

#### Parameters

- base – POWERQUAD peripheral base address
- config – Pointer to “pq\_config\_t” structure.

```
static inline void PQ_SetCoproprocessorScaler(POWERQUAD_Type *base, const pq_prescale_t
                                             *prescale)
```

set coprocessor scaler for coprocessor instructions, this function is used to set output saturation and scaling for input/output.

#### Parameters

- base – POWERQUAD peripheral base address
- prescale – Pointer to “pq\_prescale\_t” structure.

```
void PQ_Init(POWERQUAD_Type *base)
```

Initializes the POWERQUAD module.

#### Parameters

- base – POWERQUAD peripheral base address.

```
void PQ_Deinit(POWERQUAD_Type *base)
```

De-initializes the POWERQUAD module.

#### Parameters

- base – POWERQUAD peripheral base address.

```
void PQ_SetFormat(POWERQUAD_Type *base, pq_computationengine_t engine, pq_format_t
                  format)
```

Set format for non-coprocessor instructions.

#### Parameters

- base – POWERQUAD peripheral base address
- engine – Computation engine
- format – Data format

```
static inline void PQ_WaitDone(POWERQUAD_Type *base)
```

Wait for the completion.

#### Parameters

- base – POWERQUAD peripheral base address

static inline void PQ\_LnF32(float \*pSrc, float \*pDst)

Processing function for the floating-point natural log.

**Parameters**

- \*pSrc – points to the block of input data. The range of the input value is (0 +INFINITY).
- \*pDst – points to the block of output data

static inline void PQ\_InvF32(float \*pSrc, float \*pDst)

Processing function for the floating-point reciprocal.

**Parameters**

- \*pSrc – points to the block of input data. The range of the input value is non-zero.
- \*pDst – points to the block of output data

static inline void PQ\_SqrtF32(float \*pSrc, float \*pDst)

Processing function for the floating-point square-root.

**Parameters**

- \*pSrc – points to the block of input data. The range of the input value is [0 +INFINITY).
- \*pDst – points to the block of output data

static inline void PQ\_InvSqrtF32(float \*pSrc, float \*pDst)

Processing function for the floating-point inverse square-root.

**Parameters**

- \*pSrc – points to the block of input data. The range of the input value is (0 +INFINITY).
- \*pDst – points to the block of output data

static inline void PQ\_EtoxF32(float \*pSrc, float \*pDst)

Processing function for the floating-point natural exponent.

**Parameters**

- \*pSrc – points to the block of input data. The range of the input value is (-INFINITY +INFINITY).
- \*pDst – points to the block of output data

static inline void PQ\_EtonxF32(float \*pSrc, float \*pDst)

Processing function for the floating-point natural exponent with negative parameter.

**Parameters**

- \*pSrc – points to the block of input data. The range of the input value is (-INFINITY +INFINITY).
- \*pDst – points to the block of output data

static inline void PQ\_SinF32(float \*pSrc, float \*pDst)

Processing function for the floating-point sine.

**Parameters**

- \*pSrc – points to the block of input data. The input value is in radians, the range is (-INFINITY +INFINITY).
- \*pDst – points to the block of output data

static inline void PQ\_CosF32(float \*pSrc, float \*pDst)

Processing function for the floating-point cosine.

**Parameters**

- \*pSrc – points to the block of input data. The input value is in radians, the range is (-INFINITY +INFINITY).
- \*pDst – points to the block of output data

static inline void PQ\_BiquadF32(float \*pSrc, float \*pDst)

Processing function for the floating-point biquad.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data

static inline void PQ\_DivF32(float \*x1, float \*x2, float \*pDst)

Processing function for the floating-point division.

Get x1 / x2.

**Parameters**

- x1 – x1
- x2 – x2
- \*pDst – points to the block of output data

static inline void PQ\_Biquad1F32(float \*pSrc, float \*pDst)

Processing function for the floating-point biquad.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data

static inline int32\_t PQ\_LnFixed(int32\_t val)

Processing function for the fixed natural log.

**Parameters**

- val – value to be calculated. The range of the input value is (0 +INFINITY).

**Returns**

returns ln(val).

static inline int32\_t PQ\_InvFixed(int32\_t val)

Processing function for the fixed reciprocal.

**Parameters**

- val – value to be calculated. The range of the input value is non-zero.

**Returns**

returns inv(val).

static inline uint32\_t PQ\_SqrtFixed(uint32\_t val)

Processing function for the fixed square-root.

**Parameters**

- val – value to be calculated. The range of the input value is [0 +INFINITY).

**Returns**

returns sqrt(val).

static inline int32\_t PQ\_InvSqrtFixed(int32\_t val)

Processing function for the fixed inverse square-root.

**Parameters**

- val – value to be calculated. The range of the input value is (0 +INFINITY).

**Returns**

returns  $1/\sqrt{\text{val}}$ .

static inline int32\_t PQ\_EtoxFixed(int32\_t val)

Processing function for the Fixed natural exponent.

**Parameters**

- val – value to be calculated. The range of the input value is (-INFINITY +INFINITY).

**Returns**

returns  $\text{etox}^{\text{val}}$ .

static inline int32\_t PQ\_EtonxFixed(int32\_t val)

Processing function for the fixed natural exponent with negative parameter.

**Parameters**

- val – value to be calculated. The range of the input value is (-INFINITY +INFINITY).

**Returns**

returns  $\text{etox}^{\text{val}}$ .

static inline int32\_t PQ\_SinQ31(int32\_t val)

Processing function for the fixed sine.

**Parameters**

- val – value to be calculated. The input value is [-1, 1] in Q31 format, which means [-pi, pi].

**Returns**

returns  $\sin(\text{val})$ .

static inline int16\_t PQ\_SinQ15(int16\_t val)

Processing function for the fixed sine.

**Parameters**

- val – value to be calculated. The input value is [-1, 1] in Q15 format, which means [-pi, pi].

**Returns**

returns  $\sin(\text{val})$ .

static inline int32\_t PQ\_CosQ31(int32\_t val)

Processing function for the fixed cosine.

**Parameters**

- val – value to be calculated. The input value is [-1, 1] in Q31 format, which means [-pi, pi].

**Returns**

returns  $\cos(\text{val})$ .

static inline int16\_t PQ\_CosQ15(int16\_t val)

Processing function for the fixed sine.

**Parameters**

- `val` – value to be calculated. The input value is [-1, 1] in Q15 format, which means [-pi, pi].

**Returns**

returns `sin(val)`.

`static inline int32_t PQ_BiquadFixed(int32_t val)`

Processing function for the fixed biquad.

**Parameters**

- `val` – value to be calculated

**Returns**

returns `biquad(val)`.

`void PQ_VectorLnF32(float *pSrc, float *pDst, int32_t length)`

Processing function for the floating-point vectorised natural log.

**Parameters**

- `*pSrc` – points to the block of input data
- `*pDst` – points to the block of output data
- `length` – the block of input data.

`void PQ_VectorInvF32(float *pSrc, float *pDst, int32_t length)`

Processing function for the floating-point vectorised reciprocal.

**Parameters**

- `*pSrc` – points to the block of input data
- `*pDst` – points to the block of output data
- `length` – the block of input data.

`void PQ_VectorSqrtF32(float *pSrc, float *pDst, int32_t length)`

Processing function for the floating-point vectorised square-root.

**Parameters**

- `*pSrc` – points to the block of input data
- `*pDst` – points to the block of output data
- `length` – the block of input data.

`void PQ_VectorInvSqrtF32(float *pSrc, float *pDst, int32_t length)`

Processing function for the floating-point vectorised inverse square-root.

**Parameters**

- `*pSrc` – points to the block of input data
- `*pDst` – points to the block of output data
- `length` – the block of input data.

`void PQ_VectorEtoxF32(float *pSrc, float *pDst, int32_t length)`

Processing function for the floating-point vectorised natural exponent.

**Parameters**

- `*pSrc` – points to the block of input data
- `*pDst` – points to the block of output data
- `length` – the block of input data.

void PQ\_VectorEtonxF32(float \*pSrc, float \*pDst, int32\_t length)

Processing function for the floating-point vectorised natural exponent with negative parameter.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block of input data.

void PQ\_VectorSinF32(float \*pSrc, float \*pDst, int32\_t length)

Processing function for the floating-point vectorised sine.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block of input data.

void PQ\_VectorCosF32(float \*pSrc, float \*pDst, int32\_t length)

Processing function for the floating-point vectorised cosine.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block of input data.

void PQ\_VectorLnFixed32(int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)

Processing function for the Q31 vectorised natural log.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block of input data.

void PQ\_VectorInvFixed32(int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)

Processing function for the Q31 vectorised reciprocal.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block of input data.

void PQ\_VectorSqrtFixed32(int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)

Processing function for the 32-bit integer vectorised square-root.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block of input data.

void PQ\_VectorInvSqrtFixed32(int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)

Processing function for the 32-bit integer vectorised inverse square-root.

**Parameters**



- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block of input data.

void PQ\_VectorEtoxFixed32(int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)

Processing function for the 32-bit integer vectorised natural exponent.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block of input data.

void PQ\_VectorEtonxFixed32(int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)

Processing function for the 32-bit integer vectorised natural exponent with negative parameter.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block of input data.

void PQ\_VectorSinQ15(int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)

Processing function for the Q15 vectorised sine.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block of input data.

void PQ\_VectorCosQ15(int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)

Processing function for the Q15 vectorised cosine.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block of input data.

void PQ\_VectorSinQ31(int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)

Processing function for the Q31 vectorised sine.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block of input data.

void PQ\_VectorCosQ31(int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)

Processing function for the Q31 vectorised cosine.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block of input data.

void PQ\_VectorLnFixed16(int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)

Processing function for the 16-bit integer vectorised natural log.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block of input data.

void PQ\_VectorInvFixed16(int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)

Processing function for the 16-bit integer vectorised reciprocal.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block of input data.

void PQ\_VectorSqrtFixed16(int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)

Processing function for the 16-bit integer vectorised square-root.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block of input data.

void PQ\_VectorInvSqrtFixed16(int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)

Processing function for the 16-bit integer vectorised inverse square-root.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block of input data.

void PQ\_VectorEtoxFixed16(int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)

Processing function for the 16-bit integer vectorised natural exponent.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block of input data.

void PQ\_VectorEtonxFixed16(int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)

Processing function for the 16-bit integer vectorised natural exponent with negative parameter.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block of input data.

void PQ\_VectorBiquadDf2F32(float \*pSrc, float \*pDst, int32\_t length)

Processing function for the floating-point vectorised biquad direct form II.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block size of input data.

void PQ\_VectorBiquadDf2Fixed32(int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)

Processing function for the 32-bit integer vectorised biquad direct form II.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block size of input data

void PQ\_VectorBiquadDf2Fixed16(int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)

Processing function for the 16-bit integer vectorised biquad direct form II.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block size of input data

void PQ\_VectorBiquadCascadeDf2F32(float \*pSrc, float \*pDst, int32\_t length)

Processing function for the floating-point vectorised biquad direct form II.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block size of input data

void PQ\_VectorBiquadCascadeDf2Fixed32(int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)

Processing function for the 32-bit integer vectorised biquad direct form II.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block size of input data

void PQ\_VectorBiquadCascadeDf2Fixed16(int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)

Processing function for the 16-bit integer vectorised biquad direct form II.

**Parameters**

- \*pSrc – points to the block of input data
- \*pDst – points to the block of output data
- length – the block size of input data

int32\_t PQ\_ArctanFixed(POWERQUAD\_Type \*base, int32\_t x, int32\_t y, *pq\_cordic\_iter\_t* iteration)

Processing function for the fixed inverse trigonometric.

Get the inverse tangent, the behavior is like c function atan.

---

**Note:** The sum of x and y should not exceed the range of int32\_t.

---

---

**Note:** Larger input number gets higher output accuracy, for example the `arctan(0.5)`, the result of `PQ_ArctanFixed(POWERQUAD, 100000, 200000, kPQ_Iteration_24)` is more accurate than `PQ_ArctanFixed(POWERQUAD, 1, 2, kPQ_Iteration_24)`.

---

#### Parameters

- `base` – POWERQUAD peripheral base address
- `x` – value of opposite
- `y` – value of adjacent
- `iteration` – iteration times

#### Returns

The return value is in the range of  $-2^{26}$  to  $2^{26}$ , which means  $-\pi/2$  to  $\pi/2$ .

`int32_t PQ_ArctanhFixed(POWERQUAD_Type *base, int32_t x, int32_t y, pq_cordic_iter_t iteration)`

Processing function for the fixed inverse trigonometric.

---

**Note:** The sum of `x` and `y` should not exceed the range of `int32_t`.

---

---

**Note:** Larger input number gets higher output accuracy, for example the `arctanh(0.5)`, the result of `PQ_ArctanhFixed(POWERQUAD, 100000, 200000, kPQ_Iteration_24)` is more accurate than `PQ_ArctanhFixed(POWERQUAD, 1, 2, kPQ_Iteration_24)`.

---

#### Parameters

- `base` – POWERQUAD peripheral base address
- `x` – value of opposite
- `y` – value of adjacent
- `iteration` – iteration times

#### Returns

The return value is radians,  $2^{27}$  means  $\pi$ . The range is  $-1.118$  to  $1.118$  radians.

`int32_t PQ_Arctan2Fixed(POWERQUAD_Type *base, int32_t x, int32_t y, pq_cordic_iter_t iteration)`

Processing function for the fixed inverse trigonometric.

Get the inverse tangent, it calculates the angle in radians for the quadrant. The behavior is like `c` function `atan2`.

---

**Note:** The sum of `x` and `y` should not exceed the range of `int32_t`.

---

---

**Note:** Larger input number gets higher output accuracy, for example the `arctan(0.5)`, the result of `PQ_Arctan2Fixed(POWERQUAD, 100000, 200000, kPQ_Iteration_24)` is more accurate than `PQ_Arctan2Fixed(POWERQUAD, 1, 2, kPQ_Iteration_24)`.

---

#### Parameters

- base – POWERQUAD peripheral base address
- x – value of opposite
- y – value of adjacent
- iteration – iteration times

**Returns**

The return value is in the range of  $-2^{27}$  to  $2^{27}$ , which means  $-\pi$  to  $\pi$ .

```
static inline int32_t PQ_Biquad1Fixed(int32_t val)
```

Processing function for the fixed biquad.

**Parameters**

- val – value to be calculated

**Returns**

returns biquad(val).

```
void PQ_TransformCFFT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)
```

Processing function for the complex FFT.

**Parameters**

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

```
void PQ_TransformRFFT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)
```

Processing function for the real FFT.

**Parameters**

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

```
void PQ_TransformIFFT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)
```

Processing function for the inverse complex FFT.

**Parameters**

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

```
void PQ_TransformCDCT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)
```

Processing function for the complex DCT.

**Parameters**

- base – POWERQUAD peripheral base address
- length – number of input samples

- pData – input data
- pResult – output data.

```
void PQ_TransformRDCT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)
```

Processing function for the real DCT.

#### Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

```
void PQ_TransformIDCT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)
```

Processing function for the inverse complex DCT.

#### Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

```
void PQ_BiquadBackUpInternalState(POWERQUAD_Type *base, int32_t biquad_num, pq_biquad_state_t *state)
```

Processing function for backup biquad context.

#### Parameters

- base – POWERQUAD peripheral base address
- biquad\_num – biquad side
- state – point to states.

```
void PQ_BiquadRestoreInternalState(POWERQUAD_Type *base, int32_t biquad_num, pq_biquad_state_t *state)
```

Processing function for restore biquad context.

#### Parameters

- base – POWERQUAD peripheral base address
- biquad\_num – biquad side
- state – point to states.

```
void PQ_BiquadCascadeDf2Init(pq_biquad_cascade_df2_instance *S, uint8_t numStages, pq_biquad_state_t *pState)
```

Initialization function for the direct form II Biquad cascade filter.

#### Parameters

- \*S – **[inout]** points to an instance of the filter data structure.
- numStages – **[in]** number of 2nd order stages in the filter.
- \*pState – **[in]** points to the state buffer.

```
void PQ_BiquadCascadeDf2F32(const pq_biquad_cascade_df2_instance *S, float *pSrc, float *pDst, uint32_t blockSize)
```

Processing function for the floating-point direct form II Biquad cascade filter.

**Parameters**

- \*S – **[in]** points to an instance of the filter data structure.
- \*pSrc – **[in]** points to the block of input data.
- \*pDst – **[out]** points to the block of output data
- blockSize – **[in]** number of samples to process.

```
void PQ_BiquadCascadeDf2Fixed32(const pq_biquad_cascade_df2_instance *S, int32_t *pSrc,  
                                int32_t *pDst, uint32_t blockSize)
```

Processing function for the Q31 direct form II Biquad cascade filter.

**Parameters**

- \*S – **[in]** points to an instance of the filter data structure.
- \*pSrc – **[in]** points to the block of input data.
- \*pDst – **[out]** points to the block of output data
- blockSize – **[in]** number of samples to process.

```
void PQ_BiquadCascadeDf2Fixed16(const pq_biquad_cascade_df2_instance *S, int16_t *pSrc,  
                                int16_t *pDst, uint32_t blockSize)
```

Processing function for the Q15 direct form II Biquad cascade filter.

**Parameters**

- \*S – **[in]** points to an instance of the filter data structure.
- \*pSrc – **[in]** points to the block of input data.
- \*pDst – **[out]** points to the block of output data
- blockSize – **[in]** number of samples to process.

```
void PQ_FIR(POWERQUAD_Type *base, const void *pAData, int32_t ALength, const void  
            *pBData, int32_t BLength, void *pResult, uint32_t opType)
```

Processing function for the FIR.

**Parameters**

- base – POWERQUAD peripheral base address
- pAData – the first input sequence
- ALength – number of the first input sequence
- pBData – the second input sequence
- BLength – number of the second input sequence
- pResult – array for the output data
- opType – operation type, could be PQ\_FIR\_FIR, PQ\_FIR\_CONVOLUTION, PQ\_FIR\_CORRELATION.

```
void PQ_FIRIncrement(POWERQUAD_Type *base, int32_t ALength, int32_t BLength, int32_t  
                    xOffset)
```

Processing function for the incremental FIR. This function can be used after pq\_fir() for incremental FIR operation when new x data are available.

**Parameters**

- base – POWERQUAD peripheral base address
- ALength – number of input samples
- BLength – number of taps

- xOffset – offset for number of input samples

```
void PQ_MatrixAddition(POWERQUAD_Type *base, uint32_t length, void *pAData, void  
                      *pBData, void *pResult)
```

Processing function for the matrix addition.

#### Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD\_MAKE\_MATRIX\_LEN.
- pAData – input matrix A
- pBData – input matrix B
- pResult – array for the output data.

```
void PQ_MatrixSubtraction(POWERQUAD_Type *base, uint32_t length, void *pAData, void  
                          *pBData, void *pResult)
```

Processing function for the matrix subtraction.

#### Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD\_MAKE\_MATRIX\_LEN.
- pAData – input matrix A
- pBData – input matrix B
- pResult – array for the output data.

```
void PQ_MatrixMultiplication(POWERQUAD_Type *base, uint32_t length, void *pAData, void  
                             *pBData, void *pResult)
```

Processing function for the matrix multiplication.

#### Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD\_MAKE\_MATRIX\_LEN.
- pAData – input matrix A
- pBData – input matrix B
- pResult – array for the output data.

```
void PQ_MatrixProduct(POWERQUAD_Type *base, uint32_t length, void *pAData, void *pBData,  
                      void *pResult)
```

Processing function for the matrix product.

#### Parameters

- base – POWERQUAD peripheral base address



- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD\_MAKE\_MATRIX\_LEN.
- pAData – input matrix A
- pBData – input matrix B
- pResult – array for the output data.

```
void PQ_VectorDotProduct(POWERQUAD_Type *base, uint32_t length, void *pAData, void *pBData, void *pResult)
```

Processing function for the vector dot product.

#### Parameters

- base – POWERQUAD peripheral base address
- length – length of vector
- pAData – input vector A
- pBData – input vector B
- pResult – array for the output data.

```
void PQ_MatrixInversion(POWERQUAD_Type *base, uint32_t length, void *pData, void *pTmpData, void *pResult)
```

Processing function for the matrix inverse.

#### Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD\_MAKE\_MATRIX\_LEN.
- pData – input matrix
- pTmpData – input temporary matrix, pTmpData length not less than pData length and 1024 words is sufficient for the largest supported matrix.
- pResult – array for the output data, round down for fixed point.

```
void PQ_MatrixTranspose(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)
```

Processing function for the matrix transpose.

#### Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD\_MAKE\_MATRIX\_LEN.
- pData – input matrix
- pResult – array for the output data.

```
void PQ_MatrixScale(POWERQUAD_Type *base, uint32_t length, float misc, const void *pData, void *pResult)
```

Processing function for the matrix scale.

#### Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD\_MAKE\_MATRIX\_LEN.
- misc – scaling parameters
- pData – input matrix
- pResult – array for the output data.

FSL\_POWERQUAD\_DRIVER\_VERSION  
Version.

enum pq\_computationengine\_t  
powerquad computation engine

*Values:*

enumerator kPQ\_CP\_PQ  
Math engine.

enumerator kPQ\_CP\_MTX  
Matrix engine.

enumerator kPQ\_CP\_FFT  
FFT engine.

enumerator kPQ\_CP\_FIR  
FIR engine.

enumerator kPQ\_CP\_CORDIC  
CORDIC engine.

enum pq\_format\_t  
powerquad data structure format type

*Values:*

enumerator kPQ\_16Bit  
Int16 Fixed point.

enumerator kPQ\_32Bit  
Int32 Fixed point.

enumerator kPQ\_Float  
Float point.

enum pq\_cordic\_iter\_t  
CORDIC iteration.

*Values:*

enumerator kPQ\_Iteration\_8  
Iterate 8 times.

enumerator kPQ\_Iteration\_16  
Iterate 16 times.

enumerator kPQ\_Iteration\_24  
Iterate 24 times.

```
typedef struct _pq_biquad_param pq_biquad_param_t
```

Struct to save biquad parameters.

```
typedef struct _pq_biquad_state pq_biquad_state_t
```

Struct to save biquad state.

```
typedef union _pq_float pq_float_t
```

Conversion between integer and float type.

PQ\_VectorBiquadDf2F32

PQ\_VectorBiquadDf2Fixed32

PQ\_VectorBiquadDf2Fixed16

PQ\_VectorBiquadCascadeDf2F32

PQ\_VectorBiquadCascadeDf2Fixed32

PQ\_VectorBiquadCascadeDf2Fixed16

PQ\_Vector8BiquadDf2CascadeF32

PQ\_Vector8BiquadDf2CascadeFixed32

PQ\_Vector8BiquadDf2CascadeFixed16

PQ\_FLOAT32

PQ\_FIXEDPT

CP\_PQ

CP\_MTX

CP\_FFT

CP\_FIR

CP\_CORDIC

PQ\_TRANS

PQ\_TRIG

PQ\_BIQUAD

PQ\_TRANS\_FIXED

PQ\_TRIG\_FIXED

PQ\_BIQUAD\_FIXED

PQ\_INV

PQ\_LN

PQ\_SQRT

PQ\_INVSQRT

PQ\_ETOX

PQ\_ETONX

PQ\_DIV  
PQ\_SIN  
PQ\_COS  
PQ\_BIQ0\_CALC  
PQ\_BIQ1\_CALC  
PQ\_COMP0\_ONLY  
PQ\_COMP1\_ONLY  
CORDIC\_ITER(x)  
CORDIC\_MIU(x)  
CORDIC\_T(x)  
CORDIC\_ARCTAN  
CORDIC\_ARCTANH  
INST\_BUSY  
PQ\_ERRSTAT\_OVERFLOW  
PQ\_ERRSTAT\_NAN  
PQ\_ERRSTAT\_FIXEDOVERFLOW  
PQ\_ERRSTAT\_UNDERFLOW  
PQ\_TRANS\_CFFT  
PQ\_TRANS\_IFFT  
PQ\_TRANS\_CDCT  
PQ\_TRANS\_IDCT  
PQ\_TRANS\_RFFT  
PQ\_TRANS\_RDCT  
PQ\_MTX\_SCALE  
PQ\_MTX\_MULT  
PQ\_MTX\_ADD  
PQ\_MTX\_INV  
PQ\_MTX\_PROD  
PQ\_MTX\_SUB  
PQ\_VEC\_DOTP  
PQ\_MTX\_TRAN  
PQ\_FIR\_FIR  
PQ\_FIR\_CONVOLUTION

PQ\_FIR\_CORRELATION

PQ\_FIR\_INCREMENTAL

\_\_pq\_ln0(x)

\_\_pq\_inv0(x)

\_\_pq\_sqrt0(x)

\_\_pq\_invsqrt0(x)

\_\_pq\_etox0(x)

\_\_pq\_etonx0(x)

\_\_pq\_sin0(x)

\_\_pq\_cos0(x)

\_\_pq\_biquad0(x)

\_\_pq\_ln\_fx0(x)

\_\_pq\_inv\_fx0(x)

\_\_pq\_sqrt\_fx0(x)

\_\_pq\_invsqrt\_fx0(x)

\_\_pq\_etox\_fx0(x)

\_\_pq\_etonx\_fx0(x)

\_\_pq\_sin\_fx0(x)

\_\_pq\_cos\_fx0(x)

\_\_pq\_biquad0\_fx(x)

\_\_pq\_div0(x)

\_\_pq\_div1(x)

\_\_pq\_ln1(x)

\_\_pq\_inv1(x)

\_\_pq\_sqrt1(x)

\_\_pq\_invsqrt1(x)

\_\_pq\_etox1(x)

\_\_pq\_etonx1(x)

\_\_pq\_sin1(x)

\_\_pq\_cos1(x)

\_\_pq\_biquad1(x)

\_\_pq\_ln\_fx1(x)

\_\_pq\_inv\_fx1(x)

`_pq_sqrt_fx1(x)`

`_pq_invsqrt_fx1(x)`

`_pq_etox_fx1(x)`

`_pq_etonx_fx1(x)`

`_pq_sin_fx1(x)`

`_pq_cos_fx1(x)`

`_pq_biquad1_fx(x)`

`_pq_readMult0()`

`_pq_readAdd0()`

`_pq_readMult1()`

`_pq_readAdd1()`

`_pq_readMult0_fx()`

`_pq_readAdd0_fx()`

`_pq_readMult1_fx()`

`_pq_readAdd1_fx()`

`PQ_LN_INF`

Parameter used for vector  $\ln(x)$

`PQ_INV_INF`

Parameter used for vector  $1/x$

`PQ_SQRT_INF`

Parameter used for vector  $\sqrt{x}$

`PQ_ISQRT_INF`

Parameter used for vector  $1/\sqrt{x}$

`PQ_ETOX_INF`

Parameter used for vector  $e^x$

`PQ_ETONX_INF`

Parameter used for vector  $e^{-x}$

`PQ_SIN_INF`

Parameter used for vector  $\sin(x)$

`PQ_COS_INF`

Parameter used for vector  $\cos(x)$

`PQ_RUN_OPCODE_R3_R2(BATCH_OPCODE, BATCH_MACHINE)`

`PQ_RUN_OPCODE_R5_R4(BATCH_OPCODE, BATCH_MACHINE)`

`PQ_RUN_OPCODE_R7_R6(BATCH_OPCODE, BATCH_MACHINE)`

`PQ_Vector8_FP(middle, last, BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)`

`PQ_RUN_OPCODE_R2_R3(BATCH_OPCODE, BATCH_MACHINE)`

`PQ_RUN_OPCODE_R4_R5(BATCH_OPCODE, BATCH_MACHINE)`

PQ\_RUN\_OPCODE\_R6\_R7(BATCH\_OPCODE, BATCH\_MACHINE)

PQ\_Vector8\_FX(middle, last, BATCH\_OPCODE, DOUBLE\_READ\_ADDERS, BATCH\_MACHINE)

PQ\_Initiate\_Vector\_Func(pSrc, pDst)

Start 32-bit data vector calculation.

Start the vector calculation, the input data could be float, int32\_t or Q31.

**Parameters**

- pSrc – Pointer to the source data.
- pDst – Pointer to the destination data.

PQ\_End\_Vector\_Func()

End vector calculation.

This function should be called after vector calculation.

PQ\_StartVector(PSRC, PDST, LENGTH)

Start 32-bit data vector calculation.

Start the vector calculation, the input data could be float, int32\_t or Q31.

**Parameters**

- PSRC – Pointer to the source data.
- PDST – Pointer to the destination data.
- LENGTH – Number of the data, must be multiple of 8.

PQ\_StartVectorFixed16(PSRC, PDST, LENGTH)

Start 16-bit data vector calculation.

Start the vector calculation, the input data could be int16\_t. This function should be use with PQ\_Vector8Fixed16.

**Parameters**

- PSRC – Pointer to the source data.
- PDST – Pointer to the destination data.
- LENGTH – Number of the data, must be multiple of 8.

PQ\_StartVectorQ15(PSRC, PDST, LENGTH)

Start Q15-bit data vector calculation.

Start the vector calculation, the input data could be Q15. This function should be use with PQ\_Vector8Q15. This function is dedicate for SinQ15/CosQ15 vector calculation. Because PowerQuad only supports Q31 Sin/Cos fixed function, so the input Q15 data is left shift 16 bits first, after Q31 calculation, the output data is right shift 16 bits.

**Parameters**

- PSRC – Pointer to the source data.
- PDST – Pointer to the destination data.
- LENGTH – Number of the data, must be multiple of 8.

PQ\_EndVector()

End vector calculation.

This function should be called after vector calculation.

PQ\_Vector8F32(BATCH\_OPCODE, DOUBLE\_READ\_ADDERS, BATCH\_MACHINE)

Float data vector calculation.

Float data vector calculation, the input data should be float. The parameter could be PQ\_LN\_INF, PQ\_INV\_INF, PQ\_SQRT\_INF, PQ\_ISQRT\_INF, PQ\_ETOX\_INF, PQ\_ETONX\_INF. For example, to calculate sqrt of a vector, use like this:

```
#define VECTOR_LEN 8
float input[VECTOR_LEN] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0};
float output[VECTOR_LEN];

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8F32(PQ_SQRT_INF);
PQ_EndVector();
```

PQ\_Vector8Fixed32(BATCH\_OPCODE, DOUBLE\_READ\_ADDERS, BATCH\_MACHINE)

Fixed 32bits data vector calculation.

Float data vector calculation, the input data should be 32-bit integer. The parameter could be PQ\_LN\_INF, PQ\_INV\_INF, PQ\_SQRT\_INF, PQ\_ISQRT\_INF, PQ\_ETOX\_INF, PQ\_ETONX\_INF, PQ\_SIN\_INF, PQ\_COS\_INF. When this function is used for sin/cos calculation, the input data should be in the format Q1.31. For example, to calculate sqrt of a vector, use like this:

```
#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 4, 9, 16, 25, 36, 49, 64};
int32_t output[VECTOR_LEN];

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8F32(PQ_SQRT_INF);
PQ_EndVector();
```

PQ\_Vector8Fixed16(BATCH\_OPCODE, DOUBLE\_READ\_ADDERS, BATCH\_MACHINE)

Fixed 32bits data vector calculation.

Float data vector calculation, the input data should be 16-bit integer. The parameter could be PQ\_LN\_INF, PQ\_INV\_INF, PQ\_SQRT\_INF, PQ\_ISQRT\_INF, PQ\_ETOX\_INF, PQ\_ETONX\_INF. For example, to calculate sqrt of a vector, use like this:

```
#define VECTOR_LEN 8
int16_t input[VECTOR_LEN] = {1, 4, 9, 16, 25, 36, 49, 64};
int16_t output[VECTOR_LEN];

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8F32(PQ_SQRT_INF);
PQ_EndVector();
```

PQ\_Vector8Q15(BATCH\_OPCODE, DOUBLE\_READ\_ADDERS, BATCH\_MACHINE)

Q15 data vector calculation.

Q15 data vector calculation, this function should only be used for sin/cos Q15 calculation, and the coprocessor output prescaler must be set to 31 before this function. This function loads Q15 data and left shift 16 bits, calculate and right shift 16 bits, then stores to the output array. The input range -1 to 1 means -pi to pi. For example, to calculate sin of a vector, use like this:

```
#define VECTOR_LEN 8
int16_t input[VECTOR_LEN] = {...}
int16_t output[VECTOR_LEN];
const pq_prescale_t prescale =
{
    .inputPrescale = 0,
    .outputPrescale = 31,
```

(continues on next page)



(continued from previous page)

```

    .outputSaturate = 0
};

PQ_SetCoproprocessorScaler(POWERQUAD, const pq_prescale_t *prescale);

PQ_StartVectorQ15(pSrc, pDst, length);
PQ_Vector8Q15(PQ_SQRT_INF);
PQ_EndVector();

```

PQ\_DF2\_Vector8\_FP(middle, last)

Float data vector biquad direct form II calculation.

Biquad filter, the input and output data are float data. Biquad side 0 is used. Example:

```

#define VECTOR_LEN 16
float input[VECTOR_LEN] = {1024.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
float output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_Initiate_Vector_Func(pSrc,pDst);
PQ_DF2_Vector8_FP(false,false);
PQ_DF2_Vector8_FP(true,true);
PQ_End_Vector_Func();

```

PQ\_DF2\_Vector8\_FX(middle, last)

Fixed data vector biquad direct form II calculation.

Biquad filter, the input and output data are fixed data. Biquad side 0 is used. Example:

```

#define VECTOR_LEN 16
int32_t input[VECTOR_LEN] = {1024, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_Initiate_Vector_Func(pSrc,pDst);
PQ_DF2_Vector8_FX(false,false);
PQ_DF2_Vector8_FX(true,true);
PQ_End_Vector_Func();

```

PQ\_Vector8BiquadDf2F32()

Float data vector biquad direct form II calculation.

Biquad filter, the input and output data are float data. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 8
float input[VECTOR_LEN] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0};
float output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2F32();
PQ_EndVector();
```

PQ\_Vector8BiquadDf2Fixed32()

Fixed 32-bit data vector biquad direct form II calculation.

Biquad filter, the input and output data are Q31 or 32-bit integer. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2Fixed32();
PQ_EndVector();
```

PQ\_Vector8BiquadDf2Fixed16()

Fixed 16-bit data vector biquad direct form II calculation.

Biquad filter, the input and output data are Q15 or 16-bit integer. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 8
int16_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int16_t output[VECTOR_LEN];
pq_biquad_state_t state =
```

(continues on next page)

(continued from previous page)

```

{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2Fixed16();
PQ_EndVector();

```

PQ\_DF2\_Cascade\_Vector8\_FP(middle, last)

Float data vector direct form II biquad cascade filter.

The input and output data are float data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```

#define VECTOR_LEN 16
float input[VECTOR_LEN] = {1024.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
float output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_Initiate_Vector_Func(pSrc, pDst);
PQ_DF2_Cascade_Vector8_FP(false, false);
PQ_DF2_Cascade_Vector8_FP(true, true);
PQ_End_Vector_Func();

```

PQ\_DF2\_Cascade\_Vector8\_FX(middle, last)

Fixed data vector direct form II biquad cascade filter.

The input and output data are fixed data. The data flow is input -> biquad side 1 -> biquad

side 0 -> output.

```
#define VECTOR_LEN 16
int32_t input[VECTOR_LEN] = {1024.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_Initiate_Vector_Func(pSrc, pDst);
PQ_DF2_Cascade_Vector8_FX(false, false);
PQ_DF2_Cascade_Vector8_FX(true, true);
PQ_End_Vector_Func();
```

PQ\_Vector8BiquadDf2CascadeF32()

Float data vector direct form II biquad cascade filter.

The input and output data are float data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 8
float input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
float output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxx,
```

(continues on next page)

(continued from previous page)

```

        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2CascadeF32();
PQ_EndVector();

```

PQ\_Vector8BiquadDf2CascadeFixed32()

Fixed 32-bit data vector direct form II biquad cascade filter.

The input and output data are fixed 32-bit data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```

#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2CascadeFixed32();
PQ_EndVector();

```

PQ\_Vector8BiquadDf2CascadeFixed16()

Fixed 16-bit data vector direct form II biquad cascade filter.

The input and output data are fixed 16-bit data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2CascadeFixed16();
PQ_EndVector();
```

**POWERQUAD\_MAKE\_MATRIX\_LEN(mat1Row, mat1Col, mat2Col)**

Make the length used for matrix functions.

**PQ\_Q31\_2\_FLOAT(x)**

Convert Q31 to float.

**PQ\_Q15\_2\_FLOAT(x)**

Convert Q15 to float.

**struct pq\_prescale\_t**

*#include <fsl\_powerquad.h>* Coprocessor prescale.

### Public Members

**int8\_t inputPrescale**

Input prescale.

**int8\_t outputPrescale**

Output prescale.

**int8\_t outputSaturate**

Output saturate at n bits, for example 0x11 is 8 bit space, the value will be truncated at +127 or -128.

**struct pq\_config\_t**

*#include <fsl\_powerquad.h>* powerquad data structure format

**Public Members**

*pq\_format\_t* inputAFormat

Input A format.

int8\_t inputAPrescale

Input A prescale, for example 1.5 can be  $1.5 \cdot 2^n$  if you scale by 'shifting' ('scaling' by a factor of n).

*pq\_format\_t* inputBFormat

Input B format.

int8\_t inputBPrescale

Input B prescale.

*pq\_format\_t* outputFormat

Out format.

int8\_t outputPrescale

Out prescale.

*pq\_format\_t* tmpFormat

Temp format.

int8\_t tmpPrescale

Temp prescale.

*pq\_format\_t* machineFormat

Machine format.

uint32\_t \*tmpBase

Tmp base address.

struct \_\_pq\_biquad\_param

*#include <fsl\_powerquad.h>* Struct to save biquad parameters.

**Public Members**

float v\_n\_1

v[n-1], set to 0 when initialization.

float v\_n

v[n], set to 0 when initialization.

float a\_1

a[1]

float a\_2

a[2]

float b\_0

b[0]

float b\_1

b[1]

float b\_2

b[2]

struct \_\_pq\_biquad\_state

*#include <fsl\_powerquad.h>* Struct to save biquad state.

### Public Members

*pq\_biquad\_param\_t* param

Filter parameter.

uint32\_t compreg

Internal register, set to 0 when initialization.

struct pq\_biquad\_cascade\_df2\_instance

*#include <fsl\_powerquad.h>* Instance structure for the direct form II Biquad cascade filter.

### Public Members

uint8\_t numStages

Number of 2nd order stages in the filter.

*pq\_biquad\_state\_t* \*pState

Points to the array of state coefficients.

union \_\_pq\_float

*#include <fsl\_powerquad.h>* Conversion between integer and float type.

### Public Members

float floatX

Float type.

uint32\_t integerX

Unsigned interger type.

## 2.55 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>*

## 2.56 Reset Driver

enum \_RSTCTL\_RSTn

Enumeration for peripheral reset control bits.

Defines the enumeration for peripheral reset control bits in RSTCTLx registers

*Values:*

enumerator kDSP\_RST\_SHIFT\_RSTn

DSP reset control

enumerator kPOWERQUAD\_RST\_SHIFT\_RSTn

POWERQUAD reset control

enumerator kCASPER\_RST\_SHIFT\_RSTn  
CASPER reset control

enumerator kHASHCRYPT\_RST\_SHIFT\_RSTn  
HASHCRYPT reset control

enumerator kPUF\_RST\_SHIFT\_RSTn  
Physical unclonable function reset control

enumerator kRNG\_RST\_SHIFT\_RSTn  
Random number generator (RNG) reset control

enumerator kFLEXSPI\_RST\_SHIFT\_RSTn  
FLEXSPI reset control

enumerator kUSBHS\_PHY\_RST\_SHIFT\_RSTn  
High speed USB PHY reset control

enumerator kUSBHS\_DEVICE\_RST\_SHIFT\_RSTn  
High speed USB Device reset control

enumerator kUSBHS\_HOST\_RST\_SHIFT\_RSTn  
High speed USB Host reset control

enumerator kUSBHS\_SRAM\_RST\_SHIFT\_RSTn  
High speed USB SRAM reset control

enumerator kSCT\_RST\_SHIFT\_RSTn  
Standard ctimers reset control

enumerator kSDIO0\_RST\_SHIFT\_RSTn  
SDIO0 reset control

enumerator kSDIO1\_RST\_SHIFT\_RSTn  
SDIO1 reset control

enumerator kACMP0\_RST\_SHIFT\_RSTn  
Grouped interrupt (PINT) reset control.

enumerator kADC0\_RST\_SHIFT\_RSTn  
ADC0 reset control

enumerator kSHSGPIO0\_RST\_SHIFT\_RSTn  
Security HSGPIO 0 reset control

enumerator kUTICK0\_RST\_SHIFT\_RSTn  
Micro-tick timer reset control

enumerator kWWDT0\_RST\_SHIFT\_RSTn  
Windowed Watchdog timer 0 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 kFC14\_RST\_SHIFT\_RSTn  
Flexcomm Interface 14 reset control

enumerator kFC15\_RST\_SHIFT\_RSTn  
Flexcomm Interface 15 reset control

enumerator kDMIC\_RST\_SHIFT\_RSTn  
Digital microphone interface reset control

enumerator kOSEVENT\_TIMER\_RST\_SHIFT\_RSTn  
Osevent Timer reset control

enumerator kHSGPIO0\_RST\_SHIFT\_RSTn  
HSGPIO 0 reset control

enumerator kHSGPIO1\_RST\_SHIFT\_RSTn  
HSGPIO 1 reset control

enumerator kHSGPIO2\_RST\_SHIFT\_RSTn  
HSGPIO 2 reset control

enumerator kHSGPIO3\_RST\_SHIFT\_RSTn  
HSGPIO 3 reset control

enumerator kHSGPIO4\_RST\_SHIFT\_RSTn  
HSGPIO 4 reset control

enumerator kHSGPIO5\_RST\_SHIFT\_RSTn  
HSGPIO 5 reset control

enumerator kHSGPIO6\_RST\_SHIFT\_RSTn  
HSGPIO 6 reset control

enumerator kHSGPIO7\_RST\_SHIFT\_RSTn  
HSGPIO 7 reset control

enumerator kCRC\_RST\_SHIFT\_RSTn  
CRC reset control

enumerator kDMAC0\_RST\_SHIFT\_RSTn  
DMA Controller 0 reset control

enumerator kDMAC1\_RST\_SHIFT\_RSTn  
DMA Controller 1 reset control

enumerator kMU\_RST\_SHIFT\_RSTn  
Message Unit reset control

enumerator kSEMA\_RST\_SHIFT\_RSTn  
Semaphore reset control

```

enumerator kFREQME_RST_SHIFT_RSTn
    Frequency Measure reset control
enumerator kCT32B0_RST_SHIFT_RSTn
    CT32B0 reset control
enumerator kCT32B1_RST_SHIFT_RSTn
    CT32B1 reset control
enumerator kCT32B2_RST_SHIFT_RSTn
    CT32B3 reset control
enumerator kCT32B3_RST_SHIFT_RSTn
    CT32B4 reset control
enumerator kCT32B4_RST_SHIFT_RSTn
    CT32B4 reset control
enumerator kMRT0_RST_SHIFT_RSTn
    Multi-rate timer (MRT) reset control
enumerator kWWDT1_RST_SHIFT_RSTn
    Windowed Watchdog timer 1 reset control
enumerator kI3C0_RST_SHIFT_RSTn
    I3C reset control
enumerator kPINT_RST_SHIFT_RSTn
    GPIO Pin interrupt reset control
enumerator kINPUTMUX_RST_SHIFT_RSTn
    Peripheral input muxes reset control
typedef enum _RSTCTL_RSTn RSTCTL_RSTn_t
    Enumeration for peripheral reset control bits.
    Defines the enumeration for peripheral reset control bits in RSTCTLx registers
typedef RSTCTL_RSTn_t reset_ip_name_t
    IP reset handle.
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 2.2.1.

RST\_CTL0\_PSCCTL0

Reset control registers index.

RST\_CTL0\_PSCCTL1

RST\_CTL0\_PSCCTL2

RST\_CTL1\_PSCCTL0

RST\_CTL1\_PSCCTL1

RST\_CTL1\_PSCCTL2

ADC\_RSTS

Array initializers with peripheral reset bits

CASPER\_RSTS

CRC\_RSTS

DMA\_RSTS\_N

DMIC\_RSTS

FLEXCOMM\_RSTS

GPIO\_RSTS\_N

HASHCRYPT\_RSTS

I3C\_RSTS

INPUTMUX\_RSTS

MRT\_RSTS

MU\_RSTS

PINT\_RSTS

SCT\_RSTS

CTIMER\_RSTS

USB\_RSTS

USDHC\_RSTS

UTICK\_RSTS

WWDT\_RSTS

OSTIMER\_RSTS

POWERQUAD\_RSTS

PUF\_RSTS

TRNG\_RSTS

## 2.57 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\_EnableSubsecCounter(RTC\_Type \*base, bool enable)

Enable the RTC Sub-second counter (32KHZ).

---

**Note:** Only enable sub-second counter after RTC\_ENA bit has been set to 1.

---

**Parameters**

- base – RTC peripheral base address
- enable – Enable/Disable RTC sub-second counter.
  - true: Enable RTC sub-second counter.
  - false: Disable RTC sub-second counter.

static inline uint32\_t RTC\_GetSubsecValue(const RTC\_Type \*base)

A read of 32KHZ sub-seconds counter.

**Parameters**

- base – RTC peripheral base address



**Returns**

Current value of the SUBSEC register

```
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 superceded 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 superceded 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.

## 2.58 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<sub>N</sub>(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);`

## 2.59 SEMA42: Hardware Semaphores Driver

FSL\_SEMA42\_DRIVER\_VERSION

SEMA42 driver version.

SEMA42 status return codes.

*Values:*

enumerator kStatus\_SEMA42\_Busy

SEMA42 gate has been locked by other processor.

enumerator kStatus\_SEMA42\_Reseting

SEMA42 gate resetting is ongoing.

enum \_sema42\_gate\_status

SEMA42 gate lock status.

*Values:*

enumerator kSEMA42\_Unlocked

The gate is unlocked.

enumerator kSEMA42\_LockedByProc0

The gate is locked by processor 0.

enumerator kSEMA42\_LockedByProc1

The gate is locked by processor 1.

enumerator kSEMA42\_LockedByProc2

The gate is locked by processor 2.

enumerator kSEMA42\_LockedByProc3

The gate is locked by processor 3.

enumerator kSEMA42\_LockedByProc4

The gate is locked by processor 4.

enumerator kSEMA42\_LockedByProc5

The gate is locked by processor 5.

enumerator kSEMA42\_LockedByProc6

The gate is locked by processor 6.

enumerator kSEMA42\_LockedByProc7

The gate is locked by processor 7.

enumerator kSEMA42\_LockedByProc8

The gate is locked by processor 8.

enumerator kSEMA42\_LockedByProc9

The gate is locked by processor 9.

enumerator kSEMA42\_LockedByProc10

The gate is locked by processor 10.

enumerator kSEMA42\_LockedByProc11

The gate is locked by processor 11.

enumerator kSEMA42\_LockedByProc12

The gate is locked by processor 12.

enumerator kSEMA42\_LockedByProc13

The gate is locked by processor 13.

enumerator kSEMA42\_LockedByProc14

The gate is locked by processor 14.

typedef enum *\_sema42\_gate\_status* sema42\_\_gate\_\_status\_\_t  
SEMA42 gate lock status.

void SEMA42\_Init(SEMA42\_Type \*base)

Initializes the SEMA42 module.

This function initializes the SEMA42 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either SEMA42\_ResetGate or SEMA42\_ResetAllGates function.

#### Parameters

- base – SEMA42 peripheral base address.

void SEMA42\_Deinit(SEMA42\_Type \*base)

De-initializes the SEMA42 module.

This function de-initializes the SEMA42 module. It only disables the clock.

#### Parameters

- base – SEMA42 peripheral base address.

*status\_t* SEMA42\_TryLock(SEMA42\_Type \*base, uint8\_t gateNum, uint8\_t procNum)

Tries to lock the SEMA42 gate.

This function tries to lock the specific SEMA42 gate. If the gate has been locked by another processor, this function returns an error code.

#### Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

#### Return values

- kStatus\_Success – Lock the sema42 gate successfully.
- kStatus\_SEMA42\_Busy – Sema42 gate has been locked by another processor.

*status\_t* SEMA42\_Lock(SEMA42\_Type \*base, uint8\_t gateNum, uint8\_t procNum)

Locks the SEMA42 gate.

This function locks the specific SEMA42 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

If SEMA42\_BUSY\_POLL\_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus\_Timeout.

#### Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

#### Return values

- kStatus\_Success – The gate was successfully locked.
- kStatus\_Timeout – Timeout occurred while waiting for the gate to be unlocked.

**Returns**

status\_t

static inline void SEMA42\_Unlock(SEMA42\_Type \*base, uint8\_t gateNum)

Unlocks the SEMA42 gate.

This function unlocks the specific SEMA42 gate. It only writes unlock value to the SEMA42 gate register. However, it does not check whether the SEMA42 gate is locked by the current processor or not. As a result, if the SEMA42 gate is not locked by the current processor, this function has no effect.

**Parameters**

- base – SEMA42 peripheral base address.
- gateNum – Gate number to unlock.

static inline *sema42\_gate\_status\_t* SEMA42\_GetGateStatus(SEMA42\_Type \*base, uint8\_t gateNum)

Gets the status of the SEMA42 gate.

This function checks the lock status of a specific SEMA42 gate.

**Parameters**

- base – SEMA42 peripheral base address.
- gateNum – Gate number.

**Returns**

status Current status.

*status\_t* SEMA42\_ResetGate(SEMA42\_Type \*base, uint8\_t gateNum)

Resets the SEMA42 gate to an unlocked status.

This function resets a SEMA42 gate to an unlocked status.

**Parameters**

- base – SEMA42 peripheral base address.
- gateNum – Gate number.

**Return values**

- kStatus\_Success – SEMA42 gate is reset successfully.
- kStatus\_SEMA42\_Reseting – Some other reset process is ongoing.

static inline *status\_t* SEMA42\_ResetAllGates(SEMA42\_Type \*base)

Resets all SEMA42 gates to an unlocked status.

This function resets all SEMA42 gate to an unlocked status.

**Parameters**

- base – SEMA42 peripheral base address.

**Return values**

- kStatus\_Success – SEMA42 is reset successfully.
- kStatus\_SEMA42\_Reseting – Some other reset process is ongoing.

SEMA42\_GATE\_NUM\_RESET\_ALL

The number to reset all SEMA42 gates.



SEMA42\_GATE<sub>n</sub>(base, n)

SEMA42 gate n register address.

The SEMA42 gates are sorted in the order 3, 2, 1, 0, 7, 6, 5, 4, ... not in the order 0, 1, 2, 3, 4, 5, 6, 7, ... The macro SEMA42\_GATE<sub>n</sub> gets the SEMA42 gate based on the gate index.

The input gate index is XOR'ed with 3U:  $0 \wedge 3 = 3$   $1 \wedge 3 = 2$   $2 \wedge 3 = 1$   $3 \wedge 3 = 0$   $4 \wedge 3 = 7$   $5 \wedge 3 = 6$   $6 \wedge 3 = 5$   $7 \wedge 3 = 4$  ...

SEMA42\_BUSY\_POLL\_COUNT

Maximum polling iterations for SEMA42 waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the SEMA42 driver code before timing out and returning an error.

It applies to all waiting loops in SEMA42 driver, such as waiting for a gate to be unlocked, waiting for a reset to complete, or waiting for a resource to become available.

This is a count of loop iterations, not a time-based value.

If defined as 0, polling loops will continue indefinitely until their exit condition is met, which could potentially cause the system to hang if hardware doesn't respond or if a resource is never released.

## 2.60 SPI: Serial Peripheral Interface Driver

### 2.61 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 transfer remaining bytes.

This function gets the master DMA transfer remaining 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 transfer remaining bytes.

This function gets the slave DMA transfer remaining 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 2.1.1.

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**

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

*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

const uint8\_t \*txEndData

The pointer of end of data

uint8\_t \*rxNextData

The pointer of next time rx data

uint8\_t \*rxEndData

The pointer of end of rx data

uint32\_t dataBytesEveryTime

Bytes in a time for DMA transfer; default is DMA\_MAX\_TRANSFER\_COUNT

## 2.62 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 maxinum 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 maximum 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

## 2.63 TRNG: True Random Number Generator

FSL\_TRNG\_DRIVER\_VERSION

TRNG driver version 2.0.18.

Current version: 2.0.18

Change log:

- version 2.0.18
  - TRNG health checks now done in software on RT5xx and RT6xx.
- version 2.0.17
  - Added support for RT700.
- version 2.0.16
  - Added support for Dual oscillator mode.
- version 2.0.15
  - Changed TRNG\_USER\_CONFIG\_DEFAULT\_XXX values according to latest recommended by design team.
- version 2.0.14
  - add support for RW610 and RW612
- version 2.0.13
  - After deepsleep it might return error, added clearing bits in TRNG\_GetRandomData() and generating new entropy.
  - Modified reloading entropy in TRNG\_GetRandomData(), for some data length it doesn't reloading entropy correctly.
- version 2.0.12
  - For KW34A4\_SERIES, KW35A4\_SERIES, KW36A4\_SERIES set TRNG\_USER\_CONFIG\_DEFAULT\_OSC\_DIV to kTRNG\_RingOscDiv8.
- version 2.0.11
  - Add clearing pending errors in TRNG\_Init().
- version 2.0.10
  - Fixed doxygen issues.
- version 2.0.9

- Fix HIS\_CCM metrics issues.
- version 2.0.8
  - For K32L2A41A\_SERIES set TRNG\_USER\_CONFIG\_DEFAULT\_OSC\_DIV to kTRNG\_RingOscDiv4.
- version 2.0.7
  - Fix MISRA 2004 issue rule 12.5.
- version 2.0.6
  - For KW35Z4\_SERIES set TRNG\_USER\_CONFIG\_DEFAULT\_OSC\_DIV to kTRNG\_RingOscDiv8.
- version 2.0.5
  - Add possibility to define default TRNG configuration by device specific preprocessor macros for FRQMIN, FRQMAX and OSCDIV.
- version 2.0.4
  - Fix MISRA-2012 issues.
- Version 2.0.3
  - update TRNG\_Init to restart entropy generation
- Version 2.0.2
  - fix MISRA issues
- Version 2.0.1
  - add support for KL8x and KL28Z
  - update default OSCDIV for K81 to divide by 2

enum `_trng_sample_mode`

TRNG sample mode. Used by `trng_config_t`.

*Values:*

enumerator `kTRNG_SampleModeVonNeumann`

Use von Neumann data in both Entropy shifter and Statistical Checker.

enumerator `kTRNG_SampleModeRaw`

Use raw data into both Entropy shifter and Statistical Checker.

enumerator `kTRNG_SampleModeVonNeumannRaw`

Use von Neumann data in Entropy shifter. Use raw data into Statistical Checker.

enum `_trng_clock_mode`

TRNG clock mode. Used by `trng_config_t`.

*Values:*

enumerator `kTRNG_ClockModeRingOscillator`

Ring oscillator is used to operate the TRNG (default).

enumerator `kTRNG_ClockModeSystem`

System clock is used to operate the TRNG. This is for test use only, and indeterminate results may occur.

enum `_trng_ring_osc_div`

TRNG ring oscillator divide. Used by `trng_config_t`.

*Values:*

enumerator `kTRNG_RingOscDiv0`  
Ring oscillator with no divide

enumerator `kTRNG_RingOscDiv2`  
Ring oscillator divided-by-2.

enumerator `kTRNG_RingOscDiv4`  
Ring oscillator divided-by-4.

enumerator `kTRNG_RingOscDiv8`  
Ring oscillator divided-by-8.

typedef enum *trng\_sample\_mode* `trng_sample_mode_t`  
TRNG sample mode. Used by `trng_config_t`.

typedef enum *trng\_clock\_mode* `trng_clock_mode_t`  
TRNG clock mode. Used by `trng_config_t`.

typedef enum *trng\_ring\_osc\_div* `trng_ring_osc_div_t`  
TRNG ring oscillator divide. Used by `trng_config_t`.

typedef struct *trng\_statistical\_check\_limit* `trng_statistical_check_limit_t`  
Data structure for definition of statistical check limits. Used by `trng_config_t`.

typedef struct *trng\_user\_config* `trng_config_t`  
Data structure for the TRNG initialization.

This structure initializes the TRNG by calling the `TRNG_Init()` function. It contains all TRNG configurations.

*status\_t* `TRNG_GetDefaultConfig(trng_config_t *userConfig)`  
Initializes the user configuration structure to default values.

This function initializes the configuration structure to default values. The default values are platform dependent.

#### Parameters

- `userConfig` – User configuration structure.

#### Returns

If successful, returns the `kStatus_TRNG_Success`. Otherwise, it returns an error.

*status\_t* `TRNG_Init(TRNG_Type *base, const trng_config_t *userConfig)`  
Initializes the TRNG.

This function initializes the TRNG. When called, the TRNG entropy generation starts immediately.

#### Parameters

- `base` – TRNG base address
- `userConfig` – Pointer to the initialization configuration structure.

#### Returns

If successful, returns the `kStatus_TRNG_Success`. Otherwise, it returns an error.

*void* `TRNG_Deinit(TRNG_Type *base)`  
Shuts down the TRNG.

This function shuts down the TRNG.

#### Parameters

- base – TRNG base address.

*status\_t* TRNG\_GetRandomData(TRNG\_Type \*base, void \*data, size\_t dataSize)

Gets random data.

This function gets random data from the TRNG.

#### Parameters

- base – TRNG base address.
- data – Pointer address used to store random data.
- dataSize – Size of the buffer pointed by the data parameter.

#### Returns

random data

struct \_trng\_statistical\_check\_limit

*#include <fsl\_trng.h>* Data structure for definition of statistical check limits. Used by *trng\_config\_t*.

#### Public Members

uint32\_t maximum

Maximum limit.

int32\_t minimum

Minimum limit.

struct \_trng\_user\_config

*#include <fsl\_trng.h>* Data structure for the TRNG initialization.

This structure initializes the TRNG by calling the TRNG\_Init() function. It contains all TRNG configurations.

#### Public Members

bool lock

Disable programmability of TRNG registers.

*trng\_clock\_mode\_t* clockMode

Clock mode used to operate TRNG.

*trng\_ring\_osc\_div\_t* ringOscDiv

Ring oscillator divide used by TRNG.

*trng\_sample\_mode\_t* sampleMode

Sample mode of the TRNG ring oscillator.

uint16\_t entropyDelay

Entropy Delay. Defines the length (in system clocks) of each Entropy sample taken.

uint16\_t sampleSize

Sample Size. Defines the total number of Entropy samples that will be taken during Entropy generation.

uint16\_t sparseBitLimit

Sparse Bit Limit which defines the maximum number of consecutive samples that may be discarded before an error is generated. This limit is used only for during von Neumann sampling (enabled by TRNG\_HAL\_SetSampleMode()). Samples are discarded if two consecutive raw samples are both 0 or both 1. If this discarding occurs for a long period of time, it indicates that there is insufficient Entropy.



`uint8_t` retryCount

Retry count. It defines the number of times a statistical check may fails during the TRNG Entropy Generation before generating an error.

`uint8_t` longRunMaxLimit

Largest allowable number of consecutive samples of all 1, or all 0, that is allowed during the Entropy generation.

`trng_statistical_check_limit_t` monobitLimit

Maximum and minimum limits for statistical check of number of ones/zero detected during entropy generation.

`trng_statistical_check_limit_t` runBit1Limit

Maximum and minimum limits for statistical check of number of runs of length 1 detected during entropy generation.

`trng_statistical_check_limit_t` runBit2Limit

Maximum and minimum limits for statistical check of number of runs of length 2 detected during entropy generation.

`trng_statistical_check_limit_t` runBit3Limit

Maximum and minimum limits for statistical check of number of runs of length 3 detected during entropy generation.

`trng_statistical_check_limit_t` runBit4Limit

Maximum and minimum limits for statistical check of number of runs of length 4 detected during entropy generation.

`trng_statistical_check_limit_t` runBit5Limit

Maximum and minimum limits for statistical check of number of runs of length 5 detected during entropy generation.

`trng_statistical_check_limit_t` runBit6PlusLimit

Maximum and minimum limits for statistical check of number of runs of length 6 or more detected during entropy generation.

`trng_statistical_check_limit_t` pokerLimit

Maximum and minimum limits for statistical check of “Poker Test”.

`trng_statistical_check_limit_t` frequencyCountLimit

Maximum and minimum limits for statistical check of entropy sample frequency count.

## 2.64 USART: Universal Synchronous/Asynchronous Receiver/Transmitter Driver

### 2.65 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

## 2.66 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, 2000000U);
```

### 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_RTCOSCTRL_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 `remainBtyes` 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 __unnamed61__`

### 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.

## 2.67 USDHC: Ultra Secured Digital Host Controller Driver

`void USDHC_Init(USDHC_Type *base, const usdhc_config_t *config)`

USDHC module initialization function.

Configures the USDHC according to the user configuration.

Example:

```
usdhc_config_t config;
config.cardDetectDat3 = false;
config.endianMode = kUSDHC_EndianModeLittle;
config.dmaMode = kUSDHC_DmaModeAdma2;
config.readWatermarkLevel = 128U;
config.writeWatermarkLevel = 128U;
USDHC_Init(USDHC, &config);
```

### Parameters

- `base` – USDHC peripheral base address.
- `config` – USDHC configuration information.

### Return values

`kStatus_Success` – Operate successfully.

`void USDHC_Deinit(USDHC_Type *base)`

Deinitializes the USDHC.

### Parameters

- `base` – USDHC peripheral base address.

`bool USDHC_Reset(USDHC_Type *base, uint32_t mask, uint32_t timeout)`

Resets the USDHC.

### Parameters

- `base` – USDHC peripheral base address.
- `mask` – The reset type mask(`_usdhc_reset`).
- `timeout` – Timeout for reset.

### Return values

- `true` – Reset successfully.
- `false` – Reset failed.

`status_t USDHC_SetAdmaTableConfig(USDHC_Type *base, usdhc_adma_config_t *dmaConfig, usdhc_data_t *dataConfig, uint32_t flags)`

Sets the DMA descriptor table configuration. A high level DMA descriptor configuration function.

### Parameters

- `base` – USDHC peripheral base address.
- `dmaConfig` – ADMA configuration
- `dataConfig` – Data descriptor
- `flags` – ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum `_usdhc_adma_flag`.

### Return values

- `kStatus_OutOfRange` – ADMA descriptor table length isn't enough to describe data.
- `kStatus_Success` – Operate successfully.

`status_t` USDHC\_SetInternalDmaConfig(USDHC\_Type \*base, *usdhc\_adma\_config\_t* \*dmaConfig, const uint32\_t \*dataAddr, bool enAutoCmd23)

Internal DMA configuration. This function is used to config the USDHC DMA related registers.

#### Parameters

- base – USDHC peripheral base address.
- dmaConfig – ADMA configuration.
- dataAddr – Transfer data address, a simple DMA parameter, if ADMA is used, leave it to NULL.
- enAutoCmd23 – Flag to indicate Auto CMD23 is enable or not, a simple DMA parameter, if ADMA is used, leave it to false.

#### Return values

- `kStatus_OutOfRange` – ADMA descriptor table length isn't enough to describe data.
- `kStatus_Success` – Operate successfully.

`status_t` USDHC\_SetADMA2Descriptor(uint32\_t \*admaTable, uint32\_t admaTableWords, const uint32\_t \*dataBufferAddr, uint32\_t dataBytes, uint32\_t flags)

Sets the ADMA2 descriptor table configuration.

#### Parameters

- admaTable – ADMA table address.
- admaTableWords – ADMA table length.
- dataBufferAddr – Data buffer address.
- dataBytes – Data Data length.
- flags – ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum `_usdhc_adma_flag`.

#### Return values

- `kStatus_OutOfRange` – ADMA descriptor table length isn't enough to describe data.
- `kStatus_Success` – Operate successfully.

`status_t` USDHC\_SetADMA1Descriptor(uint32\_t \*admaTable, uint32\_t admaTableWords, const uint32\_t \*dataBufferAddr, uint32\_t dataBytes, uint32\_t flags)

Sets the ADMA1 descriptor table configuration.

#### Parameters

- admaTable – ADMA table address.
- admaTableWords – ADMA table length.
- dataBufferAddr – Data buffer address.
- dataBytes – Data length.
- flags – ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum `_usdhc_adma_flag`.

**Return values**

- `kStatus_OutOfRange` – ADMA descriptor table length isn't enough to describe data.
- `kStatus_Success` – Operate successfully.

`static inline void USDHC_EnableInternalDMA(USDHC_Type *base, bool enable)`

Enables internal DMA.

**Parameters**

- `base` – USDHC peripheral base address.
- `enable` – enable or disable flag

`static inline void USDHC_EnableInterruptStatus(USDHC_Type *base, uint32_t mask)`

Enables the interrupt status.

**Parameters**

- `base` – USDHC peripheral base address.
- `mask` – Interrupt status flags mask(`_usdhc_interrupt_status_flag`).

`static inline void USDHC_DisableInterruptStatus(USDHC_Type *base, uint32_t mask)`

Disables the interrupt status.

**Parameters**

- `base` – USDHC peripheral base address.
- `mask` – The interrupt status flags mask(`_usdhc_interrupt_status_flag`).

`static inline void USDHC_EnableInterruptSignal(USDHC_Type *base, uint32_t mask)`

Enables the interrupt signal corresponding to the interrupt status flag.

**Parameters**

- `base` – USDHC peripheral base address.
- `mask` – The interrupt status flags mask(`_usdhc_interrupt_status_flag`).

`static inline void USDHC_DisableInterruptSignal(USDHC_Type *base, uint32_t mask)`

Disables the interrupt signal corresponding to the interrupt status flag.

**Parameters**

- `base` – USDHC peripheral base address.
- `mask` – The interrupt status flags mask(`_usdhc_interrupt_status_flag`).

`static inline uint32_t USDHC_GetEnabledInterruptStatusFlags(USDHC_Type *base)`

Gets the enabled interrupt status.

**Parameters**

- `base` – USDHC peripheral base address.

**Returns**

Current interrupt status flags mask(`_usdhc_interrupt_status_flag`).

`static inline uint32_t USDHC_GetInterruptStatusFlags(USDHC_Type *base)`

Gets the current interrupt status.

**Parameters**

- `base` – USDHC peripheral base address.

**Returns**

Current interrupt status flags mask(`_usdhc_interrupt_status_flag`).

static inline void USDHC\_ClearInterruptStatusFlags(USDHC\_Type \*base, uint32\_t mask)  
Clears a specified interrupt status. write 1 clears.

**Parameters**

- base – USDHC peripheral base address.
- mask – The interrupt status flags mask(\_usdhc\_interrupt\_status\_flag).

static inline uint32\_t USDHC\_GetAutoCommand12ErrorStatusFlags(USDHC\_Type \*base)  
Gets the status of auto command 12 error.

**Parameters**

- base – USDHC peripheral base address.

**Returns**

Auto command 12 error status flags mask(\_usdhc\_auto\_command12\_error\_status\_flag).

static inline uint32\_t USDHC\_GetAdmaErrorStatusFlags(USDHC\_Type \*base)  
Gets the status of the ADMA error.

**Parameters**

- base – USDHC peripheral base address.

**Returns**

ADMA error status flags mask(\_usdhc\_adma\_error\_status\_flag).

static inline uint32\_t USDHC\_GetPresentStatusFlags(USDHC\_Type \*base)  
Gets a present status.

This function gets the present USDHC's status except for an interrupt status and an error status.

**Parameters**

- base – USDHC peripheral base address.

**Returns**

Present USDHC's status flags mask(\_usdhc\_present\_status\_flag).

void USDHC\_GetCapability(USDHC\_Type \*base, *usdhc\_capability\_t* \*capability)  
Gets the capability information.

**Parameters**

- base – USDHC peripheral base address.
- capability – Structure to save capability information.

static inline void USDHC\_ForceClockOn(USDHC\_Type \*base, bool enable)  
Forces the card clock on.

**Parameters**

- base – USDHC peripheral base address.
- enable – enable/disable flag

uint32\_t USDHC\_SetSdClock(USDHC\_Type \*base, uint32\_t srcClock\_Hz, uint32\_t busClock\_Hz)  
Sets the SD bus clock frequency.

**Parameters**

- base – USDHC peripheral base address.
- srcClock\_Hz – USDHC source clock frequency united in Hz.
- busClock\_Hz – SD bus clock frequency united in Hz.

**Returns**

The nearest frequency of busClock\_Hz configured for SD bus.

`bool USDHC_SetCardActive(USDHC_Type *base, uint32_t timeout)`

Sends 80 clocks to the card to set it to the active state.

This function must be called each time the card is inserted to ensure that the card can receive the command correctly.

**Parameters**

- `base` – USDHC peripheral base address.
- `timeout` – Timeout to initialize card.

**Return values**

- `true` – Set card active successfully.
- `false` – Set card active failed.

`static inline void USDHC_AssertHardwareReset(USDHC_Type *base, bool high)`

Triggers a hardware reset.

**Parameters**

- `base` – USDHC peripheral base address.
- `high` – 1 or 0 level

`static inline void USDHC_SetDataBusWidth(USDHC_Type *base, usdhc_data_bus_width_t width)`

Sets the data transfer width.

**Parameters**

- `base` – USDHC peripheral base address.
- `width` – Data transfer width.

`static inline void USDHC_WriteData(USDHC_Type *base, uint32_t data)`

Fills the data port.

This function is used to implement the data transfer by Data Port instead of DMA.

**Parameters**

- `base` – USDHC peripheral base address.
- `data` – The data about to be sent.

`static inline uint32_t USDHC_ReadData(USDHC_Type *base)`

Retrieves the data from the data port.

This function is used to implement the data transfer by Data Port instead of DMA.

**Parameters**

- `base` – USDHC peripheral base address.

**Returns**

The data has been read.

`void USDHC_SendCommand(USDHC_Type *base, usdhc_command_t *command)`

Sends command function.

**Parameters**

- `base` – USDHC peripheral base address.
- `command` – configuration



static inline void USDHC\_EnableWakeupEvent(USDHC\_Type \*base, uint32\_t mask, bool enable)

Enables or disables a wakeup event in low-power mode.

**Parameters**

- base – USDHC peripheral base address.
- mask – Wakeup events mask(\_usdhc\_wakeup\_event).
- enable – True to enable, false to disable.

static inline void USDHC\_CardDetectByData3(USDHC\_Type \*base, bool enable)

Detects card insert status.

**Parameters**

- base – USDHC peripheral base address.
- enable – enable/disable flag

static inline bool USDHC\_DetectCardInsert(USDHC\_Type \*base)

Detects card insert status.

**Parameters**

- base – USDHC peripheral base address.

static inline void USDHC\_EnableSdioControl(USDHC\_Type \*base, uint32\_t mask, bool enable)

Enables or disables the SDIO card control.

**Parameters**

- base – USDHC peripheral base address.
- mask – SDIO card control flags mask(\_usdhc\_sdio\_control\_flag).
- enable – True to enable, false to disable.

static inline void USDHC\_SetContinueRequest(USDHC\_Type \*base)

Restarts a transaction which has stopped at the block GAP for the SDIO card.

**Parameters**

- base – USDHC peripheral base address.

static inline void USDHC\_RequestStopAtBlockGap(USDHC\_Type \*base, bool enable)

Request stop at block gap function.

**Parameters**

- base – USDHC peripheral base address.
- enable – True to stop at block gap, false to normal transfer.

void USDHC\_SetMmcBootConfig(USDHC\_Type \*base, const *usdhc\_boot\_config\_t* \*config)

Configures the MMC boot feature.

Example:

```
usdhc_boot_config_t config;
config.ackTimeoutCount = 4;
config.bootMode = kUSDHC_BootModeNormal;
config.blockCount = 5;
config.enableBootAck = true;
config.enableBoot = true;
config.enableAutoStopAtBlockGap = true;
USDHC_SetMmcBootConfig(USDHC, &config);
```

**Parameters**

- base – USDHC peripheral base address.
- config – The MMC boot configuration information.

static inline void USDHC\_EnableMmcBoot(USDHC\_Type \*base, bool enable)

Enables or disables the mmc boot mode.

#### Parameters

- base – USDHC peripheral base address.
- enable – True to enable, false to disable.

static inline void USDHC\_SetForceEvent(USDHC\_Type \*base, uint32\_t mask)

Forces generating events according to the given mask.

#### Parameters

- base – USDHC peripheral base address.
- mask – The force events bit position (\_usdhc\_force\_event).

static inline bool USDHC\_RequestTuningForSDR50(USDHC\_Type \*base)

Checks the SDR50 mode request tuning bit. When this bit set, application shall perform tuning for SDR50 mode.

#### Parameters

- base – USDHC peripheral base address.

static inline bool USDHC\_RequestReTuning(USDHC\_Type \*base)

Checks the request re-tuning bit. When this bit is set, user should do manual tuning or standard tuning function.

#### Parameters

- base – USDHC peripheral base address.

static inline void USDHC\_EnableAutoTuning(USDHC\_Type \*base, bool enable)

The SDR104 mode auto tuning enable and disable. This function should be called after tuning function execute pass, auto tuning will handle by hardware.

#### Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

void USDHC\_EnableAutoTuningForCmdAndData(USDHC\_Type \*base)

The auto tuning enable for CMD/DATA line.

#### Parameters

- base – USDHC peripheral base address.

void USDHC\_EnableManualTuning(USDHC\_Type \*base, bool enable)

Manual tuning trigger or abort. User should handle the tuning cmd and find the boundary of the delay then calculate a average value which will be configured to the **CLK\_TUNE\_CTRL\_STATUS**. This function should be called before function **USDHC\_AdjustDelayForManualTuning**.

#### Parameters

- base – USDHC peripheral base address.
- enable – tuning enable flag

static inline uint32\_t USDHC\_GetTuningDelayStatus(USDHC\_Type \*base)

Get the tuning delay cell setting.

**Parameters**

- base – USDHC peripheral base address.

**Return values**

CLK – Tuning Control and Status register value.

status\_t USDHC\_SetTuningDelay(USDHC\_Type \*base, uint32\_t preDelay, uint32\_t outDelay,  
uint32\_t postDelay)

The tuning delay cell setting.

**Parameters**

- base – USDHC peripheral base address.
- preDelay – Set the number of delay cells on the feedback clock between the feedback clock and CLK\_PRE.
- outDelay – Set the number of delay cells on the feedback clock between CLK\_PRE and CLK\_OUT.
- postDelay – Set the number of delay cells on the feedback clock between CLK\_OUT and CLK\_POST.

**Return values**

- kStatus\_Fail – config the delay setting fail
- kStatus\_Success – config the delay setting success

status\_t USDHC\_AdjustDelayForManualTuning(USDHC\_Type \*base, uint32\_t delay)

Adjusts delay for mannual tuning.

*Deprecated:*

Do not use this function. It has been superceded by USDHC\_SetTuingDelay

**Parameters**

- base – USDHC peripheral base address.
- delay – setting configuration

**Return values**

- kStatus\_Fail – config the delay setting fail
- kStatus\_Success – config the delay setting success

static inline void USDHC\_SetStandardTuningCounter(USDHC\_Type \*base, uint8\_t counter)

set tuning counter tuning.

**Parameters**

- base – USDHC peripheral base address.
- counter – tuning counter

**Return values**

- kStatus\_Fail – config the delay setting fail
- kStatus\_Success – config the delay setting success

```
void USDHC__EnableStandardTuning(USDHC_Type *base, uint32_t tuningStartTap, uint32_t step,
                                bool enable)
```

The enable standard tuning function. The standard tuning window and tuning counter using the default config tuning cmd is sent by the software, user need to check whether the tuning result can be used for SDR50, SDR104, and HS200 mode tuning.

**Parameters**

- base – USDHC peripheral base address.
- tuningStartTap – start tap
- step – tuning step
- enable – enable/disable flag

```
static inline uint32_t USDHC__GetExecuteStdTuningStatus(USDHC_Type *base)
```

Gets execute STD tuning status.

**Parameters**

- base – USDHC peripheral base address.

```
static inline uint32_t USDHC__CheckStdTuningResult(USDHC_Type *base)
```

Checks STD tuning result.

**Parameters**

- base – USDHC peripheral base address.

```
static inline uint32_t USDHC__CheckTuningError(USDHC_Type *base)
```

Checks tuning error.

**Parameters**

- base – USDHC peripheral base address.

```
void USDHC__EnableDDRMMode(USDHC_Type *base, bool enable, uint32_t nibblePos)
```

The enable/disable DDR mode.

**Parameters**

- base – USDHC peripheral base address.
- enable – enable/disable flag
- nibblePos – nibble position

```
static inline void USDHC__EnableHS400Mode(USDHC_Type *base, bool enable)
```

The enable/disable HS400 mode.

**Parameters**

- base – USDHC peripheral base address.
- enable – enable/disable flag

```
static inline void USDHC__ResetStrobeDLL(USDHC_Type *base)
```

Resets the strobe DLL.

**Parameters**

- base – USDHC peripheral base address.

```
static inline void USDHC__EnableStrobeDLL(USDHC_Type *base, bool enable)
```

Enables/disables the strobe DLL.

**Parameters**

- base – USDHC peripheral base address.

- enable – enable/disable flag

```
void USDHC_ConfigStrobeDLL(USDHC_Type *base, uint32_t delayTarget, uint32_t
                           updateInterval)
```

Configs the strobe DLL delay target and update interval.

#### Parameters

- base – USDHC peripheral base address.
- delayTarget – delay target
- updateInterval – update interval

```
static inline void USDHC_SetStrobeDllOverride(USDHC_Type *base, uint32_t delayTaps)
    Enables manual override for slave delay chain using STROBE_SLV_OVERRIDE_VAL.
```

#### Parameters

- base – USDHC peripheral base address.
- delayTaps – Valid delay taps range from 1 - 128 taps. A value of 0 selects tap 1, and a value of 0x7F selects tap 128.

```
static inline uint32_t USDHC_GetStrobeDLLStatus(USDHC_Type *base)
```

Gets the strobe DLL status.

#### Parameters

- base – USDHC peripheral base address.

```
void USDHC_SetDataConfig(USDHC_Type *base, usdhc_transfer_direction_t dataDirection,
                          uint32_t blockCount, uint32_t blockSize)
```

USDHC data configuration.

#### Parameters

- base – USDHC peripheral base address.
- dataDirection – Data direction, tx or rx.
- blockCount – Data block count.
- blockSize – Data block size.

```
void USDHC_TransferCreateHandle(USDHC_Type *base, usdhc_handle_t *handle, const
                                usdhc_transfer_callback_t *callback, void *userData)
```

Creates the USDHC handle.

#### Parameters

- base – USDHC peripheral base address.
- handle – USDHC handle pointer.
- callback – Structure pointer to contain all callback functions.
- userData – Callback function parameter.

```
status_t USDHC_TransferNonBlocking(USDHC_Type *base, usdhc_handle_t *handle,
                                    usdhc_adma_config_t *dmaConfig, usdhc_transfer_t
                                    *transfer)
```

Transfers the command/data using an interrupt and an asynchronous method.

This function sends a command and data and returns immediately. It doesn't wait for the transfer to complete or to encounter an error. The application must not call this API in multiple threads at the same time. Because of that this API doesn't support the re-entry mechanism.

---

**Note:** Call API USDHC\_TransferCreateHandle when calling this API.

---

### Parameters

- base – USDHC peripheral base address.
- handle – USDHC handle.
- dmaConfig – ADMA configuration.
- transfer – Transfer content.

### Return values

- kStatus\_InvalidArgument – Argument is invalid.
- kStatus\_USDHC\_BusyTransferring – Busy transferring.
- kStatus\_USDHC\_PrepareAdmaDescriptorFailed – Prepare ADMA descriptor failed.
- kStatus\_Success – Operate successfully.

*status\_t* USDHC\_TransferBlocking(USDHC\_Type \*base, *usdhc\_adma\_config\_t* \*dmaConfig, *usdhc\_transfer\_t* \*transfer)

Transfers the command/data using a blocking method.

This function waits until the command response/data is received or the USDHC encounters an error by polling the status flag.

The application must not call this API in multiple threads at the same time. Because this API doesn't support the re-entry mechanism.

---

**Note:** There is no need to call API USDHC\_TransferCreateHandle when calling this API.

---

### Parameters

- base – USDHC peripheral base address.
- dmaConfig – adma configuration
- transfer – Transfer content.

### Return values

- kStatus\_InvalidArgument – Argument is invalid.
- kStatus\_USDHC\_PrepareAdmaDescriptorFailed – Prepare ADMA descriptor failed.
- kStatus\_USDHC\_SendCommandFailed – Send command failed.
- kStatus\_USDHC\_TransferDataFailed – Transfer data failed.
- kStatus\_Success – Operate successfully.

*void* USDHC\_TransferHandleIRQ(USDHC\_Type \*base, *usdhc\_handle\_t* \*handle)

IRQ handler for the USDHC.

This function deals with the IRQs on the given host controller.

### Parameters

- base – USDHC peripheral base address.
- handle – USDHC handle.

FSL\_USDHC\_DRIVER\_VERSION

Driver version 2.8.5.

Enum \_usdhc\_status. USDHC status.

*Values:*

enumerator kStatus\_USDHC\_BusyTransferring

Transfer is on-going.

enumerator kStatus\_USDHC\_PrepareAdmaDescriptorFailed

Set DMA descriptor failed.

enumerator kStatus\_USDHC\_SendCommandFailed

Send command failed.

enumerator kStatus\_USDHC\_TransferDataFailed

Transfer data failed.

enumerator kStatus\_USDHC\_DMADDataAddrNotAlign

Data address not aligned.

enumerator kStatus\_USDHC\_ReTuningRequest

Re-tuning request.

enumerator kStatus\_USDHC\_TuningError

Tuning error.

enumerator kStatus\_USDHC\_NotSupport

Not support.

enumerator kStatus\_USDHC\_TransferDataComplete

Transfer data complete.

enumerator kStatus\_USDHC\_SendCommandSuccess

Transfer command complete.

enumerator kStatus\_USDHC\_TransferDMAComplete

Transfer DMA complete.

Enum \_usdhc\_capability\_flag. Host controller capabilities flag mask. .

*Values:*

enumerator kUSDHC\_SupportAdmaFlag

Support ADMA.

enumerator kUSDHC\_SupportHighSpeedFlag

Support high-speed.

enumerator kUSDHC\_SupportDmaFlag

Support DMA.

enumerator kUSDHC\_SupportSuspendResumeFlag

Support suspend/resume.

enumerator kUSDHC\_SupportV330Flag

Support voltage 3.3V.

enumerator kUSDHC\_SupportV300Flag

Support voltage 3.0V.

enumerator kUSDHC\_Support4BitFlag  
Flag in HTCAPBLT\_MBL's position, supporting 4-bit mode.

enumerator kUSDHC\_Support8BitFlag  
Flag in HTCAPBLT\_MBL's position, supporting 8-bit mode.

enumerator kUSDHC\_SupportDDR50Flag  
SD version 3.0 new feature, supporting DDR50 mode.

enumerator kUSDHC\_SupportSDR104Flag  
Support SDR104 mode.

enumerator kUSDHC\_SupportSDR50Flag  
Support SDR50 mode.

Enum \_usdhc\_wakeup\_event. Wakeup event mask. .

*Values:*

enumerator kUSDHC\_WakeupEventOnCardInt  
Wakeup on card interrupt.

enumerator kUSDHC\_WakeupEventOnCardInsert  
Wakeup on card insertion.

enumerator kUSDHC\_WakeupEventOnCardRemove  
Wakeup on card removal.

enumerator kUSDHC\_WakeupEventsAll  
All wakeup events

Enum \_usdhc\_reset. Reset type mask. .

*Values:*

enumerator kUSDHC\_ResetAll  
Reset all except card detection.

enumerator kUSDHC\_ResetCommand  
Reset command line.

enumerator kUSDHC\_ResetData  
Reset data line.

enumerator kUSDHC\_ResetTuning  
Reset tuning circuit.

enumerator kUSDHC\_ResetsAll  
All reset types

Enum \_usdhc\_transfer\_flag. Transfer flag mask.

*Values:*

enumerator kUSDHC\_EnableDmaFlag  
Enable DMA.

enumerator kUSDHC\_CommandTypeSuspendFlag  
Suspend command.



enumerator kUSDHC\_CommandTypeResumeFlag  
Resume command.

enumerator kUSDHC\_CommandTypeAbortFlag  
Abort command.

enumerator kUSDHC\_EnableBlockCountFlag  
Enable block count.

enumerator kUSDHC\_EnableAutoCommand12Flag  
Enable auto CMD12.

enumerator kUSDHC\_DataReadFlag  
Enable data read.

enumerator kUSDHC\_MultipleBlockFlag  
Multiple block data read/write.

enumerator kUSDHC\_EnableAutoCommand23Flag  
Enable auto CMD23.

enumerator kUSDHC\_ResponseLength136Flag  
136-bit response length.

enumerator kUSDHC\_ResponseLength48Flag  
48-bit response length.

enumerator kUSDHC\_ResponseLength48BusyFlag  
48-bit response length with busy status.

enumerator kUSDHC\_EnableCrcCheckFlag  
Enable CRC check.

enumerator kUSDHC\_EnableIndexCheckFlag  
Enable index check.

enumerator kUSDHC\_DataPresentFlag  
Data present flag.

Enum \_usdhc\_present\_status\_flag. Present status flag mask. .

*Values:*

enumerator kUSDHC\_CommandInhibitFlag  
Command inhibit.

enumerator kUSDHC\_DataInhibitFlag  
Data inhibit.

enumerator kUSDHC\_DataLineActiveFlag  
Data line active.

enumerator kUSDHC\_SdClockStableFlag  
SD bus clock stable.

enumerator kUSDHC\_WriteTransferActiveFlag  
Write transfer active.

enumerator kUSDHC\_ReadTransferActiveFlag  
Read transfer active.

enumerator kUSDHC\_BufferWriteEnableFlag  
Buffer write enable.

enumerator kUSDHC\_BufferReadEnableFlag  
Buffer read enable.

enumerator kUSDHC\_ReTuningRequestFlag  
Re-tuning request flag, only used for SDR104 mode.

enumerator kUSDHC\_DelaySettingFinishedFlag  
Delay setting finished flag.

enumerator kUSDHC\_CardInsertedFlag  
Card inserted.

enumerator kUSDHC\_CommandLineLevelFlag  
Command line signal level.

enumerator kUSDHC\_Data0LineLevelFlag  
Data0 line signal level.

enumerator kUSDHC\_Data1LineLevelFlag  
Data1 line signal level.

enumerator kUSDHC\_Data2LineLevelFlag  
Data2 line signal level.

enumerator kUSDHC\_Data3LineLevelFlag  
Data3 line signal level.

enumerator kUSDHC\_Data4LineLevelFlag  
Data4 line signal level.

enumerator kUSDHC\_Data5LineLevelFlag  
Data5 line signal level.

enumerator kUSDHC\_Data6LineLevelFlag  
Data6 line signal level.

enumerator kUSDHC\_Data7LineLevelFlag  
Data7 line signal level.

Enum \_usdhc\_interrupt\_status\_flag. Interrupt status flag mask. .

*Values:*

enumerator kUSDHC\_CommandCompleteFlag  
Command complete.

enumerator kUSDHC\_DataCompleteFlag  
Data complete.

enumerator kUSDHC\_BlockGapEventFlag  
Block gap event.

enumerator kUSDHC\_DmaCompleteFlag  
DMA interrupt.

enumerator kUSDHC\_BufferWriteReadyFlag  
Buffer write ready.

enumerator kUSDHC\_BufferReadReadyFlag  
Buffer read ready.

enumerator kUSDHC\_CardInsertionFlag  
Card inserted.

enumerator kUSDHC\_CardRemovalFlag  
Card removed.

enumerator kUSDHC\_CardInterruptFlag  
Card interrupt.

enumerator kUSDHC\_ReTuningEventFlag  
Re-Tuning event, only for SD3.0 SDR104 mode.

enumerator kUSDHC\_TuningPassFlag  
SDR104 mode tuning pass flag.

enumerator kUSDHC\_TuningErrorFlag  
SDR104 tuning error flag.

enumerator kUSDHC\_CommandTimeoutFlag  
Command timeout error.

enumerator kUSDHC\_CommandCrcErrorFlag  
Command CRC error.

enumerator kUSDHC\_CommandEndBitErrorFlag  
Command end bit error.

enumerator kUSDHC\_CommandIndexErrorFlag  
Command index error.

enumerator kUSDHC\_DataTimeoutFlag  
Data timeout error.

enumerator kUSDHC\_DataCrcErrorFlag  
Data CRC error.

enumerator kUSDHC\_DataEndBitErrorFlag  
Data end bit error.

enumerator kUSDHC\_AutoCommand12ErrorFlag  
Auto CMD12 error.

enumerator kUSDHC\_DmaErrorFlag  
DMA error.

enumerator kUSDHC\_CommandErrorFlag  
Command error

enumerator kUSDHC\_DataErrorFlag  
Data error

enumerator kUSDHC\_ErrorFlag  
All error

enumerator kUSDHC\_DataFlag  
Data interrupts

enumerator kUSDHC\_DataDMAFlag  
Data interrupts

enumerator kUSDHC\_CommandFlag

Command interrupts

enumerator kUSDHC\_CardDetectFlag

Card detection interrupts

enumerator kUSDHC\_SDR104TuningFlag

SDR104 tuning flag.

enumerator kUSDHC\_AllInterruptFlags

All flags mask

Enum \_usdhc\_auto\_command12\_error\_status\_flag. Auto CMD12 error status flag mask. .

*Values:*

enumerator kUSDHC\_AutoCommand12NotExecutedFlag

Not executed error.

enumerator kUSDHC\_AutoCommand12TimeoutFlag

Timeout error.

enumerator kUSDHC\_AutoCommand12EndBitErrorFlag

End bit error.

enumerator kUSDHC\_AutoCommand12CrcErrorFlag

CRC error.

enumerator kUSDHC\_AutoCommand12IndexErrorFlag

Index error.

enumerator kUSDHC\_AutoCommand12NotIssuedFlag

Not issued error.

Enum \_usdhc\_standard\_tuning. Standard tuning flag.

*Values:*

enumerator kUSDHC\_ExecuteTuning

Used to start tuning procedure.

enumerator kUSDHC\_TuningSampleClockSel

When **std\_tuning\_en** bit is set, this bit is used to select sampling clock.

Enum \_usdhc\_adma\_error\_status\_flag. ADMA error status flag mask. .

*Values:*

enumerator kUSDHC\_AdmaLenghMismatchFlag

Length mismatch error.

enumerator kUSDHC\_AdmaDescriptorErrorFlag

Descriptor error.

Enum \_usdhc\_adma\_error\_state. ADMA error state.

This state is the detail state when ADMA error has occurred.

*Values:*

enumerator kUSDHC\_AdmaErrorStateStopDma

Stop DMA, previous location set in the ADMA system address is errored address.

enumerator kUSDHC\_AdmaErrorStateFetchDescriptor

Fetch descriptor, current location set in the ADMA system address is errored address.

enumerator kUSDHC\_AdmaErrorStateChangeAddress

Change address, no DMA error has occurred.

enumerator kUSDHC\_AdmaErrorStateTransferData

Transfer data, previous location set in the ADMA system address is errored address.

enumerator kUSDHC\_AdmaErrorStateInvalidLength

Invalid length in ADMA descriptor.

enumerator kUSDHC\_AdmaErrorStateInvalidDescriptor

Invalid descriptor fetched by ADMA.

enumerator kUSDHC\_AdmaErrorState

ADMA error state

Enum \_usdhc\_force\_event. Force event bit position. .

*Values:*

enumerator kUSDHC\_ForceEventAutoCommand12NotExecuted

Auto CMD12 not executed error.

enumerator kUSDHC\_ForceEventAutoCommand12Timeout

Auto CMD12 timeout error.

enumerator kUSDHC\_ForceEventAutoCommand12CrcError

Auto CMD12 CRC error.

enumerator kUSDHC\_ForceEventEndBitError

Auto CMD12 end bit error.

enumerator kUSDHC\_ForceEventAutoCommand12IndexError

Auto CMD12 index error.

enumerator kUSDHC\_ForceEventAutoCommand12NotIssued

Auto CMD12 not issued error.

enumerator kUSDHC\_ForceEventCommandTimeout

Command timeout error.

enumerator kUSDHC\_ForceEventCommandCrcError

Command CRC error.

enumerator kUSDHC\_ForceEventCommandEndBitError

Command end bit error.

enumerator kUSDHC\_ForceEventCommandIndexError

Command index error.

enumerator kUSDHC\_ForceEventDataTimeout

Data timeout error.

enumerator kUSDHC\_ForceEventDataCrcError

Data CRC error.

enumerator kUSDHC\_ForceEventDataEndBitError  
Data end bit error.

enumerator kUSDHC\_ForceEventAutoCommand12Error  
Auto CMD12 error.

enumerator kUSDHC\_ForceEventCardInt  
Card interrupt.

enumerator kUSDHC\_ForceEventDmaError  
Dma error.

enumerator kUSDHC\_ForceEventTuningError  
Tuning error.

enumerator kUSDHC\_ForceEventsAll  
All force event flags mask.

enum \_\_usdhc\_transfer\_direction  
Data transfer direction.

*Values:*

enumerator kUSDHC\_TransferDirectionReceive  
USDHC transfer direction receive.

enumerator kUSDHC\_TransferDirectionSend  
USDHC transfer direction send.

enum \_\_usdhc\_data\_bus\_width  
Data transfer width.

*Values:*

enumerator kUSDHC\_DataBusWidth1Bit  
1-bit mode

enumerator kUSDHC\_DataBusWidth4Bit  
4-bit mode

enumerator kUSDHC\_DataBusWidth8Bit  
8-bit mode

enum \_\_usdhc\_endian\_mode  
Endian mode.

*Values:*

enumerator kUSDHC\_EndianModeBig  
Big endian mode.

enumerator kUSDHC\_EndianModeHalfWordBig  
Half word big endian mode.

enumerator kUSDHC\_EndianModeLittle  
Little endian mode.

enum \_\_usdhc\_dma\_mode  
DMA mode.

*Values:*

enumerator kUSDHC\_DmaModeSimple  
External DMA.

enumerator kUSDHC\_DmaModeAdma1  
ADMA1 is selected.

enumerator kUSDHC\_DmaModeAdma2  
ADMA2 is selected.

enumerator kUSDHC\_ExternalDMA  
External DMA mode selected.

Enum \_usdhc\_sdio\_control\_flag. SDIO control flag mask. .

*Values:*

enumerator kUSDHC\_StopAtBlockGapFlag  
Stop at block gap.

enumerator kUSDHC\_ReadWaitControlFlag  
Read wait control.

enumerator kUSDHC\_InterruptAtBlockGapFlag  
Interrupt at block gap.

enumerator kUSDHC\_ReadDoneNo8CLK  
Read done without 8 clk for block gap.

enumerator kUSDHC\_ExactBlockNumberReadFlag  
Exact block number read.

enum \_usdhc\_boot\_mode  
MMC card boot mode.

*Values:*

enumerator kUSDHC\_BootModeNormal  
Normal boot

enumerator kUSDHC\_BootModeAlternative  
Alternative boot

enum \_usdhc\_card\_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

enumerator kCARD\_CommandTypeEmpty  
Empty command

enum \_usdhc\_card\_response\_type  
The command response type.

Defines the command response type from card to host controller.

*Values:*

enumerator kCARD\_ResponseTypeNone

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

Enum `_usdhc_adma1_descriptor_flag`. The mask for the control/status field in ADMA1 descriptor.

*Values:*

enumerator kUSDHC\_Adma1DescriptorValidFlag

Valid flag.

enumerator kUSDHC\_Adma1DescriptorEndFlag

End flag.

enumerator kUSDHC\_Adma1DescriptorInterruptFlag

Interrupt flag.

enumerator kUSDHC\_Adma1DescriptorActivity1Flag

Activity 1 flag.

enumerator kUSDHC\_Adma1DescriptorActivity2Flag

Activity 2 flag.

enumerator kUSDHC\_Adma1DescriptorTypeNop

No operation.

enumerator kUSDHC\_Adma1DescriptorTypeTransfer

Transfer data.

enumerator kUSDHC\_Adma1DescriptorTypeLink

Link descriptor.

enumerator kUSDHC\_Adma1DescriptorTypeSetLength

Set data length.



Enum \_usdhc\_adma2\_descriptor\_flag. ADMA1 descriptor control and status mask.

*Values:*

enumerator kUSDHC\_\_Adma2DescriptorValidFlag

Valid flag.

enumerator kUSDHC\_\_Adma2DescriptorEndFlag

End flag.

enumerator kUSDHC\_\_Adma2DescriptorInterruptFlag

Interrupt flag.

enumerator kUSDHC\_\_Adma2DescriptorActivity1Flag

Activity 1 mask.

enumerator kUSDHC\_\_Adma2DescriptorActivity2Flag

Activity 2 mask.

enumerator kUSDHC\_\_Adma2DescriptorTypeNop

No operation.

enumerator kUSDHC\_\_Adma2DescriptorTypeReserved

Reserved.

enumerator kUSDHC\_\_Adma2DescriptorTypeTransfer

Transfer type.

enumerator kUSDHC\_\_Adma2DescriptorTypeLink

Link type.

Enum \_usdhc\_adma\_flag. ADMA descriptor configuration flag. .

*Values:*

enumerator kUSDHC\_\_AdmaDescriptorSingleFlag

Try to finish the transfer in a single ADMA descriptor. If transfer size is bigger than one ADMA descriptor's ability, new another descriptor for data transfer.

enumerator kUSDHC\_\_AdmaDescriptorMultipleFlag

Create multiple ADMA descriptors within the ADMA table, this is used for mmc boot mode specifically, which need to modify the ADMA descriptor on the fly, so the flag should be used combining with stop at block gap feature.

enum \_usdhc\_burst\_len

DMA transfer burst len config.

*Values:*

enumerator kUSDHC\_\_EnBurstLenForINCR

Enable burst len for INCR.

enumerator kUSDHC\_\_EnBurstLenForINCR4816

Enable burst len for INCR4/INCR8/INCR16.

enumerator kUSDHC\_\_EnBurstLenForINCR4816WRAP

Enable burst len for INCR4/8/16 WRAP.

Enum \_usdhc\_transfer\_data\_type. Transfer data type definition.

*Values:*

enumerator `kUSDHC_TransferDataNormal`

Transfer normal read/write data.

enumerator `kUSDHC_TransferDataTuning`

Transfer tuning data.

enumerator `kUSDHC_TransferDataBoot`

Transfer boot data.

enumerator `kUSDHC_TransferDataBootcontinuous`

Transfer boot data continuously.

typedef enum `_usdhc_transfer_direction` `usdhc_transfer_direction_t`

Data transfer direction.

typedef enum `_usdhc_data_bus_width` `usdhc_data_bus_width_t`

Data transfer width.

typedef enum `_usdhc_endian_mode` `usdhc_endian_mode_t`

Endian mode.

typedef enum `_usdhc_dma_mode` `usdhc_dma_mode_t`

DMA mode.

typedef enum `_usdhc_boot_mode` `usdhc_boot_mode_t`

MMC card boot mode.

typedef enum `_usdhc_card_command_type` `usdhc_card_command_type_t`

The command type.

typedef enum `_usdhc_card_response_type` `usdhc_card_response_type_t`

The command response type.

Defines the command response type from card to host controller.

typedef enum `_usdhc_burst_len` `usdhc_burst_len_t`

DMA transfer burst len config.

typedef uint32\_t `usdhc_adma1_descriptor_t`

Defines the ADMA1 descriptor structure.

typedef struct `_usdhc_adma2_descriptor` `usdhc_adma2_descriptor_t`

Defines the ADMA2 descriptor structure.

typedef struct `_usdhc_capability` `usdhc_capability_t`

USDHC capability information.

Defines a structure to save the capability information of USDHC.

typedef struct `_usdhc_boot_config` `usdhc_boot_config_t`

Data structure to configure the MMC boot feature.

typedef struct `_usdhc_config` `usdhc_config_t`

Data structure to initialize the USDHC.

typedef struct `_usdhc_command` `usdhc_command_t`

Card command descriptor.

Defines card command-related attribute.

typedef struct `_usdhc_adma_config` `usdhc_adma_config_t`

ADMA configuration.

`typedef struct _usdhc_scatter_gather_data_list usdhc_scatter_gather_data_list_t`

Card scatter gather data list.

Allow application register uncontinuous data buffer for data transfer.

`typedef struct _usdhc_scatter_gather_data usdhc_scatter_gather_data_t`

Card scatter gather data descriptor.

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

`typedef struct _usdhc_scatter_gather_transfer usdhc_scatter_gather_transfer_t`

usdhc scatter gather transfer.

`typedef struct _usdhc_data usdhc_data_t`

Card data descriptor.

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

`typedef struct _usdhc_transfer usdhc_transfer_t`

Transfer state.

`typedef struct _usdhc_handle usdhc_handle_t`

USDHC handle typedef.

`typedef struct _usdhc_transfer_callback usdhc_transfer_callback_t`

USDHC callback functions.

`typedef status_t (*usdhc_transfer_function_t)(USDHC_Type *base, usdhc_transfer_t *content)`

USDHC transfer function.

`typedef struct _usdhc_host usdhc_host_t`

USDHC host descriptor.

`USDHC_MAX_BLOCK_COUNT`

Maximum block count can be set one time.

`FSL_USDHC_ENABLE_SCATTER_GATHER_TRANSFER`

USDHC scatter gather feature control macro.

`USDHC_ADMA1_ADDRESS_ALIGN`

The alignment size for ADDRESS filed in ADMA1's descriptor.

`USDHC_ADMA1_LENGTH_ALIGN`

The alignment size for LENGTH field in ADMA1's descriptor.

`USDHC_ADMA2_ADDRESS_ALIGN`

The alignment size for ADDRESS field in ADMA2's descriptor.

`USDHC_ADMA2_LENGTH_ALIGN`

The alignment size for LENGTH filed in ADMA2's descriptor.

`USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT`

The bit shift for ADDRESS filed in ADMA1's descriptor.

| Address/page field     | Reserved | Attribute |      |    |     |     |       |
|------------------------|----------|-----------|------|----|-----|-----|-------|
| 31 12                  | 11 6     | 05        | 04   | 03 | 02  | 01  | 00    |
| address or data length | 000000   | Act2      | Act1 | 0  | Int | End | Valid |

| Act2 | Act1 | Comment         | 31-28              | 27-12       |
|------|------|-----------------|--------------------|-------------|
| 0    | 0    | No op           | Don't care         |             |
| 0    | 1    | Set data length | 0000               | Data Length |
| 1    | 0    | Transfer data   | Data address       |             |
| 1    | 1    | Link descriptor | Descriptor address |             |

USDHC\_ADMA1\_DESCRIPTOR\_ADDRESS\_MASK

The bit mask for ADDRESS field in ADMA1's descriptor.

USDHC\_ADMA1\_DESCRIPTOR\_LENGTH\_SHIFT

The bit shift for LENGTH field in ADMA1's descriptor.

USDHC\_ADMA1\_DESCRIPTOR\_LENGTH\_MASK

The mask for LENGTH field in ADMA1's descriptor.

USDHC\_ADMA1\_DESCRIPTOR\_MAX\_LENGTH\_PER\_ENTRY

The maximum value of LENGTH field in ADMA1's descriptor. Since the max transfer size ADMA1 support is 65535 which is indivisible by 4096, so to make sure a large data load transfer (>64KB) continuously (require the data address be always align with 4096), software will set the maximum data length for ADMA1 to (64 - 4)KB.

USDHC\_ADMA2\_DESCRIPTOR\_LENGTH\_SHIFT

The bit shift for LENGTH field in ADMA2's descriptor.

| Address field  | Length        | Reserved   | Attribute |      |    |     |     |       |
|----------------|---------------|------------|-----------|------|----|-----|-----|-------|
| 63 32          | 31 16         | 15 06      | 05        | 04   | 03 | 02  | 01  | 00    |
| 32-bit address | 16-bit length | 0000000000 | Act2      | Act1 | 0  | Int | End | Valid |

| Act2 | Act1 | Comment         | Operation                                                         |
|------|------|-----------------|-------------------------------------------------------------------|
| 0    | 0    | No op           | Don't care                                                        |
| 0    | 1    | Reserved        | Read this line and go to next one                                 |
| 1    | 0    | Transfer data   | Transfer data with address and length set in this descriptor line |
| 1    | 1    | Link descriptor | Link to another descriptor                                        |

USDHC\_ADMA2\_DESCRIPTOR\_LENGTH\_MASK

The bit mask for LENGTH field in ADMA2's descriptor.

USDHC\_ADMA2\_DESCRIPTOR\_MAX\_LENGTH\_PER\_ENTRY

The maximum value of LENGTH field in ADMA2's descriptor.

struct \_usdhc\_adma2\_descriptor

#include <fsl\_usdhc.h> Defines the ADMA2 descriptor structure.

### Public Members

uint32\_t attribute  
The control and status field.

uint32\_t address  
The address field.

struct \_\_usdhc\_\_capability  
*#include <fsl\_usdhc.h>* USDHC capability information.  
Defines a structure to save the capability information of USDHC.

### 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 block count can be set one time.

uint32\_t flags  
Capability flags to indicate the support information(*\_\_usdhc\_\_capability\_flag*).

struct \_\_usdhc\_\_boot\_config  
*#include <fsl\_usdhc.h>* Data structure to configure the MMC boot feature.

### Public Members

uint32\_t ackTimeoutCount  
Timeout value for the boot ACK. The available range is 0 ~ 15.

*usdhc\_boot\_mode\_t* bootMode  
Boot mode selection.

uint32\_t blockCount  
Stop at block gap value of automatic mode. Available range is 0 ~ 65535.

size\_t blockSize  
Block size.

bool enableBootAck  
Enable or disable boot ACK.

bool enableAutoStopAtBlockGap  
Enable or disable auto stop at block gap function in boot period.

struct \_\_usdhc\_\_config  
*#include <fsl\_usdhc.h>* Data structure to initialize the USDHC.

### Public Members

uint32\_t dataTimeout

Data timeout value.

usdhc\_endian\_mode\_t endianMode

Endian mode.

uint8\_t readWatermarkLevel

Watermark level for DMA read operation. Available range is 1 ~ 128.

uint8\_t writeWatermarkLevel

Watermark level for DMA write operation. Available range is 1 ~ 128.

struct \_\_usdhc\_command

*#include <fsl\_usdhc.h>* Card command descriptor.

Defines card command-related attribute.

### Public Members

uint32\_t index

Command index.

uint32\_t argument

Command argument.

usdhc\_card\_command\_type\_t type

Command type.

usdhc\_card\_response\_type\_t responseType

Command response type.

uint32\_t response[4U]

Response for this command.

uint32\_t responseErrorFlags

Response error flag, which need to check the command reponse.

uint32\_t flags

Cmd flags.

struct \_\_usdhc\_adma\_config

*#include <fsl\_usdhc.h>* ADMA configuration.

### Public Members

usdhc\_dma\_mode\_t dmaMode

DMA mode.

uint32\_t \*admaTable

ADMA table address, can't be null if transfer way is ADMA1/ADMA2.

uint32\_t admaTableWords

ADMA table length united as words, can't be 0 if transfer way is ADMA1/ADMA2.

struct \_\_usdhc\_scatter\_gather\_data\_list

*#include <fsl\_usdhc.h>* Card scatter gather data list.

Allow application register uncontinuous data buffer for data transfer.

struct \_\_usdhc\_scatter\_gather\_data

*#include <fsl\_usdhc.h>* Card scatter gather data descriptor.

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

### Public Members

bool enableAutoCommand12

Enable auto CMD12.

bool enableAutoCommand23

Enable auto CMD23.

bool enableIgnoreError

Enable to ignore error event to read/write all the data.

usdhc\_transfer\_direction\_t dataDirection

data direction

uint8\_t dataType

this is used to distinguish the normal/tuning/boot data.

size\_t blockSize

Block size.

usdhc\_scatter\_gather\_data\_list\_t sgData

scatter gather data

struct \_\_usdhc\_scatter\_gather\_transfer

*#include <fsl\_usdhc.h>* usdhc scatter gather transfer.

### Public Members

usdhc\_scatter\_gather\_data\_t \*data

Data to transfer.

usdhc\_command\_t \*command

Command to send.

struct \_\_usdhc\_data

*#include <fsl\_usdhc.h>* Card data descriptor.

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

### Public Members

bool enableAutoCommand12

Enable auto CMD12.

bool enableAutoCommand23

Enable auto CMD23.

bool enableIgnoreError

Enable to ignore error event to read/write all the data.

uint8\_t dataType

this is used to distinguish the normal/tuning/boot data.

size\_t blockSize

Block size.

uint32\_t blockCount

Block count.

uint32\_t \*rxData

Buffer to save data read.

const uint32\_t \*txData

Data buffer to write.

struct \_\_usdhc\_transfer

*#include <fsl\_usdhc.h>* Transfer state.

### Public Members

usdhc\_data\_t \*data

Data to transfer.

usdhc\_command\_t \*command

Command to send.

struct \_\_usdhc\_transfer\_callback

*#include <fsl\_usdhc.h>* USDHC callback functions.

### Public Members

void (\*CardInserted)(USDHC\_Type \*base, void \*userData)

Card inserted occurs when DAT3/CD pin is for card detect

void (\*CardRemoved)(USDHC\_Type \*base, void \*userData)

Card removed occurs

void (\*SdioInterrupt)(USDHC\_Type \*base, void \*userData)

SDIO card interrupt occurs

void (\*BlockGap)(USDHC\_Type \*base, void \*userData)

stopped at block gap event

void (\*TransferComplete)(USDHC\_Type \*base, usdhc\_handle\_t \*handle, status\_t status, void \*userData)

Transfer complete callback.

void (\*ReTuning)(USDHC\_Type \*base, void \*userData)

Handle the re-tuning.

struct \_\_usdhc\_handle

*#include <fsl\_usdhc.h>* USDHC handle.

Defines the structure to save the USDHC state information and callback function.

---

**Note:** All the fields except interruptFlags and transferredWords must be allocated by the user.

---



**Public Members**

*usdhc\_data\_t* \*volatile data

Transfer parameter. Data to transfer.

*usdhc\_command\_t* \*volatile command

Transfer parameter. Command to send.

volatile uint32\_t transferredWords

Transfer status. Words transferred by DATAPORT way.

*usdhc\_transfer\_callback\_t* callback

Callback function.

void \*userData

Parameter for transfer complete callback.

struct \_\_usdhc\_host

#include <fsl\_usdhc.h> USDHC host descriptor.

**Public Members**

USDHC\_Type \*base

USDHC peripheral base address.

uint32\_t sourceClock\_Hz

USDHC source clock frequency united in Hz.

*usdhc\_config\_t* config

USDHC configuration.

*usdhc\_capability\_t* capability

USDHC capability information.

*usdhc\_transfer\_function\_t* transfer

USDHC transfer function.

## 2.68 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.

## 2.69 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 = 0xFFFFFfU;
config->timeoutValue = 0xFFFFFfU;
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  $TWDCLK \cdot 256 \cdot 4$ . 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 `0xFFFFF` (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

`uint32_t windowValue`

Window value, set this to 0xFFFFFFFF if windowing is not in effect

`uint32_t timeoutValue`

Timeout value

`uint32_t warningValue`

Watchdog time counter value that will generate a warning interrupt. Set this to 0 for no warning

`uint32_t clockFreq_Hz`

Watchdog clock source frequency.

# Chapter 3

## Middleware

### 3.1 Boot

#### 3.1.1 MCUXpresso SDK : mcuxsdk-middleware-mcuboot\_opensource

##### Overview

This repository is a fork of MCUboot (<https://github.com/mcu-tools/mcuboot>) for MCUXpresso SDK delivery and it contains the components officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository (mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

##### Documentation

Overall details can be reviewed here: [MCUXpresso SDK Online Documentation](#)

Visit [MCUboot - Documentation](#) to review details on the contents in this sub-repo.

##### Setup

Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest [Getting Started with SDK - Detailed Installation Instructions](#)

##### Contribution

Contributions are not currently accepted. If the intended contribution is not related to NXP specific code, consider contributing directly to the upstream MCUboot project. Once this MCUboot fork is synchronized with the upstream project, such contributions will end up here as well. If the intended contribution is a bugfix or improvement for NXP porting layer or for code added or modified by NXP, please open an issue or contact NXP support.

---

## NXP Fork

This fork of MCUboot contains specific modifications and enhancements for NXP MCUXpresso SDK integration.

See *changelog* for details.

### 3.1.2 MCUboot



This is MCUboot version 2.1.0

MCUboot is a secure bootloader for 32-bits microcontrollers. It defines a common infrastructure for the bootloader and the system flash layout on microcontroller systems, and provides a secure bootloader that enables easy software upgrade.

MCUboot is not dependent on any specific operating system and hardware and relies on hardware porting layers from the operating system it works with. Currently, MCUboot works with the following operating systems and SoCs:

- [Zephyr](#)
- [Apache Mynewt](#)
- [Apache NuttX](#)
- [RIOT](#)
- [Mbed OS](#)
- [Espressif](#)
- [Cypress/Infineon](#)

RIOT is supported only as a boot target. We will accept any new port contributed by the community once it is good enough.

#### MCUboot How-tos

See the following pages for instructions on using MCUboot with different operating systems and SoCs:

- [Zephyr](#)
- [Apache Mynewt](#)
- [Apache NuttX](#)
- [RIOT](#)
- [Mbed OS](#)
- [Espressif](#)
- [Cypress/Infineon](#)

There are also instructions for the *Simulator*.



## Roadmap

The issues being planned and worked on are tracked using GitHub issues. To give your input, visit [MCUboot GitHub Issues](#).

## Source files

You can find additional documentation on the bootloader in the source files. For more information, use the following links:

- [boot/bootutil](#) - The core of the bootloader itself.
- [boot/boot\\_serial](#) - Support for serial upgrade within the bootloader itself.
- [boot/zephyr](#) - Port of the bootloader to Zephyr.
- [boot/mynewt](#) - Bootloader application for Apache Mynewt.
- [boot/nuttX](#) - Bootloader application and port of MCUboot interfaces for Apache NuttX.
- [boot/mbed](#) - Port of the bootloader to Mbed OS.
- [boot/espressif](#) - Bootloader application and MCUboot port for Espressif SoCs.
- [boot/cypress](#) - Bootloader application and MCUboot port for Cypress/Infineon SoCs.
- [imgtool](#) - A tool to securely sign firmware images for booting by MCUboot.
- [sim](#) - A bootloader simulator for testing and regression.

## Joining the project

Developers are welcome!

Use the following links to join or see more about the project:

- [Our developer mailing list](#)
- [Our Slack channel](#) [Get your invite](#)

## 3.2 Motor Control

### 3.2.1 FreeMASTER

*Communication Driver User Guide*

#### Introduction

**What is FreeMASTER?** [FreeMASTER](#) is a PC-based application developed by NXP for NXP customers. It is a versatile tool usable as a real-time monitor, visualization tool, and a graphical control panel of embedded applications based on the NXP processing units.

This document describes the embedded-side software driver which implements an interface between the application and the host PC. The interface covers the following communication:

- **Serial** UART communication either over plain RS232 interface or more typically over a USB-to-Serial either external or built in a debugger probe.
- **USB** direct connection to target microcontroller
- **CAN bus**

- **TCP/IP network** wired or WiFi
- **Segger J-Link RTT**
- **JTAG** debug port communication
- ...and all of the above also using a **Zephyr** generic drivers.

The driver also supports so-called “packet-driven BDM” interface which enables a protocol-based communication over a debugging port. The BDM stands for Background Debugging Module and its physical implementation is different on each platform. Some platforms leverage a semi-standard JTAG interface, other platforms provide a custom implementation called BDM. Regardless of the name, this debugging interface enables non-intrusive access to the memory space while the target CPU is running. For basic memory read and write operations, there is no communication driver required on the target when communicating with the host PC. Use this driver to get more advanced FreeMASTER protocol features over the BDM interface. The driver must be configured for the packet-driven BDM mode, in which the host PC uses the debugging interface to write serial command frames directly to the target memory buffer. The same method is then used to read response frames from that memory buffer.

Similar to “packet-driven BDM”, the FreeMASTER also supports a communication over [J-Link RTT](<https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/>) interface defined by SEGGER Microcontroller GmbH for ARM CortexM-based microcontrollers. This method also uses JTAG physical interface and enables high-speed real time communication to run over the same channel as used for application debugging.

**Driver version 3** This document describes version 3 of the FreeMASTER Communication Driver. This version features the implementation of the new Serial Protocol, which significantly extends the features and security of its predecessor. The new protocol internal number is v4 and its specification is available in the documentation accompanying the driver code.

Driver V3 is deployed to modern 32-bit MCU platforms first, so the portfolio of supported platforms is smaller than for the previous V2 versions. It is recommended to keep using the V2 driver for legacy platforms, such as S08, S12, ColdFire, or Power Architecture. Reach out to [FreeMASTER community](#) or to the local NXP representative with requests for more information or to port the V3 driver to legacy MCU devices.

Thanks to a layered approach, the new driver simplifies the porting of the driver to new UART, CAN or networking communication interfaces significantly. Users are encouraged to port the driver to more NXP MCU platforms and contribute the code back to NXP for integration into future releases. Existing code and low-level driver layers may be used as an example when porting to new targets.

**Note:** Using the FreeMASTER tool and FreeMASTER Communication Driver is only allowed in systems based on NXP microcontroller or microprocessor unit. Use with non-NXP MCU platforms is **not permitted** by the license terms.

**Target platforms** The driver implementation uses the following abstraction mechanisms which simplify driver porting and supporting new communication modules:

- **General CPU Platform** (see source code in the `src/platforms` directory). The code in this layer is only specific to native data type sizes and CPU architectures (for example; alignment-aware memory copy routines). This driver version brings two generic implementations of 32-bit platforms supporting both little-endian and big-endian architectures. There are also implementations customized for the 56F800E family of digital signal controllers and S12Z MCUs. **Zephyr** is treated as a specific CPU platform as it brings unified user configuration (Kconfig) and generic hardware device drivers. With Zephyr, the transport layer and low-level communication layers described below are configured automatically using Kconfig and Device Tree technologies.

- **Transport Communication Layer** - The Serial, CAN, Networking, PD-BDM, and other methods of transport logic are implemented as a driver layer called FMSTR\_TRANSPORT with a uniform API. A support of the Network transport also extends single-client modes of operation which are native for Serial, USB and CAN by a concept of multiple client sessions.
- **Low-level Communication Driver** - Each type of transport further defines a low-level API used to access the physical communication module. For example, the Serial transport defines a character-oriented API implemented by different serial communication modules like UART, LPUART, USART, and also USB-CDC. Similarly, the CAN transport defines a message-oriented API implemented by the FlexCAN or MCAN modules. Moreover, there are multiple different implementations for the same kind of communication peripherals. The difference between the implementation is in the way the low-level hardware registers are accessed. The *mcuxsdk* folder contains implementations which use MCUXpresso SDK drivers. These drivers should be used in applications based on the NXP MCUXpresso SDK. The “ampsdk” drivers target automotive-specific MCUs and their respective SDKs. The “dreg” implementations use a plain C-language access to hardware register addresses which makes it a universal and the most portable solution. In this case, users are encouraged to add more drivers for other communication modules or other respective SDKs and contribute the code back to NXP for integration.

The low-level drivers defined for the Networking transport enable datagram-oriented UDP and stream TCP communication. This implementation is demonstrated using the lwIP software stack but shall be portable to other TCP/IP stacks. It may sound surprisingly, but also the Segger J-Link RTT communication driver is linked to the Networking transport (RTT is stream oriented communication handled similarly to TCP).

**Replacing existing drivers** For all supported platforms, the driver described in this document replaces the V2 implementation and also older driver implementations that were available separately for individual platforms (PC Master SCI drivers).

**Clocks, pins, and peripheral initialization** The FreeMASTER communication driver is only responsible for runtime processing of the communication and must be integrated with an user application code to function properly. The user application code is responsible for general initialization of clock sources, pin multiplexers, and peripheral registers related to the communication speed. Such initialization should be done before calling the FMSTR\_Init function.

It is recommended to develop the user application using one of the Software Development Kits (SDKs) available from third parties or directly from NXP, such as MCUXpresso SDK, MCUXpresso IDE, and related tools. This approach simplifies the general configuration process significantly.

**MCUXpresso SDK** The MCUXpresso SDK is a software package provided by NXP which contains the device initialization code, linker files, and software drivers with example applications for the NXP family of MCUs. The MCUXpresso Config Tools may be used to generate the clock-setup and pin-multiplexer setup code suitable for the selected processor.

The MCUXpresso SDK also contains this FreeMASTER communication driver as a “middleware” component which may be downloaded along with the example applications from <https://mcuxpresso.nxp.com/en/welcome>.

**MCUXpresso SDK on GitHub** The FreeMASTER communication driver is also released as one of the middleware components of the MCUXpresso SDK on the GitHub. This release enables direct integration of the FreeMASTER source code Git repository into a target applications including Zephyr applications.

Related links:

- [The official FreeMASTER middleware repository.](#)

- [Online version of this document](#)

**FreeMASTER in Zephyr** The FreeMASTER middleware repository can be used with MCUXpresso SDK as well as a Zephyr module. Zephyr-specific samples which include examples of Kconfig and Device Tree configurations for Serial, USB and Network communications are available in separate repository. West manifest in this sample repository fetches the full Zephyr package including the FreeMASTER middleware repository used as a Zephyr module.

## Example applications

**MCUX SDK Example applications** There are several example applications available for each supported MCU platform.

- **fmstr\_uart** demonstrates a plain serial transmission, typically connecting to a computer's physical or virtual COM port. The typical transmission speed is 115200 bps.
- **fmstr\_can** demonstrates CAN bus communication. This requires a suitable CAN interface connected to the computer and interconnected with the target MCU using a properly terminated CAN bus. The typical transmission speed is 500 kbps. A FreeMASTER-over-CAN communication plug-in must be used.
- **fmstr\_usb\_cdc** uses an on-chip USB controller to implement a CDC communication class. It is connected directly to a computer's USB port and creates a virtual COM port device. The typical transmission speed is above 1 Mbps.
- **fmstr\_net** demonstrates the Network communication over UDP or TCP protocol. Existing examples use lwIP stack to implement the communication, but in general, it shall be possible to use any other TCP/IP stack to achieve the same functionality.
- **fmstr\_wifi** is the fmstr\_net application modified to use a WiFi network interface instead of a wired Ethernet connection.
- **fmstr\_rtt** demonstrates the communication over SEGGER J-Link RTT interface. Both fmstr\_net and fmstr\_rtt examples require the FreeMASTER TCP/UDP communication plug-in to be used on the PC host side.
- **fmstr\_eonce** uses the real-time data unit on the JTAG EOnCE module of the 56F800E family to implement pseudo-serial communication over the JTAG port. The typical transmission speed is around 10 kbps. This communication requires FreeMASTER JTAG/EOnCE communication plug-in.
- **fmstr\_pdbdm** uses JTAG or BDM debugging interface to access the target RAM directly while the CPU is running. Note that such approach can be used with any MCU application, even without any special driver code. The computer reads from and writes into the RAM directly without CPU intervention. The Packet-Driven BDM (PD-BDM) communication uses the same memory access to exchange command and response frames. With PD-BDM, the FreeMASTER tool is able to go beyond basic memory read/write operations and accesses also advanced features like Recorder, TSA, or Pipes. The typical transmission speed is around 10 kbps. A PD-BDM communication plug-in must be used in FreeMASTER and configured properly for the selected debugging interface. Note that this communication cannot be used while a debugging interface is used by a debugger session.
- **fmstr\_any** is a special example application which demonstrates how the NXP MCUXpresso Config Tools can be used to configure pins, clocks, peripherals, interrupts, and even the FreeMASTER "middleware" driver features in a graphical and user friendly way. The user can switch between the Serial, CAN, and other ways of communication and generate the required initialization code automatically.

**Zephyr sample applications** Zephyr sample applications demonstrate Kconfig and Device Tree configuration which configure the FreeMASTER middleware module for a selected communication option (Serial, CAN, Network or RTT).

Refer to *readme.md* files in each sample directory for description of configuration options required to implement FreeMASTER connectivity.

## Description

This section shows how to add the FreeMASTER Communication Driver into application and how to configure the connection to the FreeMASTER visualization tool.

**Features** The FreeMASTER driver implements the FreeMASTER protocol V4 and provides the following features which may be accessed using the FreeMASTER visualization tool:

- Read/write access to any memory location on the target.
- Optional password protection of the read, read/write, and read/write/flash access levels.
- Atomic bit manipulation on the target memory (bit-wise write access).
- Optimal size-aligned access to memory which is also suitable to access the peripheral register space.
- Oscilloscope access—real-time access to target variables. The sample rate may be limited by the communication speed.
- Recorder— access to the fast transient recorder running on the board as a part of the FreeMASTER driver. The sample rate is only limited by the MCU CPU speed. The length of the data recorded depends on the amount of available memory.
- Multiple instances of Oscilloscopes and Recorders without the limitation of maximum number of variables.
- Application commands—high-level message delivery from the PC to the application.
- TSA tables—describing the data types, variables, files, or hyperlinks exported by the target application. The TSA newly supports also non-memory mapped resources like external EEPROM or SD Card files.
- Pipes—enabling the buffered stream-oriented data exchange for a general-purpose terminal-like communication, diagnostic data streaming, or other data exchange.

The FreeMASTER driver features:

- Full FreeMASTER protocol V4 implementation with a new V4 style of CRC used.
- Layered approach supporting Serial, CAN, Network, PD-BDM, and other transports.
- Layered low-level Serial transport driver architecture enabling to select UART, LPUART, USART, and other physical implementations of serial interfaces, including USB-CDC.
- Layered low-level CAN transport driver architecture enabling to select FlexCAN, msCAN, MCAN, and other physical implementations of the CAN interface.
- Layered low-level Networking transport enabling to select TCP, UDP or J-Link RTT communication.
- TSA support to write-protect memory regions or individual variables and to deny the access to the unsafe memory.
- The pipe callback handlers are invoked whenever new data is available for reading from the pipe.

- Two Serial Single-Wire modes of operation are enabled. The “external” mode has the RX and TX shorted on-board. The “true” single-wire mode interconnects internally when the MCU or UART modules support it.

The following sections briefly describe all FreeMASTER features implemented by the driver. See the PC-based FreeMASTER User Manual for more details on how to use the features to monitor, tune, or control an embedded application.

**Board Detection** The FreeMASTER protocol V4 defines the standard set of configuration values which the host PC tool reads to identify the target and to access other target resources properly. The configuration includes the following parameters:

- Version of the driver and the version of the protocol implemented.
- MTU as the Maximum size of the Transmission Unit (for example; communication buffer size).
- Application name, description, and version strings.
- Application build date and time as a string.
- Target processor byte ordering (little/big endian).
- Protection level that requires password authentication.
- Number of the Recorder and Oscilloscope instances.
- RAM Base Address for optimized memory access commands.

**Memory Read** This basic feature enables the host PC to read any data memory location by specifying the address and size of the required memory area. The device response frame must be shorter than the MTU to fit into the outgoing communication buffer. To read a device memory of any size, the host uses the information retrieved during the Board Detection and splits the large-block request to multiple partial requests.

The driver uses size-aligned operations to read the target memory (for example; uses proper read-word instruction when an address is aligned to 4 bytes).

**Memory Write** Similarly to the Memory Read operation, the Memory Write feature enables to write to any RAM memory location on the target device. A single write command frame must be shorter than the MTU to fit into the target communication buffer. Larger requests must be split into smaller ones.

The driver uses size-aligned operations to write to the target memory (for example; uses proper write-word instruction when an address is aligned to 4 bytes).

**Masked Memory Write** To implement the write access to a single bit or a group of bits of target variables, the Masked Memory Write feature is available in the FreeMASTER protocol and it is supported by the driver using the Read-Modify-Write approach.

Be careful when writing to bit fields of volatile variables that are also modified in an application interrupt. The interrupt may be serviced in the middle of a read-modify-write operation and it may cause data corruption.

**Oscilloscope** The protocol and driver enables any number of variables to be read at once with a single request from the host. This feature is called Oscilloscope and the FreeMASTER tool uses it to display a real-time graph of variable values.

The driver can be configured to support any number of Oscilloscope instances and enable simultaneously running graphs to be displayed on the host computer screen.



**Recorder** The protocol enables the host to select target variables whose values are then periodically recorded into a dedicated on-board memory buffer. After such data sampling stops (either on a host request or by evaluating a threshold-crossing condition), the data buffer is downloaded to the host and displayed as a graph. The data sampling rate is not limited by the speed of the communication line, so it enables displaying the variable transitions in a very high resolution.

The driver can be configured to support multiple Recorder instances and enable multiple recorder graphs to be displayed on the host screen. Having multiple recorders also enables setting the recording point differently for each instance. For example; one instance may be recording data in a general timer interrupt while another instance may record at a specific control algorithm time in the PWM interrupt.

**TSA** With the TSA feature, data types and variables can be described directly in the application source code. Such information is later provided to the FreeMASTER tool which may use it instead of reading symbol data from the application ELF executable file.

The information is encoded as so-called TSA tables which become direct part of the application code. The TSA tables contain descriptors of variables that shall be visible to the host tool. The descriptors can describe the memory areas by specifying the address and size of the memory block or more conveniently using the C variable names directly. Different set of TSA descriptors can be used to encode information about the structure types, unions, enumerations, or arrays.

The driver also supports special types of TSA table entries to describe user resources like external EEPROM and SD Card files, memory-mapped files, virtual directories, web URL hyperlinks, and constant enumerations.

**TSA Safety** When the TSA is enabled in the application, the TSA Safety can be enabled and validate the memory accesses directly by the embedded-side driver. When the TSA Safety is turned on, any memory request received from the host is validated and accepted only if it belongs to a TSA-described object. The TSA entries can be declared as Read-Write or Read-Only so that the driver can actively deny the write access to the Read-Only objects.

**Application commands** The Application Commands are high-level messages that can be delivered from the PC Host to the embedded application for further processing. The embedded application can either poll the status, or be called back when a new Application Command arrives to be processed. After the embedded application acknowledges that the command is handled, the host receives the Result Code and reads the other return data from memory. Both the Application Commands and the Result Codes are specific to a given application and it is user's responsibility to define them. The FreeMASTER protocol and the FreeMASTER driver only implement the delivery channel and a set of API calls to enable the Application Command processing in general.

**Pipes** The Pipes enable buffered and stream-oriented data exchange between the PC Host and the target application. Any pipe can be written to and read from at both ends (either on the PC or the MCU). The data transmission is acknowledged using the special FreeMASTER protocol commands. It is guaranteed that the data bytes are delivered from the writer to the reader in a proper order and without losses.

**Serial single-wire operation** The MCU Serial Communication Driver natively supports normal dual-wire operation. Because the protocol is half-duplex only, the driver can also operate in two single-wire modes:

- “External” single-wire operation where the Receiver and Transmitter pins are shorted on the board. This mode is supported by default in the MCU driver because the Receiver and Transmitter units are enabled or disabled whenever needed. It is also easy to extend this operation for the RS485 communication.

- “True” single-wire mode which uses only a single pin and the direction switching is made by the UART module. This mode of operation must be enabled by defining the FMSTR\_SERIAL\_SINGLEWIRE configuration option.

**Multi-session support** With networking interface it is possible for multiple clients to access the target MCU simultaneously. Reading and writing of target memory is processed atomically so there is no risk of data corruption. The state-full resources such as Recorders or Oscilloscopes are locked to a client session upon first use and access is denied to other clients until lock is released..

## Zephyr-specific

**Dedicated communication task** FreeMASTER communication may run isolated in a dedicated task. The task automates the FMSTR\_Init and FMSTR\_Poll calls together with periodic activities enabling the FreeMASTER UI to fetch information about tasks and CPU utilization. The task can be started automatically or manually, and it must be assigned a priority to be able to react on interrupts and other communication events. Refer to Zephyr FreeMASTER sample applications which all use this communication task.

**Zephyr shell and logging over FreeMASTER pipe** FreeMASTER implements a shell backend which may use FreeMASTER pipe as a I/O terminal and logging output. Refer to Zephyr FreeMASTER sample applications which all use this feature.

**Automatic TSA tables** TSA tables can be declared as “automatic” in Zephyr which make them automatically registered in the table list. This may be very useful when there are many TSA tables or when the tables are defined in different (often unrelated) libraries linked together. In this case user does not need to build a list of all tables manually.

**Driver files** The driver source files can be found in a top-level src folder, further divided into the sub-folders:

- **src/platforms** platform-specific folder—one folder exists for each supported processor platform (for example; 32-bit Little Endian platform). Each such folder contains a platform header file with data types and a code which implements the potentially platform-specific operations, such as aligned memory access.
- **src/common** folder—contains the common driver source files shared by the driver for all supported platforms. All the .c files must be added to the project, compiled, and linked together with the application.
  - *freemaster.h* - master driver header file, which declares the common data types, macros, and prototypes of the FreeMASTER driver API functions.
  - *freemaster\_cfg.h.example* - this file can serve as an example of the FreeMASTER driver configuration file. Save this file into a project source code folder and rename it to *freemaster\_cfg.h*. The FreeMASTER driver code includes this file to get the project-specific configuration options and to optimize the compilation of the driver.
  - *freemaster\_defcfg.h* - defines the default values for each FreeMASTER configuration option if the option is not set in the *freemaster\_cfg.h* file.
  - *freemaster\_protocol.h* - defines the FreeMASTER protocol constants used internally by the driver.
  - *freemaster\_protocol.c* - implements the FreeMASTER protocol decoder and handles the basic Get Configuration Value, Memory Read, and Memory Write commands.



- *freemaster\_rec.c* - handles the Recorder-specific commands and implements the Recorder sampling and triggering routines. When the Recorder is disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
- *freemaster\_scope.c* - handles the Oscilloscope-specific commands. If the Oscilloscope is disabled by the FreeMASTER driver configuration file, this file compiles as void.
- *freemaster\_pipes.c* - implements the Pipes functionality when the Pipes feature is enabled.
- *freemaster\_appcmd.c* - handles the communication commands used to deliver and execute the Application Commands within the context of the embedded application. When the Application Commands are disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
- *freemaster\_tsa.c* - handles the commands specific to the TSA feature. This feature enables the FreeMASTER host tool to obtain the TSA memory descriptors declared in the embedded application. If the TSA is disabled by the FreeMASTER driver configuration file, this file compiles as void.
- *freemaster\_tsa.h* - contains the declaration of the macros used to define the TSA memory descriptors. This file is indirectly included into the user application code (via *freemaster.h*).
- *freemaster\_sha.c* - implements the SHA-1 hash code used in the password authentication algorithm.
- *freemaster\_private.h* - contains the declarations of functions and data types used internally in the driver. It also contains the C pre-processor statements to perform the compile-time verification of the user configuration provided in the *freemaster\_cfg.h* file.
- *freemaster\_serial.c* - implements the serial protocol logic including the CRC, FIFO queuing, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a character-oriented API exported by the specific low-level driver.
- *freemaster\_serial.h* - defines the low-level character-oriented Serial API.
- *freemaster\_can.c* - implements the CAN protocol logic including the CAN message preparation, signalling using the first data byte in the CAN frame, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a message-oriented API exported by the specific low-level driver.
- *freemaster\_can.h* - defines the low-level message-oriented CAN API.
- *freemaster\_net.c* - implements the Network protocol transport logic including multiple session management code.
- *freemaster\_net.h* - definitions related to the Network transport.
- *freemaster\_pdbdm.c* - implements the packet-driven BDM communication buffer and other communication-related operations.
- *freemaster\_utils.c* - aligned memory copy routines, circular buffer management and other utility functions
- *freemaster\_utils.h* - definitions related to utility code.
- ***src/drivers/[sdk]/serial*** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster\_serial\_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the UART, LPUART, USART, and other kinds of Serial communication modules.

- **src/drivers/[sdk]/can** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster\_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the FlexCAN, msCAN, MCAN, and other kinds of CAN communication modules.
- **src/drivers/[sdk]/network** - contains low-level code adapting the FreeMASTER Network transport to an underlying TCP/IP or RTT stack.
  - *freemaster\_net\_lwip\_tcp.c* and *\_udp.c* - default networking implementation of TCP and UDP transports using lwIP stack.
  - *freemaster\_net\_segger\_rtt.c* - implementation of network transport using Segger J-Link RTT interface

**Driver configuration** The driver is configured using a single header file (*freemaster\_cfg.h*). Create this file and save it together with other project source files before compiling the driver code. All FreeMASTER driver source files include the *freemaster\_cfg.h* file and use the macros defined here for the conditional and parameterized compilation. The C compiler must locate the configuration file when compiling the driver files. Typically, it can be achieved by putting this file into a folder where the other project-specific included files are stored.

As a starting point to create the configuration file, get the *freemaster\_cfg.h.example* file, rename it to *freemaster\_cfg.h*, and save it into the project area.

**Note:** It is NOT recommended to leave the *freemaster\_cfg.h* file in the FreeMASTER driver source code folder. The configuration file must be placed at a project-specific location, so that it does not affect the other applications that use the same driver.

**Configurable items** This section describes the configuration options which can be defined in *freemaster\_cfg.h*.

### Interrupt modes

```
#define FMSTR_LONG_INTR    [0|1]
#define FMSTR_SHORT_INTR  [0|1]
#define FMSTR_POLL_DRIVEN [0|1]
```

**Value Type** boolean (0 or 1)

**Description** Exactly one of the three macros must be defined to non-zero. The others must be defined to zero or left undefined. The non-zero-defined constant selects the interrupt mode of the driver. See [Driver interrupt modes](#).

- FMSTR\_LONG\_INTR — long interrupt mode
- FMSTR\_SHORT\_INTR — short interrupt mode
- FMSTR\_POLL\_DRIVEN — poll-driven mode

**Note:** Some options may not be supported by all communication interfaces. For example, the FMSTR\_SHORT\_INTR option is not supported by the USB\_CDC interface.

### Protocol transport

```
#define FMSTR_TRANSPORT [identifier]
```

**Value Type** Driver identifiers are structure instance names defined in FreeMASTER source code. Specify one of existing instances to make use of the protocol transport.

**Description** Use one of the pre-defined constants, as implemented by the FreeMASTER code. The current driver supports the following transports:

- **FMSTR\_SERIAL** - serial communication protocol
- **FMSTR\_CAN** - using CAN communication
- **FMSTR\_PDBDM** - using packet-driven BDM communication
- **FMSTR\_NET** - network communication using TCP or UDP protocol

**Serial transport** This section describes configuration parameters used when serial transport is used:

```
#define FMSTR_TRANSPORT FMSTR_SERIAL
```

**FMSTR\_SERIAL\_DRV** Select what low-level driver interface will be used when implementing the Serial communication.

```
#define FMSTR_SERIAL_DRV [identifier]
```

**Value Type** Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing serial driver instances.

**Description** When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/serial* implementation):

- **FMSTR\_SERIAL\_MCUX\_UART** - UART driver
- **FMSTR\_SERIAL\_MCUX\_LPUART** - LPUART driver
- **FMSTR\_SERIAL\_MCUX\_USART** - USART driver
- **FMSTR\_SERIAL\_MCUX\_MINIUSART** - miniUSART driver
- **FMSTR\_SERIAL\_MCUX\_QSCI** - DSC QSCI driver
- **FMSTR\_SERIAL\_MCUX\_USB** - USB/CDC class driver (also see code in the */support/mcuxsdk\_usb* folder)
- **FMSTR\_SERIAL\_56F800E\_EONCE** - DSC JTAG EOnCE driver

Other SDKs or BSPs may define custom low-level driver interface structure which may be used as **FMSTR\_SERIAL\_DRV**. For example:

- **FMSTR\_SERIAL\_DREG\_UART** - demonstrates the low-level interface implemented without the MCUXpresso SDK and using direct access to peripheral registers.

## FMSTR\_SERIAL\_BASE

```
#define FMSTR_SERIAL_BASE [address|symbol]
```

**Value Type** Optional address value (numeric or symbolic)

**Description** Specify the base address of the UART, LPUART, USART, or other serial peripheral module to be used for the communication. This value is not defined by default. User application should call `FMSTR_SetSerialBaseAddress()` to select the peripheral module.

#### FMSTR\_COMM\_BUFFER\_SIZE

```
#define FMSTR_COMM_BUFFER_SIZE [number]
```

**Value Type** 0 or a value in range 32...255

**Description** Specify the size of the communication buffer to be allocated by the driver. Default value, which suits all driver features, is used when this option is defined as 0.

#### FMSTR\_COMM\_QUEUE\_SIZE

```
#define FMSTR_COMM_QUEUE_SIZE [number]
```

**Value Type** Value in range 0...255

**Description** Specify the size of the FIFO receiver queue used to quickly receive and store characters in the `FMSTR_SHORT_INTR` interrupt mode. The default value is 32 B.

#### FMSTR\_SERIAL\_SINGLEWIRE

```
#define FMSTR_SERIAL_SINGLEWIRE [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Set to non-zero to enable the “True” single-wire mode which uses a single MCU pin to communicate. The low-level driver enables the pin direction switching when the MCU peripheral supports it.

**CAN Bus transport** This section describes configuration parameters used when CAN transport is used:

```
#define FMSTR_TRANSPORT FMSTR_CAN
```

**FMSTR\_CAN\_DRV** Select what low-level driver interface will be used when implementing the CAN communication.

```
#define FMSTR_CAN_DRV [identifier]
```

**Value Type** Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing CAN driver instances.

**Description** When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/can implementation*):

- **FMSTR\_CAN\_MCUX\_FLEXCAN** - FlexCAN driver
- **FMSTR\_CAN\_MCUX\_MCAN** - MCAN driver
- **FMSTR\_CAN\_MCUX\_MSCAN** - msCAN driver
- **FMSTR\_CAN\_MCUX\_DSCFLEXCAN** - DSC FlexCAN driver
- **FMSTR\_CAN\_MCUX\_DSCMSCAN** - DSC msCAN driver

Other SDKs or BSPs may define the custom low-level driver interface structure which may be used as FMSTR\_CAN\_DRV.

### FMSTR\_CAN\_BASE

```
#define FMSTR_CAN_BASE [address|symbol]
```

**Value Type** Optional address value (numeric or symbolic)

**Description** Specify the base address of the FlexCAN, msCAN, or other CAN peripheral module to be used for the communication. This value is not defined by default. User application should call FMSTR\_SetCanBaseAddress() to select the peripheral module.

### FMSTR\_CAN\_CMDID

```
#define FMSTR_CAN_CMDID [number]
```

**Value Type** CAN identifier (11-bit or 29-bit number)

**Description** CAN message identifier used for FreeMASTER commands (direction from PC Host tool to target application). When declaring 29-bit identifier, combine the numeric value with FMSTR\_CAN\_EXTID bit. Default value is 0x7AA.

### FMSTR\_CAN\_RSPID

```
#define FMSTR_CAN_RSPID [number]
```

**Value Type** CAN identifier (11-bit or 29-bit number)

**Description** CAN message identifier used for responding messages (direction from target application to PC Host tool). When declaring 29-bit identifier, combine the numeric value with FMSTR\_CAN\_EXTID bit. Note that both *CMDID* and *RSPID* values may be the same. Default value is 0x7AA.

### FMSTR\_FLEXCAN\_TXMB

```
#define FMSTR_FLEXCAN_TXMB [number]
```

**Value Type** Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

**Description** Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame transmission. Default value is 0.

#### FMSTR\_FLEXCAN\_RXMB

```
#define FMSTR_FLEXCAN_RXMB [number]
```

**Value Type** Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

**Description** Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame reception. Note that the FreeMASTER driver may also operate with a common message buffer used by both TX and RX directions. Default value is 1.

**Network transport** This section describes configuration parameters used when Network transport is used:

```
#define FMSTR_TRANSPORT FMSTR_NET
```

**FMSTR\_NET\_DRV** Select network interface implementation.

```
#define FMSTR_NET_DRV [identifier]
```

**Value Type** Identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing NET driver instances.

**Description** When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/network implementation*):

- **FMSTR\_NET\_LWIP\_TCP** - TCP communication using lwIP stack
- **FMSTR\_NET\_LWIP\_UDP** - UDP communication using lwIP stack
- **FMSTR\_NET\_SEGGER\_RTT** - Communication using SEGGER J-Link RTT interface

Other SDKs or BSPs may define the custom networking interface which may be used as FMSTR\_CAN\_DRV.

Add another row below:

#### FMSTR\_NET\_PORT

```
#define FMSTR_NET_PORT [number]
```

**Value Type** TCP or UDP port number (short integer)

**Description** Specifies the server port number used by TCP or UDP protocols.

#### FMSTR\_NET\_BLOCKING\_TIMEOUT

```
#define FMSTR_NET_BLOCKING_TIMEOUT [number]
```

**Value Type** Timeout as number of milliseconds

**Description** This value specifies a timeout in milliseconds for which the network socket operations may block the execution inside *FMSTR\_Poll*. This may be set high (e.g. 250) when a dedicated RTOS task is used to handle FreeMASTER protocol polling. Set to a lower value when the polling task is also responsible for other operations. Set to 0 to attempt to use non-blocking socket operations.

### FMSTR\_NET\_AUTODISCOVERY

```
#define FMSTR_NET_AUTODISCOVERY [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** This option enables the FreeMASTER driver to use a separate UDP socket to broadcast auto-discovery messages to network. This helps the FreeMASTER tool to discover the target device address, port and protocol options.

### Debugging options

#### FMSTR\_DISABLE

```
#define FMSTR_DISABLE [0|1]
```

**Value Type** boolean (0 or 1)

**Description** Define as non-zero to disable all FreeMASTER features, exclude the driver code from build, and compile all its API functions empty. This may be useful to remove FreeMASTER without modifying any application source code. Default value is 0 (false).

#### FMSTR\_DEBUG\_TX

```
#define FMSTR_DEBUG_TX [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to enable the driver to periodically transmit test frames out on the selected communication interface (SCI or CAN). With the debug transmission enabled, it is simpler to detect problems in the baudrate or other communication configuration settings.

The test frames are transmitted until the first valid command frame is received from the PC Host tool. The test frame is a valid error status frame, as defined by the protocol format. On the serial line, the test frame consists of three printable characters (+©W) which are easy to capture using the serial terminal tools.

This feature requires the FMSTR\_Poll() function to be called periodically. Default value is 0 (false).

**FMSTR\_APPLICATION\_STR**

```
#define FMSTR_APPLICATION_STR
```

**Value Type** String.

**Description** Name of the application visible in FreeMASTER host application.

**Memory access****FMSTR\_USE\_READMEM**

```
#define FMSTR_USE_READMEM [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the Memory Read command and enable FreeMASTER to have read access to memory and variables. The access can be further restricted by using a TSA feature.  
Default value is 1 (true).

**FMSTR\_USE\_WRITEMEM**

```
#define FMSTR_USE_WRITEMEM [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the Memory Write command.  
The default value is 1 (true).

**Oscilloscope options****FMSTR\_USE\_SCOPE**

```
#define FMSTR_USE_SCOPE [number]
```

**Value Type** Integer number.

**Description** Number of Oscilloscope instances to be supported. Set to 0 to disable the Oscilloscope feature.  
Default value is 0.

**FMSTR\_MAX\_SCOPE\_VARS**

```
#define FMSTR_MAX_SCOPE_VARS [number]
```



**Value Type** Integer number larger than 2.

**Description** Number of variables to be supported by each Oscilloscope instance.  
Default value is 8.

### Recorder options

#### FMSTR\_USE\_RECORDER

```
#define FMSTR_USE_RECORDER [number]
```

**Value Type** Integer number.

**Description** Number of Recorder instances to be supported. Set to 0 to disable the Recorder feature.  
Default value is 0.

#### FMSTR\_REC\_BUFF\_SIZE

```
#define FMSTR_REC_BUFF_SIZE [number]
```

**Value Type** Integer number larger than 2.

**Description** Defines the size of the memory buffer used by the Recorder instance #0.  
Default: not defined, user shall call 'FMSTR\_RecorderCreate()' API function to specify this parameter in run time.

#### FMSTR\_REC\_TIMEBASE

```
#define FMSTR_REC_TIMEBASE [time specification]
```

**Value Type** Number (nanoseconds time).

**Description** Defines the base sampling rate in nanoseconds (sampling speed) Recorder instance #0.

Use one of the following macros:

- FMSTR\_REC\_BASE\_SECONDS(x)
- FMSTR\_REC\_BASE\_MILLISEC(x)
- FMSTR\_REC\_BASE\_MICROSEC(x)
- FMSTR\_REC\_BASE\_NANOSEC(x)

Default: not defined, user shall call 'FMSTR\_RecorderCreate()' API function to specify this parameter in run time.

#### FMSTR\_REC\_FLOAT\_TRIG

```
#define FMSTR_REC_FLOAT_TRIG [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the floating-point triggering. Be aware that floating-point triggering may grow the code size by linking the floating-point standard library. Default value is 0 (false).

### Application Commands options

#### FMSTR\_USE\_APPCMD

```
#define FMSTR_USE_APPCMD [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the Application Commands feature. Default value is 0 (false).

#### FMSTR\_APPCMD\_BUFF\_SIZE

```
#define FMSTR_APPCMD_BUFF_SIZE [size]
```

**Value Type** Numeric buffer size in range 1..255

**Description** The size of the Application Command data buffer allocated by the driver. The buffer stores the (optional) parameters of the Application Command which waits to be processed.

#### FMSTR\_MAX\_APPCMD\_CALLS

```
#define FMSTR_MAX_APPCMD_CALLS [number]
```

**Value Type** Number in range 0..255

**Description** The number of different Application Commands that can be assigned a callback handler function using FMSTR\_RegisterAppCmdCall(). Default value is 0.

### TSA options

#### FMSTR\_USE\_TSA

```
#define FMSTR_USE_TSA [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable the FreeMASTER TSA feature to be used. With this option enabled, the TSA tables defined in the applications are made available to the FreeMASTER host tool. Default value is 0 (false).

#### FMSTR\_USE\_TSA\_SAFETY

```
#define FMSTR_USE_TSA_SAFETY [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable the memory access validation in the FreeMASTER driver. With this option, the host tool is not able to access the memory which is not described by at least one TSA descriptor. Also a write access is denied for objects defined as read-only in TSA tables. Default value is 0 (false).

#### FMSTR\_USE\_TSA\_INROM

```
#define FMSTR_USE_TSA_INROM [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Declare all TSA descriptors as *const*, which enables the linker to put the data into the flash memory. The actual result depends on linker settings or the linker commands used in the project. Default value is 0 (false).

#### FMSTR\_USE\_TSA\_DYNAMIC

```
#define FMSTR_USE_TSA_DYNAMIC [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable runtime-defined TSA entries to be added to the TSA table by the FMSTR\_SetUpTsaBuff() and FMSTR\_TsaAddVar() functions. Default value is 0 (false).

### Pipes options

#### FMSTR\_USE\_PIPES

```
#define FMSTR_USE_PIPES [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable the FreeMASTER Pipes feature to be used. Default value is 0 (false).

## FMSTR\_MAX\_PIPES\_COUNT

```
#define FMSTR_MAX_PIPES_COUNT [number]
```

**Value Type** Number in range 1..63.

**Description** The number of simultaneous pipe connections to support. The default value is 1.

**Driver interrupt modes** To implement the communication, the FreeMASTER driver handles the Serial or CAN module's receive and transmit requests. Use the *freemaster\_cfg.h* configuration file to select whether the driver processes the communication automatically in the interrupt service routine handler or if it only polls the status of the module (typically during the application idle time).

This section describes each of the interrupt mode in more details.

**Completely Interrupt-Driven operation** Activated using:

```
#define FMSTR_LONG_INTR 1
```

In this mode, both the communication and the FreeMASTER protocol decoding is done in the *FMSTR\_SerialIsr*, *FMSTR\_CanIsr*, or other interrupt service routine. Because the protocol execution may be a lengthy task (especially with the TSA-Safety enabled) it is recommended to use this mode only if the interrupt prioritization scheme is possible in the application and the FreeMASTER interrupt is assigned to a lower (the lowest) priority.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR\_SerialIsr* or *FMSTR\_CanIsr* functions from that handler.

**Mixed Interrupt and Polling Modes** Activated using:

```
#define FMSTR_SHORT_INTR 1
```

In this mode, the communication processing time is split between the interrupt routine and the main application loop or task. The raw communication is handled by the *FMSTR\_SerialIsr*, *FMSTR\_CanIsr*, or other interrupt service routine, while the protocol decoding and execution is handled by the *FMSTR\_Poll* routine. Call *FMSTR\_Poll* during the idle time in the application main loop.

The interrupt processing in this mode is relatively fast and deterministic. Upon a serial-receive event, the received character is only placed into a FIFO-like queue and it is not further processed. Upon a CAN receive event, the received frame is stored into a receive buffer. When transmitting, the characters are fetched from the prepared transmit buffer.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR\_SerialIsr* or *FMSTR\_CanIsr* functions from that handler.

When the serial interface is used as the serial communication interface, ensure that the *FMSTR\_Poll* function is called at least once per *N* character time periods. *N* is the length of the FreeMASTER FIFO queue (*FMSTR\_COMM\_QUEUE\_SIZE*) and the character time is the time needed to transmit or receive a single byte over the SCI line.

## Completely Poll-driven

```
#define FMSTR_POLL_DRIVEN 1
```

In this mode, both the communication and the FreeMASTER protocol decoding are done in the *FMSTR\_Poll* routine. No interrupts are needed and the *FMSTR\_SerialIsr*, *FMSTR\_CanIsr*, and similar handlers compile to an empty code.

When using this mode, ensure that the *FMSTR\_Poll* function is called by the application at least once per the serial “character time” which is the time needed to transmit or receive a single character.

In the latter two modes (*FMSTR\_SHORT\_INTR* and *FMSTR\_POLL\_DRIVEN*), the protocol handling takes place in the *FMSTR\_Poll* routine. An application interrupt can occur in the middle of the Read Memory or Write Memory commands’ execution and corrupt the variable being accessed by the FreeMASTER driver. In these two modes, some issues or glitches may occur when using FreeMASTER to visualize or monitor volatile variables modified in interrupt servicing code.

The same issue may appear even in the full interrupt mode (*FMSTR\_LONG\_INTR*), if volatile variables are modified in the interrupt code with a priority higher than the priority of the communication interrupt.

**Data types** Simple portability was one of the main requirements when writing the FreeMASTER driver. This is why the driver code uses the privately-declared data types and the vast majority of the platform-dependent code is separated in the platform-dependent source files. The data types used in the driver API are all defined in the platform-specific header file.

To prevent name conflicts with the symbols used in the application, all data types, macros, and functions have the *FMSTR\_* prefix. The only global variables used in the driver are the transport and low-level API structures exported from the driver-implementation layer to upper layers. Other than that, all private variables are declared as static and named using the *fmstr\_* prefix.

**Communication interface initialization** The FreeMASTER driver does not perform neither the initialization nor the configuration of the peripheral module that it uses to communicate. It is the application startup code responsibility to configure the communication module before the FreeMASTER driver is initialized by the *FMSTR\_Init* call.

When the Serial communication module is used as the FreeMASTER communication interface, configure the UART receive and transmit pins, the serial communication baud rate, parity (no-parity), the character length (eight bits), and the number of stop bits (one) before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see *Driver interrupt modes*), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected serial peripheral module. Call the *FMSTR\_SerialIsr* function from the application handler.

When a CAN module is used as the FreeMASTER communication interface, configure the CAN receive and transmit pins and the CAN module bit rate before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see *Driver interrupt modes*), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected CAN peripheral module. Call the *FMSTR\_CanIsr* function from the application handler.

**Note:** It is not necessary to enable or unmask the serial nor the CAN interrupts before initializing the FreeMASTER driver. The driver enables or disables the interrupts and communication lines, as required during runtime.

**FreeMASTER Recorder calls** When using the FreeMASTER Recorder in the application (*FMSTR\_USE\_RECORDER* > 0), call the *FMSTR\_RecorderCreate* function early after *FMSTR\_Init* to set

up each recorder instance to be used in the application. Then call the `FMSTR_Recorder` function periodically in the code where the data recording should occur. A typical place to call the Recorder routine is at the timer or PWM interrupts, but it can be anywhere else. The example applications provided together with the driver code call the `FMSTR_Recorder` in the main application loop.

In applications where `FMSTR_Recorder` is called periodically with a constant period, specify the period in the Recorder configuration structure before calling `FMSTR_RecorderCreate`. This setting enables the PC Host FreeMASTER tool to display the X-axis of the Recorder graph properly scaled for the time domain.

**Driver usage** Start using or evaluating FreeMASTER by opening some of the example applications available in the driver setup package.

Follow these steps to enable the basic FreeMASTER connectivity in the application:

- Make sure that all `*.c` files of the FreeMASTER driver from the `src/common/platforms/[your_platform]` folder are a part of the project. See [Driver files](#) for more details.
- Configure the FreeMASTER driver by creating or editing the `freemaster_cfg.h` file and by saving it into the application project directory. See [Driver configuration](#) for more details.
- Include the `freemaster.h` file into any application source file that makes the FreeMASTER API calls.
- Initialize the Serial or CAN modules. Set the baud rate, parity, and other parameters of the communication. Do not enable the communication interrupts in the interrupt mask registers.
- For the `FMSTR_LONG_INTR` and `FMSTR_SHORT_INTR` modes, install the application-specific interrupt routine and call the `FMSTR_SerialIsr` or `FMSTR_CanIsr` functions from this handler.
- Call the `FMSTR_Init` function early on in the application initialization code.
- Call the `FMSTR_RecorderCreate` functions for each Recorder instance to enable the Recorder feature.
- In the main application loop, call the `FMSTR_Poll` API function periodically when the application is idle.
- For the `FMSTR_SHORT_INTR` and `FMSTR_LONG_INTR` modes, enable the interrupts globally so that the interrupts can be handled by the CPU.

**Communication troubleshooting** The most common problem that causes communication issues is a wrong baud rate setting or a wrong pin multiplexer setting of the target MCU. When a communication between the PC Host running FreeMASTER and the target MCU cannot be established, try enabling the `FMSTR_DEBUG_TX` option in the `freemaster_cfg.h` file and call the `FMSTR_Poll` function periodically in the main application task loop.

With this feature enabled, the FreeMASTER driver periodically transmits a test frame through the Serial or CAN lines. Use a logic analyzer or an oscilloscope to monitor the signals at the communication pins of the CPU device to examine whether the bit rate and signal polarity are configured properly.

## Driver API

This section describes the driver Application Programmers' Interface (API) needed to initialize and use the FreeMASTER serial communication driver.

**Control API** There are three key functions to initialize and use the driver.

## FMSTR\_Init

### Prototype

```
FMSTR_BOOL FMSTR_Init(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_protocol.c*

**Description** This function initializes the internal variables of the FreeMASTER driver and enables the communication interface. This function does not change the configuration of the selected communication module. The hardware module must be initialized before the *FMSTR\_Init* function is called.

A call to this function must occur before calling any other FreeMASTER driver API functions.

## FMSTR\_Poll

### Prototype

```
void FMSTR_Poll(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_protocol.c*

**Description** In the poll-driven or short interrupt modes, this function handles the protocol decoding and execution (see *Driver interrupt modes*). In the poll-driven mode, this function also handles the communication interface with the PC. Typically, the *FMSTR\_Poll* function is called during the “idle” time in the main application task loop.

To prevent the receive data overflow (loss) on a serial interface, make sure that the *FMSTR\_Poll* function is called at least once per the time calculated as:

$$N * Tchar$$

where:

- *N* is equal to the length of the receive FIFO queue (configured by the *FMSTR\_COMM\_QUEUE\_SIZE* macro). *N* is 1 for the poll-driven mode.
- *Tchar* is the character time, which is the time needed to transmit or receive a single byte over the SCI line.

**Note:** In the long interrupt mode, this function typically compiles as an empty function and can still be called. It is worthwhile to call this function regardless of the interrupt mode used in the application. This approach enables a convenient switching between the different interrupt modes only by changing the configuration macros in the *freemaster\_cfg.h* file.

## FMSTR\_SerialIsr / FMSTR\_CanIsr

### Prototype

```
void FMSTR_SerialIsr(void);  
void FMSTR_CanIsr(void);
```

- Declaration: *freemaster.h*
- Implementation: *hw-specific low-level driver C file*

**Description** This function contains the interrupt-processing code of the FreeMASTER driver. In long or short interrupt modes (see [Driver interrupt modes](#)), this function must be called from the application interrupt service routine registered for the communication interrupt vector. On platforms where the communication module uses multiple interrupt vectors, the application should register a handler for all vectors and call this function at each interrupt.

**Note:** In a poll-driven mode, this function is compiled as an empty function and does not have to be used.

## Recorder API

### FMSTR\_RecorderCreate

#### Prototype

```
FMSTR_BOOL FMSTR_RecorderCreate(FMSTR_INDEX recIndex, FMSTR_REC_BUFF* buffCfg);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_rec.c*

**Description** This function registers a recorder instance and enables it to be used by the PC Host tool. Call this function for all recorder instances from 0 to the maximum number defined by the FMSTR\_USE\_RECORDER configuration option (minus one). An exception to this requirement is the recorder of instance 0 which may be automatically configured by FMSTR\_Init when the *freemaster\_cfg.h* configuration file defines the *FMSTR\_REC\_BUFF\_SIZE* and *FMSTR\_REC\_TIMEBASE* options.

For more information, see [Configurable items](#).

### FMSTR\_Recorder

#### Prototype

```
void FMSTR_Recorder(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_rec.c*

**Description** This function takes a sample of the variables being recorded using the FreeMASTER Recorder instance *recIndex*. If the selected Recorder is not active when the *FMSTR\_Recorder* function is being called, the function returns immediately. When the Recorder is active, the values of the variables being recorded are copied into the recorder buffer and the trigger conditions are evaluated.



If a trigger condition is satisfied, the Recorder enters the post-trigger mode, where it counts down the follow-up samples (number of *FMSTR\_Recorder* function calls) and de-activates the Recorder when the required post-trigger samples are finished.

The *FMSTR\_Recorder* function is typically called in the timer or PWM interrupt service routines. This function can also be called in the application main loop (for testing purposes).

## FMSTR\_RecorderTrigger

### Prototype

```
void FMSTR_RecorderTrigger(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_rec.c*

**Description** This function forces the Recorder trigger condition to happen, which causes the Recorder to be automatically deactivated after the post-trigger samples are sampled. Use this function in the application code for programmatic control over the Recorder triggering. This can be useful when a more complex triggering conditions need to be used.

**Fast Recorder API** The Fast Recorder feature is not available in the FreeMASTER driver version 3. This feature was heavily dependent on the target platform and it was only available for the 56F8xxxx DSCs.

**TSA Tables** When the TSA is enabled in the FreeMASTER driver configuration file (by setting the *FMSTR\_USE\_TSA* macro to a non-zero value), it defines the so-called TSA tables in the application. This section describes the macros that must to be used to define the TSA tables.

There can be any number of TSA tables spread across the application source files. There must be always exactly one TSA Table List defined, which informs the FreeMASTER driver about the active TSA tables.

When there is at least one TSA table and one TSA Table List defined in the application, the TSA information automatically appears in the FreeMASTER symbols list. The symbols can then be used to create FreeMASTER variables for visualization or control.

**TSA table definition** The TSA table describes the static or global variables together with their address, size, type, and access-protection information. If the TSA-described variables are of a structure type, the TSA table may also describe this type and provide an access to the individual structure members of the variable.

The TSA table definition begins with the *FMSTR\_TSA\_TABLE\_BEGIN* macro with a *table\_id* identifying the table. The *table\_id* shall be a valid C-language symbol.

```
FMSTR_TSA_TABLE_BEGIN(table_id)
```

After this opening macro, the TSA descriptors are placed using these macros:

```
/* Adding variable descriptors */
FMSTR_TSA_RW_VAR(name, type) /* read/write variable entry */
FMSTR_TSA_RO_VAR(name, type) /* read-only variable entry */

/* Description of complex data types */
FMSTR_TSA_STRUCT(struct_name) /* structure or union type entry */
```

(continues on next page)

(continued from previous page)

```

FMSTR_TSA_MEMBER(struct_name, member_name, type) /* structure member entry */

/* Memory blocks */
FMSTR_TSA_RW_MEM(name, type, address, size) /* read/write memory block */
FMSTR_TSA_RO_MEM(name, type, address, size) /* read-only memory block */

```

The table is closed using the FMSTR\_TSA\_TABLE\_END macro:

```
FMSTR_TSA_TABLE_END()
```

**TSA descriptor parameters** The TSA descriptor macros accept these parameters:

- *name* — variable name. The variable must be defined before the TSA descriptor references it.
- *type* — variable or member type. Only one of the pre-defined type constants may be used (see below).
- *struct\_name* — structure type name. The type must be defined (typedef) before the TSA descriptor references it.
- *member\_name* — structure member name.

**Note:** The structure member descriptors (FMSTR\_TSA\_MEMBER) must immediately follow the parent structure descriptor (FMSTR\_TSA\_STRUCT) in the table.

**Note:** To write-protect the variables in the FreeMASTER driver (FMSTR\_TSA\_RO\_VAR), enable the TSA-Safety feature in the configuration file.

**TSA variable types** The table lists *type* identifiers which can be used in TSA descriptors:

| Constant                           | Description                                                    |
|------------------------------------|----------------------------------------------------------------|
| FMSTR_TSA_UINTn                    | Unsigned integer type of size <i>n</i> bits (n=8,16,32,64)     |
| FMSTR_TSA_SINTn                    | Signed integer type of size <i>n</i> bits (n=8,16,32,64)       |
| FMSTR_TSA_FRACn                    | Fractional number of size <i>n</i> bits (n=16,32,64).          |
| FMSTR_TSA_FRAC_Q( <i>m,n</i> )     | Signed fractional number in general Q form (m+n+1 total bits)  |
| FMSTR_TSA_FRAC_UQ( <i>m,n</i> )    | Unsigned fractional number in general UQ form (m+n total bits) |
| FMSTR_TSA_FLOAT                    | 4-byte standard IEEE floating-point type                       |
| FMSTR_TSA_DOUBLE                   | 8-byte standard IEEE floating-point type                       |
| FMSTR_TSA_POINTER                  | Generic pointer type defined (platform-specific 16 or 32 bit)  |
| FM-STR_TSA_USERTYPE( <i>name</i> ) | Structure or union type declared with FMSTR_TSA_STRUCT record  |

**TSA table list** There shall be exactly one TSA Table List in the application. The list contains one entry for each TSA table defined anywhere in the application.

The TSA Table List begins with the FMSTR\_TSA\_TABLE\_LIST\_BEGIN macro and continues with the TSA table entries for each table.

```

FMSTR_TSA_TABLE_LIST_BEGIN()

FMSTR_TSA_TABLE(table_id)
FMSTR_TSA_TABLE(table_id2)
FMSTR_TSA_TABLE(table_id3)
...

```

The list is closed with the FMSTR\_TSA\_TABLE\_LIST\_END macro:

```
FMSTR_TSA_TABLE_LIST_END()
```

**TSA Active Content entries** FreeMASTER v2.0 and higher supports TSA Active Content, enabling the TSA tables to describe the memory-mapped files, virtual directories, and URL hyperlinks. FreeMASTER can access such objects similarly to accessing the files and folders on the local hard drive.

With this set of TSA entries, the FreeMASTER pages can be embedded directly into the target MCU flash and accessed by FreeMASTER directly over the communication line. The HTML-coded pages rendered inside the FreeMASTER window can access the TSA Active Content resources using a special URL referencing the *fmstr:* protocol.

This example provides an overview of the supported TSA Active Content entries:

```
FMSTR_TSA_TABLE_BEGIN(files_and_links)

/* Directory entry applies to all subsequent MEMFILE entries */
FMSTR_TSA_DIRECTORY("/text_files") /* entering a new virtual directory */

/* The readme.txt file will be accessible at the fmstr://text_files/readme.txt URL */
FMSTR_TSA_MEMFILE("readme.txt", readme_txt, sizeof(readme_txt)) /* memory-mapped file */

/* Files can also be specified with a full path so the DIRECTORY entry does not apply */
FMSTR_TSA_MEMFILE("/index.htm", index, sizeof(index)) /* memory-mapped file */
FMSTR_TSA_MEMFILE("/prj/demo.pmp", demo_pmp, sizeof(demo_pmp)) /* memory-mapped file */

/* Hyperlinks can point to a local MEMFILE object or to the Internet */
FMSTR_TSA_HREF("Board's Built-in Welcome Page", "/index.htm")
FMSTR_TSA_HREF("FreeMASTER Home Page", "http://www.nxp.com/freemaster")

/* Project file links simplify opening the projects from any URLs */
FMSTR_TSA_PROJECT("Demonstration Project (embedded)", "/prj/demo.pmp")
FMSTR_TSA_PROJECT("Full Project (online)", "http://mycompany.com/prj/demo.pmp")

FMSTR_TSA_TABLE_END()
```

## TSA API

### FMSTR\_SetUpTsaBuff

#### Prototype

```
FMSTR_BOOL FMSTR_SetUpTsaBuff(FMSTR_ADDR buffAddr, FMSTR_SIZE buffSize);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_tsa.c*

#### Arguments

- *buffAddr* [in] - address of the memory buffer for the dynamic TSA table
- *buffSize* [in] - size of the memory buffer which determines the maximum number of TSA entries to be added in the runtime

**Description** This function must be used to assign the RAM memory buffer to the TSA subsystem when FMSTR\_USE\_TSA\_DYNAMIC is enabled. The memory buffer is then used to store the TSA entries added dynamically to the runtime TSA table using the FMSTR\_TsaAddVar function call. The runtime TSA table is processed by the FreeMASTER PC Host tool along with all static tables as soon as the communication port is open.

The size of the memory buffer determines the number of TSA entries that can be added dynamically. Depending on the MCU platform, one TSA entry takes either 8 or 16 bytes.

## FMSTR\_TsaAddVar

### Prototype

```
FMSTR_BOOL FMSTR_TsaAddVar(FMSTR_TSATBL_STRPTR tsaName, FMSTR_TSATBL_STRPTR ↵  
↵ tsaType,  
    FMSTR_TSATBL_VOIDPTR varAddr, FMSTR_SIZE32 varSize,  
    FMSTR_SIZE flags);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_tsa.c*

### Arguments

- *tsaName* [in] - name of the object
- *tsaType* [in] - name of the object type
- *varAddr* [in] - address of the object
- *varSize* [in] - size of the object
- *flags* [in] - access flags; a combination of these values:
  - FMSTR\_TSA\_INFO\_RO\_VAR — read-only memory-mapped object (typically a variable)
  - FMSTR\_TSA\_INFO\_RW\_VAR — read/write memory-mapped object
  - FMSTR\_TSA\_INFO\_NON\_VAR — other entry, describing structure types, structure members, enumerations, and other types

**Description** This function can be called only when the dynamic TSA table is enabled by the FMSTR\_USE\_TSA\_DYNAMIC configuration option and when the FMSTR\_SetUpTsaBuff function call is made to assign the dynamic TSA table memory. This function adds an entry into the dynamic TSA table. It can be used to register a read-only or read/write memory object or describe an item of the user-defined type.

See [TSA table definition](#) for more details about the TSA table entries.

## Application Commands API

### FMSTR\_GetAppCmd

#### Prototype

```
FMSTR_APPCMD_CODE FMSTR_GetAppCmd(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*

**Description** This function can be used to detect if there is an Application Command waiting to be processed by the application. If no command is pending, this function returns the FMSTR\_APPCMDRESULT\_NOCMD constant. Otherwise, this function returns the code of the Application Command that must be processed. Use the FMSTR\_AppCmdAck call to acknowledge the Application Command after it is processed and to return the appropriate result code to the host.

The FMSTR\_GetAppCmd function does not report the commands for which a callback handler function exists. If the FMSTR\_GetAppCmd function is called when a callback-registered command is pending (and before it is actually processed by the callback function), this function returns FMSTR\_APPCMDRESULT\_NOCMD.

## FMSTR\_GetAppCmdData

### Prototype

```
FMSTR_APPCMD_PDATA FMSTR_GetAppCmdData(FMSTR_SIZE* dataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*

### Arguments

- *dataLen* [out] - pointer to the variable that receives the length of the data available in the buffer. It can be NULL when this information is not needed.

**Description** This function can be used to retrieve the Application Command data when the application determines that an Application Command is pending (see [FMSTR\\_GetAppCmd](#)).

There is just a single buffer to hold the Application Command data (the buffer length is FMSTR\_APPCMD\_BUFF\_SIZE bytes). If the data are to be used in the application after the command is processed by the FMSTR\_AppCmdAck call, copy the data out to a private buffer.

## FMSTR\_AppCmdAck

### Prototype

```
void FMSTR_AppCmdAck(FMSTR_APPCMD_RESULT resultCode);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*

### Arguments

- *resultCode* [in] - the result code which is to be returned to FreeMASTER

**Description** This function is used when the Application Command processing finishes in the application. The resultCode passed to this function is returned back to the host and the driver is re-initialized to expect the next Application Command.

After this function is called and before the next Application Command arrives, the return value of the FMSTR\_GetAppCmd function is FMSTR\_APPCMDRESULT\_NOCMD.

## FMSTR\_AppCmdSetResponseData

### Prototype

```
void FMSTR_AppCmdSetResponseData(FMSTR_ADDR resultDataAddr, FMSTR_SIZE resultDataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*

### Arguments

- *resultDataAddr* [in] - pointer to the data buffer that is to be copied to the Application Command data buffer
- *resultDataLen* [in] - length of the data to be copied. It must not exceed the FMSTR\_APPCMD\_BUFF\_SIZE value.

**Description** This function can be used before the Application Command processing finishes, when there are data to be returned back to the PC.

The response data buffer is copied into the Application Command data buffer, from where it is accessed when the host requires it. Do not use FMSTR\_GetAppCmdData and the data buffer after FMSTR\_AppCmdSetResponseData is called.

**Note:** The current version of FreeMASTER does not support the Application Command response data.

## FMSTR\_RegisterAppCmdCall

### Prototype

```
FMSTR_BOOL FMSTR_RegisterAppCmdCall(FMSTR_APPCMD_CODE appCmdCode, FMSTR_
↳PAPPCMDFUNC callbackFunc);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*

### Arguments

- *appCmdCode* [in] - the Application Command code for which the callback is to be registered
- *callbackFunc* [in] - pointer to the callback function that is to be registered. Use NULL to unregister a callback registered previously with this Application Command.

**Return value** This function returns a non-zero value when the callback function was successfully registered or unregistered. It can return zero when trying to register a callback function for more than FMSTR\_MAX\_APPCMD\_CALLS different Application Commands.

**Description** This function can be used to register the given function as a callback handler for the Application Command. The Application Command is identified using single-byte code. The callback function is invoked automatically by the FreeMASTER driver when the protocol decoder obtains a request to get the application command result code.

The prototype of the callback function is

```
FMSTR_APPCMD_RESULT HandlerFunction(FMSTR_APPCMD_CODE nAppcmd,
FMSTR_APPCMD_PDATA pData, FMSTR_SIZE nDataLen);
```

Where:

- *nAppcmd* -Application Command code
- *pData* —points to the Application Command data received (if any)
- *nDataLen* —information about the Application Command data length

The return value of the callback function is used as the Application Command Result Code and returned to FreeMASTER.

**Note:** The FMSTR\_MAX\_APPCMD\_CALLS configuration macro defines how many different Application Commands may be handled by a callback function. When FMSTR\_MAX\_APPCMD\_CALLS is undefined or defined as zero, the FMSTR\_RegisterAppCmdCall function always fails.

## Pipes API

### FMSTR\_PipeOpen

#### Prototype

```
FMSTR_HPIPE FMSTR_PipeOpen(FMSTR_PIPE_PORT pipePort, FMSTR_PPIPEFUNC pipeCallback,
    FMSTR_ADDR pipeRxBuff, FMSTR_PIPE_SIZE pipeRxSize,
    FMSTR_ADDR pipeTxBuff, FMSTR_PIPE_SIZE pipeTxSize,
    FMSTR_U8 type, const FMSTR_CHAR *name);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_pipes.c*

#### Arguments

- *pipePort* [in] - port number that identifies the pipe for the client
- *pipeCallback* [in] - pointer to the callback function that is called whenever a pipe data status changes
- *pipeRxBuff* [in] - address of the receive memory buffer
- *pipeRxSize* [in] - size of the receive memory buffer
- *pipeTxBuff* [in] - address of the transmit memory buffer
- *pipeTxSize* [in] - size of the transmit memory buffer
- *type* [in] - a combination of FMSTR\_PIPE\_MODE\_xxx and FMSTR\_PIPE\_SIZE\_xxx constants describing primary pipe data format and usage. This type helps FreeMASTER decide how to access the pipe by default. Optional, use 0 when undetermined.
- *name* [in] - user name of the pipe port. This name is visible to the FreeMASTER user when creating the graphical pipe interface.

**Description** This function initializes a new pipe and makes it ready to accept or send the data to the PC Host client. The receive memory buffer is used to store the received data before they are read out by the FMSTR\_PipeRead call. When this buffer gets full, the PC Host client denies the data transmission into this pipe until there is enough free space again. The transmit memory buffer is used to store the data transmitted by the application to the PC Host client using the FMSTR\_PipeWrite call. The transmit buffer can get full when the PC Host is disconnected or when it is slow in receiving and reading out the pipe data.



The function returns the pipe handle which must be stored and used in the subsequent calls to manage the pipe object.

The callback function (if specified) is called whenever new data are received through the pipe and available for reading. This callback is also called when the data waiting in the transmit buffer are successfully pushed to the PC Host and the transmit buffer free space increases. The prototype of the callback function provided by the user application must be as follows. The *PipeHandler* name is only a placeholder and must be defined by the application.

```
void PipeHandler(FMSTR_HPIPE pipeHandle);
```

## FMSTR\_PipeClose

### Prototype

```
void FMSTR_PipeClose(FMSTR_HPIPE pipeHandle);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_pipes.c*

### Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR\_PipeOpen function call

**Description** This function de-initializes the pipe object. No data can be received or sent on the pipe after this call.

## FMSTR\_PipeWrite

### Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeWrite(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,  
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE writeGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_pipes.c*

### Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR\_PipeOpen function call
- *pipeData* [in] - address of the data to be written
- *pipeDataLen* [in] - length of the data to be written
- *writeGranularity* [in] - size of the minimum unit of data which is to be written

**Description** This function puts the user-specified data into the pipe's transmit memory buffer and schedules it for transmission. This function returns the number of bytes that were successfully written into the buffer. This number may be smaller than the number of the requested bytes if there is not enough free space in the transmit buffer.

The *writeGranularity* argument can be used to split the data into smaller chunks, each of the size given by the *writeGranularity* value. The FMSTR\_PipeWrite function writes as many data chunks as possible into the transmit buffer and does not attempt to write an incomplete chunk.



This feature can prove to be useful to avoid the intermediate caching when writing an array of integer values or other multi-byte data items. When making the `nGranularity` value equal to the `nLength` value, all data are considered as one chunk which is either written successfully as a whole or not at all. The `nGranularity` value of 0 or 1 disables the data-chunk approach.

## FMSTR\_PipeRead

### Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeRead(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE readGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_pipes.c*

### Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR\_PipeOpen function call
- *pipeData* [in] - address of the data buffer to be filled with the received data
- *pipeDataLen* [in] - length of the data to be read
- *readGranularity* [in] - size of the minimum unit of data which is to be read

**Description** This function copies the data received from the pipe from its receive buffer to the user buffer for further processing. The function returns the number of bytes that were successfully copied to the buffer. This number may be smaller than the number of the requested bytes if there is not enough data bytes available in the receive buffer.

The `readGranularity` argument can be used to copy the data in larger chunks in the same way as described in the FMSTR\_PipeWrite function.

**API data types** This section describes the data types used in the FreeMASTER driver. The information provided here can be useful when modifying or porting the FreeMASTER Communication Driver to new NXP platforms.

**Note:** The licensing conditions prohibit use of FreeMASTER and the FreeMASTER Communication Driver with non-NXP MPU or MCU products.

**Public common types** The table below describes the public data types used in the FreeMASTER driver API calls. The data types are declared in the *freemaster.h* header file.

| Type name                                                                                                                                                                                                                 | Description                                                                                                                         |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>FM-STR_ADDR</i><br>For example, this type is defined as long integer on the 56F8xxx platform where the 24-bit addresses must be supported, but the C-pointer may be only 16 bits wide in some compiler configurations. | Data type used to hold the memory address. On most platforms, this is normally a C-pointer, but it may also be a pure integer type. |
| <i>FM-STR_SIZE</i><br>It is required that this type is unsigned and at least 16 bits wide integer.                                                                                                                        | Data type used to hold the memory block size.                                                                                       |
| <i>FM-STR_BOOL</i><br>This type is used only in zero/non-zero conditions in the driver code.                                                                                                                              | Data type used as a general boolean type.                                                                                           |
| <i>FM-STR_APPCM</i><br>Generally, this is an unsigned 8-bit value.                                                                                                                                                        | Data type used to hold the Application Command code.                                                                                |
| <i>FM-STR_APPCM</i><br>Generally, this is an unsigned 8-bit value.                                                                                                                                                        | Data type used to create the Application Command data buffer.                                                                       |
| <i>FM-STR_APPCM</i><br>Generally, this is an unsigned 8-bit value.                                                                                                                                                        | Data type used to hold the Application Command result code.                                                                         |

**Public TSA types** The table describes the TSA-specific public data types. These types are declared in the *freemaster\_tsa.h* header file, which is included in the user application indirectly by the *freemaster.h* file.

|                       |                                                                                                                                                               |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>FM-STR_TSA_TII</i> | Data type used to hold a descriptor index in the TSA table or a table index in the list of TSA tables.<br>By default, this is defined as <i>FM-STR_SIZE</i> . |
| <i>FM-STR_TSA_TS</i>  | Data type used to hold a memory block size, as used in the TSA descriptors.<br>By default, this is defined as <i>FM-STR_SIZE</i> .                            |

**Public Pipes types** The table describes the data types used by the FreeMASTER Pipes API:

|                       |                                                                                                                |
|-----------------------|----------------------------------------------------------------------------------------------------------------|
| <i>FM-STR_HPIPE</i>   | Pipe handle that identifies the open-pipe object.<br>Generally, this is a pointer to a void type.              |
| <i>FM-STR_PIPE_PC</i> | Integer type required to hold at least 7 bits of data.<br>Generally, this is an unsigned 8-bit or 16-bit type. |
| <i>FM-STR_PIPE_SI</i> | Integer type required to hold at least 16 bits of data.<br>This is used to store the data buffer sizes.        |
| <i>FM-STR_PPIPEF</i>  | Pointer to the pipe handler function.<br>See <a href="#">FM-STR_PipeOpen</a> for more details.                 |

**Internal types** The table describes the data types used internally by the FreeMASTER driver. The data types are declared in the platform-specific header file and they are not available in the application code.

|                                                                                                          |                                                                         |
|----------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|
| <i>FMSTR_U8</i>                                                                                          | The smallest memory entity.                                             |
| On the vast majority of platforms, this is an unsigned 8-bit integer.                                    |                                                                         |
| On the 56F8xx DSP platform, this is defined as an unsigned 16-bit integer.                               |                                                                         |
| <i>FM-STR_U16</i>                                                                                        | Unsigned 16-bit integer.                                                |
| <i>FM-STR_U32</i>                                                                                        | Unsigned 32-bit integer.                                                |
| <i>FMSTR_S8</i>                                                                                          | Signed 8-bit integer.                                                   |
| <i>FM-STR_S16</i>                                                                                        | Signed 16-bit integer.                                                  |
| <i>FM-STR_S32</i>                                                                                        | Signed 32-bit integer.                                                  |
| <i>FM-STR_FLOAT</i>                                                                                      | 4-byte standard IEEE floating-point type.                               |
| <i>FM-STR_FLAGS</i>                                                                                      | Data type forming a union with a structure of flag bit-fields.          |
| <i>FM-STR_SIZE8</i>                                                                                      | Data type holding a general size value, at least 8 bits wide.           |
| <i>FM-STR_INDEX</i>                                                                                      | General for-loop index. Must be signed, at least 16 bits wide.          |
| <i>FM-STR_BCHR</i>                                                                                       | A single character in the communication buffer.                         |
| Typically, this is an 8-bit unsigned integer, except for the DSP platforms where it is a 16-bit integer. |                                                                         |
| <i>FM-STR_BPTR</i>                                                                                       | A pointer to the communication buffer (an array of <i>FMSTR_BCHR</i> ). |

## Document references

### Links

- This document online: <https://mcuxpresso.nxp.com/mcuxsdk/latest/html/middleware/freemaster/doc/index.html>

- FreeMASTER tool home: [www.nxp.com/freemaster](http://www.nxp.com/freemaster)
- FreeMASTER community area: [community.nxp.com/community/freemaster](http://community.nxp.com/community/freemaster)
- FreeMASTER GitHub code repo: <https://github.com/nxp-mcuxpresso/mcux-freemaster>
- MCUXpresso SDK home: [www.nxp.com/mcuxpresso](http://www.nxp.com/mcuxpresso)
- MCUXpresso SDK builder: [mcuxpresso.nxp.com/en](http://mcuxpresso.nxp.com/en)

## Documents

- *FreeMASTER Usage Serial Driver Implementation* (document [AN4752](#))
- *Integrating FreeMASTER Time Debugging Tool With CodeWarrior For Microcontrollers v10.X Project* (document [AN4771](#))
- *Flash Driver Library For MC56F847xx And MC56F827xx DSC Family* (document [AN4860](#))

**Revision history** This Table summarizes the changes done to this document since the initial release.

| Revision | Date    | Description                                                                                                                                                                                                                                                                                                          |
|----------|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.0      | 03/2006 | Limited initial release                                                                                                                                                                                                                                                                                              |
| 2.0      | 09/2007 | Updated for FreeMASTER version. New Freescale document template used.                                                                                                                                                                                                                                                |
| 2.1      | 12/2007 | Added description of the new Fast Recorder feature and its API.                                                                                                                                                                                                                                                      |
| 2.2      | 04/2010 | Added support for MPC56xx platform, Added new API for use CAN interface.                                                                                                                                                                                                                                             |
| 2.3      | 04/2011 | Added support for Kxx Kinetis platform and MQX operating system.                                                                                                                                                                                                                                                     |
| 2.4      | 06/2011 | Serial driver update, adds support for USB CDC interface.                                                                                                                                                                                                                                                            |
| 2.5      | 08/2011 | Added Packet Driven BDM interface.                                                                                                                                                                                                                                                                                   |
| 2.7      | 12/2013 | Added FLEXCAN32 interface, byte access and isr callback configuration option.                                                                                                                                                                                                                                        |
| 2.8      | 06/2014 | Removed obsolete license text, see the software package content for up-to-date license.                                                                                                                                                                                                                              |
| 2.9      | 03/2015 | Update for driver version 1.8.2 and 1.9: FreeMASTER Pipes, TSA Active Content, LIN Transport Layer support, DEBUG-TX communication troubleshooting, Kinetis SDK support.                                                                                                                                             |
| 3.0      | 08/2016 | Update for driver version 2.0: Added support for MPC56xx, MPC57xx, KEAxx and S32Kxx platforms. New NXP document template as well as new license agreement used. added MCAN interface. Folders structure at the installation destination was rearranged.                                                              |
| 4.0      | 04/2019 | Update for driver released as part of FreeMASTER v3.0 and MCUXpresso SDK 2.6. Updated to match new V4 serial communication protocol and new configuration options. This version of the document removes substantial portion of outdated information related to S08, S12, ColdFire, Power and other legacy platforms. |
| 4.1      | 04/2020 | Minor update for FreeMASTER driver included in MCUXpresso SDK 2.8.                                                                                                                                                                                                                                                   |
| 4.2      | 09/2020 | Added example applications description and information about the MCUXpresso Config Tools. Fixed the pipe-related API description.                                                                                                                                                                                    |
| 4.3      | 10/2024 | Added description of Network and Segger J-Link RTT interface configuration. Accompanying the MCUXpresso SDK version 24.12.00.                                                                                                                                                                                        |
| 4.4      | 04/2025 | Added Zephyr-specific information. Accompanying the MCUXpresso SDK version 25.06.00.                                                                                                                                                                                                                                 |

## 3.3 Multimedia

### 3.3.1 Xtensa Audio Framework (XAF)

#### Xtensa Audio Framework (XAF) Examples

**Overview** The Xtensa Audio Framework (XAF) is designed to accelerate the development of audio processing applications for the HiFi family of DSP cores. The multicore version of XAF described in these examples is designed to work with subsystems having single or multiple DSPs, enabling sophisticated audio processing capabilities in embedded systems.

Each demo showcases a dual-core architecture:

- **cm33/** - The ARM application for the Cortex-M33 core, which provides the user interface and system control
- **dsp/** - The DSP application that performs audio processing using the XAF middleware library

When an application is started, a shell interface is displayed on the terminal that executes from the ARM core. Users can control the application through shell commands, which are relayed via RPMsg-Lite IPC to the DSP core where they are processed and responses are returned. This architecture demonstrates efficient partitioning of workloads - with user interface and control tasks handled by the ARM core while computationally intensive audio processing is offloaded to the specialized DSP core.

For more information about XAF and detailed documentation on the API and available components, please refer to the Cadence XAF documentation ([/middleware/cadence/multicore-xaf/xa\\_af\\_hostless/doc](/middleware/cadence/multicore-xaf/xa_af_hostless/doc)).

**Availability Note Important:** These XAF examples are not included in the standard MCUXpresso SDK repository. They are available as part of the MCUXpresso SDK Builder package on the NXP website. To access these examples, please visit [MCUXpresso SDK Builder](#) and create a customized SDK package that includes the XAF examples for your target platform.

## Included Examples

**XAF Playback Example** The `dsp_xaf_playback` application demonstrates audio file decoding and playback capabilities using the DSP core and Xtensa Audio Framework, supporting various audio codecs while handling operations through a shell interface on the ARM core that communicates with DSP processing.

**XAF Record Example** The `dsp_xaf_record` example captures audio from digital microphones (DMIC), processes it on the DSP core using voice enhancement algorithms, performs voice recognition (VIT), and outputs the detected wake words and voice commands to the console, enabling hands-free voice control applications.

**XAF USB Example** The XAF USB example demonstrates DSP-powered USB audio processing in two configurations: USB speaker and USB microphone. The application uses shell commands to switch between modes, with the ARM core handling USB communication while the DSP processes audio.

## XAF Playback Example

### Table of Content

- [Overview](#)
- [Functionality](#)
- [Hardware Requirements](#)
- [Hardware Modifications](#)
- [Preparation](#)
- [Example Configuration](#)
- [Running the Demo](#)

- [Known Issues](#)

**Overview** The `dsp_xaf_playback` application demonstrates audio processing using the DSP core, the Xtensa Audio Framework (XAF) middleware library, and selected Xtensa audio codecs.

As shown in the table below, the application is supported on several development boards and each development board may have certain limitations, some development boards may also require hardware modifications or allow the use of an audio expansion board. Therefore, please check the supported features and [Hardware modifications](#) or [Example configuration](#) sections before running the demo.

**Limitations:**

- **MP3 encoder, G.711, G.722, BSAC, DAB+, DAB/MP2, DRM:** Provided only as linked libraries but are not enabled in the example.

**Functionality** The application includes the following main components:

1. **ARM Core (CM33)** - Handles user interface, SD card operations, and communicates with the DSP core
2. **DSP Core** - Processes audio data using the Xtensa Audio Framework (XAF)

The typical audio processing pipeline includes:

- File source component (reads from SD card)
- Decoder component (decodes compressed audio)
- Renderer component (outputs to audio hardware)

When the file playback command is issued, the ARM core reads the file from SD card and sends data to the DSP, which processes it and outputs to the audio hardware.

**Hardware Requirements**

- Development board (one of the following):
  - EVK-MIMXRT595 board
  - EVK-MIMXRT685 board
  - MIMXRT685-AUD-EVK board
  - MIMXRT700-EVK board
- Micro USB cable
- JTAG/SWD debugger
- Headphones with 3.5 mm stereo jack
- Personal Computer
- SD card with audio files (for file playback feature)

**Hardware Modifications** Some development boards need some hardware modifications to run the application.

- *EVK-MIMXRT595:*

To enable the example audio using WM8904 codec, connect pins as follows:

- JP7-1 <-> JP8-2



Note: The I3C Pin configuration in `pin_mux.c` is verified for default 1.8V, for 3.3V, need to manually configure slew rate to slow mode for I3C-SCL/SDA.

- *EVK-MIMXRT685:*

To enable the example audio using WM8904 codec, connect pins as follows:

- JP7-1 <--> JP8-2

- *MIMXRT685-AUD-EVK:*

- Set the hardware jumpers (Tower system/base module) to default settings.
- Set hardware jumpers JP2 2<-->3, JP44 1<-->2 and JP45 1<-->2.

- *MIMXRT700-EVK:*

Set the hardware jumpers to default settings.

## Preparation

1. Connect headphones to Audio HP / Line-Out connector (J4).
  - EVK-MIMXRT595 - J4
  - EVK-MIMXRT685 - J4
  - MIMXRT685-AUD-EVK - J4, J50, J51, J52
  - MIMXRT700-EVK - J29
2. Connect a micro USB cable between the PC host and the debug USB port on the development board.
  - EVK-MIMXRT595 - J40
  - EVK-MIMXRT685 - J5
  - MIMXRT685-AUD-EVK - J5
  - MIMXRT700-EVK - J54
3. Open a serial terminal with the following settings:
  - 115200 baud rate
  - 8 data bits
  - No parity
  - One stop bit
  - No flow control
4. Download the program for CM33 core to the target board.
5. Launch the debugger in your IDE to begin running the demo.
6. If building release configuration, start the `xt-ocd` daemon and download the program for DSP core to the target board. If building debug configuration, launch the Xtensa IDE or `xt-gdb` debugger to begin running the demo.

### Notes:

- DSP image can only be debugged using J-Link debugger. See the document ‘Getting Started with Xplorer’ for your particular board for more information.

**Example Configuration** The example can be configured by user. Before configuration, please check the [table](#) to see if the feature is supported on the development board.

- **MIMXRT700-EVK Decoder Configuration:**

RT700 has limited RAM on Cortex-M33 core 1 which limits the available decoders. Only SBC decoder is enabled by default. In order to enable a different decoder/encoder, it is necessary to define the appropriate define on project level. Use one of the following define from the list of the supported decoders on the HiFi1 core:

- XA\_AAC\_DECODER
- XA\_MP3\_DECODER
- XA\_SBC\_DECODER
- XA\_VORBIS\_DECODER
- XA\_OPUS\_DECODER

**Running the Demo** The ARM application will power and clock the DSP, so it must be loaded prior to loading the DSP application. The DSP application can be built by the following tools: Xtensa Xplorer or Xtensa C Compiler. Application for Cortex-M33 can be built by the other toolchains listed in MCUXpresso SDK Release Notes.

The release configurations of the demo will combine both applications into one ARM image. With this, the ARM core will load and start the DSP application on startup. Pre-compiled DSP binary images are provided under dsp/binary/ directory. If you make changes to the DSP application in release configuration, rebuild ARM application after building the DSP application. If you plan to use MCUXpresso IDE for cm33 you will have to make sure that the preprocessor symbol DSP\_IMAGE\_COPY\_TO\_RAM, found in IDE project settings, is defined to the value 1 when building release configuration.

The debug configurations will build two separate applications that need to be loaded independently. DSP application can be built by the following tools: Xtensa Xplorer or Xtensa C Compiler. Required tool versions can be found in MCUXpresso SDK Release Notes for the board. Application for cm33 can be built by the other toolchains listed there. If you plan to use MCUXpresso IDE for cm33 you will have to make sure that the preprocessor symbol DSP\_IMAGE\_COPY\_TO\_RAM, found in IDE project settings, is defined to the value 0 when building debug configuration. The ARM application will power and clock the DSP, so it must be loaded prior to loading the DSP application.

In order to debug both the Cortex-M33 and DSP side of the application, please follow the instructions:

1. It is necessary to run the Cortex-M33 side first and stop the application before the DSP\_Start function
2. Run the xt-ocd daemon with proper settings
3. Download and debug the DSP application

In order to get TRACE debug output from the XAF it is necessary to define XF\_TRACE 1 in the project settings. It is possible to save the TRACE output into RAM using DUMP\_TRACE\_TO\_BUF 1 define on project level. Please see the initialization of the TRACE function in the xaf\_main\_dsp.c file. For more details see XAF documentation.

When the demo runs successfully, the terminal will display the following output (example from MIMXRT700-EVK):

```
*****
DSP audio framework demo start
*****
```

(continues on next page)

(continued from previous page)

```
[CM33_Main] Configure codec

[DSP_Main] Cadence Xtensa Audio Framework
[DSP_Main] Library Name   : Audio Framework (Hostless)
[DSP_Main] Library Version : 3.5
[DSP_Main] API Version    : 3.2

[DSP_Main] start
[DSP_Main] established RPMsg link
[CM33_Main] DSP image copied to DSP TCM
[CM33_Main][APP_SDCARD_Task] start
[CM33_Main][APP_DSP_IPC_Task] start
[CM33_Main][APP_Shell_Task] start

Copyright 2024 NXP
```

Type `help` to see the command list. Similar description will be displayed on serial console (*If multi-channel playback mode is enabled, the description is slightly different. Available encoders/decoders may differ - refer to the [table](#).*):

```
"help": List all the registered commands

"exit": Exit program

"version": Query DSP for component versions

"file": Perform audio file decode and playback from SD card
USAGE: file [list|stop|<audio_file>]
list       List audio files on SD card available for playback
<audio_file> Select file from SD card and start playback

"decoder": Perform decode on DSP and play to speaker.
USAGE: decoder [aac|mp3|opus|sbc|vorbis_ogg|vorbis_raw]
aac:       Decode aac data
mp3:       Decode mp3 data
opus:      Decode opus data
sbc:       Decode sbc data
vorbis_ogg: Decode OGG VORBIS data
vorbis_raw: Decode raw VORBIS data

"encoder": Encode PCM data on DSP and compare with reference data.
USAGE: encoder [opus|sbc]
opus:      Encode pcm data using opus encoder
sbc:       Encode pcm data using sbc encoder

"src" Perform sample rate conversion on DSP

"gain": Perform PCM gain adjustment on DSP
```

Xtensa IDE log when command is playing a file (mp3/aac/vorbis/... ):

```
File playback start, initial buffer size: 16384
[DSP Codec] Audio Device Ready
[DSP Codec] Decoder component started
[DSP Codec] Setting decode playback format:
Decoder    : mp3_dec
Sample rate: 16000
Bit Width  : 16
Channels   : 2
[DSP Codec] Renderer component started
```

(continues on next page)

(continued from previous page)

```
[DSP Codec] Connected XA_DECODER -> XA_RENDERERER
[DSP_ProcessThread] start
[DSP_BufferThread] start
```

Xtensa IDE log when decoder command starts playback successfully:

```
[DSP_Main] Input buffer addr: 0x20020000, buffer size: 94276
[DSP Codec] Audio Device Ready
[DSP Codec] Decoder created
[DSP Codec] Decoder component started
[DSP Codec] Renderer component created
[DSP Codec] Connected XA_DECODER -> XA_RENDERERER
[DSP_ProcessThread] start
[DSP_ProcessThread] Execution complete - exiting
[DSP_ProcessThread] exiting
[DSP Codec] Audio device closed

[CM33 CMD] [APP_DSP_IPC_Task] response from DSP, cmd: 0, error: 0
[CM33 CMD] Decode complete
```

**MIMXRT685-AUD-EVK Multi-channel Support:** The MIMXRT685-AUD-EVK board supports multi-channel audio. When selecting audio files for playback, you can specify the number of channels:

```
***
file [list|stop]<audio_file> [<nchannel>]]
<nchannel>    Select the number of channels (2 or 8 can be selected).
NOTE: Selected audio file must meet the following parameters:
      - Sample rate: 96 kHz
      - Width:      32 bit
***
```

Xtensa IDE log when command is playing a PCM file:

```
***
[DSP_FILE_REN] Audio Device Ready
[DSP_FILE_REN] post-proc/pcm_gain component started
[DSP_FILE_REN] post-proc/client_proxy component started
[DSP_FILE_REN] Connected post-proc/pcm_gain -> post-proc/client_proxy
[DSP_FILE_REN] renderer component started
[DSP_FILE_REN] Connected post-proc/client_proxy -> renderer
[DSP_BufferThread] start
[DSP_ProcessThread] start
[DSP_CleanupThread] start3
***
```

### Known Issues

1. The “file stop” command doesn’t stop the playback for some small files (with low sample rate).
2. MIMXRT700-EVK: Has limited RAM on Cortex-M33 core 1 which limits the available decoders.

### XAF Record Example

### Table of Content

- [Overview](#)
- [Functionality](#)
- [Hardware Requirements](#)
- [Hardware Modifications](#)
- [Preparation](#)
- [Example Configuration](#)
- [Running the Demo](#)
- [Known Issues](#)

**Overview** The dsp\_xaf\_record application demonstrates audio processing using the DSP core, the Xtensa Audio Framework (XAF) middleware library, with a focus on audio recording, processing and voice recognition (VIT - Voice Intelligent Technology, Voice seeker).

As shown in the table below, the application is supported on several development boards and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and [Hardware modifications](#) or [Example configuration](#) sections before running the demo.

**Functionality** The application includes the following main components:

1. **ARM Core (CM33)** - Handles user interface and communicates with the DSP core
2. **DSP Core** - Processes audio data using the Xtensa Audio Framework (XAF)

The typical audio processing pipeline includes:

- Audio source component - DMIC audio
- VIT and Voice seeker component (perform voice recognition)
- Renderer component (playback on codec)

The application demonstrates recording from digital microphones (DMIC), processing the audio with voice enhancement algorithms, performing voice recognition, and prints back in console detected WakeWord and list of commands.

### Hardware Requirements

- Development board (one of the following):
  - EVK-MIMXRT595 board
  - EVK-MIMXRT685 board
  - MIMXRT685-AUD-EVK board (optionally with 8CH-DMIC expansion board - rev B required)
  - MIMXRT700-EVK board
- Micro USB cable
- JTAG/SWD debugger
- Headphones with 3.5 mm stereo jack
- Personal Computer

**Hardware Modifications** Some development boards need some hardware modifications to run the application.

- *EVK-MIMXRT595:*

To enable the example audio using WM8904 codec, connect pins as follows:

- JP7-1 <-> JP8-2

Note: The I3C Pin configuration in pin\_mux.c is verified for default 1.8V, for 3.3V, need to manually configure slew rate to slow mode for I3C-SCL/SDA.

- *EVK-MIMXRT685:*

To enable the example audio using WM8904 codec, connect pins as follows:

- JP7-1 <-> JP8-2

- *MIMXRT685-AUD-EVK*

1. Set the hardware jumpers (Tower system/base module) to default settings.
2. Set hardware jumpers JP2 2<->3, JP44 1<->2 and JP45 1<->2.

*For 8CH-DMIC expansion board (optional):*

1. Connect the 8CH-DMIC expansion board to the MIMXRT685-AUD-EVK board to the DMIC connector (J31). For safety reasons, the expansion board must be connected when the power supply is disconnected.
2. Set the hardware jumpers on the 8-DMIC expansion board to 2MIC, 3MICA, 3MICC config (Short: J6, J9, J10).
3. Set the hardware jumpers JP44 2<->3 and JP45 2<->3 on the MIMXRT685-AUD-EVK board for on-board DMIC bypass.

- *MIMXRT700-EVK:*

Set the hardware jumpers to default settings.

## Preparation

1. Connect headphones to Audio HP / Line-Out connector.
  - EVK-MIMXRT595 - J4
  - EVK-MIMXRT685 - J4
  - MIMXRT685-AUD-EVK - J4, J50 for third channel when using 3 microphones
  - MIMXRT700-EVK - J29
2. Connect a micro USB cable between the PC host and the debug USB port on the development board.
  - EVK-MIMXRT595 - J40
  - EVK-MIMXRT685 - J5
  - MIMXRT685-AUD-EVK - J5
  - MIMXRT700-EVK - J54
3. Open a serial terminal with the following settings:
  - 115200 baud rate
  - 8 data bits
  - No parity
  - One stop bit

- No flow control
4. Download the program for CM33 core to the target board.
  5. Launch the debugger in your IDE to begin running the demo.
  6. If building release configuration, start the xt-ocd daemon and download the program for DSP core to the target board. If building debug configuration, launch the Xtensa IDE or xt-gdb debugger to begin running the demo.

Notes:

- DSP image can only be debugged using J-Link debugger. See the document ‘Getting Started with Xplorer’ for your particular board for more information.

**Example Configuration** The example can be configured by user. Before configuration, please check the [table](#) to see if the feature is supported on the development board.

• **MIMXRT685-AUD-EVK 8CH-DMIC expansion board settings:**

Select how many microphones should be used

- Set the BOARD\_DMIC\_NUM preprocessor macro to 1,2, 3 (default) or 4 in the project for the CM33 core.
- When the 8CH-DMIC expansion board is used, the DMIC\_BOARD\_CONNECTED macro must be set to 1 (default) in the project for the DSP core.
- Important: When you set the value to 2, 3 or 4 you have to connect the 8CH-DMIC expansion board and set the DMIC\_BOARD\_CONNECTED macro to 1. Don’t forget set the hardware jumpers JP44 2-3 and JP45 2-3.

**Running the Demo** The ARM application will power and clock the DSP, so it must be loaded prior to loading the DSP application. The DSP application can be built by the following tools: Xtensa Xplorer or Xtensa C Compiler. Application for Cortex-M33 can be built by the other toolchains listed in MCUXpresso SDK Release Notes.

The release configurations of the demo will combine both applications into one ARM image. With this, the ARM core will load and start the DSP application on startup. Pre-compiled DSP binary images are provided under dsp/binary/ directory. If you make changes to the DSP application in release configuration, rebuild ARM application after building the DSP application. If you plan to use MCUXpresso IDE for cm33 you will have to make sure that the preprocessor symbol DSP\_IMAGE\_COPY\_TO\_RAM, found in IDE project settings, is defined to the value 1 when building release configuration.

The debug configurations will build two separate applications that need to be loaded independently. DSP application can be built by the following tools: Xtensa Xplorer or Xtensa C Compiler. Required tool versions can be found in MCUXpresso SDK Release Notes for the board. Application for cm33 can be built by the other toolchains listed there. If you plan to use MCUXpresso IDE for cm33 you will have to make sure that the preprocessor symbol DSP\_IMAGE\_COPY\_TO\_RAM, found in IDE project settings, is defined to the value 0 when building debug configuration. The ARM application will power and clock the DSP, so it must be loaded prior to loading the DSP application.

In order to debug both the Cortex-M33 and DSP side of the application, please follow the instructions:

1. It is necessary to run the Cortex-M33 side first and stop the application before the DSP\_Start function
2. Run the xt-ocd daemon with proper settings
3. Download and debug the DSP application

In order to get TRACE debug output from the XAF it is necessary to define XF\_TRACE 1 in the project settings. It is possible to save the TRACE output into RAM using DUMP\_TRACE\_TO\_BUF 1 define on project level. Please see the initialization of the TRACE function in the xaf\_main\_dsp.c file. For more details see XAF documentation.

**Running on CM33** When the demo runs successfully, the CM33 terminal will display the following output (example from MIMXRT700-EVK):

```
*****
DSP audio framework demo start
*****

[CM33 Main] Configure codec

[DSP_Main] Cadence Xtensa Audio Framework
[DSP_Main] Library Name   : Audio Framework (Hostless)
[DSP_Main] Library Version : 3.5
[DSP_Main] API Version    : 3.2

[DSP_Main] start
[DSP_Main] established RPMsg link
[CM33 Main] DSP image copied to DSP TCM
[CM33 Main][APP_DSP_IPC_Task] start
[CM33 Main][APP_Shell_Task] start

Copyright 2024 NXP

>>
```

Type help to see the command list. Similar description will be displayed on serial console (example from MIMXRT700-EVK):

```
"help": List all the registered commands

"exit": Exit program

"version": Query DSP for component versions

"record_dmic": Record DMIC audio , perform voice recognition (VIT) and playback on codec
USAGE: record_dmic [language]
For voice recognition say supported WakeWord and in 3s frame supported command.
If selected model contains strings, then WakeWord and list of commands will be printed in console.
NOTE: this command does not return to the shell
```

After running the “record\_dmic en” command, similar output will be printed

```
[CM33 CMD] Setting VIT language to en
[DSP_Main] Number of channels 1, sampling rate 16000, PCM width 32
[CM33 CMD] [APP_DSP_IPC_Task] response from DSP, cmd: 13, error: 0
[DSP Record] Audio Device Ready
[CM33 CMD] DSP DMIC Recording started
[CM33 CMD] To see VIT functionality say wakeword and command
[DSP VIT] VIT Model info
[DSP VIT] VIT Model Release = 0x40a00
[DSP VIT] Language supported : English
[DSP VIT] Number of WakeWords supported : 2
[DSP VIT] Number of Commands supported : 12
[DSP VIT] VIT_Model integrating WakeWord and Voice Commands strings : YES
[DSP VIT] WakeWords supported :
[DSP VIT] 'HEY NXP'
[DSP VIT] 'HEY TV'
```

(continues on next page)



(continued from previous page)

```

[DSP VIT] Voice commands supported :
[DSP VIT] 'MUTE'
[DSP VIT] 'NEXT'
[DSP VIT] 'SKIP'
[DSP VIT] 'PAIR DEVICE'
[DSP VIT] 'PAUSE'
[DSP VIT] 'STOP'
[DSP VIT] 'POWER OFF'
[DSP VIT] 'POWER ON'
[DSP VIT] 'PLAY MUSIC'
[DSP VIT] 'PLAY GAME'
[DSP VIT] 'WATCH CARTOON'
[DSP VIT] 'WATCH MOVIE'
[DSP Record] connected CAPTURER -> GAIN_0
[DSP Record] connected XA_GAIN_0 -> XA_VIT_PRE_PROC_0
[DSP Record] connected XA_VIT_PRE_PROC_0 -> XA_RENDERER_0
[DSP VIT] - WakeWord detected 1 HEY NXP
[DSP VIT] - Voice Command detected 6 STOP

```

**Xtensa IDE log of successful start of command:**

```

Number of channels 2, sampling rate 16000, PCM width 16
Audio Device Ready
connected CAPTURER -> GAIN_0
connected CAPTURER -> XA_VIT_PRE_PROC_0
connected XA_VIT_PRE_PROC_0 -> XA_RENDERER_0

```

**Running on DSP** Debug configuration: When the demo runs successfully, the terminal will display the following:

```

Cadence Xtensa Audio Framework
Library Name   : Audio Framework (Hostless)
Library Version : 3.2
API Version    : 3.0

[DSP_Main] start
[DSP_Main] established RPMsg link
Number of channels 2, sampling rate 16000, PCM width 16

Audio Device Ready
VoiceSeekerLight lib initialized!
===== VoiceSeekerLight Configuration =====
version = 0.6.0
num mics = 2
max num mics = 4
mic0 = (35, 0, 0)
mic1 = (-35, 0, 0)
mic2 = (0, -35, 0)
num_spks = 0
max num spks = 2
samplerate = 16000
framesize_in = 32
framesize_out = 480
create_aec = 0
create_doa = 0
buffer_length_sec = 1.5
aec_filter_length_ms = 0
===== VoiceSeekerLight Memory Allocation =====
VoiceSeekerLib allocated 80592 persistent bytes
VoiceSeekerLib allocated 3840 scratch bytes

```

(continues on next page)

(continued from previous page)

```

===== VoiceSeekerLight Memory Usage
↔ =====
=====
Total           = 72400 bytes

connected CAPTurer -> GAIN_0
connected XA_GAIN_0 -> XA_VOICE_SEEKER_0
connected XA_VOICE_SEEKER_0 -> XA_VIT_PRE_PROC_0
connected XA_VIT_PRE_PROC_0 -> XA_RENDERER_0

```

**Known Issues** There are limited features in release SRAM target because of memory limitations. To enable/disable components, set appropriate preprocessor define in project settings to 0/1 (e.g. XA\_VIT\_PRE\_PROC etc.). Debug and flash targets have full functionality enabled.

## XAF USB Example

### Table of Content

- [Overview](#)
- [Functionality](#)
- [Hardware Requirements](#)
- [Hardware Modifications](#)
- [Preparation](#)
- [Running the Demo](#)
- [Known Issues](#)

**Overview** The dsp\_xaf\_usb\_demo application demonstrates audio processing using the DSP core, the Xtensa Audio Framework (XAF) middleware library.

As shown in the table below, the application is supported on several development boards and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and [Hardware modifications](#) section before running the demo.

**Functionality** The application includes the following main components:

1. **ARM Core (CM33)** - Handles user interface, and communicates with the DSP core
2. **DSP Core** - Processes audio data using the Xtensa Audio Framework (XAF)

The XAF USB example demonstrates DSP-powered USB audio processing in two configurations: USB speaker and USB microphone. The application uses shell commands to switch between modes, with the ARM core handling USB communication while the DSP processes audio.

- **USB Speaker Mode (USB2.0 □ Line out):** Receives audio from a USB host, processes it on the DSP, and outputs through the headphone jack, making the device function as a USB speaker for your computer.
- **USB Microphone Mode (DMIC □ USB2.0):** Captures audio from the onboard digital microphones, processes it on the DSP, and streams it to a USB host as a standard audio input device.

## Hardware Requirements

- Development board (one of the following):
  - EVK-MIMXRT595 board
  - EVK-MIMXRT685 board
  - MIMXRT685-AUD-EVK board
  - MIMXRT700-EVK board
- 2x Micro USB cable
- JTAG/SWD debugger
- Headphones with 3.5 mm stereo jack
- Personal Computer

**Hardware Modifications** Some development boards need some hardware modifications to run the application.

- *EVK-MIMXRT595:*

To enable the example audio using WM8904 codec, connect pins as follows:

- JP7-1 <--> JP8-2

Note: The I3C Pin configuration in pin\_mux.c is verified for default 1.8V, for 3.3V, need to manually configure slew rate to slow mode for I3C-SCL/SDA.

- *EVK-MIMXRT685:*

To enable the example audio using WM8904 codec, connect pins as follows:

- JP7-1 <--> JP8-2

- *MIMXRT685-AUD-EVK*

- Set the hardware jumpers (Tower system/base module) to default settings.
- Set hardware jumpers JP2 2<-->3, JP44 1<-->2 and JP45 1<-->2.

- *MIMXRT700-EVK:*

Set the hardware jumpers to default settings.

## Preparation

1. Connect headphones to Audio HP / Line-Out connector.
  - EVK-MIMXRT595 - J4
  - EVK-MIMXRT685 - J4
  - MIMXRT685-AUD-EVK - J4
  - MIMXRT700-EVK - J29
2. Connect the first micro USB cable between the PC host and the debug USB port on the development board.
  - EVK-MIMXRT595 - J40
  - EVK-MIMXRT685 - J5
  - MIMXRT685-AUD-EVK - J5
  - MIMXRT700-EVK - J54

3. Connect the second micro USB cable between the PC host and the USB port on the development board.
  - EVK-MIMXRT595 - J38
  - EVK-MIMXRT685 - J7
  - MIMXRT685-AUD-EVK - J7
  - MIMXRT700-EVK - J40
4. Open a serial terminal with the following settings:
  - 115200 baud rate
  - 8 data bits
  - No parity
  - One stop bit
  - No flow control
5. Download the program for CM33 core to the target board.
6. Launch the debugger in your IDE to begin running the demo.
7. If building release configuration, start the xt-ocd daemon and download the program for DSP core to the target board. If building debug configuration, launch the Xtensa IDE or xt-gdb debugger to begin running the demo.

Notes:

- DSP image can only be debugged using J-Link debugger. See the document ‘Getting Started with Xplorer’ for your particular board for more information.

**Running the Demo** The ARM application will power and clock the DSP, so it must be loaded prior to loading the DSP application. The DSP application can be built by the following tools: Xtensa Xplorer or Xtensa C Compiler. Application for Cortex-M33 can be built by the other toolchains listed in MCUXpresso SDK Release Notes.

The release configurations of the demo will combine both applications into one ARM image. With this, the ARM core will load and start the DSP application on startup. Pre-compiled DSP binary images are provided under dsp/binary/ directory. If you make changes to the DSP application in release configuration, rebuild ARM application after building the DSP application. If you plan to use MCUXpresso IDE for cm33 you will have to make sure that the preprocessor symbol `DSP_IMAGE_COPY_TO_RAM`, found in IDE project settings, is defined to the value 1 when building release configuration.

The debug configurations will build two separate applications that need to be loaded independently. DSP application can be built by the following tools: Xtensa Xplorer or Xtensa C Compiler. Required tool versions can be found in MCUXpresso SDK Release Notes for the board. Application for cm33 can be built by the other toolchains listed there. If you plan to use MCUXpresso IDE for cm33 you will have to make sure that the preprocessor symbol `DSP_IMAGE_COPY_TO_RAM`, found in IDE project settings, is defined to the value 0 when building debug configuration. The ARM application will power and clock the DSP, so it must be loaded prior to loading the DSP application.

In order to debug both the Cortex-M33 and DSP side of the application, please follow the instructions:

1. It is necessary to run the Cortex-M33 side first and stop the application before the `DSP_Start` function
2. Run the xt-ocd daemon with proper settings
3. Download and debug the DSP application

In order to get TRACE debug output from the XAF it is necessary to define XF\_TRACE 1 in the project settings. It is possible to save the TRACE output into RAM using DUMP\_TRACE\_TO\_BUF 1 define on project level. Please see the initialization of the TRACE function in the xaf\_main\_dsp.c file. For more details see XAF documentation.

**Running on CM33** When the demo runs successfully, the CM33 terminal will display the following output (example from MIMXRT700-EVK):

```
*****
DSP audio framework demo start
*****

[CM33 Main] Configure codec

[DSP_Main] Cadence Xtensa Audio Framework
[DSP_Main] Library Name   : Audio Framework (Hostless)
[DSP_Main] Library Version : 3.5
[DSP_Main] API Version    : 3.2

[DSP_Main] start
[DSP_Main] established RPMsg link
[CM33 Main] DSP image copied to DSP TCM
[CM33 Main][APP_DSP_IPC_Task] start
[CM33 Main][APP_Shell_Task] start

Copyright 2024 NXP

>>
```

Type help to see the command list. Similar description will be displayed on serial console (example from MIMXRT700-EVK):

```
"help": List all the registered commands

"exit": Exit program

"version": Query DSP for component versions

"usb_speaker": Perform usb speaker device and playback on DSP
  USAGE: usb_speaker [start|stop]
  start      Start usb speaker device and playback on DSP
  stop       Stop usb speaker device and playback on DSP

"usb_mic": Record DMIC audio and playback on usb microphone audio device
  USAGE: usb_mic [start|stop]
  start      Start record and playback on usb microphone audio device
  stop       Stop record and playback on usb microphone audio device
```

When usb\_speaker command starts playback successfully, the terminal will display following output:

```
[APP_DSP_IPC_Task] response from DSP, cmd: 21, error: 0
DSP USB playback start
>>
```

Xtensa IDE log when command is playing a file:

```
USB speaker start, initial buffer size: 960
[DSP_USB_SPEAKER] Audio Device Ready
[DSP_USB_SPEAKER] post-proc/pcm_gain component started
[DSP_USB_SPEAKER] post-proc/client_proxy component started
```

(continues on next page)

(continued from previous page)

```
[DSP_USB_SPEAKER] Connected post-proc/pcm_gain -> post-proc/client_proxy  
[DSP_USB_SPEAKER] renderer component started  
[DSP_USB_SPEAKER] Connected post-proc/client_proxy -> renderer  
[DSP_ProcessThread] start  
[DSP_BufferThread] start  
[DSP_CleanupThread] start
```

The USB device on your host will be enumerated as XAF USB DEMO.

Xtensa IDE will not show any additional log entry.

**Running the demo DSP** Debug configuration: When the demo runs successfully, the terminal will display the following:

```
Cadence Xtensa Audio Framework  
Library Name   : Audio Framework (Hostless)  
Library Version : 2.6p1  
API Version    : 2.0  
  
[DSP_Main] start  
[DSP_Main] established RPMMsg link
```

### Known Issues

- When starting the “usb\_speaker” after the “usb\_mic” command, the sound output may be distorted. Please power cycle the board.

# Chapter 4

## RTOS

### 4.1 FreeRTOS

#### 4.1.1 FreeRTOS kernel

Open source RTOS kernel for small devices.

[FreeRTOS kernel for MCUXpresso SDK Readme](#)

[FreeRTOS kernel for MCUXpresso SDK ChangeLog](#)

[FreeRTOS kernel Readme](#)

#### 4.1.2 FreeRTOS drivers

This is set of NXP provided FreeRTOS reentrant bus drivers.

#### 4.1.3 backoffalgorithm

Algorithm for calculating exponential backoff with jitter for network retry attempts.

[Readme](#)

#### 4.1.4 corehttp

C language HTTP client library designed for embedded platforms.

#### 4.1.5 corejson

JSON parser.

## **Readme**

### **4.1.6 coremqtt**

MQTT publish/subscribe messaging library.

### **4.1.7 coremqtt-agent**

The coreMQTT Agent library is a high level API that adds thread safety to the coreMQTT library.

## **Readme**

### **4.1.8 corepkcs11**

PKCS #11 key management library.

## **Readme**

### **4.1.9 freertos-plus-tcp**

Open source RTOS FreeRTOS Plus TCP.

## **Readme**