



# MCUXpresso SDK Documentation

Release 25.12.00-pvw2



NXP  
Nov 14, 2025



# Table of contents

|          |                                                                 |           |
|----------|-----------------------------------------------------------------|-----------|
| <b>1</b> | <b>EVK-MIMX8MQ</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                      | 24        |
| 1.3.1    | Getting Started with MCUXpresso SDK Repository                  | 24        |
| 1.4      | Release Notes                                                   | 37        |
| 1.4.1    | MCUXpresso SDK Release Notes                                    | 37        |
| 1.5      | ChangeLog                                                       | 41        |
| 1.5.1    | MCUXpresso SDK Changelog                                        | 41        |
| 1.6      | Driver API Reference Manual                                     | 67        |
| 1.7      | Middleware Documentation                                        | 67        |
| 1.7.1    | Multicore                                                       | 67        |
| 1.7.2    | FreeMASTER                                                      | 68        |
| 1.7.3    | FreeRTOS                                                        | 68        |
| <b>2</b> | <b>MIMX8MQ6</b>                                                 | <b>69</b> |
| 2.1      | CACHE: LMEM CACHE Memory Controller                             | 69        |
| 2.2      | Clock                                                           | 70        |
| 2.3      | MIPI CSI2 RX: MIPI CSI2 RX Driver                               | 101       |
| 2.4      | ECSPI: Enhanced Configurable Serial Peripheral Interface Driver | 108       |
| 2.5      | ECSPI Driver                                                    | 108       |
| 2.6      | ECSPI SDMA Driver                                               | 121       |
| 2.7      | GPC: General Power Controller Driver                            | 124       |
| 2.8      | GPIO: General-Purpose Input/Output Driver                       | 126       |
| 2.9      | GPT: General Purpose Timer                                      | 130       |
| 2.10     | I2C: Inter-Integrated Circuit Driver                            | 138       |
| 2.11     | I2C Driver                                                      | 138       |
| 2.12     | Iomuxc_driver                                                   | 150       |
| 2.13     | IRQSTEER: Interrupt Request Steering Driver                     | 166       |
| 2.14     | Common Driver                                                   | 171       |
| 2.15     | LCDIF: LCD interface                                            | 183       |
| 2.16     | MCM: Miscellaneous Control Module                               | 195       |
| 2.17     | MIPI DSI Driver                                                 | 200       |
| 2.18     | MIPI_DSI: MIPI DSI Host Controller                              | 218       |
| 2.19     | MU: Messaging Unit                                              | 218       |
| 2.20     | OCOTP: On Chip One-Time Programmable controller.                | 228       |
| 2.21     | PWM: Pulse Width Modulation Driver                              | 231       |
| 2.22     | QSPI: Quad Serial Peripheral Interface                          | 237       |
| 2.23     | Quad Serial Peripheral Interface Driver                         | 237       |
| 2.24     | RDC: Resource Domain Controller                                 | 250       |
| 2.25     | RDC_SEMA42: Hardware Semaphores Driver                          | 256       |
| 2.26     | SAI: Serial Audio Interface                                     | 258       |
| 2.27     | SAI Driver                                                      | 258       |
| 2.28     | SAI SDMA Driver                                                 | 282       |
| 2.29     | SDMA: Smart Direct Memory Access (SDMA) Controller Driver       | 285       |

|          |                                                                    |            |
|----------|--------------------------------------------------------------------|------------|
| 2.30     | SEMA4: Hardware Semaphores Driver . . . . .                        | 303        |
| 2.31     | SNVS: Secure Non-Volatile Storage . . . . .                        | 306        |
| 2.32     | Secure Non-Volatile Storage High-Power . . . . .                   | 306        |
| 2.33     | Secure Non-Volatile Storage Low-Power . . . . .                    | 315        |
| 2.34     | SPDIF: Sony/Philips Digital Interface . . . . .                    | 321        |
| 2.35     | SRC: System Reset Controller Driver . . . . .                      | 335        |
| 2.36     | TMU: Thermal Management Unit Driver . . . . .                      | 338        |
| 2.37     | UART: Universal Asynchronous Receiver/Transmitter Driver . . . . . | 343        |
| 2.38     | UART Driver . . . . .                                              | 343        |
| 2.39     | UART FreeRTOS Driver . . . . .                                     | 359        |
| 2.40     | UART SDMA Driver . . . . .                                         | 359        |
| 2.41     | USDHC: Ultra Secured Digital Host Controller Driver . . . . .      | 361        |
| 2.42     | WDOG: Watchdog Timer Driver . . . . .                              | 388        |
| <b>3</b> | <b>Middleware . . . . .</b>                                        | <b>395</b> |
| 3.1      | Motor Control . . . . .                                            | 395        |
| 3.1.1    | FreeMASTER . . . . .                                               | 395        |
| <b>4</b> | <b>RTOS . . . . .</b>                                              | <b>433</b> |
| 4.1      | FreeRTOS . . . . .                                                 | 433        |
| 4.1.1    | FreeRTOS kernel . . . . .                                          | 433        |
| 4.1.2    | FreeRTOS drivers . . . . .                                         | 433        |
| 4.1.3    | backoffalgorithm . . . . .                                         | 433        |
| 4.1.4    | corehttp . . . . .                                                 | 433        |
| 4.1.5    | corejson . . . . .                                                 | 433        |
| 4.1.6    | coremqtt . . . . .                                                 | 434        |
| 4.1.7    | corepkcs11 . . . . .                                               | 434        |
| 4.1.8    | freertos-plus-tcp . . . . .                                        | 434        |

This documentation contains information specific to the evkmimx8mq board.

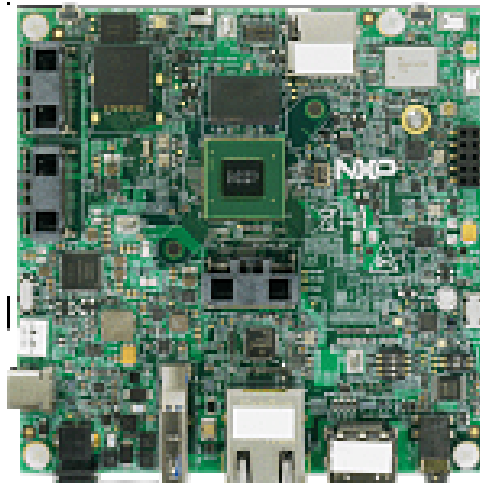


# Chapter 1

## EVK-MIMX8MQ

### 1.1 Overview

The i.MX 8MQuad family of boards provides a powerful and flexible development system for NXP's Cortex-M4 MCUs.



MCU device and part on board is shown below:

- Device: MIMX8MQ6
- PartNumber: MIMX8MQ6DVAJZ

### 1.2 Getting Started with MCUXpresso SDK Package

#### 1.2.1 Getting Started with Package

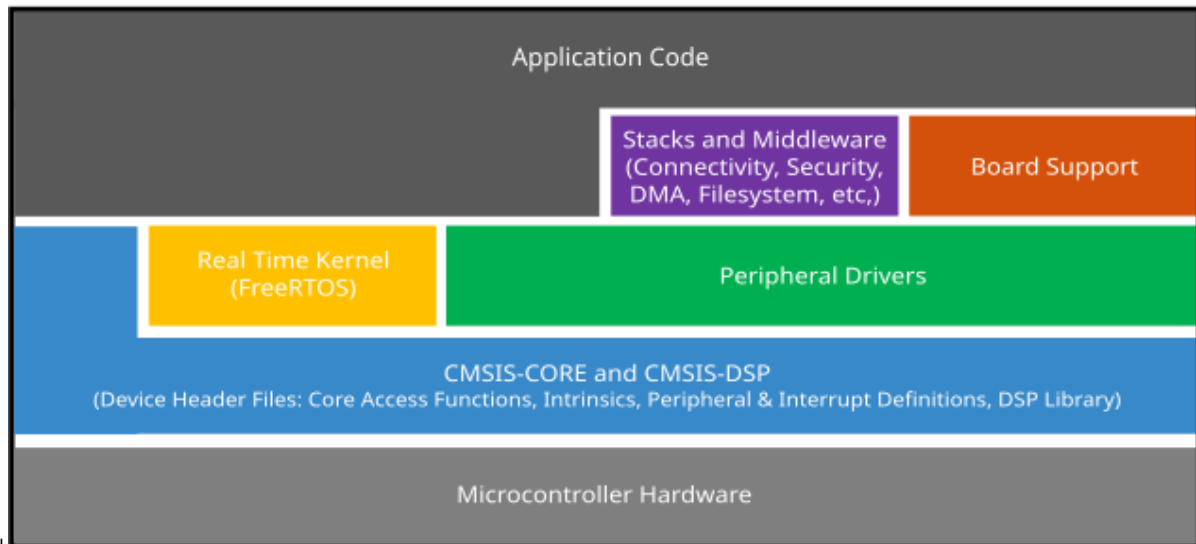
##### Overview

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 demo applications. The MCUXpresso SDK also contains optional RTOS integrations such as FreeRTOS and Azure RTOS, and device stack to support rapid development on devices.

For supported toolchain versions, see *MCUXpresso SDK Release Notes Supporting i.MX 8M Devices* (document MCUXSDKIMX8MRN).

For the latest version of this and other MCUXpresso SDK documents, see the MCUXpresso SDK homepage [MCUXpresso-SDK: Software Development Kit for MCUXpresso](#).



## MCUXpresso SDK board support folders

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

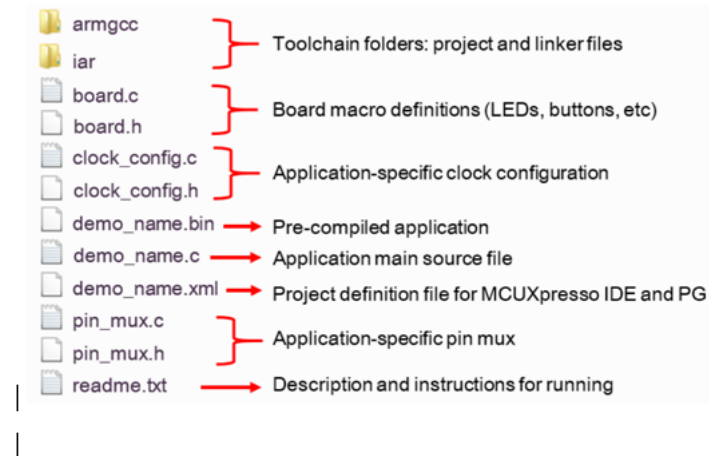
- `cmsis_driver_examples`: Simple applications intended to concisely illustrate how to use CMSIS drivers.
- `demo_apps`: Full-featured applications intended to highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- `driver_examples`: Simple applications intended to concisely illustrate how to use the MCUXpresso SDK's peripheral drivers for a single use case.
- `rtos_examples`: Basic FreeRTOS™ OS examples showcasing the use of various RTOS objects (semaphores, queues, and so on) and interfacing with the MCUXpresso SDK's RTOS drivers
- `multicore_examples`: Simple applications intended to concisely illustrate how to use middleware/multicore stack.

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



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

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



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

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

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

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

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

For examples containing 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 folders](#)

## Toolchain introduction

The MCUXpresso SDK release for i.MX 8M Devices includes the build system to be used with some toolchains. In this chapter, the toolchain support is presented and detailed.

**Compiler/Debugger** The release supports building and debugging with the toolchains listed in Table 1.

The user can choose the appropriate one for development.

- Arm GCC + SEGGER J-Link GDB Server. This is a command line tool option and it supports both Windows OS and Linux OS.
- IAR Embedded Workbench for Arm and SEGGER J-Link software. The IAR Embedded Workbench is an IDE integrated with editor, compiler, debugger, and other components. The SEGGER J-Link software provides the driver for the J-Link Plus debugger probe and supports the device to attach, debug, and download.

| Com-<br>piler/Debugger      | Supported host<br>OS   | Debug<br>probe | Tool website                                                                                                                |
|-----------------------------|------------------------|----------------|-----------------------------------------------------------------------------------------------------------------------------|
| ArmGCC/J-Link<br>GDB server | Windows<br>OS/Linux OS | J-Link<br>Plus | <a href="https://developer.arm.com/open-source/gnu-toolchain/gnu-rm">developer.arm.com/open-source/gnu-toolchain/gnu-rm</a> |

[www.segger.com](http://www.segger.com)

| | IAR/J-Link | Windows OS | J-Link Plus | [www.iar.com](http://www.iar.com)

[www.segger.com](http://www.segger.com)

|

Download the corresponding tools for the specific host OS from the website.

**Note:** To support i.MX 8M Dual/8M Quad, the patch for IAR should be installed. The patch named [iar\\_support\\_patch\\_imx8mq.zip](#) can be used with MCUXpresso SDK. See the readme.txt in the patch for additional information about patch installation.

**Parent topic:** [Toolchain introduction](#)

## Run a demo application using IAR

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK. The `hello_world` demo application targeted for the MIMX8MQ-EVK hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

**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
```

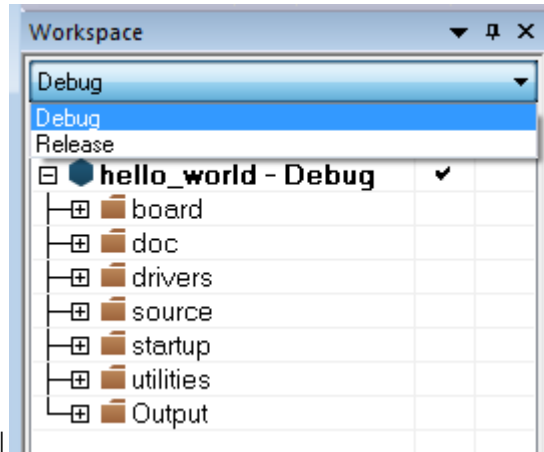
Using the MIMX8MQ-EVK hardware platform as an example, the `hello_world` workspace is located in;

```
<install_dir>/boards/evkmimx8mq/demo_apps/hello_world/iar/hello_world.eww
```

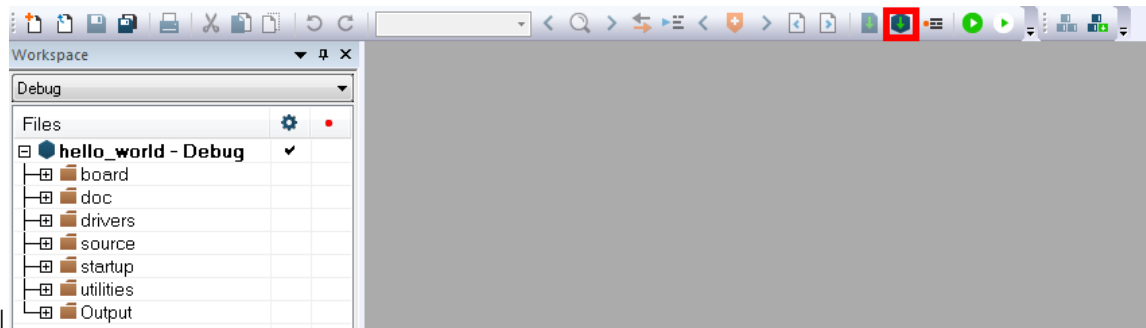
Other example applications may have additional folders in their path.

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.

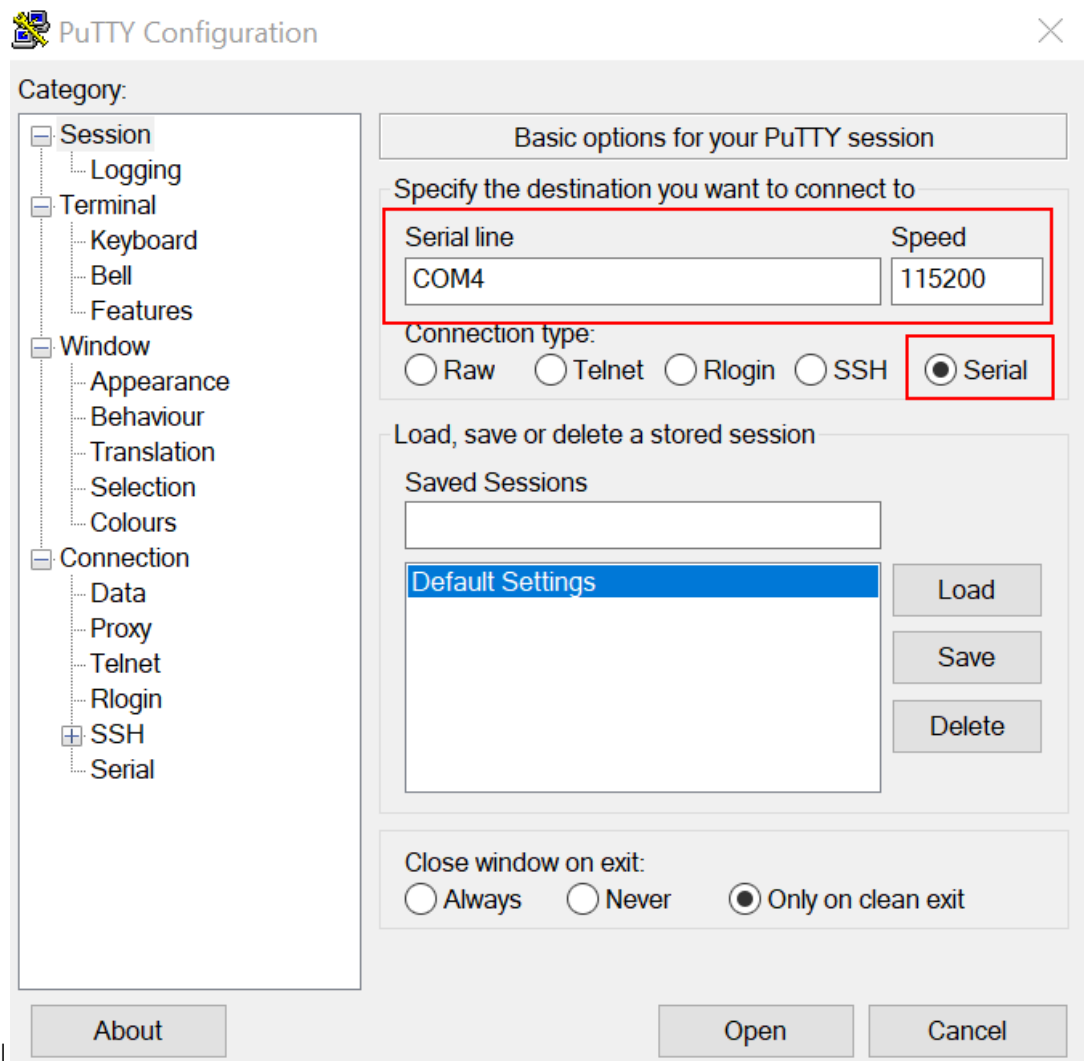


4. The build completes without errors.

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

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

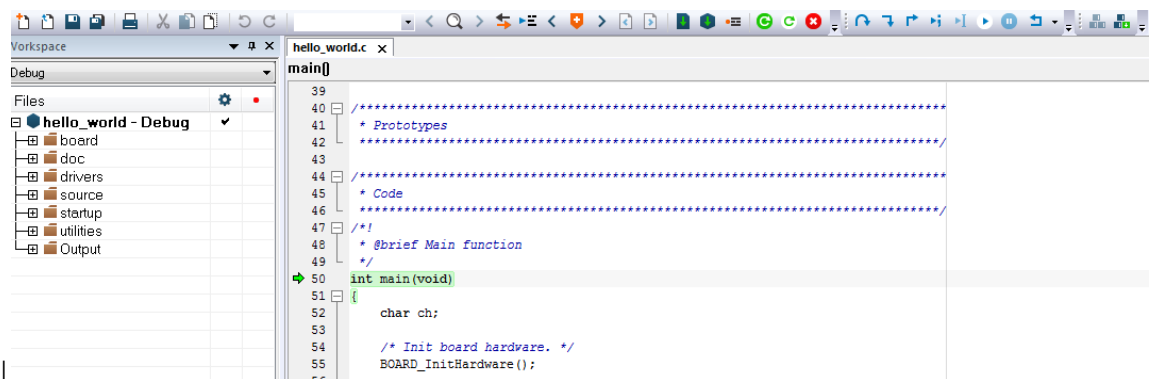
1. This board supports the J-Link PLUS debug probe. Before using it, install SEGGER J-Link software, which can be downloaded from <http://www.segger.com/downloads/jlink/>.
2. Connect the development platform to your PC via USB cable between the USB-UART MICRO USB connector and the PC USB connector, then connect 12 V power supply and J-Link Plus to the device.
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 baud rate
  2. No parity
  3. 8 data bits
  4. 1 stop bit



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



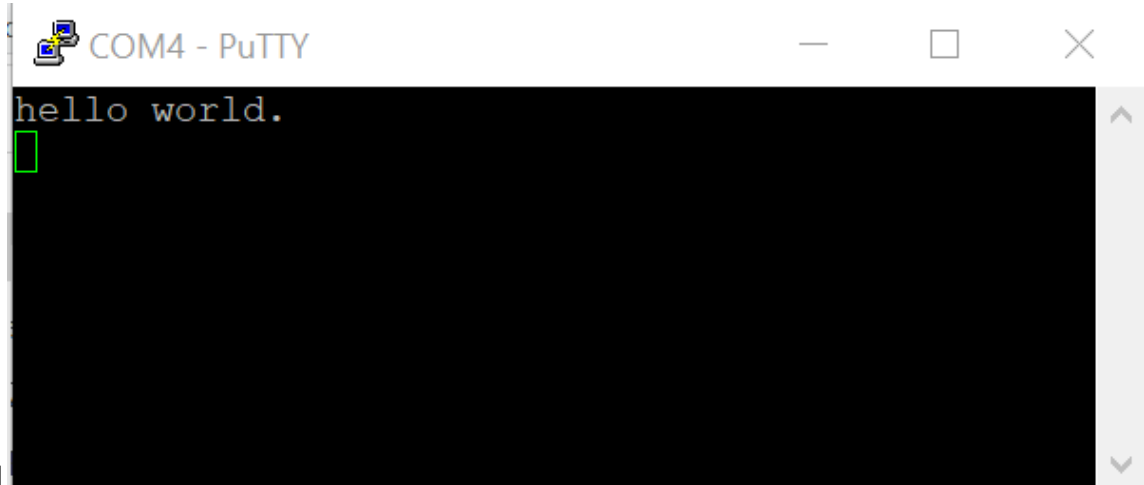
5. The application then downloads to the target and automatically runs to the `main()` function.



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



7. The `hello_world` application now runs and a banner appears on the terminal. If this does not occur, check your terminal settings and connections.



**\*\*Note:\*\*** For converting the DDR target elf to bin, run the following commands (take git bash console as example).

1. For the elf file built by Arm GCC:  
`$ <ARMGCC PATH>/bin/arm-none-eabi-objcopy.exe -Obinary --remove-section=.stacktop_and_pc <hello_world_<mcore type>.elf> <hello_world_<mcore type>.bin>`
2. For the elf/out file built by IAR:  
`$ <ARMGCC PATH>/bin/arm-none-eabi-objcopy.exe --remove-section=.stacktop_and_pc hello_world_<mcore type>.elf hello_world_<mcore type>_stripped.elf`  
`$ <IAR PATH>/arm/bin/elftool.exe --bin hello_world_<mcore type>_stripped.elf hello_world_<mcore type>.bin`

**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 targeted for i.MX 8M Quad platform is used as an example, though these steps can be applied to any board, demo or example application in the MCUXpresso SDK.

**Linux OS host** The following sections provide steps to run a demo compiled with Arm GCC on Linux host.

**Set up toolchain** This section contains the steps to install the necessary components required to build and run a MCUXpresso SDK demo application with the Arm GCC toolchain, as supported by the MCUXpresso SDK.

**Install GCC Arm embedded tool chain** Download and run the installer from [launchpad.net/gcc-arm-embedded](https://launchpad.net/gcc-arm-embedded). 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 the *MCUXpresso SDK Release Notes* (document MCUXSDKRN).

**Note:** See [Host setup](#) for Linux OS before compiling the application.

**Parent topic:**Set up toolchain

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

```
$ export ARMGCC_DIR=/work/platforms/tmp/gcc-arm-none-eabi-7-2017-q4-major
```

```
$ export PATH=$PATH:/work/platforms/tmp/gcc-arm-none-eabi-7-2017-q4-major/bin
```

**Parent topic:**Set up toolchain

**Parent topic:**Linux OS host

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

1. 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>/boards/evkmimx8mq/demo\_apps/hello\_world/armgcc

2. Run the build\_debug.sh script on the command line to perform the build. The output is shown as below:

```
$ ./build_debug.sh
-- TOOLCHAIN_DIR: /work/platforms/tmp/gcc-arm-none-eabi-7-2017-q4-major
-- BUILD_TYPE: debug
-- TOOLCHAIN_DIR: /work/platforms/tmp/gcc-arm-none-eabi-7-2017-q4-major
-- BUILD_TYPE: debug
-- The ASM compiler identification is GNU
-- Found assembler: /work/platforms/tmp/gcc-arm-none-eabi-7-2017-q4-major/bin/arm-none-eabi-gcc
-- Configuring done
-- Generating done
-- Build files have been written to:
/work/platforms/tmp/nxp/SDK\2.3.0\EVK-MIMX8MQ/boards/evkmimx8mq/demo\_apps/hello\_
↪world/armgcc
Scanning dependencies of target hello_world.elf
[ 6%] Building C object CMakeFiles/hello\_world.elf.dir/work/platforms/tmp/nxp/SDK\2.3.0\_
↪EVK-MIMX8MQ/boards/evkmimx8mq/demo\_apps/hello\_world/hello\_world.c.obj
< -- skipping lines -- >
[100%] Linking C executable debug/hello_world.elf
[100%] Built target hello_world.elf
```

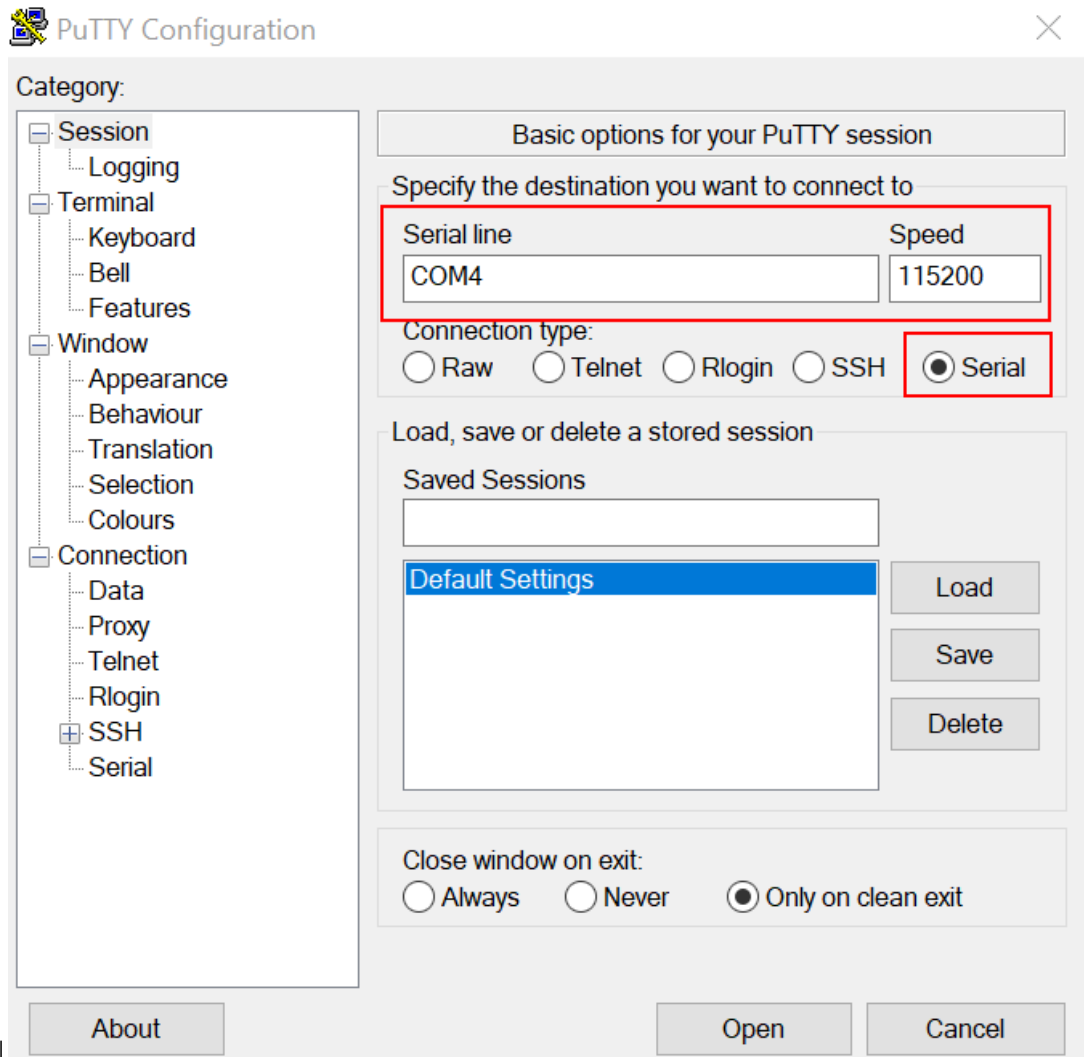
**Parent topic:**Linux OS host

**Run an example application** This section describes steps to run a demo application using J-Link GDB Server application.

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 USB-UART connector and the PC USB connector. If using a standalone J-Link debug pod, also 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 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



3. Open the J-Link GDB Server application. Assuming the J-Link software is installed, the application can be launched from a new terminal for the MIMX8MQ6\_M4 device:

```
$ JLinkGDBServer -if JTAG -device MIMX8MQ6\_M4
SEGGER J-Link GDB Server V6.22a Command Line Version
JLinkARM.dll V6.22g \(\DLL compiled Jan 17 2018 16:40:32\)
Command line: -if JTAG -device MIMX8MQ6\_M4
-----GDB Server start settings-----
GDBInit file: none
GDB Server Listening port: 2331
SWO raw output listening port: 2332
Terminal I/O port: 2333
```

(continues on next page)

(continued from previous page)

```

Accept remote connection: yes
< -- Skipping lines -- >
Target connection timeout: 0 ms
-----J-Link related settings-----
J-Link Host interface: USB
J-Link script: none
J-Link settings file: none
-----Target related settings-----
Target device: MIMX8MQ6\_M4
Target interface: JTAG
Target interface speed: 1000 kHz
Target endian: little
Connecting to J-Link...
J-Link is connected.
Firmware: J-Link V10 compiled Jan 11 2018 10:41:05
Hardware: V10.10
S/N: 600101610
Feature(s): RDI, FlashBP, FlashDL, JFlash, GDB
Checking target voltage...
Target voltage: 3.39 V
Listening on TCP/IP port 2331
Connecting to target...
J-Link found 1 JTAG device, Total IRLen = 4
JTAG ID: 0x5BA00477 \((Cortex-M4\)
Connected to target
Waiting for GDB connection...

```

4. 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/evkmimx8mq/demo\_apps/hello\_world/armgcc/debug*
```

5. Start the GDB client:

```

$ arm-none-eabi-gdb hello_world.elf
GNU gdb (GNU Tools for Arm Embedded Processors 7-2017-q4-major) 8.0.50.20171128-git
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-linux-gnu --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...
(gdb)

```

6. Connect to the GDB server and load the binary by running the following commands:

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



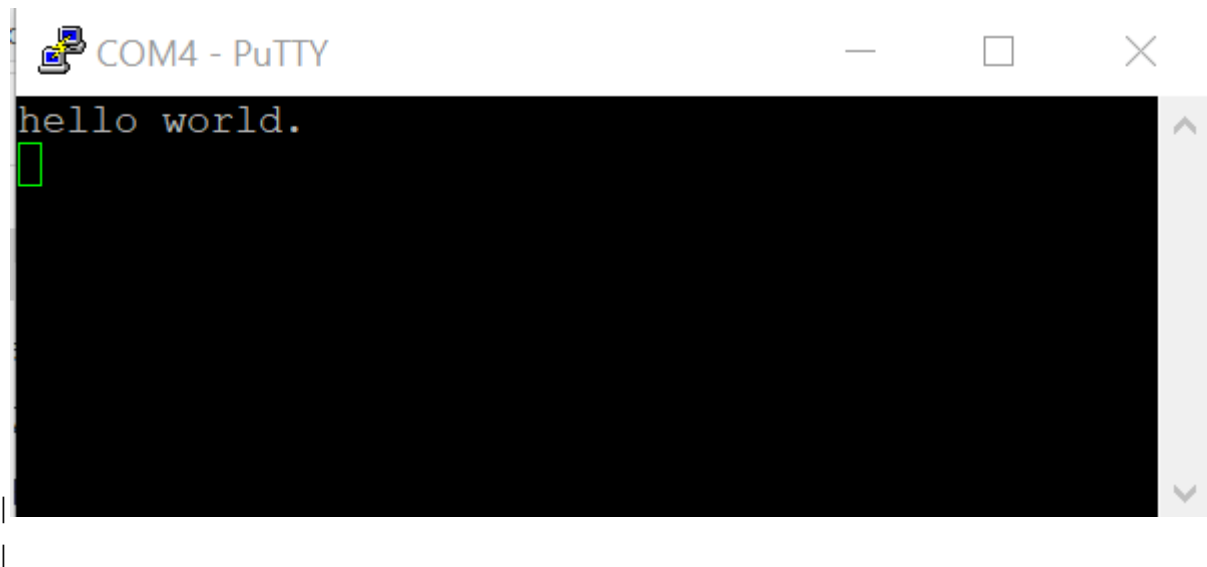
## 4. load

```
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
0x1ffe0008 in __isr_vector__
(gdb) monitor reset
Resetting target
(gdb) monitor halt
(gdb) load
Loading section .interrupts, size 0x240 lma 0x1ffe0000
Loading section .text, size 0x3858 lma 0x1ffe0240
Loading section .ARM, size 0x8 lma 0x1ffe3a98
Loading section .init__array, size 0x4 lma 0x1ffe3aa0
Loading section .fini__array, size 0x4 lma 0x1ffe3aa4
Loading section .data, size 0x64 lma 0x1ffe3aa8
Start address 0x1ffe02f4, load size 15116
Transfer rate: 81 KB/sec, 2519 bytes/write.
(gdb\)
```

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

```
(gdb) monitor go
```

The `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.



**Parent topic:**Linux OS host

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

**Windows OS host** The following sections provide steps to run a demo compiled with Arm GCC on Windows OS host.

**Set up toolchain** This section contains the steps to install the necessary components required to build and run a MCUXpresso SDK demo application with the Arm GCC toolchain on Windows OS, as supported by the MCUXpresso SDK.

**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*.

**Note:** See Appendix B for Windows OS before compiling the application.

**Parent topic:**Set up toolchain

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

Reference the installation folder of the GNU Arm GCC Embedded tools for the exact path name.

**Parent topic:**Set up toolchain

**Parent topic:**Windows OS host

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

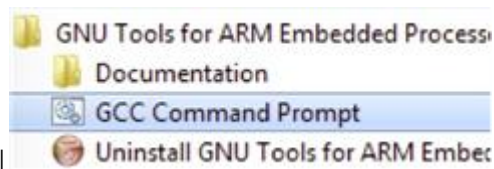
1. Change the directory to the example application project directory, which has a path similar to the following:

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

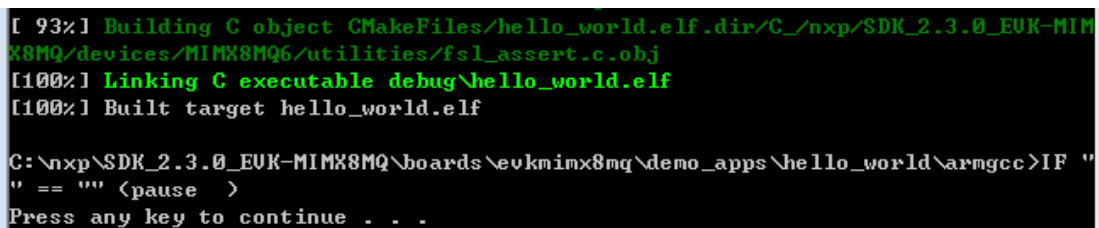
For this example, the exact path is: `<install_dir>/boards/evkmimx8mq/demo_apps/hello_world/armgcc`

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

2. 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”.



3. Type “build\_debug.bat” on the command line or double click on the “build\_debug.bat” file in Windows Explorer to perform the build. The output is shown in this figure:



```
[ 93%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.3.0_EUK-MIMX8MQ/devices/MIMX8MQ6/utilities/fsl_assert.c.obj
[100%] Linking C executable debug\hello_world.elf
[100%] Built target hello_world.elf

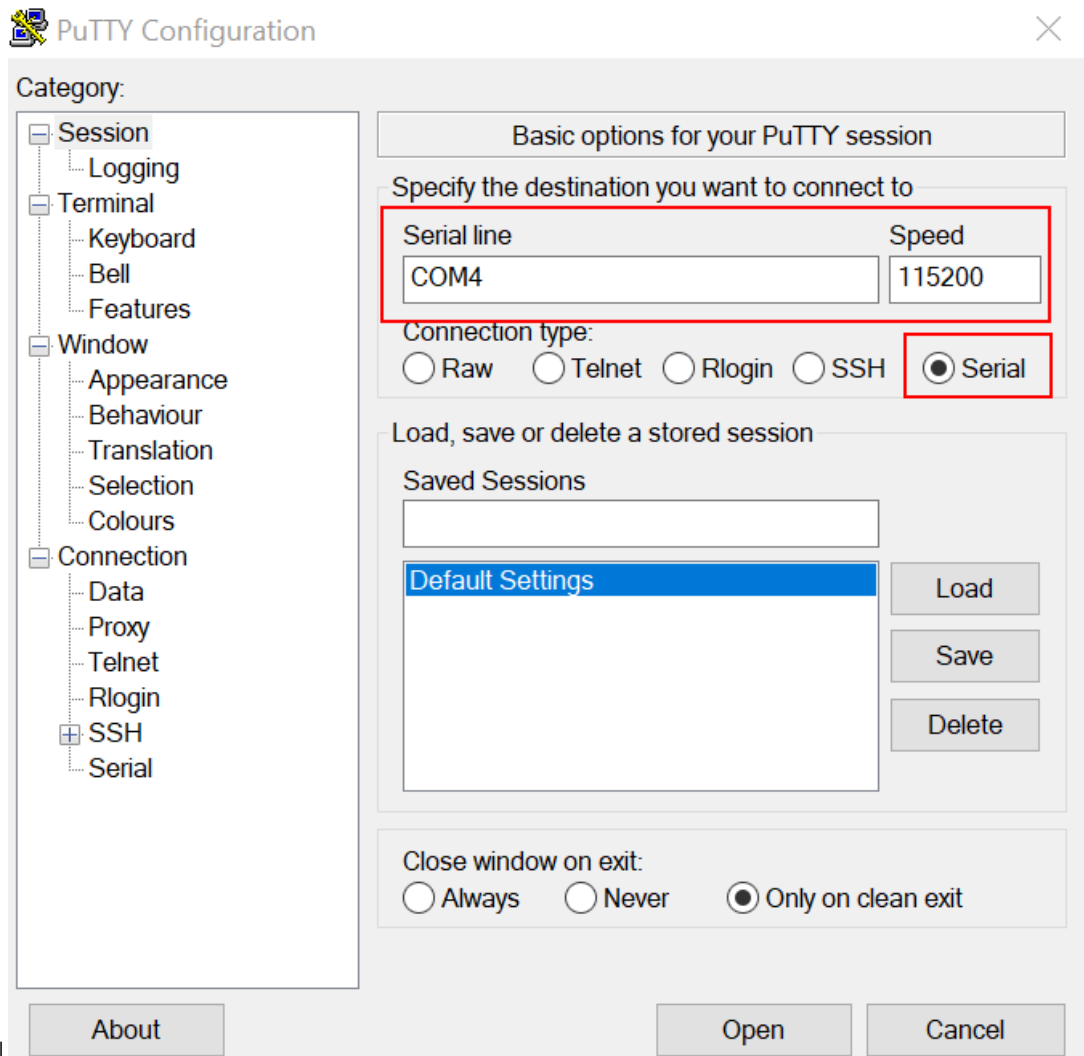
C:/nxp/SDK_2.3.0_EUK-MIMX8MQ/boards/evkmimx8mq/demo_apps/hello_world/armgcc>IF " " == "" (pause )
Press any key to continue . . .
```

**Parent topic:**Windows OS host

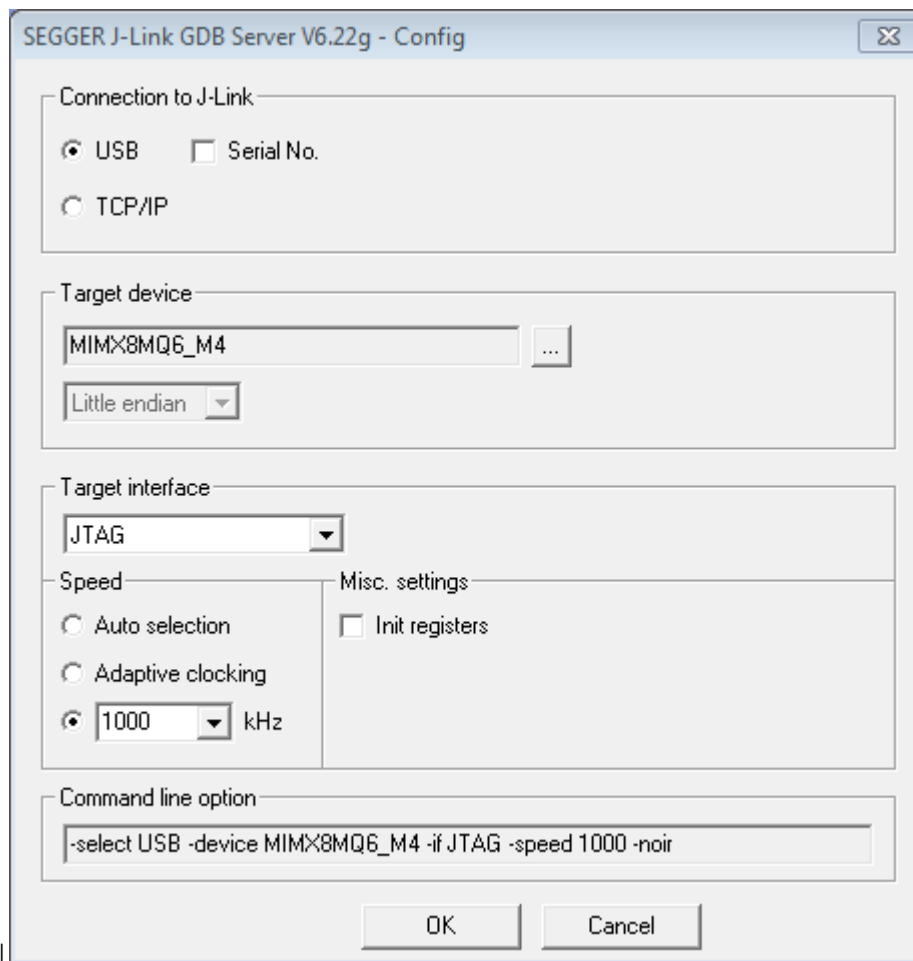
**Run an example application** This section describes steps to run a demo application using J-Link GDB Server application.

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 USB-UART connector and the PC USB connector. If using a standalone J-Link debug pod, also 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 Appendix A). Configure the terminal with these settings:
  1. 115200 baud rate
  2. No parity
  3. 8 data bits
  4. 1 stop bit

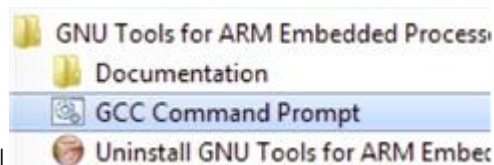


3. Open the J-Link GDB Server application. Assuming the J-Link software is installed, the application can be launched by going to the Windows operating system Start menu and selecting "Programs -> SEGGER -> J-Link <version> J-Link GDB Server".
4. Modify the settings as shown below. The target device selection chosen for this example is the MIMX8MQ6\_M4.
5. After it is connected, the screen should resemble 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".



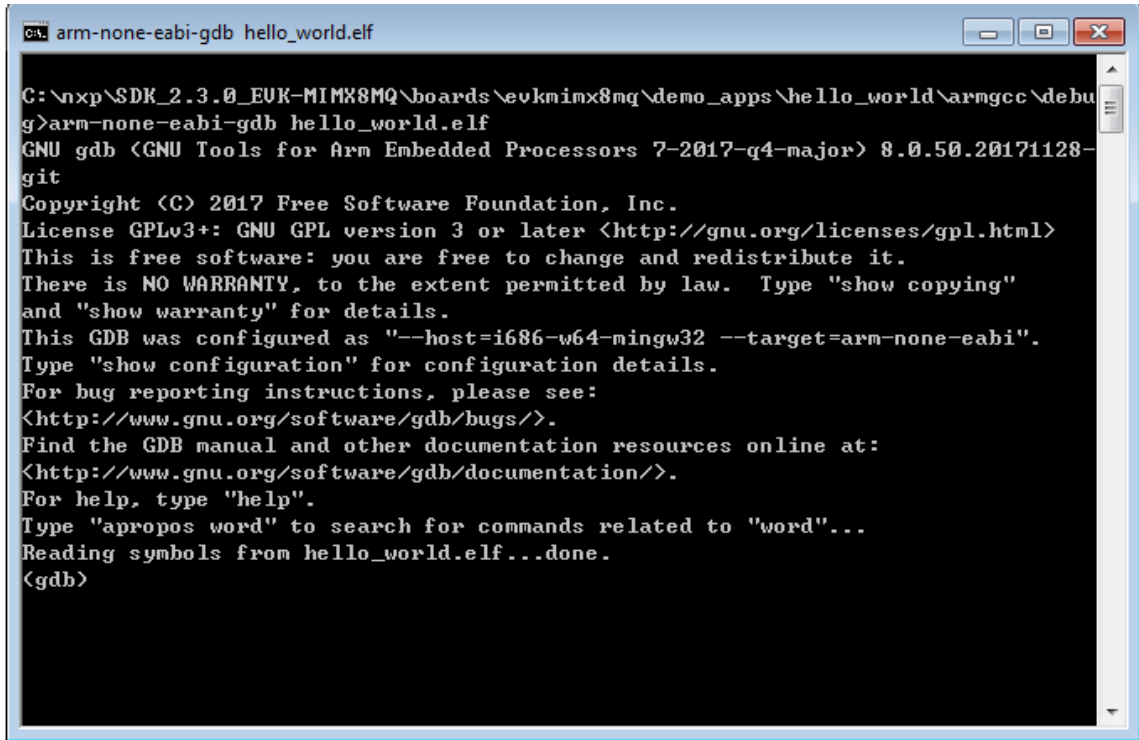
7. Change to the directory that contains the example application output. The output can be found in using one of these paths, depending on the build target selected:

```
<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/evkmimx8mq/demo_apps/hello_world/armgcc/debug
```

8. Run the command "arm-none-eabi-gdb.exe <application\_name>.elf". For this example, it is "arm-none-eabi-gdb.exe hello\_world.elf".



```

C:\nxp\SDK_2.3.0_EUK-MIMX8MQ\boards\evkmimx8mq\demo_apps\hello_world\armgcc\debug
g>arm-none-eabi-gdb hello_world.elf
GNU gdb (GNU Tools for Arm Embedded Processors 7-2017-q4-major) 8.0.50.20171128-
git
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hello_world.elf...done.
(gdb)

```

9. Run these commands:

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

10. The application is now downloaded and halted at the reset vector. 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 is not true, check your terminal settings and connections.



```

COM4 - PuTTY
hello world.

```

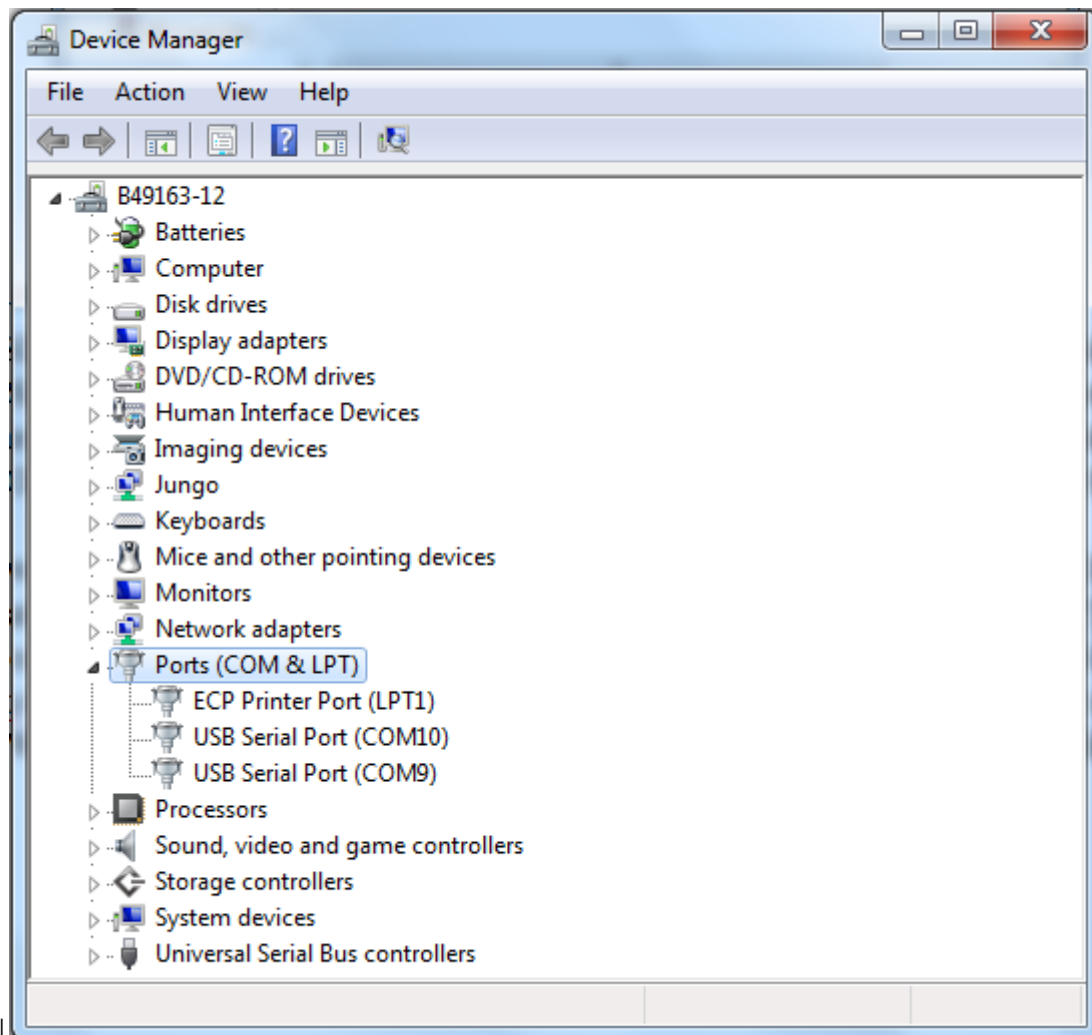
Parent topic: Windows OS host

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

## Running an application by U-Boot

This section describes the steps to write a bootable SDK bin file to TCM or DRAM with the prebuilt U-Boot image for the i.MX processor. The following steps describe how to use the U-Boot:

1. Connect the **DEBUG UART** slot on the board to your PC through the USB cable. The Windows OS installs the USB driver automatically, and the Ubuntu OS finds the serial devices as well.
2. On Windows OS, open the device manager, find **USB serial Port** in **Ports (COM and LPT)**. Assume that the ports are COM9 and COM10. One port is for the debug message from the Cortex-A53 and the other is for the Cortex-M7. The port number is allocated randomly, so opening both is beneficial for development. On Ubuntu OS, find the TTY device with name `/dev/ttyUSB*` to determine your debug port. Similar to Windows OS, opening both is beneficial for development.



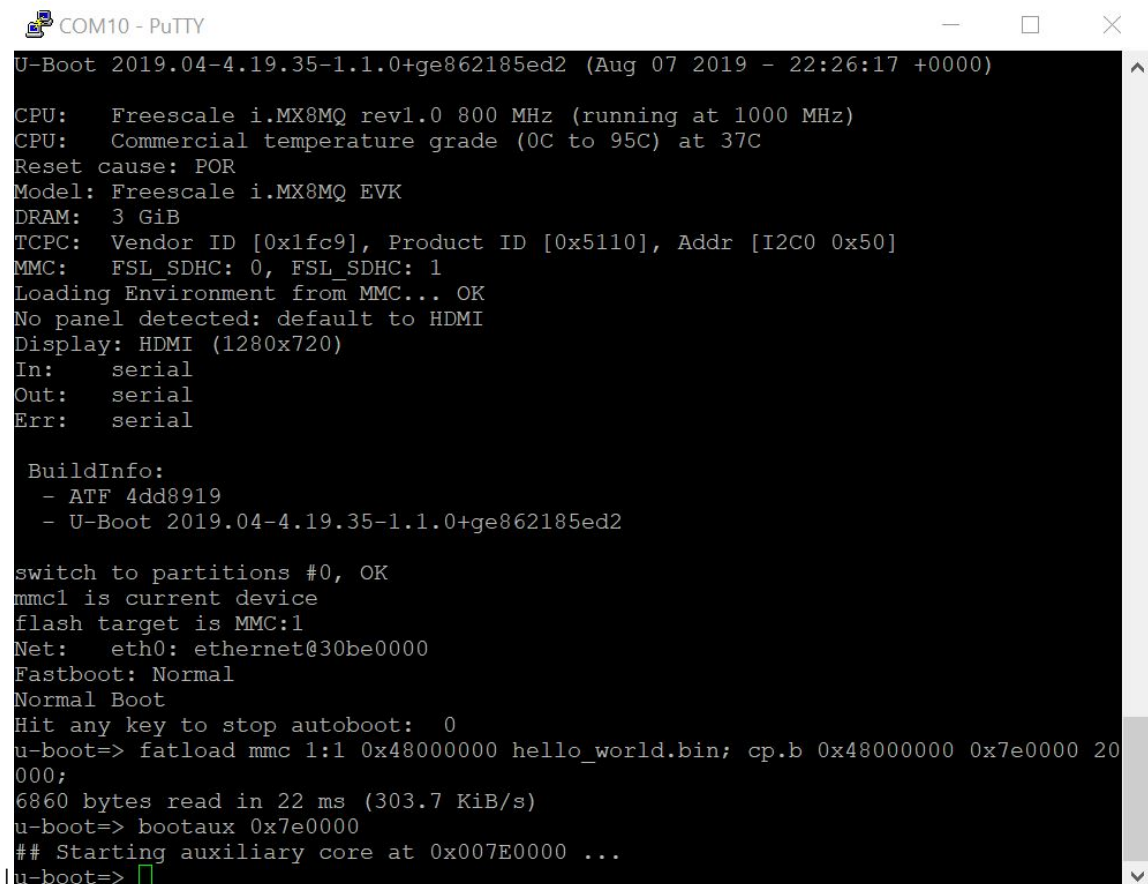
3. Build the application (for example, `hello_world`) to get the bin file (`hello_world.bin`).
4. Prepare an SD card with the prebuilt U-Boot image and copy bin file (`hello_world.bin`) into the SD card. Then, insert the SD card to the target board. Make sure to use the default boot SD slot and check the dipswitch configuration.
5. Open your preferred serial terminals for the serial devices, setting the speed to 115200 bps, 8 data bits, 1 stop bit (115200, 8N1), no parity, then power on the board.
6. Power on the board and hit any key to stop autoboot in the terminals, then enter to U-Boot command line mode. You can then write the image and run it from TCM or DRAM with the

following commands:

1. If the `hello_world.bin` is made from the debug/release target, which means the binary file will run at TCM, use the following commands to boot:
  - `fatload mmc 1:1 0x48000000 hello_world.bin`
  - `cp.b 0x48000000 0x7e0000 0x20000`
  - `bootaux 0x7e0000`
2. If the `hello_world.bin` is made from the `ddr_debug/ddr_release` target, which means the binary file runs at DRAM, use the following commands:
  - `fatload mmc 1:1 0x80000000 hello_world.bin`
  - `dcache flush`
  - `bootaux 0x80000000` **Note:** For m4 examples under the ddr target with Core A kernel boot, change the Linux dtb file specifically in U-Boot before the kernel starts. Use the following command:

```
setenv fdtfile fsl-imx8mq-evk-m4.dtb
save
```

**Note:** For Linux release version L5.15.71-2.2.0 and later, the `run prepare_mcore` command must run before the `bootaux` command.



```
COM10 - PuTTY
U-Boot 2019.04-4.19.35-1.1.0+ge862185ed2 (Aug 07 2019 - 22:26:17 +0000)

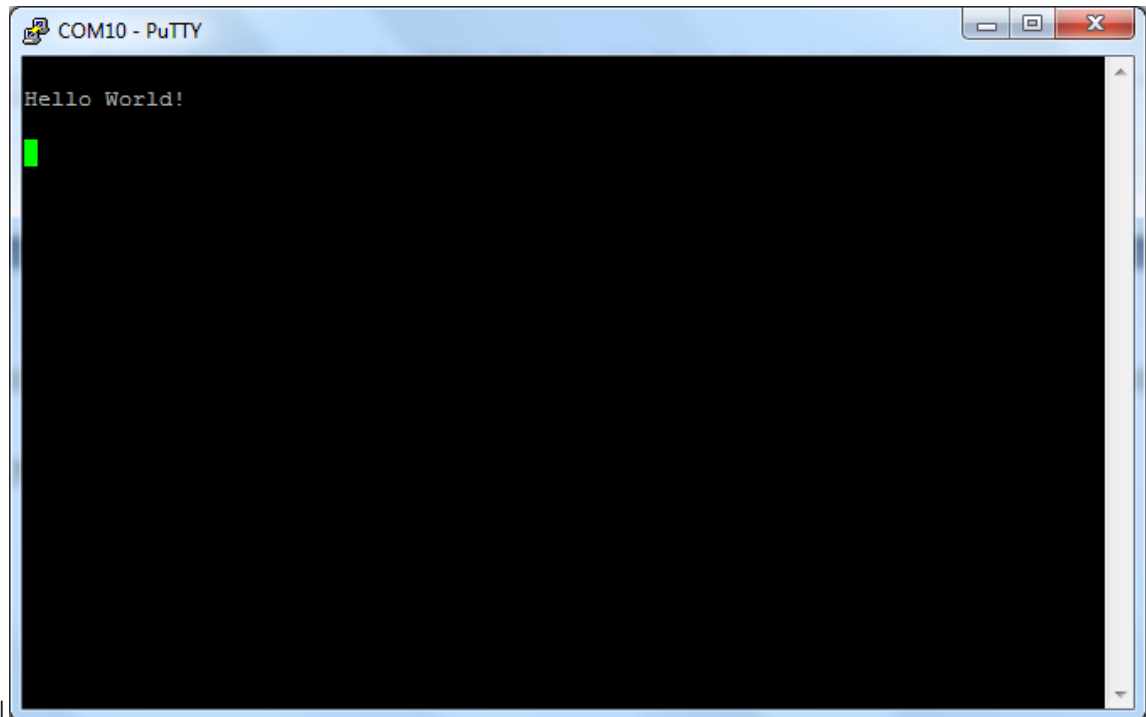
CPU:   Freescale i.MX8MQ rev1.0 800 MHz (running at 1000 MHz)
CPU:   Commercial temperature grade (0C to 95C) at 37C
Reset cause: POR
Model: Freescale i.MX8MQ EVK
DRAM:  3 GiB
TCPC:  Vendor ID [0x1fc9], Product ID [0x5110], Addr [I2C0 0x50]
MMC:    FSL SDHC: 0, FSL SDHC: 1
Loading Environment from MMC... OK
No panel detected: default to HDMI
Display: HDMI (1280x720)
In:     serial
Out:    serial
Err:    serial

BuildInfo:
- ATF 4dd8919
- U-Boot 2019.04-4.19.35-1.1.0+ge862185ed2

switch to partitions #0, OK
mmc1 is current device
flash target is MMC:1
Net:    eth0: ethernet@30be0000
Fastboot: Normal
Normal Boot
Hit any key to stop autoboot:  0
u-boot=> fatload mmc 1:1 0x48000000 hello_world.bin; cp.b 0x48000000 0x7e0000 20000;
6860 bytes read in 22 ms (303.7 KiB/s)
u-boot=> bootaux 0x7e0000
## Starting auxiliary core at 0x007E0000 ...
u-boot=>
```



7. Open another terminal application on the PC, such as PuTTY and connect to the debug COM port (to determine the COM port number, see [How to determine COM port](#)). Configure the terminal with these settings:
  - 115200
  - No parity
  - 8 data bits
  - 1 stop bit
8. The `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.

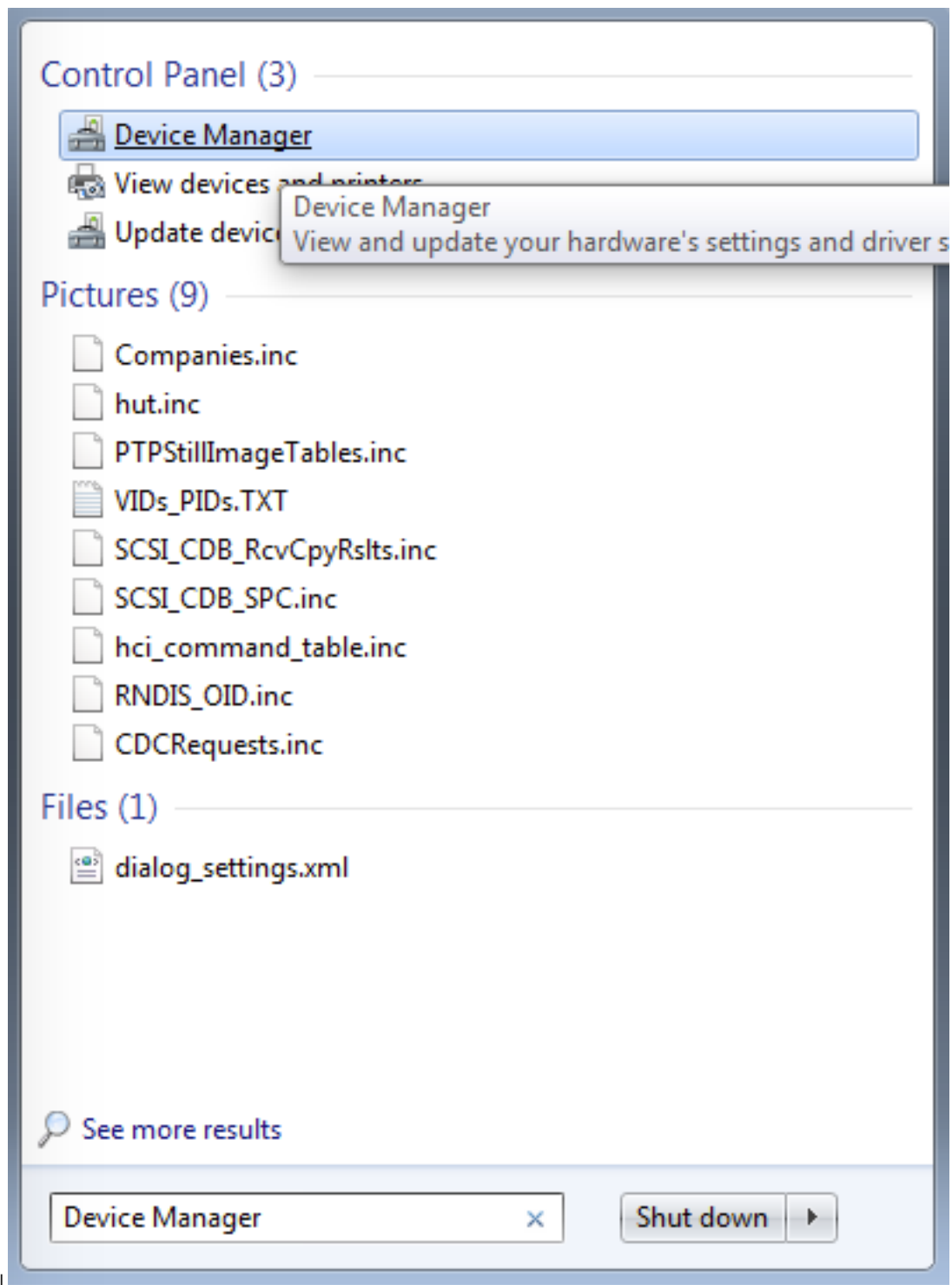


### How to determine COM port

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

1. To determine the COM port, open the Windows operating system Device Manager. This can be achieved by going to the Windows operating system Start menu and typing **Device Manager** in the search bar, as shown in *Figure 1*.





2. In the **Device Manager**, expand the **Ports (COM & LPT)** section to view the available ports. Depending on the NXP board you're using, the COM port can be named differently.

1. **USB-UART interface**



|

## 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
}
```

## Host setup

An MCUXpresso SDK build requires that some packages are installed on the Host. Depending on the used Host operating system, the following tools should be installed.

### Linux:

- Cmake

```
$ sudo apt-get install cmake
$ # Check the version >= 3.0.x
$ cmake --version
```

### Windows:

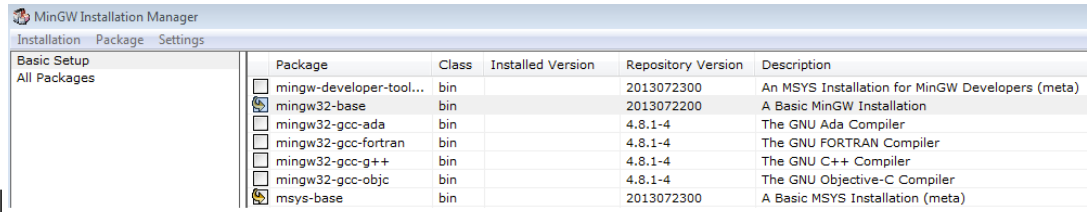
- MinGW

The Minimalist GNU for Windows OS (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 SDK does not utilize 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 [sourceforge.net/projects/mingw/files/Installer/](https://sourceforge.net/projects/mingw/files/Installer/).
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 **mingw32-base** and **msys-base** are selected under **Basic Setup**.



4. Click **Apply Changes** in the **Installation** menu 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`.

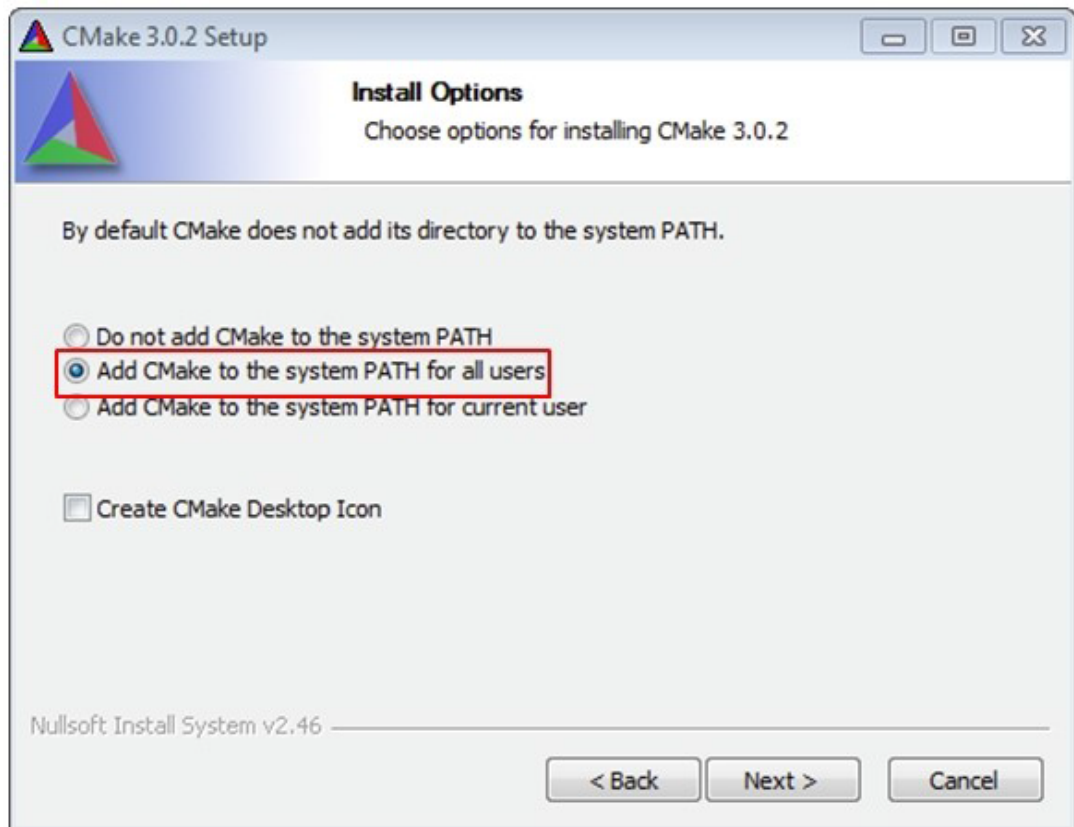
Assuming the default installation path, `C:\MinGW`, an example is as shown in [Figure 3](host\_setup.md #ADDINGPATH). If the path is not set correctly, the toolchain does not work.

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



- 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.

## 1.3 Getting Started with MCUXpresso SDK GitHub

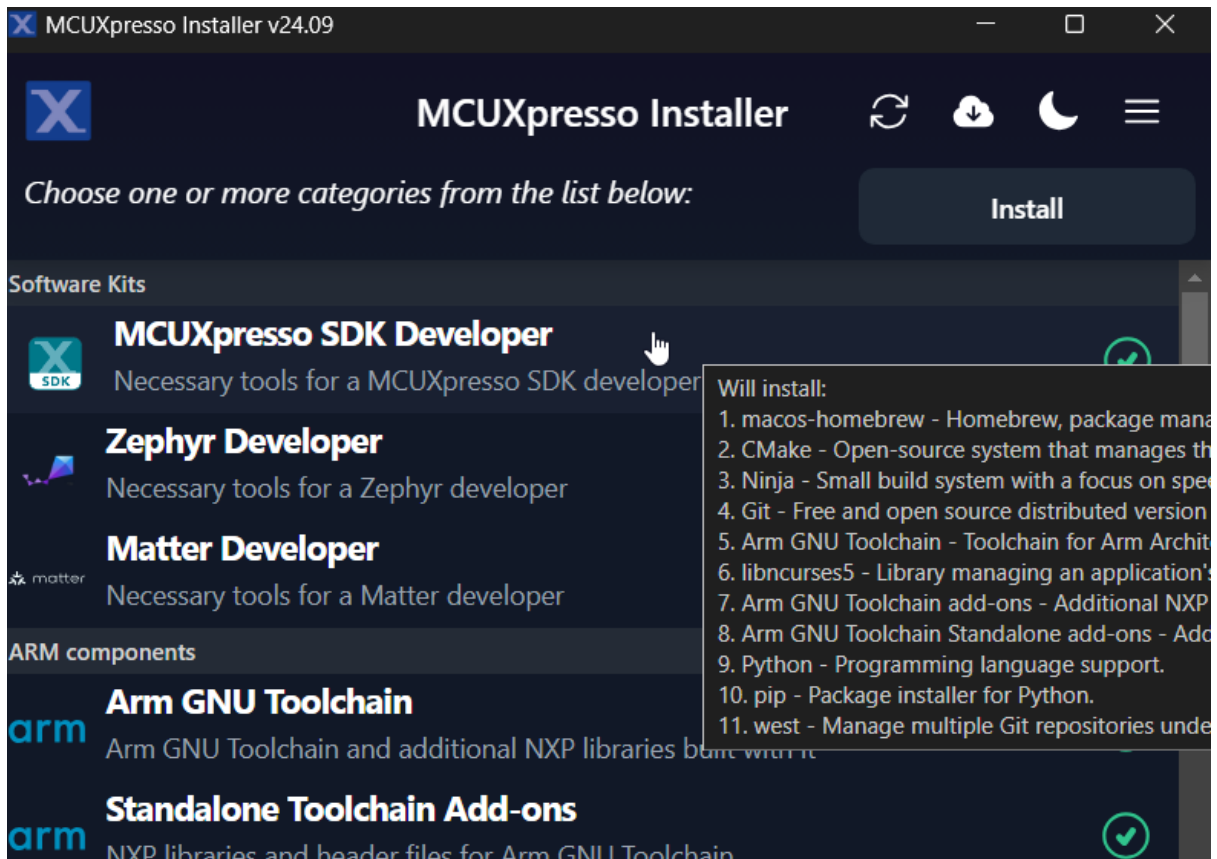
### 1.3.1 Getting Started with MCUXpresso SDK Repository

#### Installation

##### NOTE

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

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



## Alternative: Manual Installation

### Basic tools

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

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

Then configure your username and email using the commands:

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

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

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

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

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

## Build And Configuration System

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

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

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

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

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

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

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

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

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

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

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

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

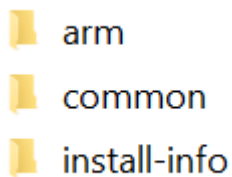
**Toolchain** MCUXpresso SDK supports all mainstream toolchains for embedded development. You can install your used or interested toolchains following the guides.

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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



## Get MCUXpresso SDK Repo

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

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

# Update the west projects
cd mcuxpresso-sdk
west update

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

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

1. Navigate to the core directory:

```
cd mcuxsdk
```

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

```
python -m venv .venv

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

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

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

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

3. Install the required Python packages:

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

## Explore Contents

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

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

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

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

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

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

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

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

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

## Run a demo using MCUXpresso for VS Code

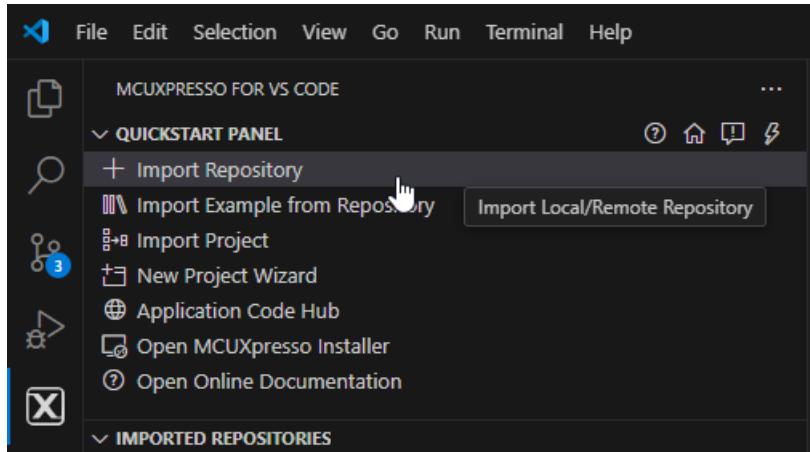
This section explains how to configure MCUXpresso for VS Code to build, run, and debug example applications. This guide uses the `hello_world` demo application as an example. However, these

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

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

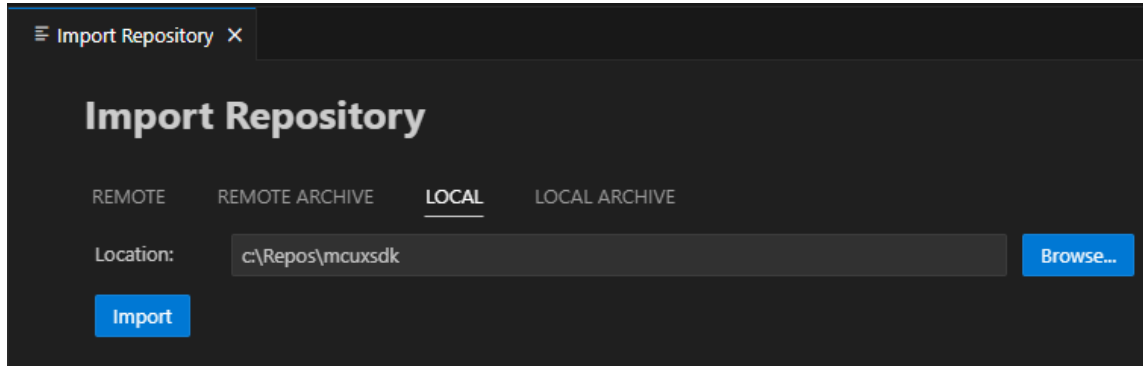
To build an example application:

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

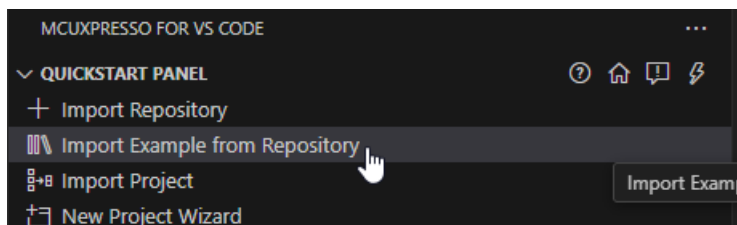


**Note:** You can import the SDK in several ways. Refer to [MCUXpresso for VS Code Wiki](#) for details.

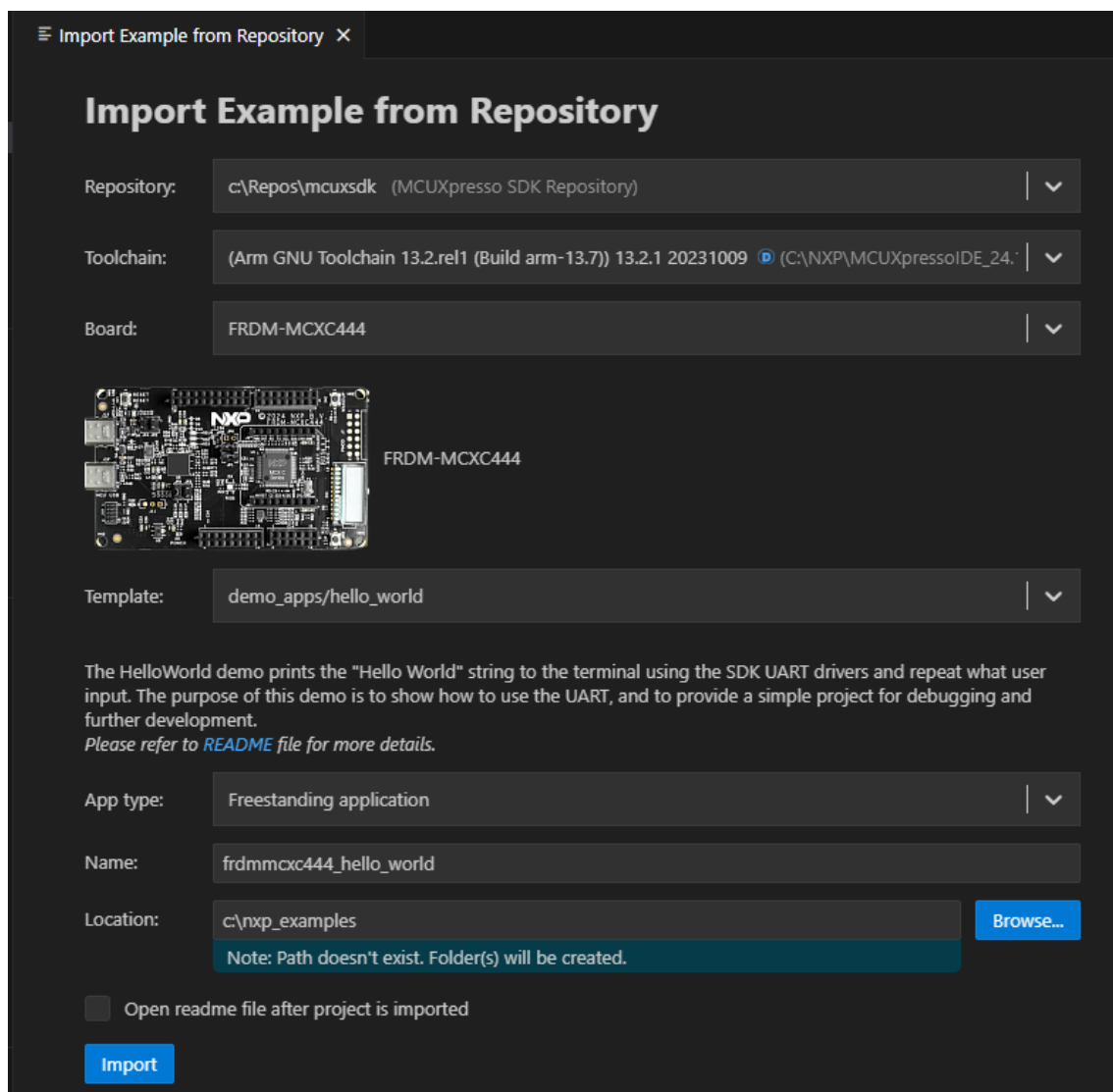
Select **Local** if you've already obtained the SDK as seen in [Get MCUXpresso SDK Repo](#). Select your location and click **Import**.



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



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



**Import Example from Repository**

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

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

Board: FRDM-MCXC444 | v

 FRDM-MCXC444

Template: demo\_apps/hello\_world | v

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

App type: Freestanding application | v

Name: frdmxc444\_hello\_world

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

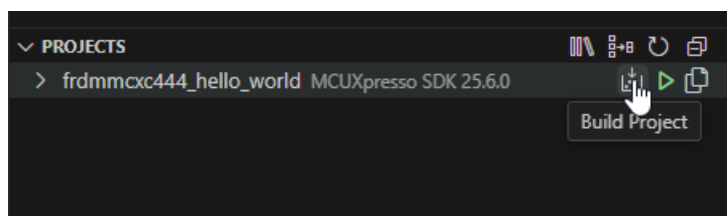
Note: Path doesn't exist. Folder(s) will be created.

☐ Open readme file after project is imported

[Import](#)

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

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



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

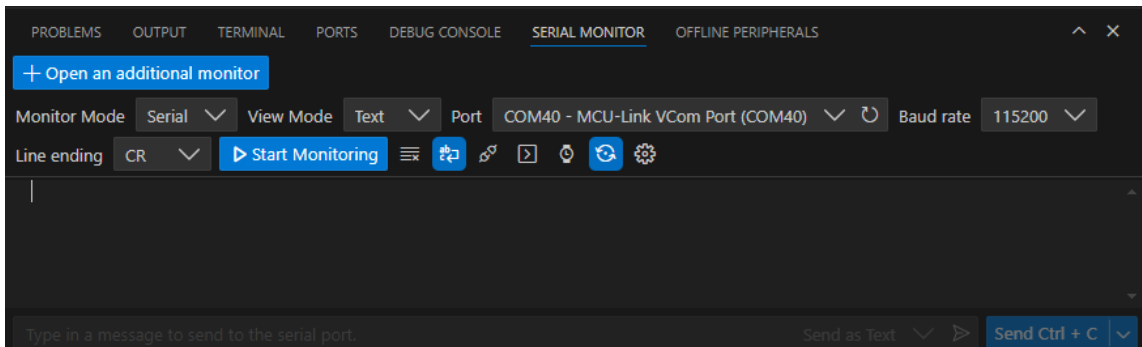
```

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

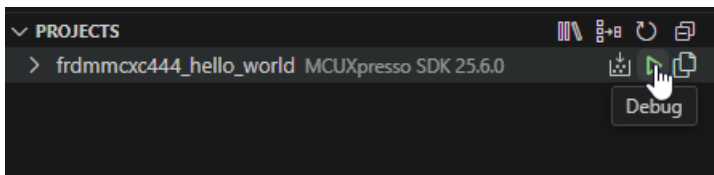
```

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

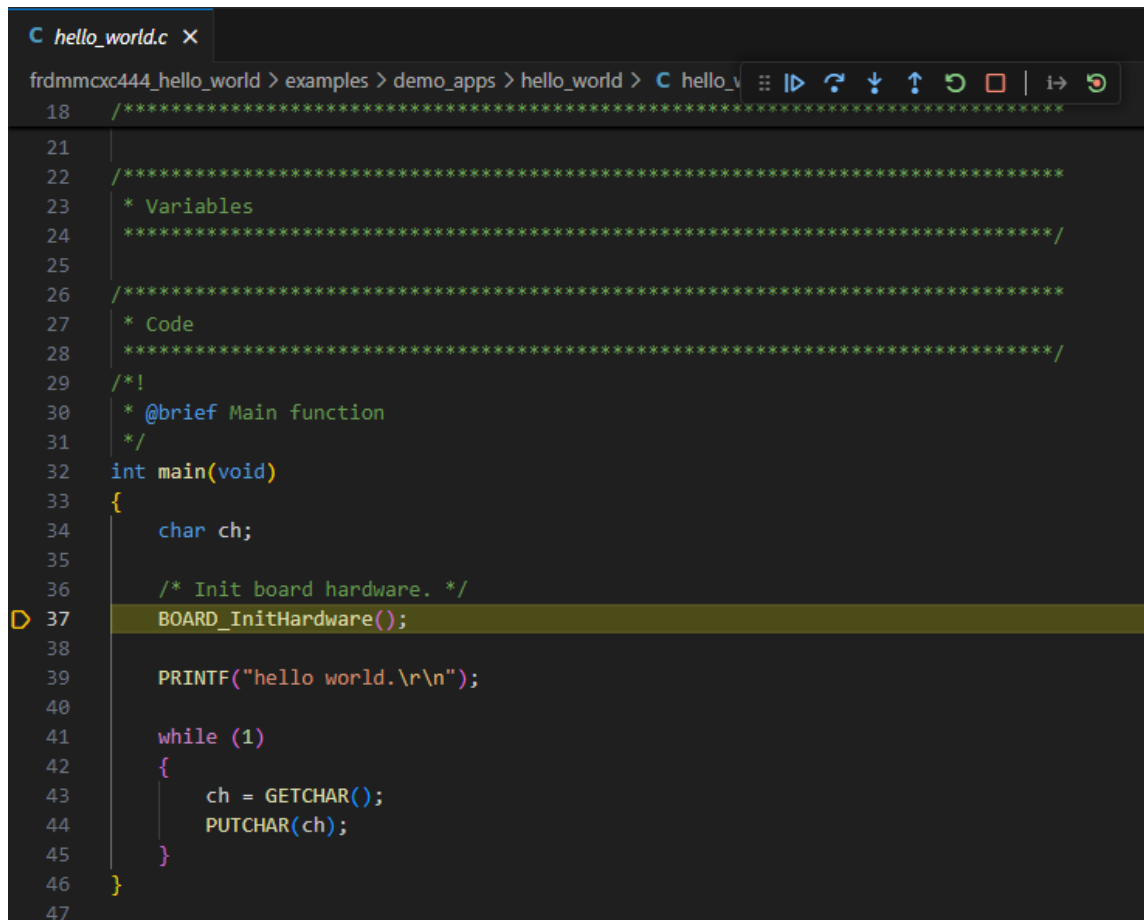
1. Open the **Serial Monitor** from the VS Code's integrated terminal. Select the VCom Port for your device and set the baud rate to 115200.



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



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

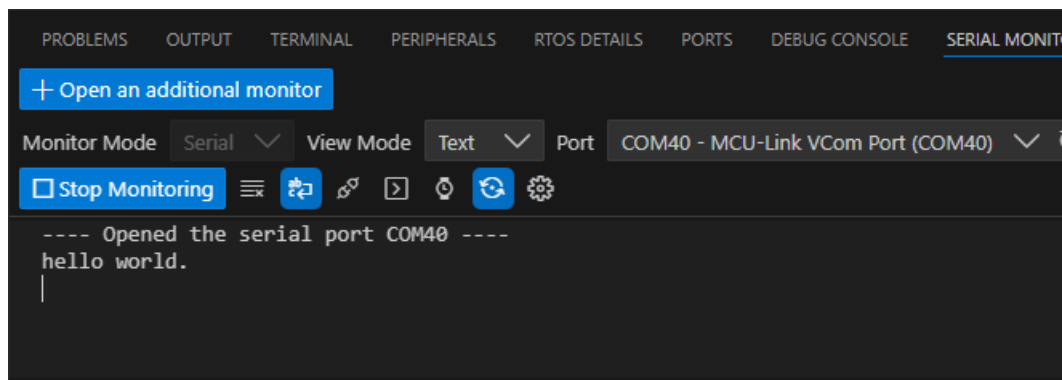


```

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

```

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



```

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

```

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```

↪ evkbbmimxrt1060]
INFO: [ 4][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbbmimxrt1170 -Dcore_id=cm4]
INFO: [ 5][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbbmimxrt1170 -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 evkbbmimxrt1170 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 evkbbmimxrt1170 --sysbuild ./examples/multicore_examples/hello_world/primary -Dcore_
↪ id=cm7 --config flexspi_nor_debug --toolchain=armgcc -p always

```

For more details, please refer to System build.

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

1. Run cmake configuration

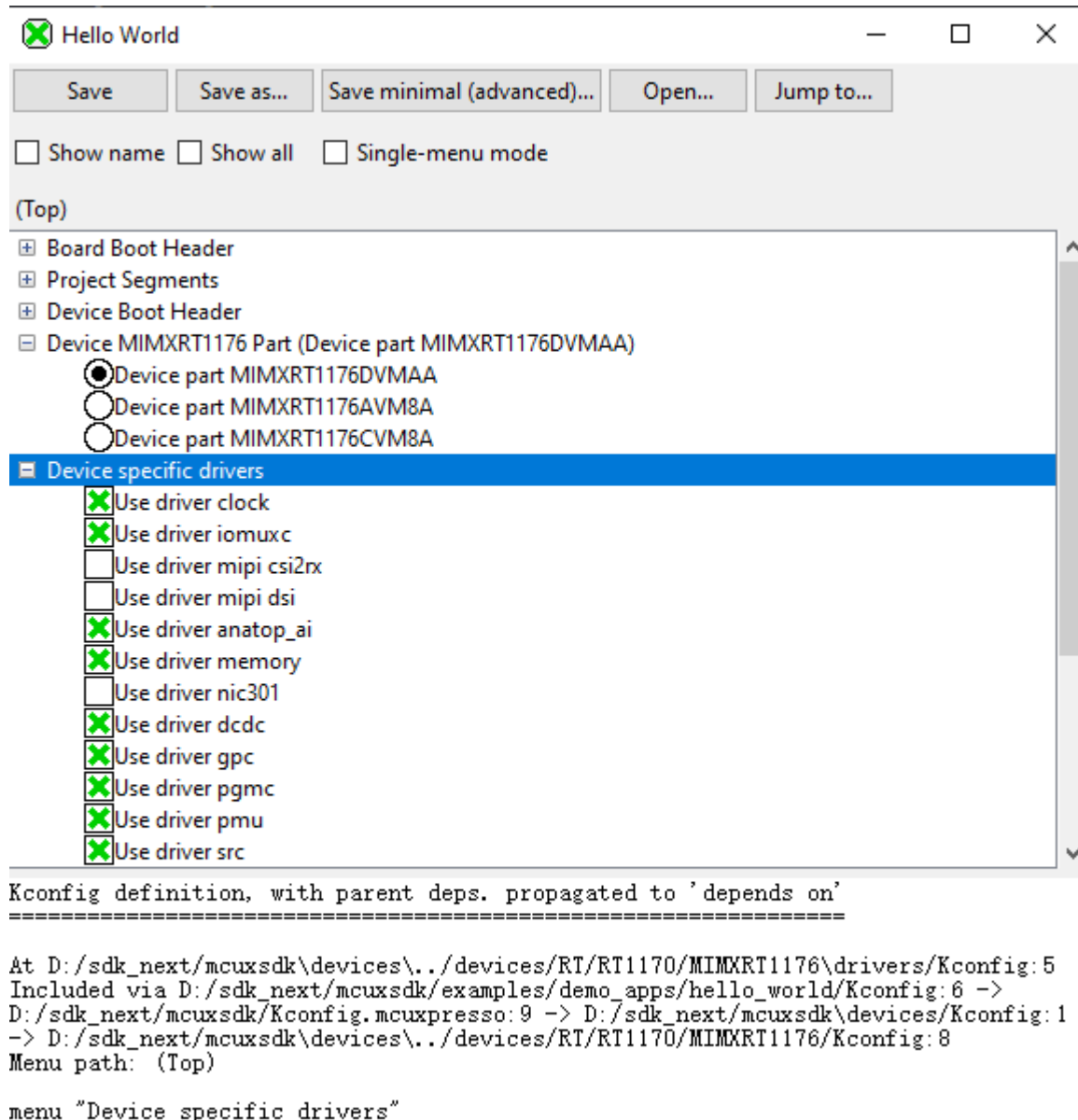
```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

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

2. Run guiconfig target

```
west build -t guiconfig
```

Then you will get the Kconfig GUI launched, like



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

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



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

Flash the hello\_world example:

```
west flash -r linkserver
```

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

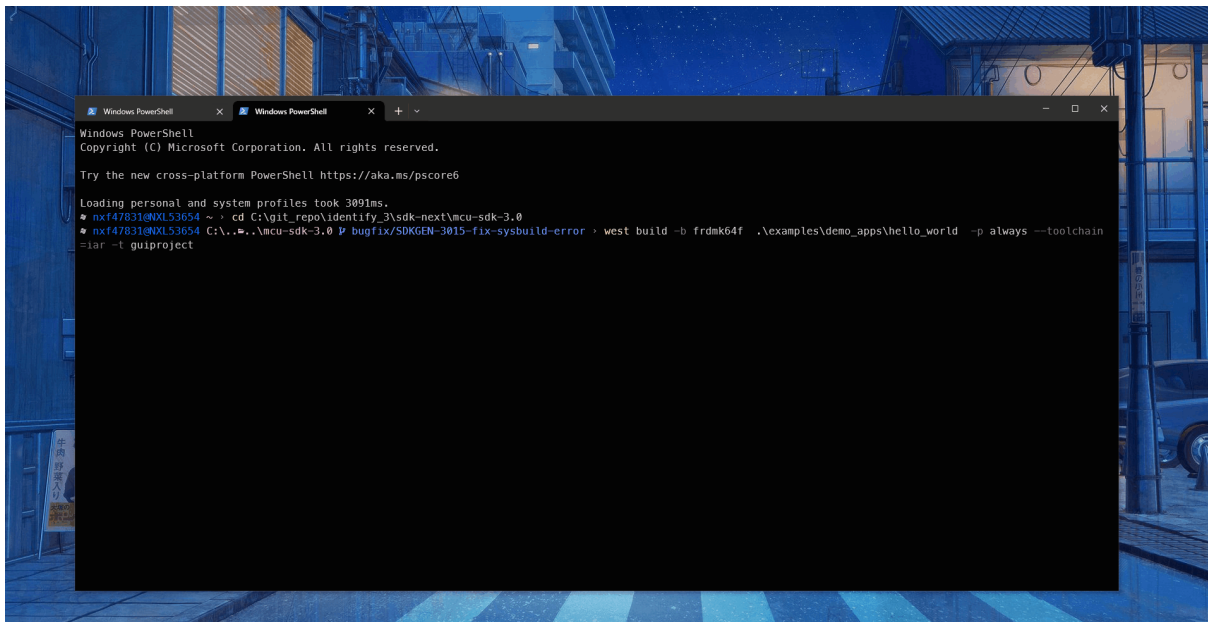
```
west debug -r linkserver
```

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

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

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

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



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

## 1.4 Release Notes

### 1.4.1 MCUXpresso SDK Release Notes

#### Overview

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

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

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

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

## MCUXpresso SDK

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

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

## Development tools

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

- IAR Embedded Workbench for Arm, version is 9.60.4
- MCUXpresso for VS Code v25.09
- GCC Arm Embedded Toolchain 14.2.x

## Supported development systems

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

| Development boards | MCU devices                                                                                                                                                  |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>EVK-MIMX8M</b>  | MIMX8MD6CVAHZ, MIMX8MD6DVAJZ, MIMX8MD7CVAHZ, MIMX8MD7DVAJZ, MIMX8MQ5CVAHZ, MIMX8MQ5DVAJZ, MIMX8MQ6CVAHZ, <b>MIMX8MQ6DVAJZ</b> , MIMX8MQ7CVAHZ, MIMX8MQ7DVAJZ |

## MCUXpresso SDK release package

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

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

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

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

## RTOS

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

## Middleware

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

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

**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

**llhttp** HTTP parser llhttp

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

## Release contents

Provides an overview of the MCUXpresso SDK release package contents and locations.

| Deliverable                                                                          | Location                                           |
|--------------------------------------------------------------------------------------|----------------------------------------------------|
| Boards                                                                               | INSTALL_DIR/boards                                 |
| Demo Applications                                                                    | INSTALL_DIR/boards/<board_name>/demo_apps          |
| Driver Examples                                                                      | INSTALL_DIR/boards/<board_name>/driver_examples    |
| eIQ examples                                                                         | INSTALL_DIR/boards/<board_name>/eiq_examples       |
| Board Project Template for MCUXpresso IDE NPW                                        | INSTALL_DIR/boards/<board_name>/project_template   |
| Driver, SoC header files, extension header files and feature header files, utilities | INSTALL_DIR/devices/<device_name>                  |
| CMSIS drivers                                                                        | INSTALL_DIR/devices/<device_name>/cmsis_drivers    |
| Peripheral drivers                                                                   | INSTALL_DIR/devices/<device_name>/drivers          |
| Toolchain linker files and startup code                                              | INSTALL_DIR/devices/<device_name>/<toolchain_name> |
| Utilities such as debug console                                                      | INSTALL_DIR/devices/<device_name>/utilities        |
| Device Project Template for MCUXpresso IDE NPW                                       | INSTALL_DIR/devices/<device_name>/project_template |
| CMSIS Arm Cortex-M header files, DSP library source                                  | INSTALL_DIR/CMSIS                                  |
| Components and board device drivers                                                  | INSTALL_DIR/components                             |
| RTOS                                                                                 | INSTALL_DIR/rtos                                   |
| Release Notes, Getting Started Document and other documents                          | INSTALL_DIR/docs                                   |
| Tools such as shared cmake files                                                     | INSTALL_DIR/tools                                  |
| Middleware                                                                           | INSTALL_DIR/middleware                             |

## Known issues

This section lists the known issues, limitations, and/or workarounds.

### Cannot add SDK components into FreeRTOS projects

It is not possible to add any SDK components into FreeRTOS project using the MCUXpresso IDE New Project wizard.

### The freertos\_lpuart example does not complete successfully

The example hangs after console output ‘FreeRTOS LPUART driver example’.

**Examples:** freertos\_lpuart

**Affected toolchains:** All

The example does not perform as expected (Ticks do not printed on the console or the application does not wake up from the sleep mode).

**Examples:** freertos\_tickless

**Affected toolchains:** All

## 1.5 ChangeLog

### 1.5.1 MCUXpresso SDK Changelog

#### Board Support Files

##### board

###### [25.06.00]

- Initial version

##### clock\_config

###### [25.06.00]

- Initial version

##### pin\_mux

###### [25.06.00]

- Initial version
- 

#### CACHE LMEM

###### [2.1.0]

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

###### [2.1.0]

- Improvements
  - Added new feature macro to support some device do not support PCCCR[ENWRBUF] bit field.

###### [2.0.6]

- Bug Fixes
  - Fixed doxygen issue.

#### [2.0.5]

- Improvements
  - Updated the cache enable function, don't enable again when it is already enabled.

#### [2.0.4]

- Bug Fixes
  - Updated full name for lmem driver.
  - Fixed doxygen issue.

#### [2.0.3]

- Bug Fixes
  - Fixed violation of MISRA C-2012 Rule 10.4 and 14.4.

#### [2.0.2]

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

#### [2.0.1]

- Bug Fixes
  - Fixed the over-4KB-size maintenance issue in invalidate/clean/clean&invalidate by range APIs.

#### [2.0.0]

- Initial version.
- 

### COMMON

#### [2.6.1]

- Improvements
  - Support Cortex M23.

#### [2.6.0]

- Bug Fixes
  - Fix CERT-C violations.

#### [2.5.0]

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

**[2.4.3]**

- Improvements
  - Enable irq 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.
- 

## ECSPI

### [2.3.3]

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

### [2.3.2]

- Improvements
  - Changed ECSPI\_DUMMYDATA to 0x00.

### [2.3.1]

- Bug Fixes
  - Fixed ECSPI\_GetInstance potential issue that return wrong instance number.

### [2.3.0]

- Bug Fixes
  - Fixed burst length issue,the burst length range shall range from 1-4096 bits, so the width shall be uint8\_t rather than uint16\_t.

### [2.2.0]

- Bug Fixes
  - Removed the useless channel configuration of waveform, since the waveform can not be configured when not using the exchange bit(ECSPIx\_CONREG[XCH]) for the transfer.
  - Fixed violations of MISRA C-2012 rules: 10.1, 11.9, 8.4.

### [2.1.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rules: 10.1, 10.3, 10.4, 11.9, 14.4, 15.7, 17.7.

**[2.1.0]**

- Improvements
  - Added timeout mechanism when waiting certain states in transfer driver.

**[2.0.2]**

- Bug Fixes
  - Fixed violations of MISRA C-2012 rules: 10.1, 10.3, 10.4

**[2.0.1]**

- Bug Fixes
  - Memset local variable SDMA transfer configuration structure to make sure unused members in structure are cleared.
  - Fixed sign-compare warning in ECSPI\_SendTransfer.

**[2.0.0]**

- Initial version.
- 

**GPIO****[2.0.7]**

- Bug Fixes
  - Fixed coverity MSG issues with CERT INT30-C compliance.

**[2.0.6]**

- Bug Fixes
  - Fixed compile warning: 'GPIO\_GetInstance' defined but not used when macro FSL\_SDK\_DISABLE\_DRIVER\_CLOCK\_CONTROL is defined.

**[2.0.5]**

- Bug Fixes
  - Fixed MISRA C-2012 issue: rule-17.7.

**[2.0.4]**

- Improvements
  - Updated the GPIO\_PinWrite to use atomic operation if possible.
- Bug Fixes
  - Fixed GPIO\_PortToggle bug with platforms don't have register DR\_TOGGLE.

### [2.0.3]

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

### [2.0.2]

- Bug Fixes
  - Fixed the bug of enabling wrong GPIO clock gate in initial API. Since some GPIO instances may not have a clock gate enabled, it checks the clock gate number and makes sure the clock gate is valid.

### [2.0.1]

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

### [2.0.0]

- Initial version.
- 

## GPT

### [2.0.6]

- Bug Fixes
  - Fix CERT INT30-C issues.

### [2.0.5]

- Improvements
  - Support workaround for ERR003777. This workaround helps switching the clock sources.

### [2.0.4]

- Bug Fixes
  - Fixed compiler warning when built with `FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL` flag enabled.

### [2.0.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 5.3 by customizing function parameter.

**[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 10.1, 10.3, 10.4, 10.6, 10.8, 17.7.

**[2.0.0]**

- Initial version.
- 

**I2C****[2.0.7]**

- Bug Fixes
  - Fixed MISRA issues.
    - \* Fixed rules 8.4, 8.5.

**[2.0.6]**

- Bug Fixes
  - Fixed the bug that, in I2C\_MasterStop after the stop command is issued, the IBB flag should be cleared rather than set.
  - Fixed the bug that to clear kI2C\_ArbitrationLostFlag and kI2C\_IntPendingFlag, their bits should be written '0' rather than '1'.

**[2.0.5]**

- Bug Fixes
  - Fixed Coverity issue of unchecked return value in I2C\_RTOS\_Transfer.
  - Fixed MISRA issues.
    - \* Fixed rules 10.1, 10.3, 10.4, 11.9, 14.4, 15.7, 16.4, 17.7.
- Improvements
  - Updated the I2C\_WAIT\_TIMEOUT macro to unified name I2C\_RETRY\_TIMES.

**[2.0.4]**

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

### [2.0.3]

- Improvements
  - Improved code readability, added new static API `I2C_WaitForStatusReady` for the status flag wait, and changed to call `I2C_WaitForStatusReady` instead of polling flags with reading register.

### [2.0.2]

- Improvements
  - Added `I2C_WATI_TIMEOUT` macro to allow users to specify the timeout times for waiting flags in functional API and blocking transfer API.

### [2.0.1]

- Bug Fixes
  - Added a proper handle for transfer config flag `kI2C_TransferNoStartFlag` to support transmit with `kI2C_TransferNoStartFlag` flag. Only supports write only or write+read with no start flag; does not support read only with no start flag.

### [2.0.0]

- Initial version.
- 

## MCM

### [2.2.0]

- Improvements
  - Support platforms with less features.

### [2.1.0]

- Others
  - Remove `byteID` from `mcm_lmem_fault_attribute_t` for document update.

### [2.0.0]

- Initial version.
- 

## MU

### [2.3.1]

- Bug Fixes
  - Fixed `FSL_FEATURE_MU_HAS_RESET_DEASSERT_INT` macro use.

### [2.3.0]

- New Features
  - Added MU\_BUSY\_POLL\_COUNT parameter to prevent infinite polling loops in MU operations.
  - Added timeout mechanism to all polling loops in MU driver code.
  - Added new function MU\_ReceiveMsgTimeout() to include timeout mechanism.
- Improvements
  - Updated function signatures to return status codes for better error handling:
    - \* Changed MU\_ResetBothSides to return status\_t instead of void
    - \* Updated MU\_SendMsg to return status\_t for timeout indication
    - \* Updated MU\_ReceiveMsg to use MU\_TIMEOUT\_VALUE (0xFFFFFFFF) as a special return value to indicate timeout
  - Enhanced documentation across all functions to clarify timeout behavior and return values.

### [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.
- 

## PWM

#### [2.0.1]

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

#### [2.0.0]

- Initial version.
-



## QSPI

### [2.3.1]

- Improvements
  - Fixed Coverity MSG issues.

### [2.3.0]

- New Features
  - Applied the QSPI IP update with register field changes.
  - Added Soc specific driver to integrate Soc configuration.
- Changed
  - Updated the QSPI LUT update function to be compatible with different sequence unit.
  - Added new feature macro `FSL_FEATURE_QSPI_HAS_SOC_SPECIFIC_CONFIG` which represents there're Soc specific QSPI configurations. Soc specific driver should cover these configurations. Previous Soc specific code in the common driver should be masked.

### [2.2.5]

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

### [2.2.4]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.3.

### [2.2.3]

- Bug Fixes
  - Cleared buffer generic configuration when do software reset.

### [2.2.2]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.1 and 11.9.

### [2.2.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.3, 10.4, 10.6, 10.8, 11.3, 11.6, 11.8, 11.9, 14.4, 16.1, 16.4, 17.7.

### [2.2.0]

- New Features
  - Added new API `QSPI_ClearCache` to clear cache for new IP feature `FSL_FEATURE_QSPI_SOCCR_HAS_CLR_LPCAC`.
- Bug Fixes
  - Fixed the `QSPI_WriteBlocking` API programming issue for low watermark, caused by previous improvement change of using TX watermark signal to fill the TX FIFO. Reverted change to previous implementation to use TX FIFO full flag for filling the FIFO. Improved previous API by accessing TX data register directly.
  - Fixed the issue that `QSPI_SetIPCommandSize` incorrectly triggered a transaction.
  - Fixed clock divider accurate issue when using internal QSPI internal divider.
  - Fixed build fail issue for some devices' not supporting API `QSPI_SetDqsConfig` for DQS configuration.

### [2.1.0]

- New Features
  - Added new API `QSPI_SetDqsConfig` for DQS configuration.
- Improvements
  - Updated the `QSPI_WriteBlocking` API to fill the TX FIFO once there are bytes of TX watermark room in the FIFO. This will improve the performance of filling TX FIFO when watermark is high.

### [2.0.2]

- Improvements
  - New Macro function:
    - \* Added `QSPI_LUT_SEQ()` function for users to set LUT table easily.
    - \* Added LUT command macros for users to easy use.
  - Comment update:
    - \* Added the comments for the limitation of `QSPI_ReadBlocking` and `QSPI_TransferReceiveBlocking`.

### [2.0.1]

- Improvements
  - New API:
    - \* `QSPI_SetReadArea` to set the read area.
- Bug Fixes
  - Fixed the issue that `QSPI_UpdateLUT` function only updated first LUT.
  - Fixed issue that some function that hardcode QSPI0 as base.

### [2.0.0]

- Initial version.

## RDC

### [2.2.0]

- New Features
  - Added APIs to get memory region or peripheral access policy for specific domain.

### [2.1.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.6.

### [2.1.0]

- Improvements
  - Enhanced to support memory region larger than 32-bit address.

### [2.0.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 10.4, 11.3, 11.8, 17.7.

### [2.0.1]

- Bug Fixes:
  - Added \_\_DSB after new configuration is set to ensure the new configuration takes effect.

### [2.0.0]

- Initial version.
- 

## RDC\_SEMA42

### [2.0.5]

- Bug Fixes
  - Fixed CERT INT30-C issues.

### [2.0.4]

- Improvements
  - Changed to implement RDC\_SEMAPHORE\_Lock base on RDC\_SEMAPHORE\_TryLock.

### [2.0.3]

- Improvements:
  - Supported the RDC\_SEMAPHORE\_Type structure whose gate registers are defined as an array.

#### [2.0.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 10.4, 10.8, 14.3, 14.4, 18.1.

#### [2.0.1]

- Improvements:
  - Added support for the platforms that don't have dedicated RDC\_SEMA42 clock gate.

#### [2.0.0]

- Initial version.
- 

### SAI

#### [2.4.10]

- Improvements
  - Allow enabling/disabling implicit channel configuration.
  - Allow NULL FIFO watermark.
- Bug Fixes
  - Fix compilation warnings when asserts are disabled

#### [2.4.9]

- Added Errata ERR051421 workaround.

#### [2.4.8]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8, 12.4.

#### [2.4.7]

- Added conditional support for bit clock swap feature
- Added common IRQ handler entry SAI\_DriverIRQHandler.

#### [2.4.6]

- Bug Fixes
  - Fixed the IAR build warning.

#### [2.4.5]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8, 12.4.

**[2.4.4]**

- Bug Fixes
  - Fixed enumeration sai\_fifo\_combine\_t - add RX configuration.

**[2.4.3]**

- Bug Fixes
  - Fixed enumeration sai\_fifo\_combine\_t value configuration issue.

**[2.4.2]**

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

**[2.4.1]**

- Bug Fixes
  - Fixed bitWidth incorrectly assigned issue.

**[2.4.0]**

- Improvements
  - Removed deprecated APIs.

**[2.3.8]**

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

**[2.3.7]**

- Improvements
  - Change feature “FSL\_FEATURE\_SAI\_FIFO\_COUNT” to “FSL\_FEATURE\_SAI\_HAS\_FIFO”.
  - Added feature “FSL\_FEATURE\_SAI\_FIFO\_COUNTn(x)” to align SAI fifo count function with IP in function

**[2.3.6]**

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 5.6.

**[2.3.5]**

- Improvements
  - Make driver to be aarch64 compatible.

#### [2.3.4]

- Bug Fixes
  - Corrected the fifo combine feature macro used in driver.

#### [2.3.3]

- Bug Fixes
  - Added bit clock polarity configuration when sai act as slave.
  - Fixed out of bound access coverity issue.
  - Fixed violations of MISRA C-2012 rule 10.3, 10.4.

#### [2.3.2]

- Bug Fixes
  - Corrected the frame sync configuration when sai act as slave.

#### [2.3.1]

- Bug Fixes
  - Corrected the peripheral name in function SAI0\_DriverIRQHandler.
  - Fixed violations of MISRA C-2012 rule 17.7.

#### [2.3.0]

- Bug Fixes
  - Fixed the build error caused by the SOC has no fifo feature.

#### [2.2.3]

- Bug Fixes
  - Corrected the peripheral name in function SAI0\_DriverIRQHandler.

#### [2.2.2]

- Bug Fixes
  - Fixed the issue of MISRA 2004 rule 9.3.
  - Fixed sign-compare warning.
  - Fixed the PA082 build warning.
  - Fixed sign-compare warning.
  - Fixed violations of MISRA C-2012 rule 10.3,17.7,10.4,8.4,10.7,10.8,14.4,17.7,11.6,10.1,10.6,8.4,14.3,16.4,18.2
  - Allow to reset Rx or Tx FIFO pointers only when Rx or Tx is disabled.
- Improvements
  - Added 24bit raw audio data width support in sai sdma driver.
  - Disabled the interrupt/DMA request in the SAI\_Init to avoid generates unexpected sai FIFO requests.

### [2.2.1]

- Improvements
  - Added mclk post divider support in function SAI\_SetMasterClockDivider.
  - Removed useless configuration code in SAI\_RxSetSerialDataConfig.
- Bug Fixes
  - Fixed the SAI SDMA driver build issue caused by the wrong structure member name used in the function SAI\_TransferRxSetConfigSDMA/SAI\_TransferTxSetConfigSDMA.
  - Fixed BAD BIT SHIFT OPERATION issue caused by the FSL\_FEATURE\_SAI\_CHANNEL\_COUNTn.
  - Applied ERR05144: not set FCONT = 1 when TMR > 0, otherwise the TX may not work.

### [2.2.0]

- Improvements
  - Added new APIs for parameters collection and simplified user interfaces:
    - \* SAI\_Init
    - \* SAI\_SetMasterClockConfig
    - \* SAI\_TxSetBitClockRate
    - \* SAI\_TxSetSerialDataConfig
    - \* SAI\_TxSetFrameSyncConfig
    - \* SAI\_TxSetFifoConfig
    - \* SAI\_TxSetBitclockConfig
    - \* SAI\_TxSetConfig
    - \* SAI\_TxSetTransferConfig
    - \* SAI\_RxSetBitClockRate
    - \* SAI\_RxSetSerialDataConfig
    - \* SAI\_RxSetFrameSyncConfig
    - \* SAI\_RxSetFifoConfig
    - \* SAI\_RxSetBitclockConfig
    - \* SAI\_RXSetConfig
    - \* SAI\_RxSetTransferConfig
    - \* SAI\_GetClassicI2SConfig
    - \* SAI\_GetLeftJustifiedConfig
    - \* SAI\_GetRightJustifiedConfig
    - \* SAI\_GetTDMConfig

### [2.1.9]

- Improvements
  - Improved SAI driver comment for clock polarity.
  - Added enumeration for SAI for sample inputs on different edges.

- Changed FSL\_FEATURE\_SAI\_CHANNEL\_COUNT to FSL\_FEATURE\_SAI\_CHANNEL\_COUNTn(base) for the difference between the different SAI instances.
- Added new APIs:
  - SAI\_TxSetBitClockDirection
  - SAI\_RxSetBitClockDirection
  - SAI\_RxSetFrameSyncDirection
  - SAI\_TxSetFrameSyncDirection

#### [2.1.8]

- Improvements
  - Added feature macro test for the sync mode2 and mode 3.
  - Added feature macro test for masterClockHz in sai\_transfer\_format\_t.

#### [2.1.7]

- Improvements
  - Added feature macro test for the mclkSource member in sai\_config\_t.
  - Changed “FSL\_FEATURE\_SAI5\_SAI6\_SHARE\_IRQ” to “FSL\_FEATURE\_SAI\_SAI5\_SAI6\_SHARE\_IRQ”.
  - Added #ifndef #endif check for SAI\_XFER\_QUEUE\_SIZE to allow redefinition.
- Bug Fixes
  - Fixed build error caused by feature macro test for mclkSource.

#### [2.1.6]

- Improvements
  - Added feature macro test for mclkSourceClockHz check.
  - Added bit clock source name for general devices.
- Bug Fixes
  - Fixed incorrect channel numbers setting while calling RX/TX set format together.

#### [2.1.5]

- Bug Fixes
  - Corrected SAI3 driver IRQ handler name.
  - Added I2S4/5/6 IRQ handler.
  - Added base in handler structure to support different instances sharing one IRQ number.
- New Features
  - Updated SAI driver for MCR bit MICS.
  - Added 192 KHZ/384 KHZ in the sample rate enumeration.
  - Added multi FIFO interrupt/SDMA transfer support for TX/RX.
  - Added an API to read/write multi FIFO data in a blocking method.
  - Added bclk bypass support when bclk is same with mclk.



**[2.1.4]**

- New Features
  - Added an API to enable/disable auto FIFO error recovery in platforms that support this feature.
  - Added an API to set data packing feature in platforms which support this feature.

**[2.1.3]**

- New Features
  - Added feature to make I2S frame sync length configurable according to bitWidth.

**[2.1.2]**

- Bug Fixes
  - Added 24-bit support for SAI eDMA transfer. All data shall be 32 bits for send/receive, as eDMA cannot directly handle 3-Byte transfer.

**[2.1.1]**

- Improvements
  - Reduced code size while not using transactional API.

**[2.1.0]**

- Improvements
  - API name changes:
    - \* SAI\_GetSendRemainingBytes -> SAI\_GetSentCount.
    - \* SAI\_GetReceiveRemainingBytes -> SAI\_GetReceivedCount.
    - \* All names of transactional APIs were added with “Transfer” prefix.
    - \* All transactional APIs use base and handle as input parameter.
    - \* Unified the parameter names.
- Bug Fixes
  - Fixed WLC bug while reading TCSR/RCSR registers.
  - Fixed MOE enable flow issue. Moved MOE enable after MICS settings in SAI\_TxInit/SAI\_RxInit.

**[2.0.0]**

- Initial version.
- 

**SEMA4****[2.2.2]**

- Improvements
  - Updated SEMA4\_TryLock function to avoid unsigned integer operations wrap issue.

### [2.2.1]

- Bug Fixes
  - Fixed violations of the CERT INT31-C, MISRA C-2012 rules 10.3, 10.4.

### [2.2.0]

- New Features
  - Added SEMA4\_BUSY\_POLL\_COUNT parameter to prevent infinite polling loops in SEMA4 operations.
  - Added timeout mechanism to all polling loops in SEMA4 driver code.
- Improvements
  - Updated SEMA4\_Lock function to return status\_t instead of void for better error handling.
  - Enhanced documentation to clarify timeout behavior and return values.

### [2.1.0]

- Improvements
  - Changed mask parameter type in SEMA4\_EnableGateNotifyInterrupt() and SEMA4\_DisableGateNotifyInterrupt() functions to avoid casting from unsigned long to unsigned short in the code when modifying the 16bits CPINE register.

### [2.0.3]

- Improvements
  - Changed to implement SEMA4\_Lock base on SEMA4\_TryLock.

### [2.0.2]

- Improvements:
  - Supported the SEMA4\_Type structure whose gate registers are defined as an array.

### [2.0.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 10.4, 15.5, 18.1, 18.4.

### [2.0.0]

- Initial version.
- 

## SNVS\_HP

### [2.3.2]

- Make SNVS\_HP\_RTC\_Init()/SNVS\_HP\_RTC\_Deinit more transparent. Use function SNVS\_HP\_Init()/SNVS\_HP\_Deinit() instead of copy of this code in SNVS\_HP\_RTC\_XXX() function.

**[2.3.1]**

- Fixed problem in SNVS\_HP\_RTC\_Init(), which is clearing bits that should stay intact.

**[2.3.0]**

- Re-map Security Violation for RT11xx specific violations.

**[2.2.0]**

- Fixed doxygen issues.
- Add SNVS HP Set locks.

**[2.1.4]**

- Fix MISRA issues.

**[2.1.3]**

- Fixed IAR Pa082 warnings.

**[2.1.2]**

- Fixed problem with initialization of the periodic interrupt frequency.
- Fixed problem with SNVS entering into fail state when HAB enters closed mode.

**[2.1.1]**

- Added APIs for HP security violation status flags.

**[2.1.0]**

- Added APIs for High Assurance Counter (HAC), Zeroizable Master Key (ZMK) and Software Security Violation.

**[2.0.0]**

- Initial version.
- 

**SNVS\_LP****[2.4.6]**

- Fix a bug in SNVS\_LP\_EnableRxActiveTamper() where assignments to base->LPATRC2R were done wrongly to LPATRC1R.

**[2.4.5]**

- Fix a bug in SNVS\_LP\_EnableRxActiveTamper() where assignments to base->LPATRC1R would overwrite previously set bits.

**[2.4.4]**

- Make SNVS\_LP\_SRTC\_Init()/SNVS\_LP\_SRTC\_Deinit more transparent. Use function SNVS\_LP\_Init()/SNVS\_LP\_Deinit() instead of copy of this code in SNVS\_LP\_SRTC\_XXX() function.

**[2.4.3]**

- Fixed problem in SNVS\_LP\_SRTC\_Init(), which is clearing bits that should stay intact.

**[2.4.2]**

- Updated driver to match with new device header files.

**[2.4.1]**

- Fixed MISRA issues.

**[2.4.0]**

- Fix backward compatibility with version 2.2.x.

**[2.3.0]**

- Add active pin, clock, voltage and temperature tamper features.

**[2.2.0]**

- Fixed doxygen issues.
- Add Transition SNVS SSM state to Trusted/Non-secure from Check state.

**[2.1.2]**

- Fix MISRA issues.

**[2.1.1]**

- Fix IAR Pa082 warning.

**[2.1.0]**

- Added APIs for Zeroizable Master Key (ZMK) and Monotonic Counter (MC).

**[2.0.0]**

- Initial version.
-

## TMU

### [2.0.3]

- Bug Fixes
  - Fixed the violations of MISRA 2012 rules:
    - \* Rule 10.1 10.3 10.4 17.7.

### [2.0.2]

- Bug Fixes
  - Fixed missing right pair definition for extern C.

### [2.0.1]

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

### [2.0.0]

- Initial version.
  - This module was first developed on i.MX 8MQuad.
- 

## UART

### [2.3.2]

- Improvements
  - Make driver aarch64 compatible

### [2.3.1]

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

### [2.3.0]

- Bug Fixes
  - Fixed DMA transfer blocking issue by enabling tx idle interrupt after DMA transmission finishes.

### [2.2.1]

- Bug Fixes
  - Fixed MISRA 2012 rule 10.4 violation.

## [2.2.0]

- New Features
  - Modified `uart_config_t`, `UART_Init` and `UART_GetDefaultConfig` APIs so that the RTS and CTS used for hardware flow control can be enabled during module initialization.
  - Added API `UART_SetRxRTSWatermark` so that the water mark level of RTS deassertion can be configured.

## [2.1.1]

- Bug Fixes
  - Fixed MISRA 8.5 violation.

## [2.1.0]

- Improvements
  - Added timeout mechanism when waiting for certain states in transfer driver.

## [2.0.2]

- Improvements
  - Added check for transmission complete in `UART_WriteBlocking`, `UART_TransferHandleIRQ` and `UART_SendSDMACallback` to ensure all the data would be sent out to bus.
  - Modified `UART_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
  - Fixed MISRA issues.
    - \* Fixed rules 10.1, 10.3, 10.4, 10.6, 10.7, 10.8, 11.9, 14.4.

## [2.0.1]

- Bug Fixes
  - Memset local variable SDMA transfer configuration structure to make sure unused members in structure are cleared.

## [2.0.0]

- Initial version.
- 

## WDOG

## [2.2.0]

- Bug Fixes
  - Fixed the wrong behavior of `workMode.enableWait`, `workMode.enableStop`, `workMode.enableDebug` in configuration structure `wdog_config_t`. When set the items to true, WDOG will continues working in those modes.

### [2.1.1]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.1, 10.3, 10.4, 10.6, 10.7 and 11.9.
  - Fixed the issue of the inseparable process interrupted by other interrupt source.
    - \* WDOG\_Init
    - \* WDOG\_Refresh

### [2.1.0]

- New Features
  - Added new API “WDOG\_TriggerSystemSoftwareReset()” to allow users to reset the system by software.
  - Added new API “WDOG\_TriggerSoftwareSignal()” to allow users to trigger a WDOG\_B signal by software.
  - Removed the parameter “softwareAssertion” and “softwareResetSignal” out of the wdog\_config\_t structure.
  - Added new parameter “enableTimeOutAssert” to the wdog\_config\_t structure. With this parameter enabled, when the WDOG timeout occurs, a WDOG\_B signal will be asserted. This signal can be routed to external pin of the chip. Note that WDOG\_B signal remains asserted until a power-on reset (POR) occurs.

### [2.0.1]

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

### [2.0.0]

- Initial version.
- 

## 1.6 Driver API Reference Manual

This section provides a link to the Driver API RM, detailing available drivers and their usage to help you integrate hardware efficiently.

[MIMX8MQ6](#)

## 1.7 Middleware Documentation

Find links to detailed middleware documentation for key components. While not all onboard middleware is covered, this serves as a useful reference for configuration and development.

### 1.7.1 Multicore

multicore

## 1.7.2 FreeMASTER

*freemaster*

## 1.7.3 FreeRTOS

*FreeRTOS*



# Chapter 2

## MIMX8MQ6

### 2.1 CACHE: LMEM CACHE Memory Controller

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 16-Byte due to the cache operation unit FSL\_FEATURE\_L1ICACHE\_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.

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 16-Byte due to the cache operation unit FSL\_FEATURE\_L1DCACHE\_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.

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 16-Byte due to the cache operation unit FSL\_FEATURE\_L1DCACHE\_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.

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 16-Byte due to the cache operation unit FSL\_FEATURE\_L1DCACHE\_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.

FSL\_CACHE\_DRIVER\_VERSION

cache driver version.

L1CODEBUSCACHE\_LINESIZE\_BYTE

code bus cache line size is equal to system bus line size, so the unified I/D cache line size equals too.

The code bus CACHE line size is 16B = 128b.

L1SYSTEMBUSCACHE\_LINESIZE\_BYTE

The system bus CACHE line size is 16B = 128b.

## 2.2 Clock

enum \_\_clock\_name

Clock name used to get clock frequency.

*Values:*

enumerator kCLOCK\_CoreM4Clk

ARM M4 Core clock

enumerator kCLOCK\_AxiClk

Main AXI bus clock.

enumerator kCLOCK\_AhbClk

AHB bus clock.

enumerator kCLOCK\_IpgClk

IPG bus clock.

enumerator kCLOCK\_Osc25MClk

OSC 25M clock.

enumerator kCLOCK\_Osc27MClk

OSC 27M clock.

enumerator kCLOCK\_ArmPllClk

Arm PLL clock.

enumerator kCLOCK\_VpuPllClk

Vpu PLL clock.

enumerator kCLOCK\_DramPllClk

Dram PLL clock.

enumerator kCLOCK\_SysPll1Clk

Sys PLL1 clock.

enumerator kCLOCK\_SysPll1Div2Clk

Sys PLL1 clock divided by 2.

enumerator kCLOCK\_SysPll1Div3Clk

Sys PLL1 clock divided by 3.

enumerator kCLOCK\_SysPll1Div4Clk

Sys PLL1 clock divided by 4.

enumerator kCLOCK\_SysPll1Div5Clk

Sys PLL1 clock divided by 5.

enumerator kCLOCK\_SysPll1Div6Clk

Sys PLL1 clock divided by 6.

enumerator kCLOCK\_SysPll1Div8Clk

Sys PLL1 clock divided by 8.

enumerator kCLOCK\_SysPll1Div10Clk

Sys PLL1 clock divided by 10.

enumerator kCLOCK\_SysPll1Div20Clk

Sys PLL1 clock divided by 20.

enumerator kCLOCK\_SysPll2Clk

Sys PLL2 clock.

enumerator kCLOCK\_SysPll2Div2Clk

Sys PLL2 clock divided by 2.

enumerator kCLOCK\_SysPll2Div3Clk

Sys PLL2 clock divided by 3.

enumerator kCLOCK\_SysPll2Div4Clk

Sys PLL2 clock divided by 4.

enumerator kCLOCK\_SysPll2Div5Clk

Sys PLL2 clock divided by 5.

enumerator kCLOCK\_SysPll2Div6Clk

Sys PLL2 clock divided by 6.

enumerator kCLOCK\_SysPll2Div8Clk

Sys PLL2 clock divided by 8.

enumerator kCLOCK\_SysPll2Div10Clk

Sys PLL2 clock divided by 10.

enumerator kCLOCK\_SysPll2Div20Clk

Sys PLL2 clock divided by 20.

enumerator kCLOCK\_SysPll3Clk

Sys PLL3 clock.

enumerator kCLOCK\_AudioPll1Clk

Audio PLL1 clock.

enumerator kCLOCK\_AudioPll2Clk  
Audio PLL2 clock.

enumerator kCLOCK\_VideoPll1Clk  
Video PLL1 clock.

enumerator kCLOCK\_ExtClk1  
External clock1.

enumerator kCLOCK\_ExtClk2  
External clock2.

enumerator kCLOCK\_ExtClk3  
External clock3.

enumerator kCLOCK\_ExtClk4  
External clock4.

enumerator kCLOCK\_NoneName  
None Clock Name.

enum \_clock\_ip\_name  
CCM CCGR gate control.

*Values:*

enumerator kCLOCK\_IpInvalid

enumerator kCLOCK\_Debug  
DEBUG Clock Gate.

enumerator kCLOCK\_Dram  
DRAM Clock Gate.

enumerator kCLOCK\_Ecspi1  
ECSPI1 Clock Gate.

enumerator kCLOCK\_Ecspi2  
ECSPI2 Clock Gate.

enumerator kCLOCK\_Ecspi3  
ECSPI3 Clock Gate.

enumerator kCLOCK\_Gpio1  
GPIO1 Clock Gate.

enumerator kCLOCK\_Gpio2  
GPIO2 Clock Gate.

enumerator kCLOCK\_Gpio3  
GPIO3 Clock Gate.

enumerator kCLOCK\_Gpio4  
GPIO4 Clock Gate.

enumerator kCLOCK\_Gpio5  
GPIO5 Clock Gate.

enumerator kCLOCK\_Gpt1  
GPT1 Clock Gate.

enumerator kCLOCK\_Gpt2  
GPT2 Clock Gate.

enumerator kCLOCK\_Gpt3  
GPT3 Clock Gate.

enumerator kCLOCK\_Gpt4  
GPT4 Clock Gate.

enumerator kCLOCK\_Gpt5  
GPT5 Clock Gate.

enumerator kCLOCK\_Gpt6  
GPT6 Clock Gate.

enumerator kCLOCK\_I2c1  
I2C1 Clock Gate.

enumerator kCLOCK\_I2c2  
I2C2 Clock Gate.

enumerator kCLOCK\_I2c3  
I2C3 Clock Gate.

enumerator kCLOCK\_I2c4  
I2C4 Clock Gate.

enumerator kCLOCK\_Iomux  
IOMUX Clock Gate.

enumerator kCLOCK\_Ipmux1  
IPMUX1 Clock Gate.

enumerator kCLOCK\_Ipmux2  
IPMUX2 Clock Gate.

enumerator kCLOCK\_Ipmux3  
IPMUX3 Clock Gate.

enumerator kCLOCK\_Ipmux4  
IPMUX4 Clock Gate.

enumerator kCLOCK\_M4  
M4 Clock Gate.

enumerator kCLOCK\_Mu  
MU Clock Gate.

enumerator kCLOCK\_Ocram  
OCRAM Clock Gate.

enumerator kCLOCK\_OcramS  
OCRAM S Clock Gate.

enumerator kCLOCK\_Pwm1  
PWM1 Clock Gate.

enumerator kCLOCK\_Pwm2  
PWM2 Clock Gate.

enumerator kCLOCK\_Pwm3  
PWM3 Clock Gate.

enumerator kCLOCK\_Pwm4  
PWM4 Clock Gate.

enumerator kCLOCK\_Qspi  
QSPI Clock Gate.

enumerator kCLOCK\_Rdc  
RDC Clock Gate.

enumerator kCLOCK\_Sai1  
SAI1 Clock Gate.

enumerator kCLOCK\_Sai2  
SAI2 Clock Gate.

enumerator kCLOCK\_Sai3  
SAI3 Clock Gate.

enumerator kCLOCK\_Sai4  
SAI4 Clock Gate.

enumerator kCLOCK\_Sai5  
SAI5 Clock Gate.

enumerator kCLOCK\_Sai6  
SAI6 Clock Gate.

enumerator kCLOCK\_Sdma1  
SDMA1 Clock Gate.

enumerator kCLOCK\_Sdma2  
SDMA2 Clock Gate.

enumerator kCLOCK\_Sec\_Debug  
SEC\_DEBUG Clock Gate.

enumerator kCLOCK\_Sema42\_1  
RDC SEMA42 Clock Gate.

enumerator kCLOCK\_Sema42\_2  
RDC SEMA42 Clock Gate.

enumerator kCLOCK\_Sim\_display  
SIM\_Display Clock Gate.

enumerator kCLOCK\_Sim\_m  
SIM\_M Clock Gate.

enumerator kCLOCK\_Sim\_main  
SIM\_MAIN Clock Gate.

enumerator kCLOCK\_Sim\_s  
SIM\_S Clock Gate.

enumerator kCLOCK\_Sim\_wakeup  
SIM\_WAKEUP Clock Gate.

enumerator kCLOCK\_Uart1  
UART1 Clock Gate.

enumerator kCLOCK\_Uart2  
UART2 Clock Gate.

enumerator kCLOCK\_Uart3  
UART3 Clock Gate.

enumerator kCLOCK\_Uart4

UART4 Clock Gate.

enumerator kCLOCK\_Wdog1

WDOG1 Clock Gate.

enumerator kCLOCK\_Wdog2

WDOG2 Clock Gate.

enumerator kCLOCK\_Wdog3

WDOG3 Clock Gate.

enumerator kCLOCK\_TempSensor

TempSensor Clock Gate.

enum \_clock\_root\_control

ccm root name used to get clock frequency.

*Values:*

enumerator kCLOCK\_RootM4

ARM Cortex-M4 Clock control name.

enumerator kCLOCK\_RootAxi

AXI Clock control name.

enumerator kCLOCK\_RootNoc

NOC Clock control name.

enumerator kCLOCK\_RootAhb

AHB Clock control name.

enumerator kCLOCK\_RootIpq

IPG Clock control name.

enumerator kCLOCK\_RootDramAlt

DRAM ALT Clock control name.

enumerator kCLOCK\_RootSai1

SAI1 Clock control name.

enumerator kCLOCK\_RootSai2

SAI2 Clock control name.

enumerator kCLOCK\_RootSai3

SAI3 Clock control name.

enumerator kCLOCK\_RootSai4

SAI4 Clock control name.

enumerator kCLOCK\_RootSai5

SAI5 Clock control name.

enumerator kCLOCK\_RootSai6

SAI6 Clock control name.

enumerator kCLOCK\_RootQspi

QSPI Clock control name.

enumerator kCLOCK\_RootI2c1

I2C1 Clock control name.

enumerator kCLOCK\_RootI2c2  
I2C2 Clock control name.

enumerator kCLOCK\_RootI2c3  
I2C3 Clock control name.

enumerator kCLOCK\_RootI2c4  
I2C4 Clock control name.

enumerator kCLOCK\_RootUart1  
UART1 Clock control name.

enumerator kCLOCK\_RootUart2  
UART2 Clock control name.

enumerator kCLOCK\_RootUart3  
UART3 Clock control name.

enumerator kCLOCK\_RootUart4  
UART4 Clock control name.

enumerator kCLOCK\_RootEcspi1  
ECSPI1 Clock control name.

enumerator kCLOCK\_RootEcspi2  
ECSPI2 Clock control name.

enumerator kCLOCK\_RootEcspi3  
ECSPI3 Clock control name.

enumerator kCLOCK\_RootPwm1  
PWM1 Clock control name.

enumerator kCLOCK\_RootPwm2  
PWM2 Clock control name.

enumerator kCLOCK\_RootPwm3  
PWM3 Clock control name.

enumerator kCLOCK\_RootPwm4  
PWM4 Clock control name.

enumerator kCLOCK\_RootGpt1  
GPT1 Clock control name.

enumerator kCLOCK\_RootGpt2  
GPT2 Clock control name.

enumerator kCLOCK\_RootGpt3  
GPT3 Clock control name.

enumerator kCLOCK\_RootGpt4  
GPT4 Clock control name.

enumerator kCLOCK\_RootGpt5  
GPT5 Clock control name.

enumerator kCLOCK\_RootGpt6  
GPT6 Clock control name.

enumerator kCLOCK\_RootWdog  
WDOG Clock control name.



enum \_clock\_root

ccm clock root used to get clock frequency.

*Values:*

enumerator kCLOCK\_M4ClkRoot

ARM Cortex-M4 Clock control name.

enumerator kCLOCK\_AxiClkRoot

AXI Clock control name.

enumerator kCLOCK\_NocClkRoot

NOC Clock control name.

enumerator kCLOCK\_AhbClkRoot

AHB Clock control name.

enumerator kCLOCK\_IpgClkRoot

IPG Clock control name.

enumerator kCLOCK\_DramAltClkRoot

DRAM ALT Clock control name.

enumerator kCLOCK\_Sai1ClkRoot

SAI1 Clock control name.

enumerator kCLOCK\_Sai2ClkRoot

SAI2 Clock control name.

enumerator kCLOCK\_Sai3ClkRoot

SAI3 Clock control name.

enumerator kCLOCK\_Sai4ClkRoot

SAI4 Clock control name.

enumerator kCLOCK\_Sai5ClkRoot

SAI5 Clock control name.

enumerator kCLOCK\_Sai6ClkRoot

SAI6 Clock control name.

enumerator kCLOCK\_QspiClkRoot

QSPI Clock control name.

enumerator kCLOCK\_I2c1ClkRoot

I2C1 Clock control name.

enumerator kCLOCK\_I2c2ClkRoot

I2C2 Clock control name.

enumerator kCLOCK\_I2c3ClkRoot

I2C3 Clock control name.

enumerator kCLOCK\_I2c4ClkRoot

I2C4 Clock control name.

enumerator kCLOCK\_Uart1ClkRoot

UART1 Clock control name.

enumerator kCLOCK\_Uart2ClkRoot

UART2 Clock control name.

enumerator kCLOCK\_Uart3ClkRoot  
UART3 Clock control name.

enumerator kCLOCK\_Uart4ClkRoot  
UART4 Clock control name.

enumerator kCLOCK\_Ecspi1ClkRoot  
ECSPI1 Clock control name.

enumerator kCLOCK\_Ecspi2ClkRoot  
ECSPI2 Clock control name.

enumerator kCLOCK\_Ecspi3ClkRoot  
ECSPI3 Clock control name.

enumerator kCLOCK\_Pwm1ClkRoot  
PWM1 Clock control name.

enumerator kCLOCK\_Pwm2ClkRoot  
PWM2 Clock control name.

enumerator kCLOCK\_Pwm3ClkRoot  
PWM3 Clock control name.

enumerator kCLOCK\_Pwm4ClkRoot  
PWM4 Clock control name.

enumerator kCLOCK\_Gpt1ClkRoot  
GPT1 Clock control name.

enumerator kCLOCK\_Gpt2ClkRoot  
GPT2 Clock control name.

enumerator kCLOCK\_Gpt3ClkRoot  
GPT3 Clock control name.

enumerator kCLOCK\_Gpt4ClkRoot  
GPT4 Clock control name.

enumerator kCLOCK\_Gpt5ClkRoot  
GPT5 Clock control name.

enumerator kCLOCK\_Gpt6ClkRoot  
GPT6 Clock control name.

enumerator kCLOCK\_WdogClkRoot  
WDOG Clock control name.

enum \_clock\_rootmux\_m4\_clk\_sel  
Root clock select enumeration for ARM Cortex-M4 core.

*Values:*

enumerator kCLOCK\_M4RootmuxOsc25m  
ARM Cortex-M4 Clock from OSC 25M.

enumerator kCLOCK\_M4RootmuxSysPll2Div5  
ARM Cortex-M4 Clock from SYSTEM PLL2 divided by 5.

enumerator kCLOCK\_M4RootmuxSysPll2Div4  
ARM Cortex-M4 Clock from SYSTEM PLL2 divided by 4.

enumerator kCLOCK\_M4RootmuxSysPll1Div3  
ARM Cortex-M4 Clock from SYSTEM PLL1 divided by 3.

enumerator kCLOCK\_M4RootmuxSysPll1  
ARM Cortex-M4 Clock from SYSTEM PLL1.

enumerator kCLOCK\_M4RootmuxAudioPll1  
ARM Cortex-M4 Clock from AUDIO PLL1.

enumerator kCLOCK\_M4RootmuxVideoPll1  
ARM Cortex-M4 Clock from VIDEO PLL1.

enumerator kCLOCK\_M4RootmuxSysPll3  
ARM Cortex-M4 Clock from SYSTEM PLL3.

enum \_clock\_rootmux\_axi\_clk\_sel  
Root clock select enumeration for AXI bus.

*Values:*

enumerator kCLOCK\_AxiRootmuxOsc25m  
ARM AXI Clock from OSC 25M.

enumerator kCLOCK\_AxiRootmuxSysPll2Div3  
ARM AXI Clock from SYSTEM PLL2 divided by 3.

enumerator kCLOCK\_AxiRootmuxSysPll1  
ARM AXI Clock from SYSTEM PLL1.

enumerator kCLOCK\_AxiRootmuxSysPll2Div4  
ARM AXI Clock from SYSTEM PLL2 divided by 4.

enumerator kCLOCK\_AxiRootmuxSysPll2  
ARM AXI Clock from SYSTEM PLL2.

enumerator kCLOCK\_AxiRootmuxAudioPll1  
ARM AXI Clock from AUDIO PLL1.

enumerator kCLOCK\_AxiRootmuxVideoPll1  
ARM AXI Clock from VIDEO PLL1.

enumerator kCLOCK\_AxiRootmuxSysPll1Div8  
ARM AXI Clock from SYSTEM PLL1 divided by 8.

enum \_clock\_rootmux\_ahb\_clk\_sel  
Root clock select enumeration for AHB bus.

*Values:*

enumerator kCLOCK\_AhbRootmuxOsc25m  
ARM AHB Clock from OSC 25M.

enumerator kCLOCK\_AhbRootmuxSysPll1Div6  
ARM AHB Clock from SYSTEM PLL1 divided by 6.

enumerator kCLOCK\_AhbRootmuxSysPll1  
ARM AHB Clock from SYSTEM PLL1.

enumerator kCLOCK\_AhbRootmuxSysPll1Div2  
ARM AHB Clock from SYSTEM PLL1 divided by 2.

enumerator kCLOCK\_AhbRootmuxSysPll2Div8  
ARM AHB Clock from SYSTEM PLL2 divided by 8.

enumerator kCLOCK\_AhbRootmuxSysPll3

ARM AHB Clock from SYSTEM PLL3.

enumerator kCLOCK\_AhbRootmuxAudioPll1

ARM AHB Clock from AUDIO PLL1.

enumerator kCLOCK\_AhbRootmuxVideoPll1

ARM AHB Clock from VIDEO PLL1.

enum \_clock\_rootmux\_qspi\_clk\_sel

Root clock select enumeration for QSPI peripheral.

*Values:*

enumerator kCLOCK\_QspiRootmuxOsc25m

ARM QSPI Clock from OSC 25M.

enumerator kCLOCK\_QspiRootmuxSysPll1Div2

ARM QSPI Clock from SYSTEM PLL1 divided by 2.

enumerator kCLOCK\_QspiRootmuxSysPll1

ARM QSPI Clock from SYSTEM PLL1.

enumerator kCLOCK\_QspiRootmuxSysPll2Div2

ARM QSPI Clock from SYSTEM PLL2 divided by 2.

enumerator kCLOCK\_QspiRootmuxAudioPll2

ARM QSPI Clock from AUDIO PLL2.

enumerator kCLOCK\_QspiRootmuxSysPll1Div3

ARM QSPI Clock from SYSTEM PLL1 divided by 3

enumerator kCLOCK\_QspiRootmuxSysPll3

ARM QSPI Clock from SYSTEM PLL3.

enumerator kCLOCK\_QspiRootmuxSysPll1Div8

ARM QSPI Clock from SYSTEM PLL1 divided by 8.

enum \_clock\_rootmux\_ecspi\_clk\_sel

Root clock select enumeration for ECSPI peripheral.

*Values:*

enumerator kCLOCK\_EcspiRootmuxOsc25m

ECSPI Clock from OSC 25M.

enumerator kCLOCK\_EcspiRootmuxSysPll2Div5

ECSPI Clock from SYSTEM PLL2 divided by 5.

enumerator kCLOCK\_EcspiRootmuxSysPll1Div20

ECSPI Clock from SYSTEM PLL1 divided by 20.

enumerator kCLOCK\_EcspiRootmuxSysPll1Div5

ECSPI Clock from SYSTEM PLL1 divided by 5.

enumerator kCLOCK\_EcspiRootmuxSysPll1

ECSPI Clock from SYSTEM PLL1.

enumerator kCLOCK\_EcspiRootmuxSysPll3

ECSPI Clock from SYSTEM PLL3.

enumerator kCLOCK\_EcspiRootmuxSysPll2Div4

ECSPI Clock from SYSTEM PLL2 divided by 4.

enumerator kCLOCK\_EcspiRootmuxAudioPll2  
ECSPI Clock from AUDIO PLL2.

enum \_clock\_rootmux\_i2c\_clk\_sel  
Root clock select enumeration for I2C peripheral.

*Values:*

enumerator kCLOCK\_I2cRootmuxOsc25m  
I2C Clock from OSC 25M.

enumerator kCLOCK\_I2cRootmuxSysPll1Div5  
I2C Clock from SYSTEM PLL1 divided by 5.

enumerator kCLOCK\_I2cRootmuxSysPll2Div20  
I2C Clock from SYSTEM PLL2 divided by 20.

enumerator kCLOCK\_I2cRootmuxSysPll3  
I2C Clock from SYSTEM PLL3 .

enumerator kCLOCK\_I2cRootmuxAudioPll1  
I2C Clock from AUDIO PLL1.

enumerator kCLOCK\_I2cRootmuxVideoPll1  
I2C Clock from VIDEO PLL1.

enumerator kCLOCK\_I2cRootmuxAudioPll2  
I2C Clock from AUDIO PLL2.

enumerator kCLOCK\_I2cRootmuxSysPll1Div6  
I2C Clock from SYSTEM PLL1 divided by 6.

enum \_clock\_rootmux\_uart\_clk\_sel  
Root clock select enumeration for UART peripheral.

*Values:*

enumerator kCLOCK\_UartRootmuxOsc25m  
UART Clock from OSC 25M.

enumerator kCLOCK\_UartRootmuxSysPll1Div10  
UART Clock from SYSTEM PLL1 divided by 10.

enumerator kCLOCK\_UartRootmuxSysPll2Div5  
UART Clock from SYSTEM PLL2 divided by 5.

enumerator kCLOCK\_UartRootmuxSysPll2Div10  
UART Clock from SYSTEM PLL2 divided by 10.

enumerator kCLOCK\_UartRootmuxSysPll3  
UART Clock from SYSTEM PLL3.

enumerator kCLOCK\_UartRootmuxExtClk2  
UART Clock from External Clock 2.

enumerator kCLOCK\_UartRootmuxExtClk34  
UART Clock from External Clock 3, External Clock 4.

enumerator kCLOCK\_UartRootmuxAudioPll2  
UART Clock from Audio PLL2.

enum \_clock\_rootmux\_gpt  
Root clock select enumeration for GPT peripheral.

*Values:*

enumerator kCLOCK\_GptRootmuxOsc25m

GPT Clock from OSC 25M.

enumerator kCLOCK\_GptRootmuxSystemPll2Div10

GPT Clock from SYSTEM PLL2 divided by 10.

enumerator kCLOCK\_GptRootmuxSysPll1Div2

GPT Clock from SYSTEM PLL1 divided by 2.

enumerator kCLOCK\_GptRootmuxSysPll1Div20

GPT Clock from SYSTEM PLL1 divided by 20.

enumerator kCLOCK\_GptRootmuxVideoPll1

GPT Clock from VIDEO PLL1.

enumerator kCLOCK\_GptRootmuxSystemPll1Div10

GPT Clock from SYSTEM PLL1 divided by 10.

enumerator kCLOCK\_GptRootmuxAudioPll1

GPT Clock from AUDIO PLL1.

enumerator kCLOCK\_GptRootmuxExtClk123

GPT Clock from External Clock1, External Clock2, External Clock3.

enum \_clock\_rootmux\_wdog\_clk\_sel

Root clock select enumeration for WDOG peripheral.

*Values:*

enumerator kCLOCK\_WdogRootmuxOsc25m

WDOG Clock from OSC 25M.

enumerator kCLOCK\_WdogRootmuxSysPll1Div6

WDOG Clock from SYSTEM PLL1 divided by 6.

enumerator kCLOCK\_WdogRootmuxSysPll1Div5

WDOG Clock from SYSTEM PLL1 divided by 5.

enumerator kCLOCK\_WdogRootmuxVpuPll

WDOG Clock from VPU DLL.

enumerator kCLOCK\_WdogRootmuxSystemPll2Div8

WDOG Clock from SYSTEM PLL2 divided by 8.

enumerator kCLOCK\_WdogRootmuxSystemPll3

WDOG Clock from SYSTEM PLL3.

enumerator kCLOCK\_WdogRootmuxSystemPll1Div10

WDOG Clock from SYSTEM PLL1 divided by 10.

enumerator kCLOCK\_WdogRootmuxSystemPll2Div6

WDOG Clock from SYSTEM PLL2 divided by 6.

enum \_clock\_rootmux\_pwm\_clk\_sel

Root clock select enumeration for PWM peripheral.

*Values:*

enumerator kCLOCK\_PwmRootmuxOsc25m

PWM Clock from OSC 25M.

enumerator kCLOCK\_PwmRootmuxSysPll2Div10

PWM Clock from SYSTEM PLL2 divided by 10.

enumerator kCLOCK\_PwmRootmuxSysPll1Div5

PWM Clock from SYSTEM PLL1 divided by 5.

enumerator kCLOCK\_PwmRootmuxSysPll1Div20

PWM Clock from SYSTEM PLL1 divided by 20.

enumerator kCLOCK\_PwmRootmuxSystemPll3

PWM Clock from SYSTEM PLL3.

enumerator kCLOCK\_PwmRootmuxExtClk12

PWM Clock from External Clock1, External Clock2.

enumerator kCLOCK\_PwmRootmuxSystemPll1Div10

PWM Clock from SYSTEM PLL1 divided by 10.

enumerator kCLOCK\_PwmRootmuxVideoPll1

PWM Clock from VIDEO PLL1.

enum \_clock\_rootmux\_sai\_clk\_sel

Root clock select enumeration for SAI peripheral.

*Values:*

enumerator kCLOCK\_SaiRootmuxOsc25m

SAI Clock from OSC 25M.

enumerator kCLOCK\_SaiRootmuxAudioPll1

SAI Clock from AUDIO PLL1.

enumerator kCLOCK\_SaiRootmuxAudioPll2

SAI Clock from AUDIO PLL2.

enumerator kCLOCK\_SaiRootmuxVideoPll1

SAI Clock from VIDEO PLL1.

enumerator kCLOCK\_SaiRootmuxSysPll1Div6

SAI Clock from SYSTEM PLL1 divided by 6.

enumerator kCLOCK\_SaiRootmuxOsc27m

SAI Clock from OSC 27M.

enumerator kCLOCK\_SaiRootmuxExtClk123

SAI Clock from External Clock1, External Clock2, External Clock3.

enumerator kCLOCK\_SaiRootmuxExtClk234

SAI Clock from External Clock2, External Clock3, External Clock4.

enum \_clock\_rootmux\_noc\_clk\_sel

Root clock select enumeration for NOC CLK.

*Values:*

enumerator kCLOCK\_NocRootmuxOsc25m

NOC Clock from OSC 25M.

enumerator kCLOCK\_NocRootmuxSysPll1

NOC Clock from SYSTEM PLL1.

enumerator kCLOCK\_NocRootmuxSysPll3

NOC Clock from SYSTEM PLL3.

enumerator kCLOCK\_NocRootmuxSysPll2

NOC Clock from SYSTEM PLL2.

enumerator kCLOCK\_\_NocRootmuxSysPll2Div2  
NOC Clock from SYSTEM PLL2 divided by 2.

enumerator kCLOCK\_\_NocRootmuxAudioPll1  
NOC Clock from AUDIO PLL1.

enumerator kCLOCK\_\_NocRootmuxVideoPll1  
NOC Clock from VIDEO PLL1.

enumerator kCLOCK\_\_NocRootmuxAudioPll2  
NOC Clock from AUDIO PLL2.

enum \_clock\_pll\_gate  
CCM PLL gate control.

*Values:*

enumerator kCLOCK\_\_ArmPllGate  
ARM PLL Gate.

enumerator kCLOCK\_\_GpuPllGate  
GPU PLL Gate.

enumerator kCLOCK\_\_VpuPllGate  
VPU PLL Gate.

enumerator kCLOCK\_\_DramPllGate  
DRAM PLL1 Gate.

enumerator kCLOCK\_\_SysPll1Gate  
SYSTEM PLL1 Gate.

enumerator kCLOCK\_\_SysPll1Div2Gate  
SYSTEM PLL1 Div2 Gate.

enumerator kCLOCK\_\_SysPll1Div3Gate  
SYSTEM PLL1 Div3 Gate.

enumerator kCLOCK\_\_SysPll1Div4Gate  
SYSTEM PLL1 Div4 Gate.

enumerator kCLOCK\_\_SysPll1Div5Gate  
SYSTEM PLL1 Div5 Gate.

enumerator kCLOCK\_\_SysPll1Div6Gate  
SYSTEM PLL1 Div6 Gate.

enumerator kCLOCK\_\_SysPll1Div8Gate  
SYSTEM PLL1 Div8 Gate.

enumerator kCLOCK\_\_SysPll1Div10Gate  
SYSTEM PLL1 Div10 Gate.

enumerator kCLOCK\_\_SysPll1Div20Gate  
SYSTEM PLL1 Div20 Gate.

enumerator kCLOCK\_\_SysPll2Gate  
SYSTEM PLL2 Gate.

enumerator kCLOCK\_\_SysPll2Div2Gate  
SYSTEM PLL2 Div2 Gate.



enumerator kCLOCK\_SysPll2Div3Gate  
SYSTEM PLL2 Div3 Gate.

enumerator kCLOCK\_SysPll2Div4Gate  
SYSTEM PLL2 Div4 Gate.

enumerator kCLOCK\_SysPll2Div5Gate  
SYSTEM PLL2 Div5 Gate.

enumerator kCLOCK\_SysPll2Div6Gate  
SYSTEM PLL2 Div6 Gate.

enumerator kCLOCK\_SysPll2Div8Gate  
SYSTEM PLL2 Div8 Gate.

enumerator kCLOCK\_SysPll2Div10Gate  
SYSTEM PLL2 Div10 Gate.

enumerator kCLOCK\_SysPll2Div20Gate  
SYSTEM PLL2 Div20 Gate.

enumerator kCLOCK\_SysPll3Gate  
SYSTEM PLL3 Gate.

enumerator kCLOCK\_AudioPll1Gate  
AUDIO PLL1 Gate.

enumerator kCLOCK\_AudioPll2Gate  
AUDIO PLL2 Gate.

enumerator kCLOCK\_VideoPll1Gate  
VIDEO PLL1 Gate.

enumerator kCLOCK\_VideoPll2Gate  
VIDEO PLL2 Gate.

enum \_clock\_gate\_value  
CCM gate control value.

*Values:*

enumerator kCLOCK\_ClockNotNeeded  
Clock always disabled.

enumerator kCLOCK\_ClockNeededRun  
Clock enabled when CPU is running.

enumerator kCLOCK\_ClockNeededRunWait  
Clock enabled when CPU is running or in WAIT mode.

enumerator kCLOCK\_ClockNeededAll  
Clock always enabled.

enum \_clock\_pll\_bypass\_ctrl  
PLL control names for PLL bypass.

These constants define the PLL control names for PLL bypass.

- 0:15: REG offset to CCM\_ANALOG\_BASE in bytes.
- 16:20: bypass bit shift.

*Values:*

enumerator kCLOCK\_AudioPll1BypassCtrl  
CCM Audio PLL1 bypass Control.

enumerator kCLOCK\_AudioPll2BypassCtrl  
CCM Audio PLL2 bypass Control.

enumerator kCLOCK\_VideoPll1BypassCtrl  
CCM Video Pll1 bypass Control.

enumerator kCLOCK\_GpuPLLpwrBypassCtrl  
CCM Gpu PLL bypass Control.

enumerator kCLOCK\_VpuPllPwrBypassCtrl  
CCM Vpu PLL bypass Control.

enumerator kCLOCK\_ArmPllPwrBypassCtrl  
CCM Arm PLL bypass Control.

enumerator kCLOCK\_SysPll1InternalPll1BypassCtrl  
CCM System PLL1 internal pll1 bypass Control.

enumerator kCLOCK\_SysPll1InternalPll2BypassCtrl  
CCM System PLL1 internal pll2 bypass Control.

enumerator kCLOCK\_SysPll2InternalPll1BypassCtrl  
CCM Analog System PLL1 internal pll1 bypass Control.

enumerator kCLOCK\_SysPll2InternalPll2BypassCtrl  
CCM Analog VIDEO System PLL1 internal pll1 bypass Control.

enumerator kCLOCK\_SysPll3InternalPll1BypassCtrl  
CCM Analog VIDEO PLL bypass Control.

enumerator kCLOCK\_SysPll3InternalPll2BypassCtrl  
CCM Analog VIDEO PLL bypass Control.

enumerator kCLOCK\_VideoPll2InternalPll1BypassCtrl  
CCM Analog 480M PLL bypass Control.

enumerator kCLOCK\_VideoPll2InternalPll2BypassCtrl  
CCM Analog 480M PLL bypass Control.

enumerator kCLOCK\_DramPllInternalPll1BypassCtrl  
CCM Analog 480M PLL bypass Control.

enumerator kCLOCK\_DramPllInternalPll2BypassCtrl  
CCM Analog 480M PLL bypass Control.

enum \_ccm\_analog\_pll\_clke

PLL clock names for clock enable/disable settings.

These constants define the PLL clock names for PLL clock enable/disable operations.

- 0:15: REG offset to CCM\_ANALOG\_BASE in bytes.
- 16:20: Clock enable bit shift.

*Values:*

enumerator kCLOCK\_AudioPll1Clke  
Audio pll1 clke

enumerator kCLOCK\_AudioPll2Clke  
Audio pll2 clke

enumerator kCLOCK\_VideoPll1Clke  
Video pll1 clke

enumerator kCLOCK\_GpuPllClke  
Gpu pll clke

enumerator kCLOCK\_VpuPllClke  
Vpu pll clke

enumerator kCLOCK\_ArmPllClke  
Arm pll clke

enumerator kCLOCK\_SystemPll1Clke  
System pll1 clke

enumerator kCLOCK\_SystemPll1Div2Clke  
System pll1 Div2 clke

enumerator kCLOCK\_SystemPll1Div3Clke  
System pll1 Div3 clke

enumerator kCLOCK\_SystemPll1Div4Clke  
System pll1 Div4 clke

enumerator kCLOCK\_SystemPll1Div5Clke  
System pll1 Div5 clke

enumerator kCLOCK\_SystemPll1Div6Clke  
System pll1 Div6 clke

enumerator kCLOCK\_SystemPll1Div8Clke  
System pll1 Div8 clke

enumerator kCLOCK\_SystemPll1Div10Clke  
System pll1 Div10 clke

enumerator kCLOCK\_SystemPll1Div20Clke  
System pll1 Div20 clke

enumerator kCLOCK\_SystemPll2Clke  
System pll2 clke

enumerator kCLOCK\_SystemPll2Div2Clke  
System pll2 Div2 clke

enumerator kCLOCK\_SystemPll2Div3Clke  
System pll2 Div3 clke

enumerator kCLOCK\_SystemPll2Div4Clke  
System pll2 Div4 clke

enumerator kCLOCK\_SystemPll2Div5Clke  
System pll2 Div5 clke

enumerator kCLOCK\_SystemPll2Div6Clke  
System pll2 Div6 clke

enumerator kCLOCK\_SystemPll2Div8Clke  
System pll2 Div8 clke

enumerator kCLOCK\_SystemPll2Div10Clke  
System pll2 Div10 clke

enumerator kCLOCK\_SystemPll2Div20Clke

System pll2 Div20 clke

enumerator kCLOCK\_SystemPll3Clke

System pll3 clke

enumerator kCLOCK\_VideoPll2Clke

Video pll2 clke

enumerator kCLOCK\_DramPllClke

Dram pll clke

enumerator kCLOCK\_OSC25MClke

OSC25M clke

enumerator kCLOCK\_OSC27MClke

OSC27M clke

enum \_clock\_pll\_ctrl

ANALOG Power down override control.

*Values:*

enumerator kCLOCK\_AudioPll1Ctrl

enumerator kCLOCK\_AudioPll2Ctrl

enumerator kCLOCK\_VideoPll1Ctrl

enumerator kCLOCK\_GpuPllCtrl

enumerator kCLOCK\_VpuPllCtrl

enumerator kCLOCK\_ArmPllCtrl

enumerator kCLOCK\_SystemPll1Ctrl

enumerator kCLOCK\_SystemPll2Ctrl

enumerator kCLOCK\_SystemPll3Ctrl

enumerator kCLOCK\_VideoPll2Ctrl

enumerator kCLOCK\_DramPllCtrl

enum \_osc\_mode

OSC work mode.

*Values:*

enumerator kOSC\_OscMode

OSC oscillator mode

enumerator kOSC\_ExtMode

OSC external mode

enum \_osc32\_src

OSC 32K input select.

*Values:*

enumerator kOSC32\_Src25MDiv800

source from 25M divide 800

enumerator kOSC32\_SrcRTC  
source from RTC

enum \_ccm\_analog\_pll\_ref\_clk  
PLL reference clock select.

*Values:*

enumerator kANALOG\_PllRefOsc25M  
reference OSC 25M

enumerator kANALOG\_PllRefOsc27M  
reference OSC 27M

enumerator kANALOG\_PllRefOscHdmiPhy27M  
reference HDMI PHY 27M

enumerator kANALOG\_PllRefClkPN  
reference CLK\_P\_N

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

typedef enum \_clock\_ip\_name clock\_ip\_name\_t  
CCM CCGR gate control.

typedef enum \_clock\_root\_control clock\_root\_control\_t  
ccm root name used to get clock frequency.

typedef enum \_clock\_root clock\_root\_t  
ccm clock root used to get clock frequency.

typedef enum \_clock\_rootmux\_m4\_clk\_sel clock\_rootmux\_m4\_clk\_sel\_t  
Root clock select enumeration for ARM Cortex-M4 core.

typedef enum \_clock\_rootmux\_axi\_clk\_sel clock\_rootmux\_axi\_clk\_sel\_t  
Root clock select enumeration for AXI bus.

typedef enum \_clock\_rootmux\_ahb\_clk\_sel clock\_rootmux\_ahb\_clk\_sel\_t  
Root clock select enumeration for AHB bus.

typedef enum \_clock\_rootmux\_qspi\_clk\_sel clock\_rootmux\_qspi\_clk\_sel\_t  
Root clock select enumeration for QSPI peripheral.

typedef enum \_clock\_rootmux\_ecspi\_clk\_sel clock\_rootmux\_ecspi\_clk\_sel\_t  
Root clock select enumeration for ECSPI peripheral.

typedef enum \_clock\_rootmux\_i2c\_clk\_sel clock\_rootmux\_i2c\_clk\_sel\_t  
Root clock select enumeration for I2C peripheral.

typedef enum \_clock\_rootmux\_uart\_clk\_sel clock\_rootmux\_uart\_clk\_sel\_t  
Root clock select enumeration for UART peripheral.

typedef enum \_clock\_rootmux\_gpt clock\_rootmux\_gpt\_t  
Root clock select enumeration for GPT peripheral.

typedef enum \_clock\_rootmux\_wdog\_clk\_sel clock\_rootmux\_wdog\_clk\_sel\_t  
Root clock select enumeration for WDOG peripheral.

typedef enum \_clock\_rootmux\_pwm\_clk\_sel clock\_rootmux\_pwm\_clk\_sel\_t  
Root clock select enumeration for PWM peripheral.

`typedef enum _clock_rootmux_sai_clk_sel clock_rootmux_sai_clk_sel_t`

Root clock select enumeration for SAI peripheral.

`typedef enum _clock_rootmux_noc_clk_sel clock_rootmux_noc_clk_sel_t`

Root clock select enumeration for NOC CLK.

`typedef enum _clock_pll_gate clock_pll_gate_t`

CCM PLL gate control.

`typedef enum _clock_gate_value clock_gate_value_t`

CCM gate control value.

`typedef enum _clock_pll_bypass_ctrl clock_pll_bypass_ctrl_t`

PLL control names for PLL bypass.

These constants define the PLL control names for PLL bypass.

- 0:15: REG offset to CCM\_ANALOG\_BASE in bytes.
- 16:20: bypass bit shift.

`typedef enum _ccm_analog_pll_clke clock_pll_clke_t`

PLL clock names for clock enable/disable settings.

These constants define the PLL clock names for PLL clock enable/disable operations.

- 0:15: REG offset to CCM\_ANALOG\_BASE in bytes.
- 16:20: Clock enable bit shift.

`typedef enum _clock_pll_ctrl clock_pll_ctrl_t`

ANALOG Power down override control.

`typedef enum _osc32_src osc32_src_t`

OSC 32K input select.

`typedef struct _osc_config osc_config_t`

OSC configuration structure.

`typedef struct _ccm_analog_frac_pll_config ccm_analog_frac_pll_config_t`

Fractional-N PLL configuration. Note: all the dividers in this configuration structure are the actually divider, software will map it to register value.

`typedef struct _ccm_analog_sscg_pll_config ccm_analog_sscg_pll_config_t`

SSCG PLL configuration. Note: all the dividers in this configuration structure are the actually divider, software will map it to register value.

`FSL_CLOCK_DRIVER_VERSION`

CLOCK driver version 2.4.1.

`SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY`

`OSC25M_CLK_FREQ`

XTAL 25M clock frequency.

`OSC27M_CLK_FREQ`

XTAL 27M clock frequency.

`HDMI_PHY_27M_FREQ`

HDMI PHY 27M clock frequency.

`CLKPN_FREQ`

clock1PN frequency.

ECSPI\_CLOCKS

Clock ip name array for ECSPI.

GPIO\_CLOCKS

Clock ip name array for GPIO.

GPT\_CLOCKS

Clock ip name array for GPT.

I2C\_CLOCKS

Clock ip name array for I2C.

IOMUX\_CLOCKS

Clock ip name array for IOMUX.

IPMUX\_CLOCKS

Clock ip name array for IPMUX.

PWM\_CLOCKS

Clock ip name array for PWM.

RDC\_CLOCKS

Clock ip name array for RDC.

SAI\_CLOCKS

Clock ip name array for SAI.

RDC\_SEMA42\_CLOCKS

Clock ip name array for RDC SEMA42.

UART\_CLOCKS

Clock ip name array for UART.

USDHC\_CLOCKS

Clock ip name array for USDHC.

WDOG\_CLOCKS

Clock ip name array for WDOG.

TMU\_CLOCKS

Clock ip name array for TEMPSENSOR.

SDMA\_CLOCKS

Clock ip name array for SDMA.

MU\_CLOCKS

Clock ip name array for MU.

QSPI\_CLOCKS

Clock ip name array for QSPI.

CCM\_BIT\_FIELD\_EXTRACTION(val, mask, shift)

CCM reg macros to extract corresponding registers bit field.

CCM\_REG\_OFF(root, off)

CCM reg macros to map corresponding registers.

CCM\_REG(root)

CCM\_REG\_SET(root)

CCM\_REG\_CLR(root)

AUDIO\_PLL1\_CFG0\_OFFSET

CCM Analog registers offset.

AUDIO\_PLL2\_CFG0\_OFFSET

VIDEO\_PLL1\_CFG0\_OFFSET

GPU\_PLL\_CFG0\_OFFSET

VPU\_PLL\_CFG0\_OFFSET

ARM\_PLL\_CFG0\_OFFSET

SYS\_PLL1\_CFG0\_OFFSET

SYS\_PLL2\_CFG0\_OFFSET

SYS\_PLL3\_CFG0\_OFFSET

VIDEO\_PLL2\_CFG0\_OFFSET

DRAM\_PLL\_CFG0\_OFFSET

OSC\_MISC\_CFG\_OFFSET

CCM\_ANALOG\_TUPLE(reg, shift)

CCM ANALOG tuple macros to map corresponding registers and bit fields.

CCM\_ANALOG\_TUPLE\_SHIFT(tuple)

CCM\_ANALOG\_TUPLE\_REG\_OFF(base, tuple, off)

CCM\_ANALOG\_TUPLE\_REG(base, tuple)

CCM\_TUPLE(ccgr, root)

CCM CCGR and root tuple.

CCM\_TUPLE\_CCGR(tuple)

CCM\_TUPLE\_ROOT(tuple)

CLOCK\_ROOT\_SOURCE

clock root source

CLOCK\_ROOT\_CONTROL\_TUPLE

kCLOCK\_CoreSysClk

For compatible with other platforms without CCM.

CLOCK\_GetCoreSysClkFreq

For compatible with other platforms without CCM.

static inline void CLOCK\_SetRootMux(*clock\_root\_control\_t* rootClk, uint32\_t mux)

Set clock root mux. User maybe need to set more than one mux ROOT according to the clock tree description in the reference manual.

#### Parameters

- rootClk – Root clock control (see *clock\_root\_control\_t* enumeration).
- mux – Root mux value (see *\_ccm\_rootmux\_xxx* enumeration).



static inline uint32\_t CLOCK\_GetRootMux(*clock\_root\_control\_t* rootClk)

Get clock root mux. In order to get the clock source of root, user maybe need to get more than one ROOT's mux value to obtain the final clock source of root.

**Parameters**

- rootClk – Root clock control (see *clock\_root\_control\_t* enumeration).

**Returns**

Root mux value (see *\_ccm\_rootmux\_xxx* enumeration).

static inline void CLOCK\_EnableRoot(*clock\_root\_control\_t* rootClk)

Enable clock root.

**Parameters**

- rootClk – Root clock control (see *clock\_root\_control\_t* enumeration)

static inline void CLOCK\_DisableRoot(*clock\_root\_control\_t* rootClk)

Disable clock root.

**Parameters**

- rootClk – Root control (see *clock\_root\_control\_t* enumeration)

static inline bool CLOCK\_IsRootEnabled(*clock\_root\_control\_t* rootClk)

Check whether clock root is enabled.

**Parameters**

- rootClk – Root control (see *clock\_root\_control\_t* enumeration)

**Returns**

CCM root enabled or not.

- true: Clock root is enabled.
- false: Clock root is disabled.

void CLOCK\_UpdateRoot(*clock\_root\_control\_t* ccmRootClk, uint32\_t mux, uint32\_t pre, uint32\_t post)

Update clock root in one step, for dynamical clock switching Note: The PRE and POST dividers in this function are the actually divider, software will map it to register value.

**Parameters**

- ccmRootClk – Root control (see *clock\_root\_control\_t* enumeration)
- mux – root mux value (see *\_ccm\_rootmux\_xxx* enumeration)
- pre – Pre divider value (0-7, divider=n+1)
- post – Post divider value (0-63, divider=n+1)

void CLOCK\_SetRootDivider(*clock\_root\_control\_t* ccmRootClk, uint32\_t pre, uint32\_t post)

Set root clock divider Note: The PRE and POST dividers in this function are the actually divider, software will map it to register value.

**Parameters**

- ccmRootClk – Root control (see *clock\_root\_control\_t* enumeration)
- pre – Pre divider value (1-8)
- post – Post divider value (1-64)

static inline uint32\_t CLOCK\_GetRootPreDivider(*clock\_root\_control\_t* rootClk)

Get clock root PRE\_PODF. In order to get the clock source of root, user maybe need to get more than one ROOT's mux value to obtain the final clock source of root.

**Parameters**

- rootClk – Root clock name (see clock\_root\_control\_t enumeration).

**Returns**

Root Pre divider value.

static inline uint32\_t CLOCK\_\_GetRootPostDivider(*clock\_root\_control\_t* rootClk)

Get clock root POST\_PODF. In order to get the clock source of root, user maybe need to get more than one ROOT's mux value to obtain the final clock source of root.

**Parameters**

- rootClk – Root clock name (see clock\_root\_control\_t enumeration).

**Returns**

Root Post divider value.

void CLOCK\_\_InitOSC25M(const *osc\_config\_t* \*config)

OSC25M init.

**Parameters**

- config – osc configuration.

void CLOCK\_\_DeinitOSC25M(void)

OSC25M deinit.

void CLOCK\_\_InitOSC27M(const *osc\_config\_t* \*config)

OSC27M init.

**Parameters**

- config – osc configuration.

void CLOCK\_\_DeinitOSC27M(void)

OSC27M deinit.

static inline void CLOCK\_\_SwitchOSC32Src(*osc32\_src\_t* sel)

switch 32KHZ OSC input

**Parameters**

- sel – OSC32 input clock select

static inline void CLOCK\_\_ControlGate(uint32\_t ccmGate, *clock\_gate\_value\_t* control)

Set PLL or CCGR gate control.

**Parameters**

- ccmGate – Gate control (see clock\_pll\_gate\_t and clock\_ip\_name\_t enumeration)
- control – Gate control value (see clock\_gate\_value\_t)

void CLOCK\_\_EnableClock(*clock\_ip\_name\_t* ccmGate)

Enable CCGR clock gate and root clock gate for each module User should set specific gate for each module according to the description of the table of system clocks, gating and override in CCM chapter of reference manual. Take care of that one module may need to set more than one clock gate.

**Parameters**

- ccmGate – Gate control for each module (see clock\_ip\_name\_t enumeration).

void CLOCK\_DisableClock(*clock\_ip\_name\_t* ccmGate)

Disable CCGR clock gate for the each module User should set specific gate for each module according to the description of the table of system clocks, gating and override in CCM chapter of reference manual. Take care of that one module may need to set more than one clock gate.

#### Parameters

- ccmGate – Gate control for each module (see *clock\_ip\_name\_t* enumeration).

static inline void CLOCK\_PowerUpPll(CCM\_ANALOG\_Type \*base, *clock\_pll\_ctrl\_t* pllControl)  
Power up PLL.

#### Parameters

- base – CCM\_ANALOG base pointer.
- pllControl – PLL control name (see *clock\_pll\_ctrl\_t* enumeration)

static inline void CLOCK\_PowerDownPll(CCM\_ANALOG\_Type \*base, *clock\_pll\_ctrl\_t* pllControl)  
Power down PLL.

#### Parameters

- base – CCM\_ANALOG base pointer.
- pllControl – PLL control name (see *clock\_pll\_ctrl\_t* enumeration)

static inline void CLOCK\_SetPllBypass(CCM\_ANALOG\_Type \*base, *clock\_pll\_bypass\_ctrl\_t* pllControl, bool bypass)

PLL bypass setting.

#### Parameters

- base – CCM\_ANALOG base pointer.
- pllControl – PLL control name (see *ccm\_analog\_pll\_control\_t* enumeration)
- bypass – Bypass the PLL.
  - true: Bypass the PLL.
  - false: Do not bypass the PLL.

static inline bool CLOCK\_IsPllBypassed(CCM\_ANALOG\_Type \*base, *clock\_pll\_bypass\_ctrl\_t* pllControl)

Check if PLL is bypassed.

#### Parameters

- base – CCM\_ANALOG base pointer.
- pllControl – PLL control name (see *ccm\_analog\_pll\_control\_t* enumeration)

#### Returns

PLL bypass status.

- true: The PLL is bypassed.
- false: The PLL is not bypassed.

static inline bool CLOCK\_IsPllLocked(CCM\_ANALOG\_Type \*base, *clock\_pll\_ctrl\_t* pllControl)  
Check if PLL clock is locked.

#### Parameters

- base – CCM\_ANALOG base pointer.
- pllControl – PLL control name (see *clock\_pll\_ctrl\_t* enumeration)

**Returns**

PLL lock status.

- true: The PLL clock is locked.
- false: The PLL clock is not locked.

```
static inline void CLOCK_EnableAnalogClock(CCM_ANALOG_Type *base, clock_pll_clke_t  
                                           pllClock)
```

Enable PLL clock.

**Parameters**

- base – CCM\_ANALOG base pointer.
- pllClock – PLL clock name (see ccm\_analog\_pll\_clock\_t enumeration)

```
static inline void CLOCK_DisableAnalogClock(CCM_ANALOG_Type *base, clock_pll_clke_t  
                                             pllClock)
```

Disable PLL clock.

**Parameters**

- base – CCM\_ANALOG base pointer.
- pllClock – PLL clock name (see ccm\_analog\_pll\_clock\_t enumeration)

```
static inline void CLOCK_OverrideAnalogClke(CCM_ANALOG_Type *base, clock_pll_clke_t  
                                             ovClock, bool override)
```

Override PLL clock output enable.

**Parameters**

- base – CCM\_ANALOG base pointer.
- ovClock – PLL clock name (see clock\_pll\_clke\_t enumeration)
- override – Override the PLL.
  - true: Override the PLL clke, CCM will handle it.
  - false: Do not override the PLL clke.

```
static inline void CLOCK_OverridePllPd(CCM_ANALOG_Type *base, clock_pll_ctrl_t pdClock,  
                                       bool override)
```

Override PLL power down.

**Parameters**

- base – CCM\_ANALOG base pointer.
- pdClock – PLL clock name (see clock\_pll\_ctrl\_t enumeration)
- override – Override the PLL.
  - true: Override the PLL clke, CCM will handle it.
  - false: Do not override the PLL clke.

```
void CLOCK_InitArmPll(const ccm_analog_frac_pll_config_t *config)
```

Initializes the ANALOG ARM PLL.

---

**Note:** This function can't detect whether the Arm PLL has been enabled and used by some IPs.

---

**Parameters**

- `config` – Pointer to the configuration structure(see `ccm_analog_frac_pll_config_t` enumeration).

`void CLOCK_DeinitArmPll(void)`

De-initialize the ARM PLL.

`void CLOCK_InitSysPll1(const ccm_analog_sscg_pll_config_t *config)`

Initializes the ANALOG SYS PLL1.

---

**Note:** This function can't detect whether the SYS PLL has been enabled and used by some IPs.

---

#### Parameters

- `config` – Pointer to the configuration structure(see `ccm_analog_sscg_pll_config_t` enumeration).

`void CLOCK_DeinitSysPll1(void)`

De-initialize the System PLL1.

`void CLOCK_InitSysPll2(const ccm_analog_sscg_pll_config_t *config)`

Initializes the ANALOG SYS PLL2.

---

**Note:** This function can't detect whether the SYS PLL has been enabled and used by some IPs.

---

#### Parameters

- `config` – Pointer to the configuration structure(see `ccm_analog_sscg_pll_config_t` enumeration).

`void CLOCK_DeinitSysPll2(void)`

De-initialize the System PLL2.

`void CLOCK_InitSysPll3(const ccm_analog_sscg_pll_config_t *config)`

Initializes the ANALOG SYS PLL3.

---

**Note:** This function can't detect whether the SYS PLL has been enabled and used by some IPs.

---

#### Parameters

- `config` – Pointer to the configuration structure(see `ccm_analog_sscg_pll_config_t` enumeration).

`void CLOCK_DeinitSysPll3(void)`

De-initialize the System PLL3.

`void CLOCK_InitDramPll(const ccm_analog_sscg_pll_config_t *config)`

Initializes the ANALOG DDR PLL.

---

**Note:** This function can't detect whether the DDR PLL has been enabled and used by some IPs.

---

#### Parameters

- `config` – Pointer to the configuration structure(see `ccm_analog_sscg_pll_config_t` enumeration).

`void CLOCK_DeinitDramPll(void)`

De-initialize the Dram PLL.

`void CLOCK_InitAudioPll1(const ccm_analog_frac_pll_config_t *config)`

Initializes the ANALOG AUDIO PLL1.

---

**Note:** This function can't detect whether the AUDIO PLL has been enabled and used by some IPs.

---

#### Parameters

- `config` – Pointer to the configuration structure(see `ccm_analog_frac_pll_config_t` enumeration).

`void CLOCK_DeinitAudioPll1(void)`

De-initialize the Audio PLL1.

`void CLOCK_InitAudioPll2(const ccm_analog_frac_pll_config_t *config)`

Initializes the ANALOG AUDIO PLL2.

---

**Note:** This function can't detect whether the AUDIO PLL has been enabled and used by some IPs.

---

#### Parameters

- `config` – Pointer to the configuration structure(see `ccm_analog_frac_pll_config_t` enumeration).

`void CLOCK_DeinitAudioPll2(void)`

De-initialize the Audio PLL2.

`void CLOCK_InitVideoPll1(const ccm_analog_frac_pll_config_t *config)`

Initializes the ANALOG VIDEO PLL1.

#### Parameters

- `config` – Pointer to the configuration structure(see `ccm_analog_frac_pll_config_t` enumeration).

`void CLOCK_DeinitVideoPll1(void)`

De-initialize the Video PLL1.

`void CLOCK_InitVideoPll2(const ccm_analog_sscg_pll_config_t *config)`

Initializes the ANALOG VIDEO PLL2.

---

**Note:** This function can't detect whether the VIDEO PLL has been enabled and used by some IPs.

---

#### Parameters

- `config` – Pointer to the configuration structure(see `ccm_analog_sscg_pll_config_t` enumeration).

`void CLOCK_DeinitVideoPll2(void)`

De-initialize the Video PLL2.

```
void CLOCK_InitSSCGPll(CCM_ANALOG_Type *base, const ccm_analog_sscg_pll_config_t
                      *config, clock_pll_ctrl_t type)
```

Initializes the ANALOG SSCG PLL.

#### Parameters

- base – CCM ANALOG base address
- config – Pointer to the configuration structure(see *ccm\_analog\_sscg\_pll\_config\_t* enumeration).
- type – sscg pll type

```
uint32_t CLOCK_GetSSCGPllFreq(CCM_ANALOG_Type *base, clock_pll_ctrl_t type, uint32_t
                             refClkFreq, bool pll1Bypass)
```

Get the ANALOG SSCG PLL clock frequency.

#### Parameters

- base – CCM ANALOG base address.
- type – sscg pll type
- refClkFreq – reference clock frequency
- pll1Bypass – pll1 bypass flag

#### Returns

Clock frequency

```
void CLOCK_InitFracPll(CCM_ANALOG_Type *base, const ccm_analog_frac_pll_config_t *config,
                      clock_pll_ctrl_t type)
```

Initializes the ANALOG Fractional PLL.

#### Parameters

- base – CCM ANALOG base address.
- config – Pointer to the configuration structure(see *ccm\_analog\_frac\_pll\_config\_t* enumeration).
- type – fractional pll type.

```
uint32_t CLOCK_GetFracPllFreq(CCM_ANALOG_Type *base, clock_pll_ctrl_t type, uint32_t
                             refClkFreq)
```

Gets the ANALOG Fractional PLL clock frequency.

#### Parameters

- base – CCM\_ANALOG base pointer.
- type – fractional pll type.
- refClkFreq – reference clock frequency

#### Returns

Clock frequency

```
uint32_t CLOCK_GetPllFreq(clock_pll_ctrl_t pll)
```

Gets PLL clock frequency.

#### Parameters

- pll – fractional pll type.

#### Returns

Clock frequency

uint32\_t CLOCK\_GetPllRefClkFreq(*clock\_pll\_ctrl\_t* ctrl)

Gets PLL reference clock frequency.

**Parameters**

- ctrl – fractional pll type.

**Returns**

Clock frequency

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

Gets the clock frequency for a specific clock name.

This function checks the current clock configurations and then calculates the clock frequency for a specific clock name defined in clock\_name\_t.

**Parameters**

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

**Returns**

Clock frequency value in hertz

uint32\_t CLOCK\_GetClockRootFreq(*clock\_root\_t* clockRoot)

Gets the frequency of selected clock root.

**Parameters**

- clockRoot – The clock root used to get the frequency, please refer to clock\_root\_t.

**Returns**

The frequency of selected clock root.

uint32\_t CLOCK\_GetCoreM4Freq(void)

Get the CCM Cortex M4 core frequency.

**Returns**

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

uint32\_t CLOCK\_GetAxiFreq(void)

Get the CCM Axi bus frequency.

**Returns**

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

uint32\_t CLOCK\_GetAhbFreq(void)

Get the CCM Ahb bus frequency.

**Returns**

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

uint8\_t oscMode

ext or osc mode

uint8\_t oscDiv

osc divider

uint8\_t refSel

pll reference clock sel

uint8\_t refDiv

A 6bit divider to make sure the REF must be within the range 10MHZ~300MHZ

uint32\_t fractionDiv

Inlcude fraction divider(divider:1:2^24) output clock range is 2000MHZ-4000MHZ



uint8\_t intDiv

uint8\_t outDiv

output clock divide, output clock range is 30MHZ to 2000MHZ, must be a even value

uint8\_t refSel

pll reference clock sel

uint8\_t refDiv1

A 3bit divider to make sure the REF must be within the range 25MHZ~235MHZ ,post\_divide REF must be within the range 25MHZ~54MHZ

uint8\_t refDiv2

A 6bit divider to make sure the post\_divide REF must be within the range 54MHZ~75MHZ

uint32\_t loopDivider1

A 6bit internal PLL1 feedback clock divider, output clock range must be within the range 1600MHZ-2400MHZ

uint32\_t loopDivider2

A 6bit internal PLL2 feedback clock divider, output clock range must be within the range 1200MHZ-2400MHZ

uint8\_t outDiv

A 6bit output clock divide, output clock range is 20MHZ to 1200MHZ

struct \_\_osc\_config

*#include <fsl\_clock.h>* OSC configuration structure.

struct \_\_ccm\_analog\_frac\_pll\_config

*#include <fsl\_clock.h>* Fractional-N PLL configuration. Note: all the dividers in this configuration structure are the actually divider, software will map it to register value.

struct \_\_ccm\_analog\_sscg\_pll\_config

*#include <fsl\_clock.h>* SSCG PLL configuration. Note: all the dividers in this configuration structure are the actually divider, software will map it to register value.

## 2.3 MIPI CSI2 RX: MIPI CSI2 RX Driver

FSL\_CSI2RX\_DRIVER\_VERSION

CSI2RX driver version.

enum \_\_csi2rx\_data\_lane

CSI2RX data lanes.

*Values:*

enumerator kCSI2RX\_DataLane0

Data lane 0.

enumerator kCSI2RX\_DataLane1

Data lane 1.

enumerator kCSI2RX\_DataLane2

Data lane 2.

enumerator kCSI2RX\_DataLane3

Data lane 3.

enum \_csi2rx\_payload

CSI2RX payload type.

*Values:*

enumerator kCSI2RX\_PayloadGroup0Null  
NULL.

enumerator kCSI2RX\_PayloadGroup0Blank  
Blank.

enumerator kCSI2RX\_PayloadGroup0Embedded  
Embedded.

enumerator kCSI2RX\_PayloadGroup0YUV420\_8Bit  
Legacy YUV420 8 bit.

enumerator kCSI2RX\_PayloadGroup0YUV422\_8Bit  
YUV422 8 bit.

enumerator kCSI2RX\_PayloadGroup0YUV422\_10Bit  
YUV422 10 bit.

enumerator kCSI2RX\_PayloadGroup0RGB444  
RGB444.

enumerator kCSI2RX\_PayloadGroup0RGB555  
RGB555.

enumerator kCSI2RX\_PayloadGroup0RGB565  
RGB565.

enumerator kCSI2RX\_PayloadGroup0RGB666  
RGB666.

enumerator kCSI2RX\_PayloadGroup0RGB888  
RGB888.

enumerator kCSI2RX\_PayloadGroup0Raw6  
Raw 6.

enumerator kCSI2RX\_PayloadGroup0Raw7  
Raw 7.

enumerator kCSI2RX\_PayloadGroup0Raw8  
Raw 8.

enumerator kCSI2RX\_PayloadGroup0Raw10  
Raw 10.

enumerator kCSI2RX\_PayloadGroup0Raw12  
Raw 12.

enumerator kCSI2RX\_PayloadGroup0Raw14  
Raw 14.

enumerator kCSI2RX\_PayloadGroup1UserDefined1  
User defined 8-bit data type 1, 0x30.

enumerator kCSI2RX\_PayloadGroup1UserDefined2  
User defined 8-bit data type 2, 0x31.

enumerator kCSI2RX\_PayloadGroup1UserDefined3

User defined 8-bit data type 3, 0x32.

enumerator kCSI2RX\_PayloadGroup1UserDefined4

User defined 8-bit data type 4, 0x33.

enumerator kCSI2RX\_PayloadGroup1UserDefined5

User defined 8-bit data type 5, 0x34.

enumerator kCSI2RX\_PayloadGroup1UserDefined6

User defined 8-bit data type 6, 0x35.

enumerator kCSI2RX\_PayloadGroup1UserDefined7

User defined 8-bit data type 7, 0x36.

enumerator kCSI2RX\_PayloadGroup1UserDefined8

User defined 8-bit data type 8, 0x37.

enum \_csi2rx\_bit\_error

MIPI CSI2RX bit errors.

*Values:*

enumerator kCSI2RX\_BitErrorEccTwoBit

ECC two bit error has occurred.

enumerator kCSI2RX\_BitErrorEccOneBit

ECC one bit error has occurred.

enum \_csi2rx\_ppi\_error

MIPI CSI2RX PPI error types.

*Values:*

enumerator kCSI2RX\_PpiErrorSotHs

CSI2RX DPHY PPI error ErrSotHS.

enumerator kCSI2RX\_PpiErrorSotSyncHs

CSI2RX DPHY PPI error ErrSotSync\_HS.

enumerator kCSI2RX\_PpiErrorEsc

CSI2RX DPHY PPI error ErrEsc.

enumerator kCSI2RX\_PpiErrorSyncEsc

CSI2RX DPHY PPI error ErrSyncEsc.

enumerator kCSI2RX\_PpiErrorControl

CSI2RX DPHY PPI error ErrControl.

enum \_csi2rx\_interrupt

MIPI CSI2RX interrupt.

*Values:*

enumerator kCSI2RX\_InterruptCrcError

enumerator kCSI2RX\_InterruptEccOneBitError

enumerator kCSI2RX\_InterruptEccTwoBitError

enumerator kCSI2RX\_InterruptUlpsStatusChange

enumerator kCSI2RX\_InterruptErrorSotHs

enumerator kCSI2RX\_InterruptErrorSotSyncHs

enumerator kCSI2RX\_InterruptErrorEsc

enumerator kCSI2RX\_InterruptErrorSyncEsc

enumerator kCSI2RX\_InterruptErrorControl

enum \_csi2rx\_ulps\_status

MIPI CSI2RX D-PHY ULPS state.

*Values:*

enumerator kCSI2RX\_ClockLaneUlps

Clock lane is in ULPS state.

enumerator kCSI2RX\_DataLane0Ulps

Data lane 0 is in ULPS state.

enumerator kCSI2RX\_DataLane1Ulps

Data lane 1 is in ULPS state.

enumerator kCSI2RX\_DataLane2Ulps

Data lane 2 is in ULPS state.

enumerator kCSI2RX\_DataLane3Ulps

Data lane 3 is in ULPS state.

enumerator kCSI2RX\_ClockLaneMark

Clock lane is in mark state.

enumerator kCSI2RX\_DataLane0Mark

Data lane 0 is in mark state.

enumerator kCSI2RX\_DataLane1Mark

Data lane 1 is in mark state.

enumerator kCSI2RX\_DataLane2Mark

Data lane 2 is in mark state.

enumerator kCSI2RX\_DataLane3Mark

Data lane 3 is in mark state.

typedef struct \_csi2rx\_config csi2rx\_config\_t

CSI2RX configuration.

typedef enum \_csi2rx\_ppi\_error csi2rx\_ppi\_error\_t

MIPI CSI2RX PPI error types.

void CSI2RX\_Init(MIPI\_CSI2RX\_Type \*base, const csi2rx\_config\_t \*config)

Enables and configures the CSI2RX peripheral module.

#### Parameters

- base – CSI2RX peripheral address.
- config – CSI2RX module configuration structure.

void CSI2RX\_Deinit(MIPI\_CSI2RX\_Type \*base)

Disables the CSI2RX peripheral module.

#### Parameters

- base – CSI2RX peripheral address.

static inline uint32\_t CSI2RX\_GetBitError(MIPI\_CSI2RX\_Type \*base)

Gets the MIPI CSI2RX bit error status.

This function gets the RX bit error status, the return value could be compared with `_csi2rx_bit_error`. If one bit ECC error detected, the return value could be passed to the function `CSI2RX_GetEccBitErrorPosition` to get the position of the ECC error bit.

Example:

```
uint32_t bitError;
uint32_t bitErrorPosition;

bitError = CSI2RX_GetBitError(MIPI_CSI2RX);

if (kCSI2RX_BitErrorEccTwoBit & bitError)
{
    Two bits error;
}
else if (kCSI2RX_BitErrorEccOneBit & bitError)
{
    One bits error;
    bitErrorPosition = CSI2RX_GetEccBitErrorPosition(bitError);
}
```

#### Parameters

- `base` – CSI2RX peripheral address.

#### Returns

The RX bit error status.

static inline uint32\_t CSI2RX\_GetEccBitErrorPosition(uint32\_t bitError)

Get ECC one bit error bit position.

If `CSI2RX_GetBitError` detects ECC one bit error, this function could extract the error bit position from the return value of `CSI2RX_GetBitError`.

#### Parameters

- `bitError` – The bit error returned by `CSI2RX_GetBitError`.

#### Returns

The position of error bit.

static inline uint32\_t CSI2RX\_GetUlpsStatus(MIPI\_CSI2RX\_Type \*base)

Gets the MIPI CSI2RX D-PHY ULPS status.

Example to check whether data lane 0 is in ULPS status.

```
uint32_t status = CSI2RX_GetUlpsStatus(MIPI_CSI2RX);

if (kCSI2RX_DataLane0Ulps & status)
{
    Data lane 0 is in ULPS status.
}
```

#### Parameters

- `base` – CSI2RX peripheral address.

#### Returns

The MIPI CSI2RX D-PHY ULPS status, it is OR'ed value or `_csi2rx_ulps_status`.

```
static inline uint32_t CSI2RX_GetPpiErrorDataLanes(MIPI_CSI2RX_Type *base,  
                                                  csi2rx_ppi_error_t errorType)
```

Gets the MIPI CSI2RX D-PHY PPI error lanes.

This function checks the PPI error occurred on which data lanes, the returned value is OR'ed value of `csi2rx_ppi_error_t`. For example, if the ErrSotHS is detected, to check the ErrSotHS occurred on which data lanes, use like this:

```
uint32_t errorDataLanes = CSI2RX_GetPpiErrorDataLanes(MIPI_CSI2RX, kCSI2RX_  
↳PpiErrorSotHS);  
  
if (kCSI2RX_DataLane0 & errorDataLanes)  
{  
    ErrSotHS occurred on data lane 0.  
}  
  
if (kCSI2RX_DataLane1 & errorDataLanes)  
{  
    ErrSotHS occurred on data lane 1.  
}
```

### Parameters

- base – CSI2RX peripheral address.
- errorType – What kind of error to check.

### Returns

The data lane mask that error `errorType` occurred.

```
static inline void CSI2RX_EnableInterrupts(MIPI_CSI2RX_Type *base, uint32_t mask)
```

Enable the MIPI CSI2RX interrupts.

This function enables the MIPI CSI2RX interrupts. The interrupts to enable are passed in as an OR'ed value of `_csi2rx_interrupt`. For example, to enable one bit and two bit ECC error interrupts, use like this:

```
CSI2RX_EnableInterrupts(MIPI_CSI2RX, kCSI2RX_InterruptEccOneBitError | kCSI2RX_  
↳InterruptEccTwoBitError);
```

### Parameters

- base – CSI2RX peripheral address.
- mask – OR'ed value of `_csi2rx_interrupt`.

```
static inline void CSI2RX_DisableInterrupts(MIPI_CSI2RX_Type *base, uint32_t mask)
```

Disable the MIPI CSI2RX interrupts.

This function disables the MIPI CSI2RX interrupts. The interrupts to disable are passed in as an OR'ed value of `_csi2rx_interrupt`. For example, to disable one bit and two bit ECC error interrupts, use like this:

```
CSI2RX_DisableInterrupts(MIPI_CSI2RX, kCSI2RX_InterruptEccOneBitError | kCSI2RX_  
↳InterruptEccTwoBitError);
```

### Parameters

- base – CSI2RX peripheral address.
- mask – OR'ed value of `_csi2rx_interrupt`.

```
static inline uint32_t CSI2RX_GetInterruptStatus(MIPI_CSI2RX_Type *base)
```

Get the MIPI CSI2RX interrupt status.

This function returns the MIPI CSI2RX interrupts status as an OR'ed value of `_csi2rx_interrupt`.

#### Parameters

- `base` – CSI2RX peripheral address.

#### Returns

OR'ed value of `_csi2rx_interrupt`.

```
CSI2RX_REG_CFG_NUM_LANES(base)
```

```
CSI2RX_REG_CFG_DISABLE_DATA_LANES(base)
```

```
CSI2RX_REG_BIT_ERR(base)
```

```
CSI2RX_REG_IRQ_STATUS(base)
```

```
CSI2RX_REG_IRQ_MASK(base)
```

```
CSI2RX_REG_ULPS_STATUS(base)
```

```
CSI2RX_REG_PPI_ERRSOT_HS(base)
```

```
CSI2RX_REG_PPI_ERRSOTSYNC_HS(base)
```

```
CSI2RX_REG_PPI_ERRESC(base)
```

```
CSI2RX_REG_PPI_ERRSYNCESC(base)
```

```
CSI2RX_REG_PPI_ERRCONTROL(base)
```

```
CSI2RX_REG_CFG_DISABLE_PAYLOAD_0(base)
```

```
CSI2RX_REG_CFG_DISABLE_PAYLOAD_1(base)
```

```
CSI2RX_REG_CFG_IGNORE_VC(base)
```

```
CSI2RX_REG_CFG_VID_VC(base)
```

```
CSI2RX_REG_CFG_VID_P_FIFO_SEND_LEVEL(base)
```

```
CSI2RX_REG_CFG_VID_VSYNC(base)
```

```
CSI2RX_REG_CFG_VID_HSYNC_FP(base)
```

```
CSI2RX_REG_CFG_VID_HSYNC(base)
```

```
CSI2RX_REG_CFG_VID_HSYNC_BP(base)
```

```
MIPI_CSI2RX_CSI2RX_CFG_NUM_LANES_csi2rx_cfg_num_lanes_MASK
```

```
MIPI_CSI2RX_CSI2RX_IRQ_MASK_csi2rx_irq_mask_MASK
```

```
struct _csi2rx_config
```

*#include <fsl\_mipi\_csi2rx.h>* CSI2RX configuration.

### Public Members

uint8\_t laneNum

Number of active lanes used for receiving data.

uint8\_t tHsSettle\_EscClk

Number of rx\_clk\_esc clock periods for T\_HS\_SETTLE. The T\_HS\_SETTLE should be in the range of 85ns + 6UI to 145ns + 10UI.

## 2.4 ECSPI: Enhanced Configurable Serial Peripheral Interface Driver

### 2.5 ECSPI Driver

void ECSPI\_MasterGetDefaultConfig(*ecspi\_master\_config\_t* \*config)

Sets the ECSPI configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in ECSPI\_MasterInit(). User may use the initialized structure unchanged in ECSPI\_MasterInit, or modify some fields of the structure before calling ECSPI\_MasterInit. After calling this API, the master is ready to transfer. Example:

```
ecspi_master_config_t config;  
ECSPI_MasterGetDefaultConfig(&config);
```

#### Parameters

- config – pointer to config structure

void ECSPI\_MasterInit(ECSPI\_Type \*base, const *ecspi\_master\_config\_t* \*config, uint32\_t srcClock\_Hz)

Initializes the ECSPI with configuration.

The configuration structure can be filled by user from scratch, or be set with default values by ECSPI\_MasterGetDefaultConfig(). After calling this API, the slave is ready to transfer. Example

```
ecspi_master_config_t config = {  
    .baudRate_Bps = 400000,  
    ...  
};  
ECSPI_MasterInit(ECSPI0, &config);
```

#### Parameters

- base – ECSPI base pointer
- config – pointer to master configuration structure
- srcClock\_Hz – Source clock frequency.

void ECSPI\_SlaveGetDefaultConfig(*ecspi\_slave\_config\_t* \*config)

Sets the ECSPI configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in ECSPI\_SlaveInit(). User may use the initialized structure unchanged in ECSPI\_SlaveInit(), or modify some fields of the structure before calling ECSPI\_SlaveInit(). After calling this API, the master is ready to transfer. Example:



```
ecspi_Slaveconfig_t config;
ECSPI_SlaveGetDefaultConfig(&config);
```

### Parameters

- config – pointer to config structure

```
void ECSPI_SlaveInit(ECSPI_Type *base, const ecspi_slave_config_t *config)
```

Initializes the ECSPI with configuration.

The configuration structure can be filled by user from scratch, or be set with default values by ECSPI\_SlaveGetDefaultConfig(). After calling this API, the slave is ready to transfer.

Example

```
ecspi_Slaveconfig_t config = {
    .baudRate_Bps = 400000,
    ...
};
ECSPI_SlaveInit(ECSPI1, &config);
```

### Parameters

- base – ECSPI base pointer
- config – pointer to master configuration structure

```
void ECSPI_Deinit(ECSPI_Type *base)
```

De-initializes the ECSPI.

Calling this API resets the ECSPI module, gates the ECSPI clock. The ECSPI module can't work unless calling the ECSPI\_MasterInit/ECSPI\_SlaveInit to initialize module.

### Parameters

- base – ECSPI base pointer

```
static inline void ECSPI_Enable(ECSPI_Type *base, bool enable)
```

Enables or disables the ECSPI.

### Parameters

- base – ECSPI base pointer
- enable – pass true to enable module, false to disable module

```
static inline uint32_t ECSPI_GetStatusFlags(ECSPI_Type *base)
```

Gets the status flag.

### Parameters

- base – ECSPI base pointer

### Returns

ECSPI Status, use status flag to AND `_ecspi_flags` could get the related status.

```
static inline void ECSPI_ClearStatusFlags(ECSPI_Type *base, uint32_t mask)
```

Clear the status flag.

### Parameters

- base – ECSPI base pointer
- mask – ECSPI Status, use status flag to AND `_ecspi_flags` could get the related status.

```
static inline void ECSPI_EnableInterrupts(ECSPI_Type *base, uint32_t mask)
```

Enables the interrupt for the ECSPI.

#### Parameters

- base – ECSPI base pointer
- mask – ECSPI interrupt source. The parameter can be any combination of the following values:
  - kECSPI\_TxfifoEmptyInterruptEnable
  - kECSPI\_TxFifoDataRequestInterruptEnable
  - kECSPI\_TxFifoFullInterruptEnable
  - kECSPI\_RxFifoReadyInterruptEnable
  - kECSPI\_RxFifoDataRequestInterruptEnable
  - kECSPI\_RxFifoFullInterruptEnable
  - kECSPI\_RxFifoOverflowInterruptEnable
  - kECSPI\_TransferCompleteInterruptEnable
  - kECSPI\_AllInterruptEnable

```
static inline void ECSPI_DisableInterrupts(ECSPI_Type *base, uint32_t mask)
```

Disables the interrupt for the ECSPI.

#### Parameters

- base – ECSPI base pointer
- mask – ECSPI interrupt source. The parameter can be any combination of the following values:
  - kECSPI\_TxfifoEmptyInterruptEnable
  - kECSPI\_TxFifoDataRequestInterruptEnable
  - kECSPI\_TxFifoFullInterruptEnable
  - kECSPI\_RxFifoReadyInterruptEnable
  - kECSPI\_RxFifoDataRequestInterruptEnable
  - kECSPI\_RxFifoFullInterruptEnable
  - kECSPI\_RxFifoOverflowInterruptEnable
  - kECSPI\_TransferCompleteInterruptEnable
  - kECSPI\_AllInterruptEnable

```
static inline void ECSPI_SoftwareReset(ECSPI_Type *base)
```

Software reset.

#### Parameters

- base – ECSPI base pointer

```
static inline bool ECSPI_IsMaster(ECSPI_Type *base, ecspi_channel_source_t channel)
```

Mode check.

#### Parameters

- base – ECSPI base pointer
- channel – ECSPI channel source

#### Returns

mode of channel

```
static inline void ECSPI_EnableDMA(ECSPI_Type *base, uint32_t mask, bool enable)
```

Enables the DMA source for ECSPI.

#### Parameters

- base – ECSPI base pointer
- mask – ECSPI DMA source. The parameter can be any of the following values:
  - kECSPI\_TxDmaEnable
  - kECSPI\_RxDmaEnable
  - kECSPI\_DmaAllEnable
- enable – True means enable DMA, false means disable DMA

```
static inline uint8_t ECSPI_GetTxFifoCount(ECSPI_Type *base)
```

Get the Tx FIFO data count.

#### Parameters

- base – ECSPI base pointer.

#### Returns

the number of words in Tx FIFO buffer.

```
static inline uint8_t ECSPI_GetRxFifoCount(ECSPI_Type *base)
```

Get the Rx FIFO data count.

#### Parameters

- base – ECSPI base pointer.

#### Returns

the number of words in Rx FIFO buffer.

```
static inline void ECSPI_SetChannelSelect(ECSPI_Type *base, ecspi_channel_source_t channel)
```

Set channel select for transfer.

#### Parameters

- base – ECSPI base pointer
- channel – Channel source.

```
void ECSPI_SetChannelConfig(ECSPI_Type *base, ecspi_channel_source_t channel, const  
                           ecspi_channel_config_t *config)
```

Set channel select configuration for transfer.

The purpose of this API is to set the channel will be use to transfer. User may use this API after instance has been initialized or before transfer start. The configuration structure *ecspi\_channel\_config* can be filled by user from scratch. After calling this API, user can select this channel as transfer channel.

#### Parameters

- base – ECSPI base pointer
- channel – Channel source.
- config – Configuration struct of channel

```
void ECSPI_SetBaudRate(ECSPI_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
```

Sets the baud rate for ECSPI transfer. This is only used in master.

#### Parameters

- base – ECSPI base pointer

- baudRate\_Bps – baud rate needed in Hz.
- srcClock\_Hz – ECSPI source clock frequency in Hz.

*status\_t* ECSPI\_WriteBlocking(ECSPI\_Type \*base, const uint32\_t \*buffer, size\_t size)

Sends a buffer of data bytes using a blocking method.

---

**Note:** This function blocks via polling until all bytes have been sent.

---

#### Parameters

- base – ECSPI base pointer
- buffer – The data bytes to send
- size – The number of data bytes to send

#### Return values

- kStatus\_Success – Successfully start a transfer.
- kStatus\_ECSPI\_Timeout – The transfer timed out and was aborted.

static inline void ECSPI\_WriteData(ECSPI\_Type \*base, uint32\_t data)

Writes a data into the ECSPI data register.

#### Parameters

- base – ECSPI base pointer
- data – Data needs to be write.

static inline uint32\_t ECSPI\_ReadData(ECSPI\_Type \*base)

Gets a data from the ECSPI data register.

#### Parameters

- base – ECSPI base pointer

#### Returns

Data in the register.

void ECSPI\_MasterTransferCreateHandle(ECSPI\_Type \*base, *ecspi\_master\_handle\_t* \*handle, *ecspi\_master\_callback\_t* callback, void \*userData)

Initializes the ECSPI master handle.

This function initializes the ECSPI master handle which can be used for other ECSPI master transactional APIs. Usually, for a specified ECSPI instance, call this API once to get the initialized handle.

#### Parameters

- base – ECSPI peripheral base address.
- handle – ECSPI handle pointer.
- callback – Callback function.
- userData – User data.

*status\_t* ECSPI\_MasterTransferBlocking(ECSPI\_Type \*base, *ecspi\_transfer\_t* \*xfer)

Transfers a block of data using a polling method.

#### Parameters

- base – SPI base pointer
- xfer – pointer to spi\_xfer\_config\_t structure

**Return values**

- `kStatus_Success` – Successfully start a transfer.
- `kStatus_InvalidArgument` – Input argument is invalid.
- `kStatus_ECSPI_Timeout` – The transfer timed out and was aborted.

*status\_t* `ECSPI_MasterTransferNonBlocking(ECSPI_Type *base, ecspi_master_handle_t *handle, ecspi_transfer_t *xfer)`

Performs a non-blocking ECSPI interrupt transfer.

---

**Note:** The API immediately returns after transfer initialization is finished.

---

---

**Note:** If ECSPI transfer data frame size is 16 bits, the transfer size cannot be an odd number.

---

**Parameters**

- `base` – ECSPI peripheral base address.
- `handle` – pointer to `ecspi_master_handle_t` structure which stores the transfer state
- `xfer` – pointer to `ecspi_transfer_t` structure

**Return values**

- `kStatus_Success` – Successfully start a transfer.
- `kStatus_InvalidArgument` – Input argument is invalid.
- `kStatus_ECSPI_Busy` – ECSPI is not idle, is running another transfer.

*status\_t* `ECSPI_MasterTransferGetCount(ECSPI_Type *base, ecspi_master_handle_t *handle, size_t *count)`

Gets the bytes of the ECSPI interrupt transferred.

**Parameters**

- `base` – ECSPI peripheral base address.
- `handle` – Pointer to ECSPI transfer handle, this should be a static variable.
- `count` – Transferred bytes of ECSPI master.

**Return values**

- `kStatus_ECSPI_Success` – Succeed get the transfer count.
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

`void` `ECSPI_MasterTransferAbort(ECSPI_Type *base, ecspi_master_handle_t *handle)`

Aborts an ECSPI transfer using interrupt.

**Parameters**

- `base` – ECSPI peripheral base address.
- `handle` – Pointer to ECSPI transfer handle, this should be a static variable.

`void` `ECSPI_MasterTransferHandleIRQ(ECSPI_Type *base, ecspi_master_handle_t *handle)`

Interrupts the handler for the ECSPI.

**Parameters**

- `base` – ECSPI peripheral base address.

- `handle` – pointer to `ecspi_master_handle_t` structure which stores the transfer state.

```
void ECSPISlaveTransferCreateHandle(ECSPI_Type *base, ecspi_slave_handle_t *handle,  
                                   ecspi_slave_callback_t callback, void *userData)
```

Initializes the ECSPISlave handle.

This function initializes the ECSPISlave handle which can be used for other ECSPISlave transactional APIs. Usually, for a specified ECSPISlave instance, call this API once to get the initialized handle.

#### Parameters

- `base` – ECSPISlave peripheral base address.
- `handle` – ECSPISlave handle pointer.
- `callback` – Callback function.
- `userData` – User data.

```
static inline status_t ECSPISlaveTransferNonBlocking(ECSPI_Type *base, ecspi_slave_handle_t  
                                                    *handle, ecspi_transfer_t *xfer)
```

Performs a non-blocking ECSPISlave interrupt transfer.

---

**Note:** The API returns immediately after the transfer initialization is finished.

---

#### Parameters

- `base` – ECSPISlave peripheral base address.
- `handle` – pointer to `ecspi_master_handle_t` structure which stores the transfer state
- `xfer` – pointer to `ecspi_transfer_t` structure

#### Return values

- `kStatus_Success` – Successfully start a transfer.
- `kStatus_InvalidArgument` – Input argument is invalid.
- `kStatus_ECSPISlave_Busy` – ECSPISlave is not idle, is running another transfer.

```
static inline status_t ECSPISlaveTransferGetCount(ECSPI_Type *base, ecspi_slave_handle_t  
                                                    *handle, size_t *count)
```

Gets the bytes of the ECSPISlave interrupt transferred.

#### Parameters

- `base` – ECSPISlave peripheral base address.
- `handle` – Pointer to ECSPISlave transfer handle, this should be a static variable.
- `count` – Transferred bytes of ECSPISlave.

#### Return values

- `kStatus_ECSPISlave_Success` – Succeed get the transfer count.
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

```
static inline void ECSPISlaveTransferAbort(ECSPI_Type *base, ecspi_slave_handle_t *handle)
```

Aborts an ECSPISlave transfer using interrupt.

#### Parameters

- base – ECSPI peripheral base address.
- handle – Pointer to ECSPI transfer handle, this should be a static variable.

void ECSPI\_SlaveTransferHandleIRQ(ECSPI\_Type \*base, *ecspi\_slave\_handle\_t* \*handle)

Interrupts a handler for the ECSPI slave.

#### Parameters

- base – ECSPI peripheral base address.
- handle – pointer to *ecspi\_slave\_handle\_t* structure which stores the transfer state

FSL\_ECSPi\_DRIVER\_VERSION

ECSPI driver version.

Return status for the ECSPI driver.

*Values:*

enumerator kStatus\_ECSPi\_Busy  
ECSPI bus is busy

enumerator kStatus\_ECSPi\_Idle  
ECSPI is idle

enumerator kStatus\_ECSPi\_Error  
ECSPI error

enumerator kStatus\_ECSPi\_HardwareOverflow  
ECSPI hardware overflow

enumerator kStatus\_ECSPi\_Timeout  
ECSPI timeout polling status flags.

enum *\_ecspi\_clock\_polarity*  
ECSPI clock polarity configuration.

*Values:*

enumerator kECSPi\_PolarityActiveHigh  
Active-high ECSPI polarity high (idles low).

enumerator kECSPi\_PolarityActiveLow  
Active-low ECSPI polarity low (idles high).

enum *\_ecspi\_clock\_phase*  
ECSPI clock phase configuration.

*Values:*

enumerator kECSPi\_ClockPhaseFirstEdge  
First edge on SPCK occurs at the middle of the first cycle of a data transfer.

enumerator kECSPi\_ClockPhaseSecondEdge  
First edge on SPCK occurs at the start of the first cycle of a data transfer.

ECSPI interrupt sources.

*Values:*

enumerator kECSPi\_TxifoEmptyInterruptEnable  
Transmit FIFO buffer empty interrupt

enumerator kECSPI\_TxFifoDataRequestInterruptEnable  
Transmit FIFO data request interrupt

enumerator kECSPI\_TxFifoFullInterruptEnable  
Transmit FIFO full interrupt

enumerator kECSPI\_RxFifoReadyInterruptEnable  
Receiver FIFO ready interrupt

enumerator kECSPI\_RxFifoDataRequestInterruptEnable  
Receiver FIFO data request interrupt

enumerator kECSPI\_RxFifoFullInterruptEnable  
Receiver FIFO full interrupt

enumerator kECSPI\_RxFifoOverflowInterruptEnable  
Receiver FIFO buffer overflow interrupt

enumerator kECSPI\_TransferCompleteInterruptEnable  
Transfer complete interrupt

enumerator kECSPI\_AllInterruptEnable  
All interrupt

ECSPI status flags.

*Values:*

enumerator kECSPI\_TxfifoEmptyFlag  
Transmit FIFO buffer empty flag

enumerator kECSPI\_TxFifoDataRequestFlag  
Transmit FIFO data request flag

enumerator kECSPI\_TxFifoFullFlag  
Transmit FIFO full flag

enumerator kECSPI\_RxFifoReadyFlag  
Receiver FIFO ready flag

enumerator kECSPI\_RxFifoDataRequestFlag  
Receiver FIFO data request flag

enumerator kECSPI\_RxFifoFullFlag  
Receiver FIFO full flag

enumerator kECSPI\_RxFifoOverflowFlag  
Receiver FIFO buffer overflow flag

enumerator kECSPI\_TransferCompleteFlag  
Transfer complete flag

ECSPI DMA enable.

*Values:*

enumerator kECSPI\_TxDmaEnable  
Tx DMA request source

enumerator kECSPI\_RxDmaEnable  
Rx DMA request source



enumerator kECSPI\_DmaAllEnable

All DMA request source

enum \_ecspi\_data\_ready

ECSPI SPI\_RDY signal configuration.

*Values:*

enumerator kECSPI\_DataReadyIgnore

SPI\_RDY signal is ignored

enumerator kECSPI\_DataReadyFallingEdge

SPI\_RDY signal will be triggered by the falling edge

enumerator kECSPI\_DataReadyLowLevel

SPI\_RDY signal will be triggered by a low level

enum \_ecspi\_channel\_source

ECSPI channel select source.

*Values:*

enumerator kECSPI\_Channel0

Channel 0 is selected

enumerator kECSPI\_Channel1

Channel 1 is selected

enumerator kECSPI\_Channel2

Channel 2 is selected

enumerator kECSPI\_Channel3

Channel 3 is selected

enum \_ecspi\_master\_slave\_mode

ECSPI master or slave mode configuration.

*Values:*

enumerator kECSPI\_Slave

ECSPI peripheral operates in slave mode.

enumerator kECSPI\_Master

ECSPI peripheral operates in master mode.

enum \_ecspi\_data\_line\_inactive\_state\_t

ECSPI data line inactive state configuration.

*Values:*

enumerator kECSPI\_DataLineInactiveStateHigh

The data line inactive state stays high.

enumerator kECSPI\_DataLineInactiveStateLow

The data line inactive state stays low.

enum \_ecspi\_clock\_inactive\_state\_t

ECSPI clock inactive state configuration.

*Values:*

enumerator kECSPI\_ClockInactiveStateLow

The SCLK inactive state stays low.

enumerator kECSPI\_ClockInactiveStateHigh

The SCLK inactive state stays high.

enum \_ecspi\_chip\_select\_active\_state\_t

ECSPI active state configuration.

*Values:*

enumerator kECSPI\_ChipSelectActiveStateLow

The SS signal line active stays low.

enumerator kECSPI\_ChipSelectActiveStateHigh

The SS signal line active stays high.

enum \_ecspi\_sample\_period\_clock\_source

ECSPI sample period clock configuration.

*Values:*

enumerator kECSPI\_spiClock

The sample period clock source is SCLK.

enumerator kECSPI\_lowFreqClock

The sample period clock source is low\_frequency reference clock(32.768 kHz).

typedef enum \_ecspi\_clock\_polarity ecspi\_clock\_polarity\_t

ECSPI clock polarity configuration.

typedef enum \_ecspi\_clock\_phase ecspi\_clock\_phase\_t

ECSPI clock phase configuration.

typedef enum \_ecspi\_data\_ready ecspi\_Data\_ready\_t

ECSPI SPI\_RDY signal configuration.

typedef enum \_ecspi\_channel\_source ecspi\_channel\_source\_t

ECSPI channel select source.

typedef enum \_ecspi\_master\_slave\_mode ecspi\_master\_slave\_mode\_t

ECSPI master or slave mode configuration.

typedef enum \_ecspi\_data\_line\_inactive\_state\_t ecspi\_data\_line\_inactive\_state\_t

ECSPI data line inactive state configuration.

typedef enum \_ecspi\_clock\_inactive\_state\_t ecspi\_clock\_inactive\_state\_t

ECSPI clock inactive state configuration.

typedef enum \_ecspi\_chip\_select\_active\_state\_t ecspi\_chip\_select\_active\_state\_t

ECSPI active state configuration.

typedef enum \_ecspi\_sample\_period\_clock\_source ecspi\_sample\_period\_clock\_source\_t

ECSPI sample period clock configuration.

typedef struct \_ecspi\_channel\_config ecspi\_channel\_config\_t

ECSPI user channel configure structure.

typedef struct \_ecspi\_master\_config ecspi\_master\_config\_t

ECSPI master configure structure.

typedef struct \_ecspi\_slave\_config ecspi\_slave\_config\_t

ECSPI slave configure structure.

typedef struct \_ecspi\_transfer ecspi\_transfer\_t

ECSPI transfer structure.

```
typedef struct _ecspi_master_handle ecspi_master_handle_t
```

```
typedef ecspi_master_handle_t ecspi_slave_handle_t
```

Slave handle is the same with master handle

```
typedef void (*ecspi_master_callback_t)(ECSPI_Type *base, ecspi_master_handle_t *handle,
status_t status, void *userData)
```

ECSPI master callback for finished transmit.

```
typedef void (*ecspi_slave_callback_t)(ECSPI_Type *base, ecspi_slave_handle_t *handle, status_t
status, void *userData)
```

ECSPI slave callback for finished transmit.

```
uint32_t ECSPI_GetInstance(ECSPI_Type *base)
```

Get the instance for ECSPI module.

### Parameters

- base – ECSPI base address

```
ECSPI_DUMMYDATA
```

ECSPI dummy transfer data, the data is sent while txBuff is NULL.

```
SPI_RETRY_TIMES
```

Retry times for waiting flag.

```
struct _ecspi_channel_config
```

*#include <fsl\_ecspi.h>* ECSPI user channel configure structure.

### Public Members

*ecspi\_master\_slave\_mode\_t* channelMode

Channel mode

*ecspi\_clock\_inactive\_state\_t* clockInactiveState

Clock line (SCLK) inactive state

*ecspi\_data\_line\_inactive\_state\_t* dataLineInactiveState

Data line (MOSI&MISO) inactive state

*ecspi\_chip\_select\_active\_state\_t* chipSlectActiveState

Chip select(SS) line active state

*ecspi\_clock\_polarity\_t* polarity

Clock polarity

*ecspi\_clock\_phase\_t* phase

Clock phase

```
struct _ecspi_master_config
```

*#include <fsl\_ecspi.h>* ECSPI master configure structure.

### Public Members

*ecspi\_channel\_source\_t* channel

Channel number

*ecspi\_channel\_config\_t* channelConfig

Channel configuration

*ecspi\_sample\_period\_clock\_source\_t* samplePeriodClock  
Sample period clock source

uint16\_t burstLength  
Burst length. The length shall be less than 4096 bits

uint8\_t chipSelectDelay  
SS delay time

uint16\_t samplePeriod  
Sample period

uint8\_t txFifoThreshold  
TX Threshold

uint8\_t rxFifoThreshold  
RX Threshold

uint32\_t baudRate\_Bps  
ECSPI baud rate for master mode

bool enableLoopback  
Enable the ECSPI loopback test.

struct \_\_ecspi\_slave\_config  
*#include <fsl\_ecspi.h>* ECSPI slave configure structure.

### Public Members

uint16\_t burstLength  
Burst length. The length shall be less than 4096 bits

uint8\_t txFifoThreshold  
TX Threshold

uint8\_t rxFifoThreshold  
RX Threshold

*ecspi\_channel\_config\_t* channelConfig  
Channel configuration

struct \_\_ecspi\_transfer  
*#include <fsl\_ecspi.h>* ECSPI transfer structure.

### Public Members

const uint32\_t \*txData  
Send buffer

uint32\_t \*rxData  
Receive buffer

size\_t dataSize  
Transfer bytes

*ecspi\_channel\_source\_t* channel  
ECSPI channel select

struct \_\_ecspi\_master\_handle  
*#include <fsl\_ecspi.h>* ECSPI master handle structure.

**Public Members**

*ecspi\_channel\_source\_t* channel  
Channel number

const uint32\_t \*volatile txData  
Transfer buffer

uint32\_t \*volatile rxData  
Receive buffer

volatile size\_t txRemainingBytes  
Send data remaining in bytes

volatile size\_t rxRemainingBytes  
Receive data remaining in bytes

volatile uint32\_t state  
ECSPI internal state

size\_t transferSize  
Bytes to be transferred

*ecspi\_master\_callback\_t* callback  
ECSPI callback

void \*userData  
Callback parameter

## 2.6 ECSPI SDMA Driver

```
void ECSPi_MasterTransferCreateHandleSDMA(ECSPi_Type *base, ecspi_sdma_handle_t *handle,
ecspi_sdma_callback_t callback, void *userData,
sdma_handle_t *txHandle, sdma_handle_t
*rxHandle, uint32_t eventSourceTx, uint32_t
eventSourceRx, uint32_t TxChannel, uint32_t
RxChannel)
```

Initialize the ECSPI master SDMA handle.

This function initializes the ECSPI master SDMA handle which can be used for other SPI master transactional APIs. Usually, for a specified ECSPI instance, user need only call this API once to get the initialized handle.

**Parameters**

- base – ECSPI peripheral base address.
- handle – ECSPI handle pointer.
- callback – User callback function called at the end of a transfer.
- userData – User data for callback.
- txHandle – SDMA handle pointer for ECSPI Tx, the handle shall be static allocated by users.
- rxHandle – SDMA handle pointer for ECSPI Rx, the handle shall be static allocated by users.
- eventSourceTx – event source for ECSPI send, which can be found in SDMA mapping.

- eventSourceRx – event source for ECSPI receive, which can be found in SDMA mapping.
- TxChannel – SDMA channel for ECSPI send.
- RxChannel – SDMA channel for ECSPI receive.

```
void ECSPI_SlaveTransferCreateHandleSDMA(ECSPI_Type *base, ecspi_sdma_handle_t *handle,  
                                         ecspi_sdma_callback_t callback, void *userData,  
                                         sdma_handle_t *txHandle, sdma_handle_t  
                                         *rxHandle, uint32_t eventSourceTx, uint32_t  
                                         eventSourceRx, uint32_t TxChannel, uint32_t  
                                         RxChannel)
```

Initialize the ECSPI Slave SDMA handle.

This function initializes the ECSPI Slave SDMA handle which can be used for other SPI Slave transactional APIs. Usually, for a specified ECSPI instance, user need only call this API once to get the initialized handle.

#### Parameters

- base – ECSPI peripheral base address.
- handle – ECSPI handle pointer.
- callback – User callback function called at the end of a transfer.
- userData – User data for callback.
- txHandle – SDMA handle pointer for ECSPI Tx, the handle shall be static allocated by users.
- rxHandle – SDMA handle pointer for ECSPI Rx, the handle shall be static allocated by users.
- eventSourceTx – event source for ECSPI send, which can be found in SDMA mapping.
- eventSourceRx – event source for ECSPI receive, which can be found in SDMA mapping.
- TxChannel – SDMA channel for ECSPI send.
- RxChannel – SDMA channel for ECSPI receive.

```
status_t ECSPI_MasterTransferSDMA(ECSPI_Type *base, ecspi_sdma_handle_t *handle,  
                                   ecspi_transfer_t *xfer)
```

Perform a non-blocking ECSPI master transfer using SDMA.

---

**Note:** This interface returned immediately after transfer initiates.

---

#### Parameters

- base – ECSPI peripheral base address.
- handle – ECSPI SDMA handle pointer.
- xfer – Pointer to sdma transfer structure.

#### Return values

- kStatus\_Success – Successfully start a transfer.
- kStatus\_InvalidArgument – Input argument is invalid.
- kStatus\_ECSPI\_Busy – ECSPI is not idle, is running another transfer.

```
status_t ECSPI_SlaveTransferSDMA(ECSPI_Type *base, ecspi_sdma_handle_t *handle,  
                                ecspi_transfer_t *xfer)
```

Perform a non-blocking ECSPI slave transfer using SDMA.

---

**Note:** This interface returned immediately after transfer initiates.

---

#### Parameters

- base – ECSPI peripheral base address.
- handle – ECSPI SDMA handle pointer.
- xfer – Pointer to sdma transfer structure.

#### Return values

- kStatus\_Success – Successfully start a transfer.
- kStatus\_InvalidArgument – Input argument is invalid.
- kStatus\_ECSPI\_Busy – ECSPI is not idle, is running another transfer.

```
void ECSPI_MasterTransferAbortSDMA(ECSPI_Type *base, ecspi_sdma_handle_t *handle)
```

Abort a ECSPI master transfer using SDMA.

#### Parameters

- base – ECSPI peripheral base address.
- handle – ECSPI SDMA handle pointer.

```
void ECSPI_SlaveTransferAbortSDMA(ECSPI_Type *base, ecspi_sdma_handle_t *handle)
```

Abort a ECSPI slave transfer using SDMA.

#### Parameters

- base – ECSPI peripheral base address.
- handle – ECSPI SDMA handle pointer.

```
FSL_ECSPI_FREERTOS_DRIVER_VERSION
```

ECSPI FreeRTOS driver version.

```
typedef struct _ecspi_sdma_handle ecspi_sdma_handle_t
```

```
typedef void (*ecspi_sdma_callback_t)(ECSPI_Type *base, ecspi_sdma_handle_t *handle, status_t  
status, void *userData)
```

ECSPI SDMA callback called at the end of transfer.

```
struct _ecspi_sdma_handle
```

*#include <fsl\_ecspi\_sdma.h>* ECSPI SDMA transfer handle, users should not touch the content of the handle.

#### Public Members

```
bool txInProgress
```

Send transfer finished

```
bool rxInProgress
```

Receive transfer finished

```
sdma_handle_t *txSdmaHandle
```

DMA handler for ECSPI send

*sdma\_handle\_t* \*rxSdmaHandle  
DMA handler for ECSPI receive

*ecspi\_sdma\_callback\_t* callback  
Callback for ECSPI SDMA transfer

void \*userData  
User Data for ECSPI SDMA callback

uint32\_t state  
Internal state of ECSPI SDMA transfer

uint32\_t ChannelTx  
Channel for send handle

uint32\_t ChannelRx  
Channel for receive handler

## 2.7 GPC: General Power Controller Driver

static inline void GPC\_EnableMemoryGate(GPC\_Type \*base, uint32\_t modules, bool enable)  
Control power for memory.

### Parameters

- base – GPC peripheral base address.
- modules – Mask value for Modules to be operated, see to `_gpc_memory_power_gate`.
- enable – Enable the power or not.

void GPC\_EnablePartialSleepWakeupSource(GPC\_Type \*base, gpc\_wakeup\_source\_t source, bool enable)

Enable the modules as wakeup sources of PSLEEP (Partial Sleep) mode.

In PSLEEP mode, HP domain is powered down, while LP domain remains powered on so peripherals in LP domain can wakeup the system from PSLEEP mode via interrupts. In PSLEEP mode, system clocks are stopped and peripheral clocks of LP domain can be optionally on. LP domain peripherals can generate interrupt either asynchronously or need its peripheral clock on, depending on what kind of wakeup event is expected. Refer to the corresponding module description about what kind of interrupts are supported by the module.

### Parameters

- base – GPC peripheral base address.
- source – Wakeup source for responding module, see to `gpc_wakeup_source_t`.
- enable – Enable the wakeup source or not.

bool GPC\_GetPartialSleepWakeupFlag(GPC\_Type \*base, gpc\_wakeup\_source\_t source)

Get if indicated wakeup module just caused the wakeup event to exit PSLEEP mode.

This function returns if the responding wakeup module just caused the MCU to exit PSLEEP mode. In hardware level, the flags of wakeup source are read only and will be cleared by cleaning the interrupt status of the corresponding wakeup module.

### Parameters

- base – GPC peripheral base address.



- `source` – Wakeup source for responding module, see to `gpc_wakeup_source_t`.

**Return values**

- `true` – Indicated wakeup flag is asserted.
- `false` – Indicated wakeup flag is not asserted.

`static inline void GPC_EnablePartialSleepMode(GPC_Type *base, bool enable)`

Switch to the Partial Sleep mode.

This function controls if the system will enter Partial SLEEP mode or remain in STOP mode.

**Parameters**

- `base` – GPC peripheral base address.
- `enable` – Enable the gate or not.

`static inline void GPC_ConfigPowerUpSequence(GPC_Type *base, uint32_t sw, uint32_t sw2iso)`

Configure the power up sequence.

There will be two steps for power up sequence:

- After a power up request, GPC waits for a number of IP BUS clocks equal to the value of SW before turning on the power of HP domain. SW must not be programmed to zero.
- After GPC turning on the power of HP domain, it waits for a number of IP BUS clocks equal to the value of SW2ISO before disable the isolation of HP domain. SW2ISO must not be programmed to zero.

**Parameters**

- `base` – GPC peripheral base address.
- `sw` – Count of IP BUS clocks before disabling the isolation of HP domain.
- `sw2iso` – Count of IP BUS clocks before turning on the power of HP domain.

`static inline void GPC_ConfigPowerDownSequence(GPC_Type *base, uint32_t iso, uint32_t iso2w)`

Configure the power down sequence.

There will be two steps for power down sequence:

- After a power down request, the GPC waits for a number of IP BUS clocks equal to the value of ISO before it enables the isolation of HP domain. ISO must not be programmed to zero.
- After HP domain is isolated, GPC waits for a number of IPG BUS clocks equal to the value of ISO2SW before it turning off the power of HP domain. ISO2SW must not be programmed to zero.

**Parameters**

- `base` – GPC peripheral base address.
- `iso` – Count of IP BUS clocks before it enables the isolation of HP domain.
- `iso2w` – Count of IP BUS clocks before it turning off the power of HP domain.

`static inline uint32_t GPC_GetStatusFlags(GPC_Type *base)`

Get the status flags of GPC.

**Parameters**

- `base` – GPC peripheral base address.

**Returns**

Mask value of flags, see to `_gpc_status_flags`.

static inline void GPC\_ClearStatusFlags(GPC\_Type \*base, uint32\_t flags)

Clear the status flags of GPC.

**Parameters**

- base – GPC peripheral base address.
- flags – Mask value of flags to be cleared, see to `_gpc_status_flags`.

FSL\_GPC\_DRIVER\_VERSION

GPC driver version 2.0.0.

enum \_gpc\_memory\_power\_gate

Enumeration of the memory power gate control.

Once the clock gate is enabled, the responding part would be powered off and contents are not retained in Partial SLEEP mode.

*Values:*

enumerator kGPC\_MemoryPowerGateL2Cache

L2 Cache Power Gate.

enumerator kGPC\_MemoryPowerGateITCM

ITCM Power Gate Enable.

enumerator kGPC\_MemoryPowerGateDTCM

DTCM Power Gate Enable.

enum \_gpc\_status\_flags

GPC flags.

*Values:*

enumerator kGPC\_PoweredDownFlag

Power status. HP domain was powered down for the previous power down request.

## 2.8 GPIO: General-Purpose Input/Output Driver

void GPIO\_PinInit(GPIO\_Type \*base, uint32\_t pin, const *gpio\_pin\_config\_t* \*Config)

Initializes the GPIO peripheral according to the specified parameters in the initConfig.

**Parameters**

- base – GPIO base pointer.
- pin – Specifies the pin number
- Config – pointer to a `gpio_pin_config_t` structure that contains the configuration information.

void GPIO\_PinWrite(GPIO\_Type \*base, uint32\_t pin, uint8\_t output)

Sets the output level of the individual GPIO pin to logic 1 or 0.

**Parameters**

- base – GPIO base pointer.
- pin – GPIO port pin number.
- output – GPIO pin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

static inline void GPIO\_WritePinOutput(GPIO\_Type \*base, uint32\_t pin, uint8\_t output)

Sets the output level of the individual GPIO pin to logic 1 or 0.

*Deprecated:*

Do not use this function. It has been superceded by GPIO\_PinWrite.

static inline void GPIO\_PortSet(GPIO\_Type \*base, uint32\_t mask)

Sets the output level of the multiple GPIO pins to the logic 1.

**Parameters**

- base – GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.)
- mask – GPIO pin number macro

static inline void GPIO\_SetPinsOutput(GPIO\_Type \*base, uint32\_t mask)

Sets the output level of the multiple GPIO pins to the logic 1.

*Deprecated:*

Do not use this function. It has been superceded by GPIO\_PortSet.

static inline void GPIO\_PortClear(GPIO\_Type \*base, uint32\_t mask)

Sets the output level of the multiple GPIO pins to the logic 0.

**Parameters**

- base – GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.)
- mask – GPIO pin number macro

static inline void GPIO\_ClearPinsOutput(GPIO\_Type \*base, uint32\_t mask)

Sets the output level of the multiple GPIO pins to the logic 0.

*Deprecated:*

Do not use this function. It has been superceded by GPIO\_PortClear.

static inline void GPIO\_PortToggle(GPIO\_Type \*base, uint32\_t mask)

Reverses the current output logic of the multiple GPIO pins.

**Parameters**

- base – GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.)
- mask – GPIO pin number macro

static inline uint32\_t GPIO\_PinRead(GPIO\_Type \*base, uint32\_t pin)

Reads the current input value of the GPIO port.

**Parameters**

- base – GPIO base pointer.
- pin – GPIO port pin number.

**Return values**

GPIO – port input value.

static inline uint32\_t GPIO\_ReadPinInput(GPIO\_Type \*base, uint32\_t pin)

Reads the current input value of the GPIO port.

*Deprecated:*

Do not use this function. It has been superceded by GPIO\_PinRead.

```
static inline uint8_t GPIO_PinReadPadStatus(GPIO_Type *base, uint32_t pin)
```

Reads the current GPIO pin pad status.

**Parameters**

- base – GPIO base pointer.
- pin – GPIO port pin number.

**Return values**

GPIO – pin pad status value.

```
static inline uint8_t GPIO_ReadPadStatus(GPIO_Type *base, uint32_t pin)
```

Reads the current GPIO pin pad status.

*Deprecated:*

Do not use this function. It has been superceded by GPIO\_PinReadPadStatus.

```
void GPIO_PinSetInterruptConfig(GPIO_Type *base, uint32_t pin, gpio_interrupt_mode_t  
                               pinInterruptMode)
```

Sets the current pin interrupt mode.

**Parameters**

- base – GPIO base pointer.
- pin – GPIO port pin number.
- pinInterruptMode – pointer to a gpio\_interrupt\_mode\_t structure that contains the interrupt mode information.

```
static inline void GPIO_SetPinInterruptConfig(GPIO_Type *base, uint32_t pin,  
                                              gpio_interrupt_mode_t pinInterruptMode)
```

Sets the current pin interrupt mode.

*Deprecated:*

Do not use this function. It has been superceded by GPIO\_PinSetInterruptConfig.

```
static inline void GPIO_PortEnableInterrupts(GPIO_Type *base, uint32_t mask)
```

Enables the specific pin interrupt.

**Parameters**

- base – GPIO base pointer.
- mask – GPIO pin number macro.

```
static inline void GPIO_EnableInterrupts(GPIO_Type *base, uint32_t mask)
```

Enables the specific pin interrupt.

**Parameters**

- base – GPIO base pointer.
- mask – GPIO pin number macro.

```
static inline void GPIO_PortDisableInterrupts(GPIO_Type *base, uint32_t mask)
```

Disables the specific pin interrupt.

**Parameters**

- base – GPIO base pointer.

- mask – GPIO pin number macro.

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

Disables the specific pin interrupt.

*Deprecated:*

Do not use this function. It has been superceded by GPIO\_PortDisableInterrupts.

static inline uint32\_t GPIO\_PortGetInterruptFlags(GPIO\_Type \*base)

Reads individual pin interrupt status.

**Parameters**

- base – GPIO base pointer.

**Return values**

current – pin interrupt status flag.

static inline uint32\_t GPIO\_GetPinsInterruptFlags(GPIO\_Type \*base)

Reads individual pin interrupt status.

**Parameters**

- base – GPIO base pointer.

**Return values**

current – pin interrupt status flag.

static inline void GPIO\_PortClearInterruptFlags(GPIO\_Type \*base, uint32\_t mask)

Clears pin interrupt flag. Status flags are cleared by writing a 1 to the corresponding bit position.

**Parameters**

- base – GPIO base pointer.
- mask – GPIO pin number macro.

static inline void GPIO\_ClearPinsInterruptFlags(GPIO\_Type \*base, uint32\_t mask)

Clears pin interrupt flag. Status flags are cleared by writing a 1 to the corresponding bit position.

**Parameters**

- base – GPIO base pointer.
- mask – GPIO pin number macro.

FSL\_GPIO\_DRIVER\_VERSION

GPIO driver version.

enum \_gpio\_pin\_direction

GPIO direction definition.

*Values:*

enumerator kGPIO\_DigitalInput

Set current pin as digital input.

enumerator kGPIO\_DigitalOutput

Set current pin as digital output.

enum \_gpio\_interrupt\_mode

GPIO interrupt mode definition.

*Values:*

enumerator kGPIO\_NoIntmode

Set current pin general IO functionality.

enumerator kGPIO\_IntLowLevel

Set current pin interrupt is low-level sensitive.

enumerator kGPIO\_IntHighLevel

Set current pin interrupt is high-level sensitive.

enumerator kGPIO\_IntRisingEdge

Set current pin interrupt is rising-edge sensitive.

enumerator kGPIO\_IntFallingEdge

Set current pin interrupt is falling-edge sensitive.

enumerator kGPIO\_IntRisingOrFallingEdge

Enable the edge select bit to override the ICR register's configuration.

typedef enum *\_gpio\_pin\_direction* gpio\_pin\_direction\_t

GPIO direction definition.

typedef enum *\_gpio\_interrupt\_mode* gpio\_interrupt\_mode\_t

GPIO interrupt mode definition.

typedef struct *\_gpio\_pin\_config* gpio\_pin\_config\_t

GPIO Init structure definition.

struct *\_gpio\_pin\_config*

*#include <fsl\_gpio.h>* GPIO Init structure definition.

### Public Members

*gpio\_pin\_direction\_t* direction

Specifies the pin direction.

uint8\_t outputLogic

Set a default output logic, which has no use in input

*gpio\_interrupt\_mode\_t* interruptMode

Specifies the pin interrupt mode, a value of *gpio\_interrupt\_mode\_t*.

## 2.9 GPT: General Purpose Timer

void GPT\_Init(GPT\_Type \*base, const *gpt\_config\_t* \*initConfig)

Initialize GPT to reset state and initialize running mode.

### Parameters

- base – GPT peripheral base address.
- initConfig – GPT mode setting configuration.

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

Disables the module and gates the GPT clock.

### Parameters

- base – GPT peripheral base address.

`void GPT_GetDefaultConfig(gpt_config_t *config)`

Fills in the GPT configuration structure with default settings.

The default values are:

```
config->clockSource = kGPT_ClockSource_Periph;
config->divider = 1U;
config->enableRunInStop = true;
config->enableRunInWait = true;
config->enableRunInDoze = false;
config->enableRunInDbg = false;
config->enableFreeRun = false;
config->enableMode = true;
```

#### Parameters

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

`static inline void GPT_SoftwareReset(GPT_Type *base)`

Software reset of GPT module.

#### Parameters

- `base` – GPT peripheral base address.

`static inline void GPT_SetClockSource(GPT_Type *base, gpt_clock_source_t gptClkSource)`

Set clock source of GPT.

#### Parameters

- `base` – GPT peripheral base address.
- `gptClkSource` – Clock source (see `gpt_clock_source_t` typedef enumeration).

`static inline gpt_clock_source_t GPT_GetClockSource(GPT_Type *base)`

Get clock source of GPT.

#### Parameters

- `base` – GPT peripheral base address.

#### Returns

clock source (see `gpt_clock_source_t` typedef enumeration).

`static inline void GPT_SetClockDivider(GPT_Type *base, uint32_t divider)`

Set pre scaler of GPT.

#### Parameters

- `base` – GPT peripheral base address.
- `divider` – Divider of GPT (1-4096).

`static inline uint32_t GPT_GetClockDivider(GPT_Type *base)`

Get clock divider in GPT module.

#### Parameters

- `base` – GPT peripheral base address.

#### Returns

clock divider in GPT module (1-4096).

`static inline void GPT_SetOscClockDivider(GPT_Type *base, uint32_t divider)`

OSC 24M pre-scaler before selected by clock source.

#### Parameters

- `base` – GPT peripheral base address.

- divider – OSC Divider(1-16).

static inline uint32\_t GPT\_GetOscClockDivider(GPT\_Type \*base)

Get OSC 24M clock divider in GPT module.

**Parameters**

- base – GPT peripheral base address.

**Returns**

OSC clock divider in GPT module (1-16).

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

Start GPT timer.

**Parameters**

- base – GPT peripheral base address.

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

Stop GPT timer.

**Parameters**

- base – GPT peripheral base address.

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

Reads the current GPT counting value.

**Parameters**

- base – GPT peripheral base address.

**Returns**

Current GPT counter value.

static inline void GPT\_SetInputOperationMode(GPT\_Type \*base, *gpt\_input\_capture\_channel\_t* channel, *gpt\_input\_operation\_mode\_t* mode)

Set GPT operation mode of input capture channel.

**Parameters**

- base – GPT peripheral base address.
- channel – GPT capture channel (see *gpt\_input\_capture\_channel\_t* typedef enumeration).
- mode – GPT input capture operation mode (see *gpt\_input\_operation\_mode\_t* typedef enumeration).

static inline *gpt\_input\_operation\_mode\_t* GPT\_GetInputOperationMode(GPT\_Type \*base, *gpt\_input\_capture\_channel\_t* channel)

Get GPT operation mode of input capture channel.

**Parameters**

- base – GPT peripheral base address.
- channel – GPT capture channel (see *gpt\_input\_capture\_channel\_t* typedef enumeration).

**Returns**

GPT input capture operation mode (see *gpt\_input\_operation\_mode\_t* typedef enumeration).



```
static inline uint32_t GPT_GetInputCaptureValue(GPT_Type *base, gpt_input_capture_channel_t
                                              channel)
```

Get GPT input capture value of certain channel.

#### Parameters

- base – GPT peripheral base address.
- channel – GPT capture channel (see `gpt_input_capture_channel_t` typedef enumeration).

#### Returns

GPT input capture value.

```
static inline void GPT_SetOutputOperationMode(GPT_Type *base,
                                              gpt_output_compare_channel_t channel,
                                              gpt_output_operation_mode_t mode)
```

Set GPT operation mode of output compare channel.

#### Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see `gpt_output_compare_channel_t` typedef enumeration).
- mode – GPT output operation mode (see `gpt_output_operation_mode_t` typedef enumeration).

```
static inline gpt_output_operation_mode_t GPT_GetOutputOperationMode(GPT_Type *base,
                                                                    gpt_output_compare_channel_t
                                                                    channel)
```

Get GPT operation mode of output compare channel.

#### Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see `gpt_output_compare_channel_t` typedef enumeration).

#### Returns

GPT output operation mode (see `gpt_output_operation_mode_t` typedef enumeration).

```
static inline void GPT_SetOutputCompareValue(GPT_Type *base, gpt_output_compare_channel_t
                                              channel, uint32_t value)
```

Set GPT output compare value of output compare channel.

#### Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see `gpt_output_compare_channel_t` typedef enumeration).
- value – GPT output compare value.

```
static inline uint32_t GPT_GetOutputCompareValue(GPT_Type *base,
                                                  gpt_output_compare_channel_t channel)
```

Get GPT output compare value of output compare channel.

#### Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see `gpt_output_compare_channel_t` typedef enumeration).

**Returns**

GPT output compare value.

static inline void GPT\_ForceOutput(GPT\_Type \*base, *gpt\_output\_compare\_channel\_t* channel)  
Force GPT output action on output compare channel, ignoring comparator.

**Parameters**

- base – GPT peripheral base address.
- channel – GPT output compare channel (see *gpt\_output\_compare\_channel\_t* typedef enumeration).

static inline void GPT\_EnableInterrupts(GPT\_Type \*base, uint32\_t mask)  
Enables the selected GPT interrupts.

**Parameters**

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

static inline void GPT\_DisableInterrupts(GPT\_Type \*base, uint32\_t mask)  
Disables the selected GPT interrupts.

**Parameters**

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

static inline uint32\_t GPT\_GetEnabledInterrupts(GPT\_Type \*base)  
Gets the enabled GPT interrupts.

**Parameters**

- base – GPT peripheral base address

**Returns**

The enabled interrupts. This is the logical OR of members of the enumeration *gpt\_interrupt\_enable\_t*

static inline uint32\_t GPT\_GetStatusFlags(GPT\_Type \*base, *gpt\_status\_flag\_t* flags)  
Get GPT status flags.

**Parameters**

- base – GPT peripheral base address.
- flags – GPT status flag mask (see *gpt\_status\_flag\_t* for bit definition).

**Returns**

GPT status, each bit represents one status flag.

static inline void GPT\_ClearStatusFlags(GPT\_Type \*base, *gpt\_status\_flag\_t* flags)  
Clears the GPT status flags.

**Parameters**

- base – GPT peripheral base address.
- flags – GPT status flag mask (see *gpt\_status\_flag\_t* for bit definition).

FSL\_GPT\_DRIVER\_VERSION

enum \_gpt\_clock\_source  
List of clock sources.

---

**Note:** Actual number of clock sources is SoC dependent

---

*Values:*

enumerator kGPT\_ClockSource\_Off  
GPT Clock Source Off.

enumerator kGPT\_ClockSource\_Periph  
GPT Clock Source from Peripheral Clock.

enumerator kGPT\_ClockSource\_HighFreq  
GPT Clock Source from High Frequency Reference Clock.

enumerator kGPT\_ClockSource\_Ext  
GPT Clock Source from external pin.

enumerator kGPT\_ClockSource\_LowFreq  
GPT Clock Source from Low Frequency Reference Clock.

enumerator kGPT\_ClockSource\_Osc  
GPT Clock Source from Crystal oscillator.

enum \_gpt\_input\_capture\_channel  
List of input capture channel number.

*Values:*

enumerator kGPT\_InputCapture\_Channel1  
GPT Input Capture Channel1.

enumerator kGPT\_InputCapture\_Channel2  
GPT Input Capture Channel2.

enum \_gpt\_input\_operation\_mode  
List of input capture operation mode.

*Values:*

enumerator kGPT\_InputOperation\_Disabled  
Don't capture.

enumerator kGPT\_InputOperation\_RiseEdge  
Capture on rising edge of input pin.

enumerator kGPT\_InputOperation\_FallEdge  
Capture on falling edge of input pin.

enumerator kGPT\_InputOperation\_BothEdge  
Capture on both edges of input pin.

enum \_gpt\_output\_compare\_channel  
List of output compare channel number.

*Values:*

enumerator kGPT\_OutputCompare\_Channel1  
Output Compare Channel1.

enumerator kGPT\_OutputCompare\_Channel2  
Output Compare Channel2.

enumerator kGPT\_OutputCompare\_Channel3  
Output Compare Channel3.

enum \_gpt\_output\_operation\_mode  
List of output compare operation mode.

*Values:*

enumerator kGPT\_OutputOperation\_Disconnected  
Don't change output pin.

enumerator kGPT\_OutputOperation\_Toggle  
Toggle output pin.

enumerator kGPT\_OutputOperation\_Clear  
Set output pin low.

enumerator kGPT\_OutputOperation\_Set  
Set output pin high.

enumerator kGPT\_OutputOperation\_Activelow  
Generate a active low pulse on output pin.

enum \_gpt\_interrupt\_enable  
List of GPT interrupts.

*Values:*

enumerator kGPT\_OutputCompare1InterruptEnable  
Output Compare Channel1 interrupt enable

enumerator kGPT\_OutputCompare2InterruptEnable  
Output Compare Channel2 interrupt enable

enumerator kGPT\_OutputCompare3InterruptEnable  
Output Compare Channel3 interrupt enable

enumerator kGPT\_InputCapture1InterruptEnable  
Input Capture Channel1 interrupt enable

enumerator kGPT\_InputCapture2InterruptEnable  
Input Capture Channel1 interrupt enable

enumerator kGPT\_RollOverFlagInterruptEnable  
Counter rolled over interrupt enable

enum \_gpt\_status\_flag  
Status flag.

*Values:*

enumerator kGPT\_OutputCompare1Flag  
Output compare channel 1 event.

enumerator kGPT\_OutputCompare2Flag  
Output compare channel 2 event.

enumerator kGPT\_OutputCompare3Flag  
Output compare channel 3 event.

enumerator kGPT\_InputCapture1Flag

Input Capture channel 1 event.

enumerator kGPT\_InputCapture2Flag

Input Capture channel 2 event.

enumerator kGPT\_RollOverFlag

Counter reaches maximum value and rolled over to 0 event.

typedef enum *\_gpt\_clock\_source* gpt\_clock\_source\_t

List of clock sources.

---

**Note:** Actual number of clock sources is SoC dependent

---

typedef enum *\_gpt\_input\_capture\_channel* gpt\_input\_capture\_channel\_t

List of input capture channel number.

typedef enum *\_gpt\_input\_operation\_mode* gpt\_input\_operation\_mode\_t

List of input capture operation mode.

typedef enum *\_gpt\_output\_compare\_channel* gpt\_output\_compare\_channel\_t

List of output compare channel number.

typedef enum *\_gpt\_output\_operation\_mode* gpt\_output\_operation\_mode\_t

List of output compare operation mode.

typedef enum *\_gpt\_interrupt\_enable* gpt\_interrupt\_enable\_t

List of GPT interrupts.

typedef enum *\_gpt\_status\_flag* gpt\_status\_flag\_t

Status flag.

typedef struct *\_gpt\_init\_config* gpt\_config\_t

Structure to configure the running mode.

struct *\_gpt\_init\_config*

*#include <fsl\_gpt.h>* Structure to configure the running mode.

## Public Members

*gpt\_clock\_source\_t* clockSource

clock source for GPT module.

uint32\_t divider

clock divider (prescaler+1) from clock source to counter.

bool enableFreeRun

true: FreeRun mode, false: Restart mode.

bool enableRunInWait

GPT enabled in wait mode.

bool enableRunInStop

GPT enabled in stop mode.

bool enableRunInDoze

GPT enabled in doze mode.

bool enableRunInDbg

GPT enabled in debug mode.

bool enableMode

true: counter reset to 0 when enabled; false: counter retain its value when enabled.

## 2.10 I2C: Inter-Integrated Circuit Driver

### 2.11 I2C Driver

void I2C\_MasterInit(I2C\_Type \*base, const *i2c\_master\_config\_t* \*masterConfig, uint32\_t srcClock\_Hz)

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

---

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

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

---

#### Parameters

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

void I2C\_MasterDeinit(I2C\_Type \*base)

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

#### Parameters

- base – I2C base pointer

void I2C\_MasterGetDefaultConfig(*i2c\_master\_config\_t* \*masterConfig)

Sets the I2C master configuration structure to default values.

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

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

#### Parameters

- masterConfig – A pointer to the master configuration structure.

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

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

**Note:** This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can partly be set with default values by `I2C_SlaveGetDefaultConfig()` or it can be custom filled by the user. This is an example.

```
i2c_slave_config_t config = {
    .enableSlave = true,
    .slaveAddress = 0x1DU,
};
I2C_SlaveInit(I2C0, &config);
```

### Parameters

- `base` – I2C base pointer
- `slaveConfig` – A pointer to the slave configuration structure

`void I2C_SlaveDeinit(I2C_Type *base)`

De-initializes the I2C slave peripheral. Calling this API gates the I2C clock. The I2C slave module can't work unless the `I2C_SlaveInit` is called to enable the clock.

### Parameters

- `base` – I2C base pointer

`void I2C_SlaveGetDefaultConfig(i2c_slave_config_t *slaveConfig)`

Sets the I2C slave configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in the `I2C_SlaveInit()`. Modify fields of the structure before calling the `I2C_SlaveInit()`. This is an example.

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

### Parameters

- `slaveConfig` – A pointer to the slave configuration structure.

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

Enables or disables the I2C peripheral operation.

### Parameters

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

`static inline uint32_t I2C_MasterGetStatusFlags(I2C_Type *base)`

Gets the I2C status flags.

### Parameters

- `base` – I2C base pointer

### Returns

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

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

Clears the I2C status flag state.

The following status register flags can be cleared kI2C\_ArbitrationLostFlag and kI2C\_IntPendingFlag.

#### Parameters

- base – I2C base pointer
- statusMask – The status flag mask, defined in type i2c\_status\_flag\_t. The parameter can be any combination of the following values:
  - kI2C\_ArbitrationLostFlag
  - kI2C\_IntPendingFlag

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

Gets the I2C status flags.

#### Parameters

- base – I2C base pointer

#### Returns

status flag, use status flag to AND \_i2c\_flags to get the related status.

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

Clears the I2C status flag state.

The following status register flags can be cleared kI2C\_ArbitrationLostFlag and kI2C\_IntPendingFlag

#### Parameters

- base – I2C base pointer
- statusMask – The status flag mask, defined in type i2c\_status\_flag\_t. The parameter can be any combination of the following values:
  - kI2C\_IntPendingFlag

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

Enables I2C interrupt requests.

#### Parameters

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

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

Disables I2C interrupt requests.

#### Parameters

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



– kI2C\_SdaTimeoutInterruptEnable

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

Sets the I2C master transfer baud rate.

**Parameters**

- `base` – I2C base pointer
- `baudRate_Bps` – the baud rate value in bps
- `srcClock_Hz` – Source clock

`status_t I2C_MasterStart(I2C_Type *base, uint8_t address, i2c_direction_t direction)`

Sends a START on the I2C bus.

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

**Parameters**

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

**Return values**

- `kStatus_Success` – Successfully send the start signal.
- `kStatus_I2C_Busy` – Current bus is busy.

`status_t I2C_MasterStop(I2C_Type *base)`

Sends a STOP signal on the I2C bus.

**Return values**

- `kStatus_Success` – Successfully send the stop signal.
- `kStatus_I2C_Timeout` – Send stop signal failed, timeout.

`status_t I2C_MasterRepeatedStart(I2C_Type *base, uint8_t address, i2c_direction_t direction)`

Sends a REPEATED START on the I2C bus.

**Parameters**

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

**Return values**

- `kStatus_Success` – Successfully send the start signal.
- `kStatus_I2C_Busy` – Current bus is busy but not occupied by current I2C master.

`status_t I2C_MasterWriteBlocking(I2C_Type *base, const uint8_t *txBuff, size_t txSize, uint32_t flags)`

Performs a polling send transaction on the I2C bus.

**Parameters**

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

- flags – Transfer control flag to decide whether need to send a stop, use kI2C\_TransferDefaultFlag to issue a stop and kI2C\_TransferNoStop to not send a stop.

**Return values**

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

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

Performs a polling receive transaction on the I2C bus.

---

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

---

**Parameters**

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

**Return values**

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

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

Performs a polling send transaction on the I2C bus.

**Parameters**

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

**Return values**

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

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

Performs a polling receive transaction on the I2C bus.

**Parameters**

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

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

Performs a master polling transfer on the I2C bus.

---

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

---

#### Parameters

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

#### Return values

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

void I2C\_MasterTransferCreateHandle(I2C\_Type \*base, *i2c\_master\_handle\_t* \*handle,  
*i2c\_master\_transfer\_callback\_t* callback, void \*userData)

Initializes the I2C handle which is used in transactional functions.

#### Parameters

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

*status\_t* I2C\_MasterTransferNonBlocking(I2C\_Type \*base, *i2c\_master\_handle\_t* \*handle,  
*i2c\_master\_transfer\_t* \*xfer)

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

---

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

---

#### Parameters

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

#### Return values

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

*status\_t* I2C\_MasterTransferGetCount(I2C\_Type \*base, *i2c\_master\_handle\_t* \*handle, size\_t \*count)

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

#### Parameters

- base – I2C base pointer.
- handle – pointer to *i2c\_master\_handle\_t* structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

#### Return values

- *kStatus\_InvalidArgument* – count is Invalid.
- *kStatus\_Success* – Successfully return the count.

*status\_t* I2C\_MasterTransferAbort(I2C\_Type \*base, *i2c\_master\_handle\_t* \*handle)

Aborts an interrupt non-blocking transfer early.

---

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

---

#### Parameters

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

#### Return values

- *kStatus\_I2C\_Timeout* – Timeout during polling flag.
- *kStatus\_Success* – Successfully abort the transfer.

void I2C\_MasterTransferHandleIRQ(I2C\_Type \*base, void \*i2cHandle)

Master interrupt handler.

#### Parameters

- base – I2C base pointer.
- i2cHandle – pointer to *i2c\_master\_handle\_t* structure.

void I2C\_SlaveTransferCreateHandle(I2C\_Type \*base, *i2c\_slave\_handle\_t* \*handle, *i2c\_slave\_transfer\_callback\_t* callback, void \*userData)

Initializes the I2C handle which is used in transactional functions.

#### Parameters

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

*status\_t* I2C\_SlaveTransferNonBlocking(I2C\_Type \*base, *i2c\_slave\_handle\_t* \*handle, uint32\_t eventMask)

Starts accepting slave transfers.

Call this API after calling the *I2C\_SlaveInit()* and *I2C\_SlaveTransferCreateHandle()* to start processing transactions driven by an I2C master. The slave monitors the I2C bus and passes

events to the callback that was passed into the call to `I2C_SlaveTransferCreateHandle()`. The callback is always invoked from the interrupt context.

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

#### Parameters

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

#### Return values

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

`void I2C_SlaveTransferAbort(I2C_Type *base, i2c_slave_handle_t *handle)`

Aborts the slave transfer.

---

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

---

#### Parameters

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

`status_t I2C_SlaveTransferGetCount(I2C_Type *base, i2c_slave_handle_t *handle, size_t *count)`

Gets the slave transfer remaining bytes during a interrupt non-blocking transfer.

#### Parameters

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

#### Return values

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

`void I2C_SlaveTransferHandleIRQ(I2C_Type *base, void *i2cHandle)`

Slave interrupt handler.

#### Parameters

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

FSL\_I2C\_DRIVER\_VERSION

I2C driver version.

I2C status return codes.

*Values:*

enumerator kStatus\_I2C\_Busy

I2C is busy with current transfer.

enumerator kStatus\_I2C\_Idle

Bus is Idle.

enumerator kStatus\_I2C\_Nak

NAK received during transfer.

enumerator kStatus\_I2C\_ArbitrationLost

Arbitration lost during transfer.

enumerator kStatus\_I2C\_Timeout

Timeout polling status flags.

enumerator kStatus\_I2C\_Addr\_Nak

NAK received during the address probe.

enum \_i2c\_flags

I2C peripheral flags.

The following status register flags can be cleared:

- kI2C\_ArbitrationLostFlag
- kI2C\_IntPendingFlag

---

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

---

*Values:*

enumerator kI2C\_ReceiveNakFlag

I2C receive NAK flag.

enumerator kI2C\_IntPendingFlag

I2C interrupt pending flag.

enumerator kI2C\_TransferDirectionFlag

I2C transfer direction flag.

enumerator kI2C\_ArbitrationLostFlag

I2C arbitration lost flag.

enumerator kI2C\_BusBusyFlag

I2C bus busy flag.

enumerator kI2C\_AddressMatchFlag

I2C address match flag.

enumerator kI2C\_TransferCompleteFlag

I2C transfer complete flag.

enum \_i2c\_interrupt\_enable

I2C feature interrupt source.

*Values:*

enumerator kI2C\_GlobalInterruptEnable

I2C global interrupt.

enum \_i2c\_direction

The direction of master and slave transfers.

*Values:*

enumerator kI2C\_Write

Master transmits to the slave.

enumerator kI2C\_Read

Master receives from the slave.

enum \_i2c\_master\_transfer\_flags

I2C transfer control flag.

*Values:*

enumerator kI2C\_TransferDefaultFlag

A transfer starts with a start signal, stops with a stop signal.

enumerator kI2C\_TransferNoStartFlag

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

enumerator kI2C\_TransferRepeatedStartFlag

A transfer starts with a repeated start signal.

enumerator kI2C\_TransferNoStopFlag

A transfer ends without a stop signal.

enum \_i2c\_slave\_transfer\_event

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

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

---

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

---

*Values:*

enumerator kI2C\_SlaveAddressMatchEvent

Received the slave address after a start or repeated start.

enumerator kI2C\_SlaveTransmitEvent

A callback is requested to provide data to transmit (slave-transmitter role).

enumerator kI2C\_SlaveReceiveEvent

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

enumerator kI2C\_SlaveTransmitAckEvent

A callback needs to either transmit an ACK or NACK.

enumerator kI2C\_SlaveCompletionEvent

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

enumerator kI2C\_SlaveAllEvents

A bit mask of all available events.

```
typedef enum _i2c_direction i2c_direction_t
```

The direction of master and slave transfers.

```
typedef struct _i2c_master_config i2c_master_config_t
```

I2C master user configuration.

```
typedef struct _i2c_master_handle i2c_master_handle_t
```

I2C master handle typedef.

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

I2C master transfer callback typedef.

```
typedef struct _i2c_master_transfer i2c_master_transfer_t
```

I2C master transfer structure.

```
typedef enum _i2c_slave_transfer_event i2c_slave_transfer_event_t
```

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

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

---

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

---

```
typedef struct _i2c_slave_handle i2c_slave_handle_t
```

I2C slave handle typedef.

```
typedef struct _i2c_slave_config i2c_slave_config_t
```

I2C slave user configuration.

```
typedef struct _i2c_slave_transfer i2c_slave_transfer_t
```

I2C slave transfer structure.

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

I2C slave transfer callback typedef.

```
I2C_RETRY_TIMES
```

Retry times for waiting flag.

```
struct _i2c_master_config
```

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

### Public Members

```
bool enableMaster
```

Enables the I2C peripheral at initialization time.

```
uint32_t baudRate_Bps
```

Baud rate configuration of I2C peripheral.

```
struct _i2c_master_transfer
```

```
#include <fsl_i2c.h> I2C master transfer structure.
```



**Public Members**

uint32\_t flags

A transfer flag which controls the transfer.

uint8\_t slaveAddress

7-bit slave address.

i2c\_direction\_t direction

A transfer direction, read or write.

uint32\_t subaddress

A sub address. Transferred MSB first.

uint8\_t subaddressSize

A size of the command buffer.

uint8\_t \*volatile data

A transfer buffer.

volatile size\_t dataSize

A transfer size.

struct \_\_i2c\_master\_handle

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

**Public Members**

i2c\_master\_transfer\_t transfer

I2C master transfer copy.

size\_t transferSize

Total bytes to be transferred.

uint8\_t state

A transfer state maintained during transfer.

i2c\_master\_transfer\_callback\_t completionCallback

A callback function called when the transfer is finished.

void \*userData

A callback parameter passed to the callback function.

struct \_\_i2c\_slave\_config

*#include <fsl\_i2c.h>* I2C slave user configuration.

**Public Members**

bool enableSlave

Enables the I2C peripheral at initialization time.

uint16\_t slaveAddress

A slave address configuration.

struct \_\_i2c\_slave\_transfer

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

### Public Members

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

A reason that the callback is invoked.

uint8\_t \*volatile data

A transfer buffer.

volatile size\_t dataSize

A transfer size.

*status\_t* completionStatus

Success or error code describing how the transfer completed. Only applies for kI2C\_SlaveCompletionEvent.

size\_t transferredCount

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

struct \_i2c\_slave\_handle

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

### Public Members

volatile uint8\_t state

A transfer state maintained during transfer.

*i2c\_slave\_transfer\_t* transfer

I2C slave transfer copy.

uint32\_t eventMask

A mask of enabled events.

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

A callback function called at the transfer event.

void \*userData

A callback parameter passed to the callback.

## 2.12 Iomuxc\_driver

```
static inline void IOMUXC_SetPinMux(uint32_t muxRegister, uint32_t muxMode, uint32_t
                                     inputRegister, uint32_t inputDaisy, uint32_t
                                     configRegister, uint32_t inputOnfield)
```

Sets the IOMUXC pin mux mode.

This is an example to set the I2C4\_SDA as the pwm1\_OUT:

```
IOMUXC_SetPinMux(IOMUXC_I2C4_SDA_PWM1_OUT, 0);
```

---

**Note:** The first five parameters can be filled with the pin function ID macros.

---

### Parameters

- muxRegister – The pin mux register\_
- muxMode – The pin mux mode\_

- inputRegister – The select input register\_
- inputDaisy – The input daisy\_
- configRegister – The config register\_
- inputOnfield – The pad->module input inversion\_

```
static inline void IOMUXC_SetPinConfig(uint32_t muxRegister, uint32_t muxMode, uint32_t
                                     inputRegister, uint32_t inputDaisy, uint32_t
                                     configRegister, uint32_t configValue)
```

Sets the IOMUXC pin configuration.

This is an example to set pin configuration for IOMUXC\_I2C4\_SDA\_PWM1\_OUT:

```
IOMUXC_SetPinConfig(IOMUXC_I2C4_SDA_PWM1_OUT, IOMUXC_SW_PAD_CTL_PAD_
↪ODE_MASK | IOMUXC0_SW_PAD_CTL_PAD_DSE(2U))
```

---

**Note:** The previous five parameters can be filled with the pin function ID macros.

---

### Parameters

- muxRegister – The pin mux register\_
- muxMode – The pin mux mode\_
- inputRegister – The select input register\_
- inputDaisy – The input daisy\_
- configRegister – The config register\_
- configValue – The pin config value\_

FSL\_IOMUXC\_DRIVER\_VERSION

IOMUXC driver version 2.0.1.

IOMUXC\_PMIC\_STBY\_REQ

IOMUXC\_PMIC\_ON\_REQ

IOMUXC\_ONOFF

IOMUXC\_POR\_B

IOMUXC\_RTC\_RESET\_B

IOMUXC\_GPIO1\_IO00\_GPIO1\_IO00

IOMUXC\_GPIO1\_IO00\_CCM\_ENET\_PHY\_REF\_CLK\_ROOT

IOMUXC\_GPIO1\_IO00\_XTALOSC\_REF\_CLK\_32K

IOMUXC\_GPIO1\_IO00\_CCM\_EXT\_CLK1

IOMUXC\_GPIO1\_IO01\_GPIO1\_IO01

IOMUXC\_GPIO1\_IO01\_PWM1\_OUT

IOMUXC\_GPIO1\_IO01\_XTALOSC\_REF\_CLK\_24M

IOMUXC\_GPIO1\_IO01\_CCM\_EXT\_CLK2

IOMUXC\_GPIO1\_IO02\_GPIO1\_IO02  
IOMUXC\_GPIO1\_IO02\_WDOG1\_WDOG\_B  
IOMUXC\_GPIO1\_IO02\_WDOG1\_WDOG\_ANY  
IOMUXC\_GPIO1\_IO03\_GPIO1\_IO03  
IOMUXC\_GPIO1\_IO03\_USDHC1\_VSELECT  
IOMUXC\_GPIO1\_IO03\_SDMA1\_EXT\_EVENT0  
IOMUXC\_GPIO1\_IO04\_GPIO1\_IO04  
IOMUXC\_GPIO1\_IO04\_USDHC2\_VSELECT  
IOMUXC\_GPIO1\_IO04\_SDMA1\_EXT\_EVENT1  
IOMUXC\_GPIO1\_IO05\_GPIO1\_IO05  
IOMUXC\_GPIO1\_IO05\_M4\_NMI  
IOMUXC\_GPIO1\_IO05\_CCM\_PMIC\_READY  
IOMUXC\_GPIO1\_IO06\_GPIO1\_IO06  
IOMUXC\_GPIO1\_IO06\_ENET1\_MDC  
IOMUXC\_GPIO1\_IO06\_USDHC1\_CD\_B  
IOMUXC\_GPIO1\_IO06\_CCM\_EXT\_CLK3  
IOMUXC\_GPIO1\_IO07\_GPIO1\_IO07  
IOMUXC\_GPIO1\_IO07\_ENET1\_MDIO  
IOMUXC\_GPIO1\_IO07\_USDHC1\_WP  
IOMUXC\_GPIO1\_IO07\_CCM\_EXT\_CLK4  
IOMUXC\_GPIO1\_IO08\_GPIO1\_IO08  
IOMUXC\_GPIO1\_IO08\_ENET1\_1588\_EVENT0\_IN  
IOMUXC\_GPIO1\_IO08\_USDHC2\_RESET\_B  
IOMUXC\_GPIO1\_IO09\_GPIO1\_IO09  
IOMUXC\_GPIO1\_IO09\_ENET1\_1588\_EVENT0\_OUT  
IOMUXC\_GPIO1\_IO09\_SDMA2\_EXT\_EVENT0  
IOMUXC\_GPIO1\_IO10\_GPIO1\_IO10  
IOMUXC\_GPIO1\_IO10\_USB1\_OTG\_ID  
IOMUXC\_GPIO1\_IO11\_GPIO1\_IO11  
IOMUXC\_GPIO1\_IO11\_USB2\_OTG\_ID  
IOMUXC\_GPIO1\_IO11\_CCM\_PMIC\_READY  
IOMUXC\_GPIO1\_IO12\_GPIO1\_IO12  
IOMUXC\_GPIO1\_IO12\_USB1\_OTG\_PWR

IOMUXC\_GPIO1\_IO12\_SDMA2\_EXT\_EVENT1  
IOMUXC\_GPIO1\_IO13\_GPIO1\_IO13  
IOMUXC\_GPIO1\_IO13\_USB1\_OTG\_OC  
IOMUXC\_GPIO1\_IO13\_PWM2\_OUT  
IOMUXC\_GPIO1\_IO14\_GPIO1\_IO14  
IOMUXC\_GPIO1\_IO14\_USB2\_OTG\_PWR  
IOMUXC\_GPIO1\_IO14\_PWM3\_OUT  
IOMUXC\_GPIO1\_IO14\_CCM\_CLKO1  
IOMUXC\_GPIO1\_IO15\_GPIO1\_IO15  
IOMUXC\_GPIO1\_IO15\_USB2\_OTG\_OC  
IOMUXC\_GPIO1\_IO15\_PWM4\_OUT  
IOMUXC\_GPIO1\_IO15\_CCM\_CLKO2  
IOMUXC\_ENET\_MDC\_ENET1\_MDC  
IOMUXC\_ENET\_MDC\_GPIO1\_IO16  
IOMUXC\_ENET\_MDIO\_ENET1\_MDIO  
IOMUXC\_ENET\_MDIO\_GPIO1\_IO17  
IOMUXC\_ENET\_TD3\_ENET1\_RGMII\_TD3  
IOMUXC\_ENET\_TD3\_GPIO1\_IO18  
IOMUXC\_ENET\_TD2\_ENET1\_RGMII\_TD2  
IOMUXC\_ENET\_TD2\_ENET1\_TX\_CLK  
IOMUXC\_ENET\_TD2\_GPIO1\_IO19  
IOMUXC\_ENET\_TD1\_ENET1\_RGMII\_TD1  
IOMUXC\_ENET\_TD1\_GPIO1\_IO20  
IOMUXC\_ENET\_TD0\_ENET1\_RGMII\_TD0  
IOMUXC\_ENET\_TD0\_GPIO1\_IO21  
IOMUXC\_ENET\_TX\_CTL\_ENET1\_RGMII\_TX\_CTL  
IOMUXC\_ENET\_TX\_CTL\_GPIO1\_IO22  
IOMUXC\_ENET\_TXC\_ENET1\_RGMII\_TXC  
IOMUXC\_ENET\_TXC\_ENET1\_TX\_ER  
IOMUXC\_ENET\_TXC\_GPIO1\_IO23  
IOMUXC\_ENET\_RX\_CTL\_ENET1\_RGMII\_RX\_CTL  
IOMUXC\_ENET\_RX\_CTL\_GPIO1\_IO24  
IOMUXC\_ENET\_RXC\_ENET1\_RGMII\_RXC

IOMUXC\_ENET\_RXC\_ENET1\_RX\_ER  
IOMUXC\_ENET\_RXC\_GPIO1\_IO25  
IOMUXC\_ENET\_RD0\_ENET1\_RGMII\_RD0  
IOMUXC\_ENET\_RD0\_GPIO1\_IO26  
IOMUXC\_ENET\_RD1\_ENET1\_RGMII\_RD1  
IOMUXC\_ENET\_RD1\_GPIO1\_IO27  
IOMUXC\_ENET\_RD2\_ENET1\_RGMII\_RD2  
IOMUXC\_ENET\_RD2\_GPIO1\_IO28  
IOMUXC\_ENET\_RD3\_ENET1\_RGMII\_RD3  
IOMUXC\_ENET\_RD3\_GPIO1\_IO29  
IOMUXC\_SD1\_CLK\_USDHC1\_CLK  
IOMUXC\_SD1\_CLK\_GPIO2\_IO00  
IOMUXC\_SD1\_CMD\_USDHC1\_CMD  
IOMUXC\_SD1\_CMD\_GPIO2\_IO01  
IOMUXC\_SD1\_DATA0\_USDHC1\_DATA0  
IOMUXC\_SD1\_DATA0\_GPIO2\_IO02  
IOMUXC\_SD1\_DATA1\_USDHC1\_DATA1  
IOMUXC\_SD1\_DATA1\_GPIO2\_IO03  
IOMUXC\_SD1\_DATA2\_USDHC1\_DATA2  
IOMUXC\_SD1\_DATA2\_GPIO2\_IO04  
IOMUXC\_SD1\_DATA3\_USDHC1\_DATA3  
IOMUXC\_SD1\_DATA3\_GPIO2\_IO05  
IOMUXC\_SD1\_DATA4\_USDHC1\_DATA4  
IOMUXC\_SD1\_DATA4\_GPIO2\_IO06  
IOMUXC\_SD1\_DATA5\_USDHC1\_DATA5  
IOMUXC\_SD1\_DATA5\_GPIO2\_IO07  
IOMUXC\_SD1\_DATA6\_USDHC1\_DATA6  
IOMUXC\_SD1\_DATA6\_GPIO2\_IO08  
IOMUXC\_SD1\_DATA7\_USDHC1\_DATA7  
IOMUXC\_SD1\_DATA7\_GPIO2\_IO09  
IOMUXC\_SD1\_RESET\_B\_USDHC1\_RESET\_B  
IOMUXC\_SD1\_RESET\_B\_GPIO2\_IO10  
IOMUXC\_SD1\_STROBE\_USDHC1\_STROBE

IOMUXC\_SD1\_STROBE\_GPIO2\_IO11  
IOMUXC\_SD2\_CD\_B\_USDHC2\_CD\_B  
IOMUXC\_SD2\_CD\_B\_GPIO2\_IO12  
IOMUXC\_SD2\_CLK\_USDHC2\_CLK  
IOMUXC\_SD2\_CLK\_GPIO2\_IO13  
IOMUXC\_SD2\_CMD\_USDHC2\_CMD  
IOMUXC\_SD2\_CMD\_GPIO2\_IO14  
IOMUXC\_SD2\_DATA0\_USDHC2\_DATA0  
IOMUXC\_SD2\_DATA0\_GPIO2\_IO15  
IOMUXC\_SD2\_DATA1\_USDHC2\_DATA1  
IOMUXC\_SD2\_DATA1\_GPIO2\_IO16  
IOMUXC\_SD2\_DATA2\_USDHC2\_DATA2  
IOMUXC\_SD2\_DATA2\_GPIO2\_IO17  
IOMUXC\_SD2\_DATA3\_USDHC2\_DATA3  
IOMUXC\_SD2\_DATA3\_GPIO2\_IO18  
IOMUXC\_SD2\_RESET\_B\_USDHC2\_RESET\_B  
IOMUXC\_SD2\_RESET\_B\_GPIO2\_IO19  
IOMUXC\_SD2\_WP\_USDHC2\_WP  
IOMUXC\_SD2\_WP\_GPIO2\_IO20  
IOMUXC\_NAND\_ALE\_RAWNAND\_ALE  
IOMUXC\_NAND\_ALE\_QSPI\_A\_SCLK  
IOMUXC\_NAND\_ALE\_GPIO3\_IO00  
IOMUXC\_NAND\_CE0\_B\_RAWNAND\_CE0\_B  
IOMUXC\_NAND\_CE0\_B\_QSPI\_A\_SS0\_B  
IOMUXC\_NAND\_CE0\_B\_GPIO3\_IO01  
IOMUXC\_NAND\_CE1\_B\_RAWNAND\_CE1\_B  
IOMUXC\_NAND\_CE1\_B\_QSPI\_A\_SS1\_B  
IOMUXC\_NAND\_CE1\_B\_GPIO3\_IO02  
IOMUXC\_NAND\_CE2\_B\_RAWNAND\_CE2\_B  
IOMUXC\_NAND\_CE2\_B\_QSPI\_B\_SS0\_B  
IOMUXC\_NAND\_CE2\_B\_GPIO3\_IO03  
IOMUXC\_NAND\_CE3\_B\_RAWNAND\_CE3\_B  
IOMUXC\_NAND\_CE3\_B\_QSPI\_B\_SS1\_B

IOMUXC\_NAND\_CE3\_B\_GPIO3\_IO04  
IOMUXC\_NAND\_CLE\_RAWNAND\_CLE  
IOMUXC\_NAND\_CLE\_QSPI\_B\_SCLK  
IOMUXC\_NAND\_CLE\_GPIO3\_IO05  
IOMUXC\_NAND\_DATA00\_RAWNAND\_DATA00  
IOMUXC\_NAND\_DATA00\_QSPI\_A\_DATA0  
IOMUXC\_NAND\_DATA00\_GPIO3\_IO06  
IOMUXC\_NAND\_DATA01\_RAWNAND\_DATA01  
IOMUXC\_NAND\_DATA01\_QSPI\_A\_DATA1  
IOMUXC\_NAND\_DATA01\_GPIO3\_IO07  
IOMUXC\_NAND\_DATA02\_RAWNAND\_DATA02  
IOMUXC\_NAND\_DATA02\_QSPI\_A\_DATA2  
IOMUXC\_NAND\_DATA02\_GPIO3\_IO08  
IOMUXC\_NAND\_DATA03\_RAWNAND\_DATA03  
IOMUXC\_NAND\_DATA03\_QSPI\_A\_DATA3  
IOMUXC\_NAND\_DATA03\_GPIO3\_IO09  
IOMUXC\_NAND\_DATA04\_RAWNAND\_DATA04  
IOMUXC\_NAND\_DATA04\_QSPI\_B\_DATA0  
IOMUXC\_NAND\_DATA04\_GPIO3\_IO10  
IOMUXC\_NAND\_DATA05\_RAWNAND\_DATA05  
IOMUXC\_NAND\_DATA05\_QSPI\_B\_DATA1  
IOMUXC\_NAND\_DATA05\_GPIO3\_IO11  
IOMUXC\_NAND\_DATA06\_RAWNAND\_DATA06  
IOMUXC\_NAND\_DATA06\_QSPI\_B\_DATA2  
IOMUXC\_NAND\_DATA06\_GPIO3\_IO12  
IOMUXC\_NAND\_DATA07\_RAWNAND\_DATA07  
IOMUXC\_NAND\_DATA07\_QSPI\_B\_DATA3  
IOMUXC\_NAND\_DATA07\_GPIO3\_IO13  
IOMUXC\_NAND\_DQS\_RAWNAND\_DQS  
IOMUXC\_NAND\_DQS\_QSPI\_A\_DQS  
IOMUXC\_NAND\_DQS\_GPIO3\_IO14  
IOMUXC\_NAND\_RE\_B\_RAWNAND\_RE\_B  
IOMUXC\_NAND\_RE\_B\_QSPI\_B\_DQS



IOMUXC\_NAND\_RE\_B\_GPIO3\_IO15  
IOMUXC\_NAND\_READY\_B\_RAWNAND\_READY\_B  
IOMUXC\_NAND\_READY\_B\_GPIO3\_IO16  
IOMUXC\_NAND\_WE\_B\_RAWNAND\_WE\_B  
IOMUXC\_NAND\_WE\_B\_GPIO3\_IO17  
IOMUXC\_NAND\_WP\_B\_RAWNAND\_WP\_B  
IOMUXC\_NAND\_WP\_B\_GPIO3\_IO18  
IOMUXC\_SAI5\_RXFS\_SAI5\_RX\_SYNC  
IOMUXC\_SAI5\_RXFS\_SAI1\_TX\_DATA0  
IOMUXC\_SAI5\_RXFS\_GPIO3\_IO19  
IOMUXC\_SAI5\_RXC\_SAI5\_RX\_BCLK  
IOMUXC\_SAI5\_RXC\_SAI1\_TX\_DATA1  
IOMUXC\_SAI5\_RXC\_GPIO3\_IO20  
IOMUXC\_SAI5\_RXD0\_SAI5\_RX\_DATA0  
IOMUXC\_SAI5\_RXD0\_SAI1\_TX\_DATA2  
IOMUXC\_SAI5\_RXD0\_GPIO3\_IO21  
IOMUXC\_SAI5\_RXD1\_SAI5\_RX\_DATA1  
IOMUXC\_SAI5\_RXD1\_SAI1\_TX\_DATA3  
IOMUXC\_SAI5\_RXD1\_SAI1\_TX\_SYNC  
IOMUXC\_SAI5\_RXD1\_SAI5\_TX\_SYNC  
IOMUXC\_SAI5\_RXD1\_GPIO3\_IO22  
IOMUXC\_SAI5\_RXD2\_SAI5\_RX\_DATA2  
IOMUXC\_SAI5\_RXD2\_SAI1\_TX\_DATA4  
IOMUXC\_SAI5\_RXD2\_SAI1\_TX\_SYNC  
IOMUXC\_SAI5\_RXD2\_SAI5\_TX\_BCLK  
IOMUXC\_SAI5\_RXD2\_GPIO3\_IO23  
IOMUXC\_SAI5\_RXD3\_SAI5\_RX\_DATA3  
IOMUXC\_SAI5\_RXD3\_SAI1\_TX\_DATA5  
IOMUXC\_SAI5\_RXD3\_SAI1\_TX\_SYNC  
IOMUXC\_SAI5\_RXD3\_SAI5\_TX\_DATA0  
IOMUXC\_SAI5\_RXD3\_GPIO3\_IO24  
IOMUXC\_SAI5\_MCLK\_SAI5\_MCLK  
IOMUXC\_SAI5\_MCLK\_SAI1\_TX\_BCLK

IOMUXC\_SAI5\_MCLK\_SAI4\_MCLK  
IOMUXC\_SAI5\_MCLK\_GPIO3\_IO25  
IOMUXC\_SAI1\_RXFS\_SAI1\_RX\_SYNC  
IOMUXC\_SAI1\_RXFS\_SAI5\_RX\_SYNC  
IOMUXC\_SAI1\_RXFS\_CORESIGHT\_TRACE\_CLK  
IOMUXC\_SAI1\_RXFS\_GPIO4\_IO00  
IOMUXC\_SAI1\_RXC\_SAI1\_RX\_BCLK  
IOMUXC\_SAI1\_RXC\_SAI5\_RX\_BCLK  
IOMUXC\_SAI1\_RXC\_CORESIGHT\_TRACE\_CTL  
IOMUXC\_SAI1\_RXC\_GPIO4\_IO01  
IOMUXC\_SAI1\_RXD0\_SAI1\_RX\_DATA0  
IOMUXC\_SAI1\_RXD0\_SAI5\_RX\_DATA0  
IOMUXC\_SAI1\_RXD0\_CORESIGHT\_TRACE0  
IOMUXC\_SAI1\_RXD0\_GPIO4\_IO02  
IOMUXC\_SAI1\_RXD0\_SRC\_BOOT\_CFG0  
IOMUXC\_SAI1\_RXD1\_SAI1\_RX\_DATA1  
IOMUXC\_SAI1\_RXD1\_SAI5\_RX\_DATA1  
IOMUXC\_SAI1\_RXD1\_CORESIGHT\_TRACE1  
IOMUXC\_SAI1\_RXD1\_GPIO4\_IO03  
IOMUXC\_SAI1\_RXD1\_SRC\_BOOT\_CFG1  
IOMUXC\_SAI1\_RXD2\_SAI1\_RX\_DATA2  
IOMUXC\_SAI1\_RXD2\_SAI5\_RX\_DATA2  
IOMUXC\_SAI1\_RXD2\_CORESIGHT\_TRACE2  
IOMUXC\_SAI1\_RXD2\_GPIO4\_IO04  
IOMUXC\_SAI1\_RXD2\_SRC\_BOOT\_CFG2  
IOMUXC\_SAI1\_RXD3\_SAI1\_RX\_DATA3  
IOMUXC\_SAI1\_RXD3\_SAI5\_RX\_DATA3  
IOMUXC\_SAI1\_RXD3\_CORESIGHT\_TRACE3  
IOMUXC\_SAI1\_RXD3\_GPIO4\_IO05  
IOMUXC\_SAI1\_RXD3\_SRC\_BOOT\_CFG3  
IOMUXC\_SAI1\_RXD4\_SAI1\_RX\_DATA4  
IOMUXC\_SAI1\_RXD4\_SAI6\_TX\_BCLK  
IOMUXC\_SAI1\_RXD4\_SAI6\_RX\_BCLK

IOMUXC\_SAI1\_RXD4\_CORESIGHT\_TRACE4  
IOMUXC\_SAI1\_RXD4\_GPIO4\_IO06  
IOMUXC\_SAI1\_RXD4\_SRC\_BOOT\_CFG4  
IOMUXC\_SAI1\_RXD5\_SAI1\_RX\_DATA5  
IOMUXC\_SAI1\_RXD5\_SAI6\_TX\_DATA0  
IOMUXC\_SAI1\_RXD5\_SAI6\_RX\_DATA0  
IOMUXC\_SAI1\_RXD5\_SAI1\_RX\_SYNC  
IOMUXC\_SAI1\_RXD5\_CORESIGHT\_TRACE5  
IOMUXC\_SAI1\_RXD5\_GPIO4\_IO07  
IOMUXC\_SAI1\_RXD5\_SRC\_BOOT\_CFG5  
IOMUXC\_SAI1\_RXD6\_SAI1\_RX\_DATA6  
IOMUXC\_SAI1\_RXD6\_SAI6\_TX\_SYNC  
IOMUXC\_SAI1\_RXD6\_SAI6\_RX\_SYNC  
IOMUXC\_SAI1\_RXD6\_CORESIGHT\_TRACE6  
IOMUXC\_SAI1\_RXD6\_GPIO4\_IO08  
IOMUXC\_SAI1\_RXD6\_SRC\_BOOT\_CFG6  
IOMUXC\_SAI1\_RXD7\_SAI1\_RX\_DATA7  
IOMUXC\_SAI1\_RXD7\_SAI6\_MCLK  
IOMUXC\_SAI1\_RXD7\_SAI1\_TX\_SYNC  
IOMUXC\_SAI1\_RXD7\_SAI1\_TX\_DATA4  
IOMUXC\_SAI1\_RXD7\_CORESIGHT\_TRACE7  
IOMUXC\_SAI1\_RXD7\_GPIO4\_IO09  
IOMUXC\_SAI1\_RXD7\_SRC\_BOOT\_CFG7  
IOMUXC\_SAI1\_TXFS\_SAI1\_TX\_SYNC  
IOMUXC\_SAI1\_TXFS\_SAI5\_TX\_SYNC  
IOMUXC\_SAI1\_TXFS\_CORESIGHT\_EVENT0  
IOMUXC\_SAI1\_TXFS\_GPIO4\_IO10  
IOMUXC\_SAI1\_TXC\_SAI1\_TX\_BCLK  
IOMUXC\_SAI1\_TXC\_SAI5\_TX\_BCLK  
IOMUXC\_SAI1\_TXC\_CORESIGHT\_EVENT1  
IOMUXC\_SAI1\_TXC\_GPIO4\_IO11  
IOMUXC\_SAI1\_TXD0\_SAI1\_TX\_DATA0  
IOMUXC\_SAI1\_TXD0\_SAI5\_TX\_DATA0

IOMUXC\_SAI1\_TXD0\_CORESIGHT\_TRACE8  
IOMUXC\_SAI1\_TXD0\_GPIO4\_IO12  
IOMUXC\_SAI1\_TXD0\_SRC\_BOOT\_CFG8  
IOMUXC\_SAI1\_TXD1\_SAI1\_TX\_DATA1  
IOMUXC\_SAI1\_TXD1\_SAI5\_TX\_DATA1  
IOMUXC\_SAI1\_TXD1\_CORESIGHT\_TRACE9  
IOMUXC\_SAI1\_TXD1\_GPIO4\_IO13  
IOMUXC\_SAI1\_TXD1\_SRC\_BOOT\_CFG9  
IOMUXC\_SAI1\_TXD2\_SAI1\_TX\_DATA2  
IOMUXC\_SAI1\_TXD2\_SAI5\_TX\_DATA2  
IOMUXC\_SAI1\_TXD2\_CORESIGHT\_TRACE10  
IOMUXC\_SAI1\_TXD2\_GPIO4\_IO14  
IOMUXC\_SAI1\_TXD2\_SRC\_BOOT\_CFG10  
IOMUXC\_SAI1\_TXD3\_SAI1\_TX\_DATA3  
IOMUXC\_SAI1\_TXD3\_SAI5\_TX\_DATA3  
IOMUXC\_SAI1\_TXD3\_CORESIGHT\_TRACE11  
IOMUXC\_SAI1\_TXD3\_GPIO4\_IO15  
IOMUXC\_SAI1\_TXD3\_SRC\_BOOT\_CFG11  
IOMUXC\_SAI1\_TXD4\_SAI1\_TX\_DATA4  
IOMUXC\_SAI1\_TXD4\_SAI6\_RX\_BCLK  
IOMUXC\_SAI1\_TXD4\_SAI6\_TX\_BCLK  
IOMUXC\_SAI1\_TXD4\_CORESIGHT\_TRACE12  
IOMUXC\_SAI1\_TXD4\_GPIO4\_IO16  
IOMUXC\_SAI1\_TXD4\_SRC\_BOOT\_CFG12  
IOMUXC\_SAI1\_TXD5\_SAI1\_TX\_DATA5  
IOMUXC\_SAI1\_TXD5\_SAI6\_RX\_DATA0  
IOMUXC\_SAI1\_TXD5\_SAI6\_TX\_DATA0  
IOMUXC\_SAI1\_TXD5\_CORESIGHT\_TRACE13  
IOMUXC\_SAI1\_TXD5\_GPIO4\_IO17  
IOMUXC\_SAI1\_TXD5\_SRC\_BOOT\_CFG13  
IOMUXC\_SAI1\_TXD6\_SAI1\_TX\_DATA6  
IOMUXC\_SAI1\_TXD6\_SAI6\_RX\_SYNC  
IOMUXC\_SAI1\_TXD6\_SAI6\_TX\_SYNC

IOMUXC\_SAI1\_TXD6\_CORESIGHT\_TRACE14  
IOMUXC\_SAI1\_TXD6\_GPIO4\_IO18  
IOMUXC\_SAI1\_TXD6\_SRC\_BOOT\_CFG14  
IOMUXC\_SAI1\_TXD7\_SAI1\_TX\_DATA7  
IOMUXC\_SAI1\_TXD7\_SAI6\_MCLK  
IOMUXC\_SAI1\_TXD7\_CORESIGHT\_TRACE15  
IOMUXC\_SAI1\_TXD7\_GPIO4\_IO19  
IOMUXC\_SAI1\_TXD7\_SRC\_BOOT\_CFG15  
IOMUXC\_SAI1\_MCLK\_SAI1\_MCLK  
IOMUXC\_SAI1\_MCLK\_SAI5\_MCLK  
IOMUXC\_SAI1\_MCLK\_SAI1\_TX\_BCLK  
IOMUXC\_SAI1\_MCLK\_GPIO4\_IO20  
IOMUXC\_SAI2\_RXFS\_SAI2\_RX\_SYNC  
IOMUXC\_SAI2\_RXFS\_SAI5\_TX\_SYNC  
IOMUXC\_SAI2\_RXFS\_GPIO4\_IO21  
IOMUXC\_SAI2\_RXC\_SAI2\_RX\_BCLK  
IOMUXC\_SAI2\_RXC\_SAI5\_TX\_BCLK  
IOMUXC\_SAI2\_RXC\_GPIO4\_IO22  
IOMUXC\_SAI2\_RXD0\_SAI2\_RX\_DATA0  
IOMUXC\_SAI2\_RXD0\_SAI5\_TX\_DATA0  
IOMUXC\_SAI2\_RXD0\_GPIO4\_IO23  
IOMUXC\_SAI2\_TXFS\_SAI2\_TX\_SYNC  
IOMUXC\_SAI2\_TXFS\_SAI5\_TX\_DATA1  
IOMUXC\_SAI2\_TXFS\_GPIO4\_IO24  
IOMUXC\_SAI2\_TXC\_SAI2\_TX\_BCLK  
IOMUXC\_SAI2\_TXC\_SAI5\_TX\_DATA2  
IOMUXC\_SAI2\_TXC\_GPIO4\_IO25  
IOMUXC\_SAI2\_TXD0\_SAI2\_TX\_DATA0  
IOMUXC\_SAI2\_TXD0\_SAI5\_TX\_DATA3  
IOMUXC\_SAI2\_TXD0\_GPIO4\_IO26  
IOMUXC\_SAI2\_MCLK\_SAI2\_MCLK  
IOMUXC\_SAI2\_MCLK\_SAI5\_MCLK  
IOMUXC\_SAI2\_MCLK\_GPIO4\_IO27

IOMUXC\_SAI3\_RXFS\_SAI3\_RX\_SYNC  
IOMUXC\_SAI3\_RXFS\_GPT1\_CAPTURE1  
IOMUXC\_SAI3\_RXFS\_SAI5\_RX\_SYNC  
IOMUXC\_SAI3\_RXFS\_GPIO4\_IO28  
IOMUXC\_SAI3\_RXC\_SAI3\_RX\_BCLK  
IOMUXC\_SAI3\_RXC\_GPT1\_CAPTURE2  
IOMUXC\_SAI3\_RXC\_SAI5\_RX\_BCLK  
IOMUXC\_SAI3\_RXC\_GPIO4\_IO29  
IOMUXC\_SAI3\_RXD\_SAI3\_RX\_DATA0  
IOMUXC\_SAI3\_RXD\_GPT1\_COMPARE1  
IOMUXC\_SAI3\_RXD\_SAI5\_RX\_DATA0  
IOMUXC\_SAI3\_RXD\_GPIO4\_IO30  
IOMUXC\_SAI3\_TXFS\_SAI3\_TX\_SYNC  
IOMUXC\_SAI3\_TXFS\_GPT1\_CLK  
IOMUXC\_SAI3\_TXFS\_SAI5\_RX\_DATA1  
IOMUXC\_SAI3\_TXFS\_GPIO4\_IO31  
IOMUXC\_SAI3\_TXC\_SAI3\_TX\_BCLK  
IOMUXC\_SAI3\_TXC\_GPT1\_COMPARE2  
IOMUXC\_SAI3\_TXC\_SAI5\_RX\_DATA2  
IOMUXC\_SAI3\_TXC\_GPIO5\_IO00  
IOMUXC\_SAI3\_TXD\_SAI3\_TX\_DATA0  
IOMUXC\_SAI3\_TXD\_GPT1\_COMPARE3  
IOMUXC\_SAI3\_TXD\_SAI5\_RX\_DATA3  
IOMUXC\_SAI3\_TXD\_GPIO5\_IO01  
IOMUXC\_SAI3\_MCLK\_SAI3\_MCLK  
IOMUXC\_SAI3\_MCLK\_PWM4\_OUT  
IOMUXC\_SAI3\_MCLK\_SAI5\_MCLK  
IOMUXC\_SAI3\_MCLK\_GPIO5\_IO02  
IOMUXC\_SPDIF\_TX\_SPDIF1\_OUT  
IOMUXC\_SPDIF\_TX\_PWM3\_OUT  
IOMUXC\_SPDIF\_TX\_GPIO5\_IO03  
IOMUXC\_SPDIF\_RX\_SPDIF1\_IN  
IOMUXC\_SPDIF\_RX\_PWM2\_OUT

IOMUXC\_SPDIF\_RX\_GPIO5\_IO04  
IOMUXC\_SPDIF\_EXT\_CLK\_SPDIF1\_EXT\_CLK  
IOMUXC\_SPDIF\_EXT\_CLK\_PWM1\_OUT  
IOMUXC\_SPDIF\_EXT\_CLK\_GPIO5\_IO05  
IOMUXC\_ECSP11\_SCLK\_ECSP11\_SCLK  
IOMUXC\_ECSP11\_SCLK\_UART3\_RX  
IOMUXC\_ECSP11\_SCLK\_UART3\_TX  
IOMUXC\_ECSP11\_SCLK\_GPIO5\_IO06  
IOMUXC\_ECSP11\_MOSI\_ECSP11\_MOSI  
IOMUXC\_ECSP11\_MOSI\_UART3\_TX  
IOMUXC\_ECSP11\_MOSI\_UART3\_RX  
IOMUXC\_ECSP11\_MOSI\_GPIO5\_IO07  
IOMUXC\_ECSP11\_MISO\_ECSP11\_MISO  
IOMUXC\_ECSP11\_MISO\_UART3\_CTS\_B  
IOMUXC\_ECSP11\_MISO\_UART3\_RTS\_B  
IOMUXC\_ECSP11\_MISO\_GPIO5\_IO08  
IOMUXC\_ECSP11\_SS0\_ECSP11\_SS0  
IOMUXC\_ECSP11\_SS0\_UART3\_RTS\_B  
IOMUXC\_ECSP11\_SS0\_UART3\_CTS\_B  
IOMUXC\_ECSP11\_SS0\_GPIO5\_IO09  
IOMUXC\_ECSP12\_SCLK\_ECSP12\_SCLK  
IOMUXC\_ECSP12\_SCLK\_UART4\_RX  
IOMUXC\_ECSP12\_SCLK\_UART4\_TX  
IOMUXC\_ECSP12\_SCLK\_GPIO5\_IO10  
IOMUXC\_ECSP12\_MOSI\_ECSP12\_MOSI  
IOMUXC\_ECSP12\_MOSI\_UART4\_TX  
IOMUXC\_ECSP12\_MOSI\_UART4\_RX  
IOMUXC\_ECSP12\_MOSI\_GPIO5\_IO11  
IOMUXC\_ECSP12\_MISO\_ECSP12\_MISO  
IOMUXC\_ECSP12\_MISO\_UART4\_CTS\_B  
IOMUXC\_ECSP12\_MISO\_UART4\_RTS\_B  
IOMUXC\_ECSP12\_MISO\_GPIO5\_IO12  
IOMUXC\_ECSP12\_SS0\_ECSP12\_SS0

IOMUXC\_ECSPi2\_SS0\_UART4\_RTS\_B  
IOMUXC\_ECSPi2\_SS0\_UART4\_CTS\_B  
IOMUXC\_ECSPi2\_SS0\_GPIO5\_IO13  
IOMUXC\_I2C1\_SCL\_I2C1\_SCL  
IOMUXC\_I2C1\_SCL\_ENET1\_MDC  
IOMUXC\_I2C1\_SCL\_GPIO5\_IO14  
IOMUXC\_I2C1\_SDA\_I2C1\_SDA  
IOMUXC\_I2C1\_SDA\_ENET1\_MDIO  
IOMUXC\_I2C1\_SDA\_GPIO5\_IO15  
IOMUXC\_I2C2\_SCL\_I2C2\_SCL  
IOMUXC\_I2C2\_SCL\_ENET1\_1588\_EVENT1\_IN  
IOMUXC\_I2C2\_SCL\_GPIO5\_IO16  
IOMUXC\_I2C2\_SDA\_I2C2\_SDA  
IOMUXC\_I2C2\_SDA\_ENET1\_1588\_EVENT1\_OUT  
IOMUXC\_I2C2\_SDA\_GPIO5\_IO17  
IOMUXC\_I2C3\_SCL\_I2C3\_SCL  
IOMUXC\_I2C3\_SCL\_PWM4\_OUT  
IOMUXC\_I2C3\_SCL\_GPT2\_CLK  
IOMUXC\_I2C3\_SCL\_GPIO5\_IO18  
IOMUXC\_I2C3\_SDA\_I2C3\_SDA  
IOMUXC\_I2C3\_SDA\_PWM3\_OUT  
IOMUXC\_I2C3\_SDA\_GPT3\_CLK  
IOMUXC\_I2C3\_SDA\_GPIO5\_IO19  
IOMUXC\_I2C4\_SCL\_I2C4\_SCL  
IOMUXC\_I2C4\_SCL\_PWM2\_OUT  
IOMUXC\_I2C4\_SCL\_PCIE1\_CLKREQ\_B  
IOMUXC\_I2C4\_SCL\_GPIO5\_IO20  
IOMUXC\_I2C4\_SDA\_I2C4\_SDA  
IOMUXC\_I2C4\_SDA\_PWM1\_OUT  
IOMUXC\_I2C4\_SDA\_PCIE2\_CLKREQ\_B  
IOMUXC\_I2C4\_SDA\_GPIO5\_IO21  
IOMUXC\_UART1\_RXD\_UART1\_RX  
IOMUXC\_UART1\_RXD\_UART1\_TX



IOMUXC\_UART1\_RXD\_ECSP13\_SCLK  
IOMUXC\_UART1\_RXD\_GPIO5\_IO22  
IOMUXC\_UART1\_TXD\_UART1\_TX  
IOMUXC\_UART1\_TXD\_UART1\_RX  
IOMUXC\_UART1\_TXD\_ECSP13\_MOSI  
IOMUXC\_UART1\_TXD\_GPIO5\_IO23  
IOMUXC\_UART2\_RXD\_UART2\_RX  
IOMUXC\_UART2\_RXD\_UART2\_TX  
IOMUXC\_UART2\_RXD\_ECSP13\_MISO  
IOMUXC\_UART2\_RXD\_GPIO5\_IO24  
IOMUXC\_UART2\_TXD\_UART2\_TX  
IOMUXC\_UART2\_TXD\_UART2\_RX  
IOMUXC\_UART2\_TXD\_ECSP13\_SS0  
IOMUXC\_UART2\_TXD\_GPIO5\_IO25  
IOMUXC\_UART3\_RXD\_UART3\_RX  
IOMUXC\_UART3\_RXD\_UART3\_TX  
IOMUXC\_UART3\_RXD\_UART1\_CTS\_B  
IOMUXC\_UART3\_RXD\_UART1\_RTS\_B  
IOMUXC\_UART3\_RXD\_GPIO5\_IO26  
IOMUXC\_UART3\_TXD\_UART3\_TX  
IOMUXC\_UART3\_TXD\_UART3\_RX  
IOMUXC\_UART3\_TXD\_UART1\_RTS\_B  
IOMUXC\_UART3\_TXD\_UART1\_CTS\_B  
IOMUXC\_UART3\_TXD\_GPIO5\_IO27  
IOMUXC\_UART4\_RXD\_UART4\_RX  
IOMUXC\_UART4\_RXD\_UART4\_TX  
IOMUXC\_UART4\_RXD\_UART2\_CTS\_B  
IOMUXC\_UART4\_RXD\_UART2\_RTS\_B  
IOMUXC\_UART4\_RXD\_PCIE1\_CLKREQ\_B  
IOMUXC\_UART4\_RXD\_GPIO5\_IO28  
IOMUXC\_UART4\_TXD\_UART4\_TX  
IOMUXC\_UART4\_TXD\_UART4\_RX  
IOMUXC\_UART4\_TXD\_UART2\_RTS\_B

IOMUXC\_UART4\_TXD\_UART2\_CTS\_B  
IOMUXC\_UART4\_TXD\_PCIE2\_CLKREQ\_B  
IOMUXC\_UART4\_TXD\_GPIO5\_IO29  
IOMUXC\_TEST\_MODE  
IOMUXC\_BOOT\_MODE0  
IOMUXC\_BOOT\_MODE1  
IOMUXC\_JTAG\_MOD  
IOMUXC\_JTAG\_TRST\_B  
IOMUXC\_JTAG\_TDI  
IOMUXC\_JTAG\_TMS  
IOMUXC\_JTAG\_TCK  
IOMUXC\_JTAG\_TDO  
IOMUXC\_RTC  
FSL\_COMPONENT\_ID

## 2.13 IRQSTEER: Interrupt Request Steering Driver

`void IRQSTEER_Init(IRQSTEER_Type *base)`

Initializes the IRQSTEER module.

This function enables the clock gate for the specified IRQSTEER.

### Parameters

- `base` – IRQSTEER peripheral base address.

`void IRQSTEER_Deinit(IRQSTEER_Type *base)`

Deinitializes an IRQSTEER instance for operation.

The clock gate for the specified IRQSTEER is disabled.

### Parameters

- `base` – IRQSTEER peripheral base address.

`static inline void IRQSTEER_EnableInterrupt(IRQSTEER_Type *base, IRQn_Type irq)`

Enables an interrupt source.

### Parameters

- `base` – IRQSTEER peripheral base address.
- `irq` – Interrupt to be routed. The interrupt must be an IRQSTEER source.

`static inline void IRQSTEER_DisableInterrupt(IRQSTEER_Type *base, IRQn_Type irq)`

Disables an interrupt source.

### Parameters

- `base` – IRQSTEER peripheral base address.
- `irq` – Interrupt source number. The interrupt must be an IRQSTEER source.

```
static inline bool IRQSTEER_InterruptIsEnabled(IRQSTEER_Type *base, IRQn_Type irq)
```

Check if an interrupt source is enabled.

#### Parameters

- *base* – IRQSTEER peripheral base address.
- *irq* – Interrupt to be queried. The interrupt must be an IRQSTEER source.

#### Returns

true if the interrupt is not masked, false otherwise.

```
static inline void IRQSTEER_SetInterrupt(IRQSTEER_Type *base, IRQn_Type irq, bool set)
```

Sets/Forces an interrupt.

---

**Note:** This function is not affected by the function `IRQSTEER_DisableInterrupt` and `IRQSTEER_EnableInterrupt`.

---

#### Parameters

- *base* – IRQSTEER peripheral base address.
- *irq* – Interrupt to be set/forced. The interrupt must be an IRQSTEER source.
- *set* – Switcher of the interrupt set/force function. “true” means to set. “false” means not (normal operation).

```
static inline void IRQSTEER_EnableMasterInterrupt(IRQSTEER_Type *base,
                                                  irqsteer_int_master_t intMasterIndex)
```

Enables a master interrupt. By default, all the master interrupts are enabled.

For example, to enable the interrupt sources of master 1:

```
IRQSTEER_EnableMasterInterrupt(IRQSTEER_M4_0, kIRQSTEER_InterruptMaster1);
```

#### Parameters

- *base* – IRQSTEER peripheral base address.
- *intMasterIndex* – Master index of interrupt sources to be routed, options available in enumeration `irqsteer_int_master_t`.

```
static inline void IRQSTEER_DisableMasterInterrupt(IRQSTEER_Type *base,
                                                  irqsteer_int_master_t intMasterIndex)
```

Disables a master interrupt.

For example, to disable the interrupt sources of master 1:

```
IRQSTEER_DisableMasterInterrupt(IRQSTEER_M4_0, kIRQSTEER_InterruptMaster1);
```

#### Parameters

- *base* – IRQSTEER peripheral base address.
- *intMasterIndex* – Master index of interrupt sources to be disabled, options available in enumeration `irqsteer_int_master_t`.

static inline bool IRQSTEER\_IsInterruptSet(IRQSTEER\_Type \*base, IRQn\_Type irq)  
Checks the status of one specific IRQSTEER interrupt.

For example, to check whether interrupt from output 0 of Display 1 is set:

```
if (IRQSTEER_IsInterruptSet(IRQSTEER_DISPLAY1_INT_OUT0)
{
    ...
}
```

#### Parameters

- base – IRQSTEER peripheral base address.
- irq – Interrupt source status to be checked. The interrupt must be an IRQSTEER source.

#### Returns

The interrupt status. “true” means interrupt set. “false” means not.

static inline bool IRQSTEER\_IsMasterInterruptSet(IRQSTEER\_Type \*base)  
Checks the status of IRQSTEER master interrupt. The master interrupt status represents at least one interrupt is asserted or not among ALL interrupts.

---

**Note:** The master interrupt status is not affected by the function IRQSTEER\_DisableMasterInterrupt.

---

#### Parameters

- base – IRQSTEER peripheral base address.

#### Returns

The master interrupt status. “true” means at least one interrupt set. “false” means not.

static inline uint32\_t IRQSTEER\_GetGroupInterruptStatus(IRQSTEER\_Type \*base,  
irqsteer\_int\_group\_t intGroupIndex)

Gets the status of IRQSTEER group interrupt. The group interrupt status represents all the interrupt status within the group specified. This API aims for facilitating the status return of one set of interrupts.

#### Parameters

- base – IRQSTEER peripheral base address.
- intGroupIndex – Index of the interrupt group status to get.

#### Returns

The mask of the group interrupt status. Bit[n] set means the source with bit offset n in group intGroupIndex of IRQSTEER is asserted.

IRQn\_Type IRQSTEER\_GetMasterNextInterrupt(IRQSTEER\_Type \*base, irqsteer\_int\_master\_t  
intMasterIndex)

Gets the next interrupt source (currently set) of one specific master.

#### Parameters

- base – IRQSTEER peripheral base address.
- intMasterIndex – Master index of interrupt sources, options available in enumeration irqsteer\_int\_master\_t.

**Returns**

The current set next interrupt source number of one specific master. Return IRQSTEER\_INT\_Invalid if no interrupt set.

```
uint32_t IRQSTEER_GetMasterIrqCount(IRQSTEER_Type *base, irqsteer_int_master_t
                                   intMasterIndex)
```

Get the number of interrupt for a given master.

**Parameters**

- base – IRQSTEER peripheral base address.
- intMasterIndex – Master index of interrupt sources, options available in enumeration irqsteer\_int\_master\_t.

**Returns**

Number of interrupts for a given master.

```
uint64_t IRQSTEER_GetMasterInterruptsStatus(IRQSTEER_Type *base, irqsteer_int_master_t
                                             intMasterIndex)
```

Get the status of the interrupts a master is in charge of.

What this function does is it takes the CHn\_STATUS registers associated with the interrupts a master is in charge of and puts them in 64-bit variable. The order they are put in the 64-bit variable is the following: CHn\_STATUS[i] : CHn\_STATUS[i + 1], where CHn\_STATUS[i + 1] is placed in the least significant half of the 64-bit variable. Assuming a master is in charge of 64 interrupts, the user may use the result of this function as such: BIT(i) & IRQSTEER\_GetMasterInterrupts() to check if interrupt i is asserted.

**Parameters**

- base – IRQSTEER peripheral base address.
- intMasterIndex – Master index of interrupt sources, options available in enumeration irqsteer\_int\_master\_t.

**Returns**

64-bit variable containing the status of the interrupts a master is in charge of.

```
FSL_IRQSTEER_DRIVER_VERSION
```

Driver version.

```
enum _irqsteer_int_group
```

IRQSTEER interrupt groups.

*Values:*

```
enumerator kIRQSTEER_InterruptGroup0
```

Interrupt Group 0: interrupt source 31 - 0

```
enumerator kIRQSTEER_InterruptGroup1
```

Interrupt Group 1: interrupt source 63 - 32

```
enumerator kIRQSTEER_InterruptGroup2
```

Interrupt Group 2: interrupt source 95 - 64

```
enumerator kIRQSTEER_InterruptGroup3
```

Interrupt Group 3: interrupt source 127 - 96

```
enumerator kIRQSTEER_InterruptGroup4
```

Interrupt Group 4: interrupt source 159 - 128

```
enumerator kIRQSTEER_InterruptGroup5
```

Interrupt Group 5: interrupt source 191 - 160

enumerator kIRQSTEER\_InterruptGroup6  
Interrupt Group 6: interrupt source 223 - 192

enumerator kIRQSTEER\_InterruptGroup7  
Interrupt Group 7: interrupt source 255 - 224

enumerator kIRQSTEER\_InterruptGroup8  
Interrupt Group 8: interrupt source 287 - 256

enumerator kIRQSTEER\_InterruptGroup9  
Interrupt Group 9: interrupt source 319 - 288

enumerator kIRQSTEER\_InterruptGroup10  
Interrupt Group 10: interrupt source 351 - 320

enumerator kIRQSTEER\_InterruptGroup11  
Interrupt Group 11: interrupt source 383 - 352

enumerator kIRQSTEER\_InterruptGroup12  
Interrupt Group 12: interrupt source 415 - 384

enumerator kIRQSTEER\_InterruptGroup13  
Interrupt Group 13: interrupt source 447 - 416

enumerator kIRQSTEER\_InterruptGroup14  
Interrupt Group 14: interrupt source 479 - 448

enumerator kIRQSTEER\_InterruptGroup15  
Interrupt Group 15: interrupt source 511 - 480

enum \_irqsteer\_int\_master  
IRQSTEER master interrupts mapping.

*Values:*

enumerator kIRQSTEER\_InterruptMaster0  
Interrupt Master 0: interrupt source 63 - 0

enumerator kIRQSTEER\_InterruptMaster1  
Interrupt Master 1: interrupt source 127 - 64

enumerator kIRQSTEER\_InterruptMaster2  
Interrupt Master 2: interrupt source 191 - 128

enumerator kIRQSTEER\_InterruptMaster3  
Interrupt Master 3: interrupt source 255 - 192

enumerator kIRQSTEER\_InterruptMaster4  
Interrupt Master 4: interrupt source 319 - 256

enumerator kIRQSTEER\_InterruptMaster5  
Interrupt Master 5: interrupt source 383 - 320

enumerator kIRQSTEER\_InterruptMaster6  
Interrupt Master 6: interrupt source 447 - 384

enumerator kIRQSTEER\_InterruptMaster7  
Interrupt Master 7: interrupt source 511 - 448

typedef enum \_irqsteer\_int\_group irqsteer\_int\_group\_t  
IRQSTEER interrupt groups.

`typedef enum _irqsteer_int_master irqsteer_int_master_t`

IRQSTEER master interrupts mapping.

`FSL_IRQSTEER_USE_DRIVER_IRQ_HANDLER`

Use the IRQSTEER driver IRQ Handler or not.

When define as 1, IRQSTEER driver implements the IRQSTEER ISR, otherwise user shall implement it. Currently the IRQSTEER ISR is only available for Cortex-M platforms.

`FSL_IRQSTEER_ENABLE_MASTER_INT`

IRQSTEER\_Init/IRQSTEER\_Deinit enables/disables IRQSTEER master interrupt or not.

When define as 1, IRQSTEER\_Init will enable the IRQSTEER master interrupt in system level interrupt controller (such as NVIC, GIC), IRQSTEER\_Deinit will disable it. Otherwise IRQSTEER\_Init/IRQSTEER\_Deinit won't touch.

`IRQSTEER_INT_SRC_REG_WIDTH`

IRQSTEER interrupt source register width.

`IRQSTEER_INT_MASTER_AGGREGATED_INT_NUM`

IRQSTEER aggregated interrupt source number for each master.

`IRQSTEER_INT_SRC_REG_INDEX(irq)`

IRQSTEER interrupt source mapping register index.

`IRQSTEER_INT_SRC_BIT_OFFSET(irq)`

IRQSTEER interrupt source mapping bit offset.

`IRQSTEER_INT_SRC_NUM(regIndex, bitOffset)`

IRQSTEER interrupt source number.

## 2.14 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 specifled by *bits* to the variable at address *address*.

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

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

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

Toggle the bits specifled 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 specifled by *clearBits* and set the bits specifled by *setBits*.

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

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

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

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

USEC\_TO\_COUNT(us, clockFreqInHz)

Macro to convert a microsecond period to raw count value

COUNT\_TO\_USEC(count, clockFreqInHz)

Macro to convert a raw count value to microsecond

MSEC\_TO\_COUNT(ms, clockFreqInHz)

Macro to convert a millisecond period to raw count value

COUNT\_TO\_MSEC(count, clockFreqInHz)

Macro to convert a raw count value to millisecond

SDK\_ISR\_EXIT\_BARRIER

SDK\_ALIGN(var, alignbytes)

Macro to define a variable with alignbytes alignment



SDK\_SIZEALIGN(var, alignbytes)

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

Macro to define a variable with L2 cache line size alignment

Macro to change a value to a given size aligned value

AT\_NONCACHEABLE\_SECTION(var)

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

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

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

AT\_NONCACHEABLE\_SECTION\_INIT(var)

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

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

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

AT\_CACHE\_LINE\_SECTION(var)

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

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

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

AT\_QUICKACCESS\_SECTION\_CODE(func)

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

AT\_QUICKACCESS\_SECTION\_DATA(var)

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

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

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

RAMFUNCTION\_SECTION\_CODE(func)

Place function in ram.

enum \_status\_groups

Status group numbers.

*Values:*

enumerator kStatusGroup\_Generic

Group number for generic status codes.

enumerator kStatusGroup\_FLASH

Group number for FLASH status codes.

enumerator kStatusGroup\_LPSPI

Group number for LPSPI status codes.

enumerator kStatusGroup\_FLEXIO\_SPI

Group number for FLEXIO SPI status codes.

enumerator kStatusGroup\_DSPI

Group number for DSPI status codes.

enumerator kStatusGroup\_FLEXIO\_UART

Group number for FLEXIO UART status codes.

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

enumerator kStatusGroup\_LPI2C  
Group number for LPI2C status codes.

enumerator kStatusGroup\_UART  
Group number for UART status codes.

enumerator kStatusGroup\_I2C  
Group number for UART status codes.

enumerator kStatusGroup\_LPSCI  
Group number for LPSCI status codes.

enumerator kStatusGroup\_LPUART  
Group number for LPUART status codes.

enumerator kStatusGroup\_SPI  
Group number for SPI status code.

enumerator kStatusGroup\_XRDC  
Group number for XRDC status code.

enumerator kStatusGroup\_SEMA42  
Group number for SEMA42 status code.

enumerator kStatusGroup\_SDHC  
Group number for SDHC status code

enumerator kStatusGroup\_SDMMC  
Group number for SDMMC status code

enumerator kStatusGroup\_SAI  
Group number for SAI status code

enumerator kStatusGroup\_MCG  
Group number for MCG status codes.

enumerator kStatusGroup\_SCG  
Group number for SCG status codes.

enumerator kStatusGroup\_SDSPI  
Group number for SDSPI status codes.

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

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

enumerator kStatusGroup\_FLASHIAP  
Group number for FLASHIAP status codes

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

enumerator kStatusGroup\_I2S  
Group number for I2S status codes

enumerator kStatusGroup\_IUART  
Group number for IUART status codes

enumerator kStatusGroup\_CSI  
Group number for CSI status codes

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

enumerator kStatusGroup\_SDRAMC  
Group number for SDRAMC status codes.

enumerator kStatusGroup\_POWER  
Group number for POWER status codes.

enumerator kStatusGroup\_ENET  
Group number for ENET status codes.

enumerator kStatusGroup\_PHY  
Group number for PHY status codes.

enumerator kStatusGroup\_TRGMUX  
Group number for TRGMUX status codes.

enumerator kStatusGroup\_SMARTCARD  
Group number for SMARTCARD status codes.

enumerator kStatusGroup\_LMEM  
Group number for LMEM status codes.

enumerator kStatusGroup\_QSPI  
Group number for QSPI status codes.

enumerator kStatusGroup\_DMA  
Group number for DMA status codes.

enumerator kStatusGroup\_EDMA  
Group number for EDMA status codes.

enumerator kStatusGroup\_DMAMGR  
Group number for DMAMGR status codes.

enumerator kStatusGroup\_FLEXCAN  
Group number for FlexCAN status codes.

enumerator kStatusGroup\_LTC  
Group number for LTC status codes.

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

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

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

enumerator kStatusGroup\_DMIC  
Group number for DMIC status codes.

enumerator kStatusGroup\_SDIF  
Group number for SDIF status codes.

enumerator kStatusGroup\_SPIFI  
Group number for SPIFI status codes.

enumerator kStatusGroup\_OTP  
Group number for OTP status codes.

enumerator kStatusGroup\_MCAN  
Group number for MCAN status codes.

enumerator kStatusGroup\_CAAM  
Group number for CAAM status codes.

enumerator kStatusGroup\_ECSPI  
Group number for ECSPI status codes.

enumerator kStatusGroup\_USDHC  
Group number for USDHC status codes.

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

enumerator kStatusGroup\_DCP  
Group number for DCP status codes.

enumerator kStatusGroup\_MSCAN  
Group number for MSCAN status codes.

enumerator kStatusGroup\_ESAI  
Group number for ESAI status codes.

enumerator kStatusGroup\_FLEXSPI  
Group number for FLEXSPI status codes.

enumerator kStatusGroup\_MMDC  
Group number for MMDC status codes.

enumerator kStatusGroup\_PDM  
Group number for MIC status codes.

enumerator kStatusGroup\_SDMA  
Group number for SDMA status codes.

enumerator kStatusGroup\_ICS  
Group number for ICS status codes.

enumerator kStatusGroup\_SPDIF  
Group number for SPDIF status codes.

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

enumerator kStatusGroup\_HASHCRYPT  
Group number for Hashcrypt status codes

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

enumerator kStatusGroup\_I3C  
Group number for I3C status codes

enumerator kStatusGroup\_LPC\_I2C\_1  
Group number for LPC\_I2C\_1 status codes.

enumerator kStatusGroup\_NOTIFIER  
Group number for NOTIFIER status codes.

enumerator kStatusGroup\_\_DebugConsole  
Group number for debug console status codes.

enumerator kStatusGroup\_\_SEMC  
Group number for SEMC status codes.

enumerator kStatusGroup\_\_ApplicationRangeStart  
Starting number for application groups.

enumerator kStatusGroup\_\_IAP  
Group number for IAP status codes

enumerator kStatusGroup\_\_SFA  
Group number for SFA status codes

enumerator kStatusGroup\_\_SPC  
Group number for SPC status codes.

enumerator kStatusGroup\_\_PUF  
Group number for PUF status codes.

enumerator kStatusGroup\_\_TOUCH\_PANEL  
Group number for touch panel status codes

enumerator kStatusGroup\_\_VBAT  
Group number for VBAT status codes

enumerator kStatusGroup\_\_XSPI  
Group number for XSPI status codes

enumerator kStatusGroup\_\_PNGDEC  
Group number for PNGDEC status codes

enumerator kStatusGroup\_\_JPEGDEC  
Group number for JPEGDEC status codes

enumerator kStatusGroup\_\_AUDMIX  
Group number for AUDMIX status codes

enumerator kStatusGroup\_\_HAL\_GPIO  
Group number for HAL GPIO status codes.

enumerator kStatusGroup\_\_HAL\_UART  
Group number for HAL UART status codes.

enumerator kStatusGroup\_\_HAL\_TIMER  
Group number for HAL TIMER status codes.

enumerator kStatusGroup\_\_HAL\_SPI  
Group number for HAL SPI status codes.

enumerator kStatusGroup\_\_HAL\_I2C  
Group number for HAL I2C status codes.

enumerator kStatusGroup\_\_HAL\_FLASH  
Group number for HAL FLASH status codes.

enumerator kStatusGroup\_\_HAL\_PWM  
Group number for HAL PWM status codes.

enumerator kStatusGroup\_\_HAL\_RNG  
Group number for HAL RNG status codes.

enumerator `kStatusGroup_HAL_I2S`  
Group number for HAL I2S status codes.

enumerator `kStatusGroup_HAL_ADC_SENSOR`  
Group number for HAL ADC SENSOR status codes.

enumerator `kStatusGroup_TIMERMANAGER`  
Group number for TiMER MANAGER status codes.

enumerator `kStatusGroup_SERIALMANAGER`  
Group number for SERIAL MANAGER status codes.

enumerator `kStatusGroup_LED`  
Group number for LED status codes.

enumerator `kStatusGroup_BUTTON`  
Group number for BUTTON status codes.

enumerator `kStatusGroup_EXTERN_EEPROM`  
Group number for EXTERN EEPROM status codes.

enumerator `kStatusGroup_SHELL`  
Group number for SHELL status codes.

enumerator `kStatusGroup_MEM_MANAGER`  
Group number for MEM MANAGER status codes.

enumerator `kStatusGroup_LIST`  
Group number for List status codes.

enumerator `kStatusGroup_OSA`  
Group number for OSA status codes.

enumerator `kStatusGroup_COMMON_TASK`  
Group number for Common task status codes.

enumerator `kStatusGroup_MSG`  
Group number for messaging status codes.

enumerator `kStatusGroup_SDK_OCOTP`  
Group number for OCOTP status codes.

enumerator `kStatusGroup_SDK_FLEXSPINOR`  
Group number for FLEXSPINOR status codes.

enumerator `kStatusGroup_CODEC`  
Group number for codec status codes.

enumerator `kStatusGroup_ASRC`  
Group number for codec status ASRC.

enumerator `kStatusGroup_OTFAD`  
Group number for codec status codes.

enumerator `kStatusGroup_SDIOSLV`  
Group number for SDIOSLV status codes.

enumerator `kStatusGroup_MECC`  
Group number for MECC status codes.

enumerator `kStatusGroup_ENET_QOS`  
Group number for ENET\_QOS status codes.

enumerator kStatusGroup\_LOG  
Group number for LOG status codes.

enumerator kStatusGroup\_I3CBUS  
Group number for I3CBUS status codes.

enumerator kStatusGroup\_QSCI  
Group number for QSCI status codes.

enumerator kStatusGroup\_ELEMU  
Group number for ELEMU status codes.

enumerator kStatusGroup\_QUEUEDSPI  
Group number for QSPI status codes.

enumerator kStatusGroup\_POWER\_MANAGER  
Group number for POWER\_MANAGER status codes.

enumerator kStatusGroup\_IPED  
Group number for IPED status codes.

enumerator kStatusGroup\_ELS\_PKC  
Group number for ELS PKC status codes.

enumerator kStatusGroup\_CSS\_PKC  
Group number for CSS PKC status codes.

enumerator kStatusGroup\_HOSTIF  
Group number for HOSTIF status codes.

enumerator kStatusGroup\_CLIF  
Group number for CLIF status codes.

enumerator kStatusGroup\_BMA  
Group number for BMA status codes.

enumerator kStatusGroup\_NETC  
Group number for NETC status codes.

enumerator kStatusGroup\_ELE  
Group number for ELE status codes.

enumerator kStatusGroup\_GLIKEY  
Group number for GLIKEY status codes.

enumerator kStatusGroup\_AON\_POWER  
Group number for AON\_POWER status codes.

enumerator kStatusGroup\_AON\_COMMON  
Group number for AON\_COMMON status codes.

enumerator kStatusGroup\_ENDAT3  
Group number for ENDAT3 status codes.

enumerator kStatusGroup\_HIPERFACE  
Group number for HIPERFACE status codes.

enumerator kStatusGroup\_NPX  
Group number for NPX status codes.

enumerator kStatusGroup\_ELA\_CSEC  
Group number for ELA\_CSEC status codes.

enumerator kStatusGroup\_FLEXIO\_T\_FORMAT

Group number for T-format status codes.

enumerator kStatusGroup\_FLEXIO\_A\_FORMAT

Group number for A-format status codes.

Generic status return codes.

*Values:*

enumerator kStatus\_Success

Generic status for Success.

enumerator kStatus\_Fail

Generic status for Fail.

enumerator kStatus\_ReadOnly

Generic status for read only failure.

enumerator kStatus\_OutOfRange

Generic status for out of range access.

enumerator kStatus\_InvalidArgument

Generic status for invalid argument check.

enumerator kStatus\_Timeout

Generic status for timeout.

enumerator kStatus\_NoTransferInProgress

Generic status for no transfer in progress.

enumerator kStatus\_Busy

Generic status for module is busy.

enumerator kStatus\_NoData

Generic status for no data is found for the operation.

typedef int32\_t status\_t

Type used for all status and error return values.

void \*SDK\_Malloc(size\_t size, size\_t alignbytes)

Allocate memory with given alignment and aligned size.

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

#### **Parameters**

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

#### **Return values**

The – allocated memory.

void SDK\_Free(void \*ptr)

Free memory.

#### **Parameters**

- ptr – The memory to be release.

void SDK\_DelayAtLeastUs(uint32\_t delayTime\_us, uint32\_t coreClock\_Hz)

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



**Parameters**

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

static inline *status\_t* EnableIRQ(IRQn\_Type interrupt)

Enable specific interrupt.

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

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

**Parameters**

- interrupt – The IRQ number.

**Return values**

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

static inline *status\_t* DisableIRQ(IRQn\_Type interrupt)

Disable specific interrupt.

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

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

**Parameters**

- interrupt – The IRQ number.

**Return values**

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

static inline *status\_t* EnableIRQWithPriority(IRQn\_Type interrupt, uint8\_t priNum)

Enable the IRQ, and also set the interrupt priority.

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

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

**Parameters**

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

**Return values**

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

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

Set the IRQ priority.

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

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

#### Parameters

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

#### Return values

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

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

Clear the pending IRQ flag.

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

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

#### Parameters

- interrupt – The flag which IRQ to clear.

#### Return values

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

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

Disable the global IRQ.

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

#### Returns

Current primask value.

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

Enable the global IRQ.

Set the primask register with the provided primask value but not just enable the primask. The idea is for the convenience of integration of RTOS. some RTOS get its own management mechanism of primask. User is required to use the EnableGlobalIRQ() and DisableGlobalIRQ() in pair.

#### Parameters

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

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

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

```
FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ
```

Macro to use the default weak IRQ handler in drivers.

```
MAKE_STATUS(group, code)
```

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

```
MAKE_VERSION(major, minor, bugfix)
```

Construct the version number for drivers.

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

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

```
ARRAY_SIZE(x)
```

Computes the number of elements in an array.

```
UINT64_H(X)
```

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

```
UINT64_L(X)
```

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

```
SUPPRESS_FALL_THROUGH_WARNING()
```

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

```
MSDK_REG_SECURE_ADDR(x)
```

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

```
MSDK_REG_NONSECURE_ADDR(x)
```

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

```
MSDK_HAS_DWT_CYCCNT
```

The chip supports DWT CYCCNT or not.

```
MSDK_INVALID_IRQ_HANDLER
```

Invalid IRQ handler address.

## 2.15 LCDIF: LCD interface

```
status_t LCDIF_Init(LCDIF_Type *base)
```

Initialize the LCDIF.

This function initializes the LCDIF to work.

### Parameters

- base – LCDIF peripheral base address.

### Return values

kStatus\_Success – Initialize successfully.

`void LCDIF_Deinit(LCDIF_Type *base)`

De-initialize the LCDIF.

This function disables the LCDIF peripheral clock.

#### Parameters

- `base` – LCDIF peripheral base address.

`void LCDIF_DpiModeGetDefaultConfig(lcdif_dpi_config_t *config)`

Get the default configuration for to initialize the LCDIF.

The default configuration value is:

```
config->panelWidth = 0;
config->panelHeight = 0;
config->hsw = 0;
config->hfp = 0;
config->hbp = 0;
config->vsw = 0;
config->vfp = 0;
config->vbp = 0;
config->polarityFlags = kLCDIF_VsyncActiveLow | kLCDIF_HsyncActiveLow | kLCDIF_
↪DataEnableActiveHigh |
kLCDIF_DriveDataOnFallingClkEdge; config->format = kLCDIF_Output24Bit;
```

#### Parameters

- `config` – Pointer to the LCDIF configuration.

`status_t LCDIF_DpiModeSetConfig(LCDIF_Type *base, uint8_t displayIndex, const lcdif_dpi_config_t *config)`

Initialize the LCDIF to work in DPI mode.

This function configures the LCDIF DPI display.

#### Parameters

- `base` – LCDIF peripheral base address.
- `displayIndex` – Display index.
- `config` – Pointer to the configuration structure.

#### Return values

- `kStatus_Success` – Initialize successfully.
- `kStatus_InvalidArgument` – Initialize failed because of invalid argument.

`status_t LCDIF_DbiModeSetConfig(LCDIF_Type *base, uint8_t displayIndex, const lcdif_dbi_config_t *config)`

Initialize the LCDIF to work in DBI mode.

This function configures the LCDIF DBI display.

#### Parameters

- `base` – LCDIF peripheral base address.
- `displayIndex` – Display index.
- `config` – Pointer to the configuration structure.

#### Return values

- `kStatus_Success` – Initialize successfully.
- `kStatus_InvalidArgument` – Initialize failed because of invalid argument.

`void LCDIF_DbiModeGetDefaultConfig(lcdif_dbi_config_t *config)`

Get the default configuration to initialize the LCDIF DBI mode.

The default configuration value is:

```
config->swizzle      = kLCDIF_DbiOutSwizzleRGB;
config->format        = kLCDIF_DbiOutD8RGB332;
config->acTimeUnit    = 0;
config->type          = kLCDIF_DbiTypeA_ClockedE;
config->reversePolarity = false;
config->writeWRPeriod  = 3U;
config->writeWRAssert  = 0U;
config->writeCSAssert  = 0U;
config->writeWRDeassert = 0U;
config->writeCSDeassert = 0U;
config->typeCTas       = 1U;
config->typeCSCLTwrl   = 1U;
config->typeCSCLTwrh   = 1U;
```

### Parameters

- `config` – Pointer to the LCDIF DBI configuration.

`static inline void LCDIF_DbiReset(LCDIF_Type *base, uint8_t displayIndex)`

Reset the DBI module.

### Parameters

- `displayIndex` – Display index.
- `base` – LCDIF peripheral base address.

`void LCDIF_DbiSelectArea(LCDIF_Type *base, uint8_t displayIndex, uint16_t startX, uint16_t startY, uint16_t endX, uint16_t endY, bool isTiled)`

Select the update area in DBI mode.

### Parameters

- `base` – LCDIF peripheral base address.
- `displayIndex` – Display index.
- `startX` – X coordinate for start pixel.
- `startY` – Y coordinate for start pixel.
- `endX` – X coordinate for end pixel.
- `endY` – Y coordinate for end pixel.
- `isTiled` – true if the pixel data is tiled.

`static inline void LCDIF_DbiSendCommand(LCDIF_Type *base, uint8_t displayIndex, uint8_t cmd)`

Send command to DBI port.

### Parameters

- `base` – LCDIF peripheral base address.
- `displayIndex` – Display index.
- `cmd` – the DBI command to send.

`void LCDIF_DbiSendData(LCDIF_Type *base, uint8_t displayIndex, const uint8_t *data, uint32_t dataLen_Byte)`

brief Send data to DBI port.

Can be used to send light weight data to panel. To send pixel data in frame buffer, use LCDIF\_DbiWriteMem.

param base LCDIF peripheral base address. param displayIndex Display index. param data pointer to data buffer. param dataLen\_Byte data buffer length in byte.

```
void LCDIF__DbiSendCommandAndData(LCDIF_Type *base, uint8_t displayIndex, uint8_t cmd,
                                   const uint8_t *data, uint32_t dataLen_Byte)
```

Send command followed by data to DBI port.

#### Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- cmd – the DBI command to send.
- data – pointer to data buffer.
- dataLen\_Byte – data buffer length in byte.

```
static inline void LCDIF__DbiWriteMem(LCDIF_Type *base, uint8_t displayIndex)
```

Send pixel data in frame buffer to panel controller memory.

This function starts sending the pixel data in frame buffer to panel controller, user can monitor interrupt kLCDIF\_Display0FrameDoneInterrupt to know when then data sending finished.

#### Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.

```
void LCDIF__SetFrameBufferConfig(LCDIF_Type *base, uint8_t displayIndex, const
                                  lcdif_fb_config_t *config)
```

Configure the LCDIF frame buffer.

@Note: For LCDIF of version DC8000 there can be 3 layers in the pre-processing, compared with the older version. Apart from the video layer, there are also 2 overlay layers which shares the same configurations. Use this API to configure the legacy video layer, and use LCDIF\_SetOverlayFrameBufferConfig to configure the overlay layers.

#### Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- config – Pointer to the configuration structure.

```
void LCDIF__FrameBufferGetDefaultConfig(lcdif_fb_config_t *config)
```

Get default frame buffer configuration.

@Note: For LCDIF of version DC8000 there can be 3 layers in the pre-processing, compared with the older version. Apart from the video layer, there are also 2 overlay layers which shares the same configurations. Use this API to get the default configuration for all the 3 layers.

The default configuration is

```
config->enable = true;
config->enableGamma = false;
config->format = kLCDIF_PixelFormatRGB565;
```

**Parameters**

- config – Pointer to the configuration structure.

```
static inline void LCDIF_SetFrameBufferAddr(LCDIF_Type *base, uint8_t displayIndex, uint32_t address)
```

Set the frame buffer to LCDIF.

---

**Note:** The address must be 128 bytes aligned.

---

**Parameters**

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- address – Frame buffer address.

```
void LCDIF_SetFrameBufferStride(LCDIF_Type *base, uint8_t displayIndex, uint32_t strideBytes)
```

Set the frame buffer stride.

**Parameters**

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- strideBytes – The stride in byte.

```
void LCDIF_SetDitherConfig(LCDIF_Type *base, uint8_t displayIndex, const lcdif_dither_config_t *config)
```

Set the dither configuration.

**Parameters**

- base – LCDIF peripheral base address.
- displayIndex – Index to configure.
- config – Pointer to the configuration structure.

```
void LCDIF_SetGammaData(LCDIF_Type *base, uint8_t displayIndex, uint16_t startIndex, const uint32_t *gamma, uint16_t gammaLen)
```

Set the gamma translation values to the LCDIF gamma table.

**Parameters**

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- startIndex – Start index in the gamma table that the value will be set to.
- gamma – The gamma values to set to the gamma table in LCDIF, could be defined using LCDIF\_MAKE\_GAMMA\_VALUE.
- gammaLen – The length of the gamma.

```
static inline void LCDIF_EnableInterrupts(LCDIF_Type *base, uint32_t mask)
```

Enables LCDIF interrupt requests.

**Parameters**

- base – LCDIF peripheral base address.
- mask – The interrupts to enable, pass in as OR'ed value of `_lcdif_interrupt`.

```
static inline void LCDIF_DisableInterrupts(LCDIF_Type *base, uint32_t mask)
```

Disable LCDIF interrupt requests.

#### Parameters

- base – LCDIF peripheral base address.
- mask – The interrupts to disable, pass in as OR'ed value of `_lcdif_interrupt`.

```
static inline uint32_t LCDIF_GetAndClearInterruptPendingFlags(LCDIF_Type *base)
```

Get and clear LCDIF interrupt pending status.

---

**Note:** The interrupt must be enabled, otherwise the interrupt flags will not assert.

---

#### Parameters

- base – LCDIF peripheral base address.

#### Returns

The interrupt pending status.

```
void LCDIF_CursorGetDefaultConfig(lcdif_cursor_config_t *config)
```

Get the hardware cursor default configuration.

The default configuration values are:

```
config->enable = true;  
config->format = kLCDIF_CursorMasked;  
config->hotspotOffsetX = 0;  
config->hotspotOffsetY = 0;
```

#### Parameters

- config – Pointer to the hardware cursor configuration structure.

```
void LCDIF_SetCursorConfig(LCDIF_Type *base, const lcdif_cursor_config_t *config)
```

Configure the cursor.

#### Parameters

- base – LCDIF peripheral base address.
- config – Cursor configuration.

```
static inline void LCDIF_SetCursorHotspotPosition(LCDIF_Type *base, uint16_t x, uint16_t y)
```

Set the cursor hotspot position.

#### Parameters

- base – LCDIF peripheral base address.
- x – X coordinate of the hotspot, range 0 ~ 8191.
- y – Y coordinate of the hotspot, range 0 ~ 8191.

```
static inline void LCDIF_SetCursorBufferAddress(LCDIF_Type *base, uint32_t address)
```

Set the cursor memory address.

#### Parameters

- base – LCDIF peripheral base address.
- address – Memory address.



```
void LCDIF_SetCursorColor(LCDIF_Type *base, uint32_t background, uint32_t foreground)
```

Set the cursor color.

#### Parameters

- base – LCDIF peripheral base address.
- background – Background color; could be defined use LCDIF\_MAKE\_CURSOR\_COLOR
- foreground – Foreground color; could be defined use LCDIF\_MAKE\_CURSOR\_COLOR

```
FSL_LCDIF_DRIVER_VERSION
```

```
enum _lcdif_polarity_flags
```

LCDIF signal polarity flags.

*Values:*

```
enumerator kLCDIF_VsyncActiveLow
VSYNC active low.
```

```
enumerator kLCDIF_VsyncActiveHigh
VSYNC active high.
```

```
enumerator kLCDIF_HsyncActiveLow
HSYNC active low.
```

```
enumerator kLCDIF_HsyncActiveHigh
HSYNC active high.
```

```
enumerator kLCDIF_DataEnableActiveLow
Data enable line active low.
```

```
enumerator kLCDIF_DataEnableActiveHigh
Data enable line active high.
```

```
enumerator kLCDIF_DriveDataOnFallingClkEdge
Drive data on falling clock edge, capture data on rising clock edge.
```

```
enumerator kLCDIF_DriveDataOnRisingClkEdge
Drive data on falling clock edge, capture data on rising clock edge.
```

```
enum _lcdif_output_format
```

LCDIF DPI output format.

*Values:*

```
enumerator kLCDIF_Output16BitConfig1
16-bit configuration 1. RGB565: XXXXXXXX_RRRRRGGG_GGGBBBBB.
```

```
enumerator kLCDIF_Output16BitConfig2
16-bit configuration 2. RGB565: XXRRRRRR_XXGGGGGG_XXBBBBBB.
```

```
enumerator kLCDIF_Output16BitConfig3
16-bit configuration 3. RGB565: XXRRRRRX_XXGGGGGG_XXBBBBBX.
```

```
enumerator kLCDIF_Output18BitConfig1
18-bit configuration 1. RGB666: XXXXXXRR_RRRRGGGG_GGBBBBBB.
```

```
enumerator kLCDIF_Output18BitConfig2
18-bit configuration 2. RGB666: XXRRRRRR_XXGGGGGG_XXBBBBBB.
```

enumerator kLCDIF\_Output24Bit  
24-bit.

enum \_lcdif\_fb\_format  
LCDIF frame buffer pixel format.

*Values:*

enumerator kLCDIF\_PixelFormatXRGB444  
XRGB4444, deprecated, use kLCDIF\_PixelFormatXRGB4444 instead.

enumerator kLCDIF\_PixelFormatXRGB4444  
XRGB4444, 16-bit each pixel, 4-bit each element. R4G4B4 in reference manual.

enumerator kLCDIF\_PixelFormatXRGB1555  
XRGB1555, 16-bit each pixel, 5-bit each element. R5G5B5 in reference manual.

enumerator kLCDIF\_PixelFormatRGB565  
RGB565, 16-bit each pixel. R5G6B5 in reference manual.

enumerator kLCDIF\_PixelFormatXRGB8888  
XRGB8888, 32-bit each pixel, 8-bit each element. R8G8B8 in reference manual.

enum \_lcdif\_interrupt  
LCDIF interrupt and status.

*Values:*

enumerator kLCDIF\_Display0FrameDoneInterrupt  
The last pixel of visible area in frame is shown.

enum \_lcdif\_cursor\_format  
LCDIF cursor format.

*Values:*

enumerator kLCDIF\_CursorMasked  
Masked format.

enumerator kLCDIF\_CursorARGB8888  
ARGB8888.

enum \_lcdif\_dbi\_cmd\_flag  
LCDIF DBI command flag.

*Values:*

enumerator kLCDIF\_DbiCmdAddress  
Send address (or command).

enumerator kLCDIF\_DbiCmdWriteMem  
Start write memory.

enumerator kLCDIF\_DbiCmdData  
Send data.

enumerator kLCDIF\_DbiCmdReadMem  
Start read memory.

enum \_lcdif\_dbi\_out\_format  
LCDIF DBI output format.

*Values:*

enumerator kLCDIF\_DbiOutD8RGB332

8-bit data bus width, pixel RGB332. For type A or B. 1 pixel sent in 1 cycle.

enumerator kLCDIF\_DbiOutD8RGB444

8-bit data bus width, pixel RGB444. For type A or B. 2 pixels sent in 3 cycles.

enumerator kLCDIF\_DbiOutD8RGB565

8-bit data bus width, pixel RGB565. For type A or B. 1 pixel sent in 2 cycles.

enumerator kLCDIF\_DbiOutD8RGB666

8-bit data bus width, pixel RGB666. For type A or B. 1 pixel sent in 3 cycles, data bus 2 LSB not used.

enumerator kLCDIF\_DbiOutD8RGB888

8-bit data bus width, pixel RGB888. For type A or B. 1 pixel sent in 3 cycles.

enumerator kLCDIF\_DbiOutD9RGB666

9-bit data bus width, pixel RGB666. For type A or B. 1 pixel sent in 2 cycles.

enumerator kLCDIF\_DbiOutD16RGB332

16-bit data bus width, pixel RGB332. For type A or B. 2 pixels sent in 1 cycle.

enumerator kLCDIF\_DbiOutD16RGB444

16-bit data bus width, pixel RGB444. For type A or B. 1 pixel sent in 1 cycle, data bus 4 MSB not used.

enumerator kLCDIF\_DbiOutD16RGB565

16-bit data bus width, pixel RGB565. For type A or B. 1 pixel sent in 1 cycle.

enumerator kLCDIF\_DbiOutD16RGB666Option1

16-bit data bus width, pixel RGB666. For type A or B. 2 pixels sent in 3 cycles.

enumerator kLCDIF\_DbiOutD16RGB666Option2

16-bit data bus width, pixel RGB666. For type A or B. 1 pixel sent in 2 cycles.

enumerator kLCDIF\_DbiOutD16RGB888Option1

16-bit data bus width, pixel RGB888. For type A or B. 2 pixels sent in 3 cycles.

enumerator kLCDIF\_DbiOutD16RGB888Option2

16-bit data bus width, pixel RGB888. For type A or B. 1 pixel sent in 2 cycles.

enum \_lcdif\_dbi\_type

LCDIF DBI type.

*Values:*

enumerator kLCDIF\_DbiTypeA\_FixedE

Selects DBI type A fixed E mode, 68000, Motorola mode.

enumerator kLCDIF\_DbiTypeA\_ClockedE

Selects DBI Type A Clocked E mode, 68000, Motorola mode.

enumerator kLCDIF\_DbiTypeB

Selects DBI type B, 8080, Intel mode.

enum \_lcdif\_dbi\_out\_swizzle

LCDIF DBI output swizzle.

*Values:*

enumerator kLCDIF\_DbiOutSwizzleRGB

RGB

enumerator `kLCDIF_DbiOutSwizzleBGR`  
BGR

typedef enum `_lcdif_output_format` `lcdif__output__format__t`  
LCDIF DPI output format.

typedef struct `_lcdif_dpi_config` `lcdif__dpi__config__t`  
Configuration for LCDIF module to work in DBI mode.

typedef enum `_lcdif_fb_format` `lcdif__fb__format__t`  
LCDIF frame buffer pixel format.

typedef struct `_lcdif_fb_config` `lcdif__fb__config__t`  
LCDIF frame buffer configuration.

typedef enum `_lcdif_cursor_format` `lcdif__cursor__format__t`  
LCDIF cursor format.

typedef struct `_lcdif_cursor_config` `lcdif__cursor__config__t`  
LCDIF cursor configuration.

typedef struct `_lcdif_dither_config` `lcdif__dither__config__t`  
LCDIF dither configuration.

a. Decide which bit of pixel color to enhance. This is configured by the `lcdif_dither_config_t::redSize`, `lcdif_dither_config_t::greenSize`, and `lcdif_dither_config_t::blueSize`. For example, setting `redSize=6` means it is the 6th bit starting from the MSB that we want to enhance, in other words, it is the `RedColor[2]` bit from `RedColor[7:0]`. `greenSize` and `blueSize` function in the same way.

b. Create the look-up table. a. The Look-Up Table includes 16 entries, 4 bits for each. b. The Look-Up Table provides a value `U[3:0]` through the index `X[1:0]` and `Y[1:0]`. c. The color value `RedColor[3:0]` is used to compare with this `U[3:0]`. d. If `RedColor[3:0] > U[3:0]`, and `RedColor[7:2]` is not 6'b111111, then the final color value is: `NewRedColor = RedColor[7:2] + 1'b1`. e. If `RedColor[3:0] <= U[3:0]`, then `NewRedColor = RedColor[7:2]`.

typedef enum `_lcdif_dbi_out_format` `lcdif__dbi__out__format__t`  
LCDIF DBI output format.

typedef enum `_lcdif_dbi_type` `lcdif__dbi__type__t`  
LCDIF DBI type.

typedef enum `_lcdif_dbi_out_swizzle` `lcdif__dbi__out__swizzle__t`  
LCDIF DBI output swizzle.

typedef struct `_lcdif_dbi_config` `lcdif__dbi__config__t`  
LCDIF DBI configuration.

`LCDIF_MAKE_CURSOR_COLOR(r, g, b)`  
Construct the cursor color, every element should be in the range of 0 ~ 255.

`LCDIF_MAKE_GAMMA_VALUE(r, g, b)`  
Construct the gamma value set to LCDIF gamma table, every element should be in the range of 0~255.

`LCDIF_ALIGN_ADDR(addr, align)`  
Calculate the aligned address for LCDIF buffer.

`LCDIF_FB_ALIGN`  
The frame buffer should be 128 byte aligned.

LCDIF\_GAMMA\_INDEX\_MAX

Gamma index max value.

LCDIF\_CURSOR\_SIZE

The cursor size is 32 x 32.

LCDIF\_FRAMEBUFFERCONFIG0\_OUTPUT\_MASK

LCDIF\_ADDR\_CPU\_2\_IP(addr)

struct \_\_lcdif\_dpi\_config

*#include <fsl\_lcdif.h>* Configuration for LCDIF module to work in DBI mode.

### Public Members

uint16\_t panelWidth

Display panel width, pixels per line.

uint16\_t panelHeight

Display panel height, how many lines per panel.

uint8\_t hsw

HSYNC pulse width.

uint8\_t hfp

Horizontal front porch.

uint8\_t hbp

Horizontal back porch.

uint8\_t vsw

VSYNC pulse width.

uint8\_t vfp

Vrtical front porch.

uint8\_t vbp

Vertical back porch.

uint32\_t polarityFlags

OR'ed value of \_lcdif\_polarity\_flags, used to contol the signal polarity.

*lcdif\_output\_format\_t* format

DPI output format.

struct \_\_lcdif\_fb\_config

*#include <fsl\_lcdif.h>* LCDIF frame buffer configuration.

### Public Members

bool enable

Enable the frame buffer output.

bool enableGamma

Enable the gamma correction.

*lcdif\_fb\_format\_t* format

Frame buffer pixel format.

struct \_\_lcdif\_cursor\_config

*#include <fsl\_lcdif.h>* LCDIF cursor configuration.

### Public Members

bool enable

Enable the cursor or not.

*lcdif\_cursor\_format\_t* format

Cursor format.

uint8\_t hotspotOffsetX

Offset of the hotspot to top left point, range 0 ~ 31

uint8\_t hotspotOffsetY

Offset of the hotspot to top left point, range 0 ~ 31

struct *\_lcdif\_dither\_config*

*#include <fsl\_lcdif.h>* LCDIF dither configuration.

- a. Decide which bit of pixel color to enhance. This is configured by the *lcdif\_dither\_config\_t::redSize*, *lcdif\_dither\_config\_t::greenSize*, and *lcdif\_dither\_config\_t::blueSize*. For example, setting *redSize*=6 means it is the 6th bit starting from the MSB that we want to enhance, in other words, it is the *RedColor[2]bit* from *RedColor[7:0]*. *greenSize* and *blueSize* function in the same way.
- b. Create the look-up table. a. The Look-Up Table includes 16 entries, 4 bits for each. b. The Look-Up Table provides a value *U[3:0]* through the index *X[1:0]* and *Y[1:0]*. c. The color value *RedColor[3:0]* is used to compare with this *U[3:0]*. d. If *RedColor[3:0] > U[3:0]*, and *RedColor[7:2]* is not 6'b111111, then the final color value is: *NewRedColor = RedColor[7:2] + 1'b1*. e. If *RedColor[3:0] <= U[3:0]*, then *NewRedColor = RedColor[7:2]*.

### Public Members

bool enable

Enable or not.

uint8\_t redSize

Red color size, valid region 4-8.

uint8\_t greenSize

Green color size, valid region 4-8.

uint8\_t blueSize

Blue color size, valid region 4-8.

uint32\_t low

Low part of the look up table.

uint32\_t high

High part of the look up table.

struct *\_lcdif\_dbi\_config*

*#include <fsl\_lcdif.h>* LCDIF DBI configuration.

### Public Members

*lcdif\_dbi\_out\_swizzle\_t* swizzle

Swizzle.

*lcdif\_dbi\_out\_format\_t* format

Output format.

uint8\_t acTimeUnit

Time unit for AC characteristics.

lcdif\_dbi\_type\_t type

DBI type.

uint16\_t writeWRPeriod

WR signal period, Cycle number = writeWRPeriod \* (acTimeUnit + 1), must be no less than 3. Only for type A and type b.

uint8\_t writeWRAssert

Cycle number = writeWRAssert \* (acTimeUnit + 1), only for type A and type B. With kLCDIF\_DbiTypeA\_FixedE: Not used. With kLCDIF\_DbiTypeA\_ClockedE: Time to assert E. With kLCDIF\_DbiTypeB: Time to assert WRX.

uint8\_t writeCSAssert

Cycle number = writeCSAssert \* (acTimeUnit + 1), only for type A and type B. With kLCDIF\_DbiTypeA\_FixedE: Time to assert CSX. With kLCDIF\_DbiTypeA\_ClockedE: Not used. With kLCDIF\_DbiTypeB: Time to assert CSX.

uint16\_t writeWRDeassert

Cycle number = writeWRDeassert \* (acTimeUnit + 1), only for type A and type B. With kLCDIF\_DbiTypeA\_FixedE: Not used. With kLCDIF\_DbiTypeA\_ClockedE: Time to de-assert E. With kLCDIF\_DbiTypeB: Time to de-assert WRX.

uint16\_t writeCSDeassert

Cycle number = writeCSDeassert \* (acTimeUnit + 1), only for type A and type B. With kLCDIF\_DbiTypeA\_FixedE: Time to de-assert CSX. With kLCDIF\_DbiTypeA\_ClockedE: Not used. With kLCDIF\_DbiTypeB: Time to de-assert CSX.

## 2.16 MCM: Miscellaneous Control Module

FSL\_MCM\_DRIVER\_VERSION

MCM driver version.

Enum\_mcm\_interrupt\_flag. Interrupt status flag mask. .

Values:

enumerator kMCM\_CacheWriteBuffer

Cache Write Buffer Error Enable.

enumerator kMCM\_ParityError

Cache Parity Error Enable.

enumerator kMCM\_FPUInvalidOperation

FPU Invalid Operation Interrupt Enable.

enumerator kMCM\_FPUDivideByZero

FPU Divide-by-zero Interrupt Enable.

enumerator kMCM\_FPUOverflow

FPU Overflow Interrupt Enable.

enumerator kMCM\_FPUUnderflow

FPU Underflow Interrupt Enable.

enumerator kMCM\_FPUInexact

FPU Inexact Interrupt Enable.

enumerator kMCM\_FPUInputDenormalInterrupt

FPU Input Denormal Interrupt Enable.

typedef union *mcm\_buffer\_fault\_attribute* mcm\_buffer\_fault\_attribute\_t

The union of buffer fault attribute.

typedef union *mcm\_lmem\_fault\_attribute* mcm\_lmem\_fault\_attribute\_t

The union of LMEM fault attribute.

static inline void MCM\_EnableCrossbarRoundRobin(MCM\_Type \*base, bool enable)

Enables/Disables crossbar round robin.

#### Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable crossbar round robin.
  - **true** Enable crossbar round robin.
  - **false** disable crossbar round robin.

static inline void MCM\_EnableInterruptStatus(MCM\_Type \*base, uint32\_t mask)

Enables the interrupt.

#### Parameters

- base – MCM peripheral base address.
- mask – Interrupt status flags mask(*mcm\_interrupt\_flag*).

static inline void MCM\_DisableInterruptStatus(MCM\_Type \*base, uint32\_t mask)

Disables the interrupt.

#### Parameters

- base – MCM peripheral base address.
- mask – Interrupt status flags mask(*mcm\_interrupt\_flag*).

static inline uint16\_t MCM\_GetInterruptStatus(MCM\_Type \*base)

Gets the Interrupt status .

#### Parameters

- base – MCM peripheral base address.

static inline void MCM\_ClearCacheWriteBufferErrorStatus(MCM\_Type \*base)

Clears the Interrupt status .

#### Parameters

- base – MCM peripheral base address.

static inline uint32\_t MCM\_GetBufferFaultAddress(MCM\_Type \*base)

Gets buffer fault address.

#### Parameters

- base – MCM peripheral base address.

static inline void MCM\_GetBufferFaultAttribute(MCM\_Type \*base, *mcm\_buffer\_fault\_attribute\_t* \*bufferfault)

Gets buffer fault attributes.

#### Parameters



- base – MCM peripheral base address.

static inline uint32\_t MCM\_GetBufferFaultData(MCM\_Type \*base)

Gets buffer fault data.

#### Parameters

- base – MCM peripheral base address.

static inline void MCM\_LimitCodeCachePeripheralWriteBuffering(MCM\_Type \*base, bool enable)

Limit code cache peripheral write buffering.

#### Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable limit code cache peripheral write buffering.
  - **true** Enable limit code cache peripheral write buffering.
  - **false** disable limit code cache peripheral write buffering.

static inline void MCM\_BypassFixedCodeCacheMap(MCM\_Type \*base, bool enable)

Bypass fixed code cache map.

#### Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable bypass fixed code cache map.
  - **true** Enable bypass fixed code cache map.
  - **false** disable bypass fixed code cache map.

static inline void MCM\_EnableCodeBusCache(MCM\_Type \*base, bool enable)

Enables/Disables code bus cache.

#### Parameters

- base – MCM peripheral base address.
- enable – Used to disable/enable code bus cache.
  - **true** Enable code bus cache.
  - **false** disable code bus cache.

static inline void MCM\_ForceCodeCacheToNoAllocation(MCM\_Type \*base, bool enable)

Force code cache to no allocation.

#### Parameters

- base – MCM peripheral base address.
- enable – Used to force code cache to allocation or no allocation.
  - **true** Force code cache to no allocation.
  - **false** Force code cache to allocation.

static inline void MCM\_EnableCodeCacheWriteBuffer(MCM\_Type \*base, bool enable)

Enables/Disables code cache write buffer.

#### Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable code cache write buffer.
  - **true** Enable code cache write buffer.
  - **false** Disable code cache write buffer.

static inline void MCM\_ClearCodeBusCache(MCM\_Type \*base)

Clear code bus cache.

**Parameters**

- base – MCM peripheral base address.

static inline void MCM\_EnablePcParityFaultReport(MCM\_Type \*base, bool enable)

Enables/Disables PC Parity Fault Report.

**Parameters**

- base – MCM peripheral base address.
- enable – Used to enable/disable PC Parity Fault Report.
  - **true** Enable PC Parity Fault Report.
  - **false** disable PC Parity Fault Report.

static inline void MCM\_EnablePcParity(MCM\_Type \*base, bool enable)

Enables/Disables PC Parity.

**Parameters**

- base – MCM peripheral base address.
- enable – Used to enable/disable PC Parity.
  - **true** Enable PC Parity.
  - **false** disable PC Parity.

static inline void MCM\_LockConfigState(MCM\_Type \*base)

Lock the configuration state.

**Parameters**

- base – MCM peripheral base address.

static inline void MCM\_EnableCacheParityReporting(MCM\_Type \*base, bool enable)

Enables/Disables cache parity reporting.

**Parameters**

- base – MCM peripheral base address.
- enable – Used to enable/disable cache parity reporting.
  - **true** Enable cache parity reporting.
  - **false** disable cache parity reporting.

static inline uint32\_t MCM\_GetLmemFaultAddress(MCM\_Type \*base)

Gets LMEM fault address.

**Parameters**

- base – MCM peripheral base address.

static inline void MCM\_GetLmemFaultAttribute(MCM\_Type \*base, *mcm\_lmem\_fault\_attribute\_t* \*lmemFault)

Get LMEM fault attributes.

**Parameters**

- base – MCM peripheral base address.

```
static inline uint64_t MCM_GetLmemFaultData(MCM_Type *base)
```

Gets LMEM fault data.

#### Parameters

- base – MCM peripheral base address.

```
MCM_LMFATR_TYPE_MASK
```

```
MCM_LMFATR_MODE_MASK
```

```
MCM_LMFATR_BUFF_MASK
```

```
MCM_LMFATR_CACH_MASK
```

```
MCM_ISCR_STAT_MASK
```

```
FSL_COMPONENT_ID
```

```
union __mcm_buffer_fault_attribute
```

*#include <fsl\_mcm.h>* The union of buffer fault attribute.

#### Public Members

```
uint32_t attribute
```

Indicates the faulting attributes, when a properly-enabled cache write buffer error interrupt event is detected.

```
struct __mcm_buffer_fault_attribute __mcm_buffer_fault_attrib attribute_memory
```

```
struct __mcm_buffer_fault_attrib
```

*#include <fsl\_mcm.h>*

#### Public Members

```
uint32_t busErrorDataAccessType
```

Indicates the type of cache write buffer access.

```
uint32_t busErrorPrivilegeLevel
```

Indicates the privilege level of the cache write buffer access.

```
uint32_t busErrorSize
```

Indicates the size of the cache write buffer access.

```
uint32_t busErrorAccess
```

Indicates the type of system bus access.

```
uint32_t busErrorMasterID
```

Indicates the crossbar switch bus master number of the captured cache write buffer bus error.

```
uint32_t busErrorOverrun
```

Indicates if another cache write buffer bus error is detected.

```
union __mcm_lmem_fault_attribute
```

*#include <fsl\_mcm.h>* The union of LMEM fault attribute.

### Public Members

uint32\_t attribute

Indicates the attributes of the LMEM fault detected.

struct \_mcm\_lmem\_fault\_attribute \_mcm\_lmem\_fault\_attrib attribute\_memory

struct \_mcm\_lmem\_fault\_attrib

#include <fsl\_mcm.h>

### Public Members

uint32\_t parityFaultProtectionSignal

Indicates the features of parity fault protection signal.

uint32\_t parityFaultMasterSize

Indicates the parity fault master size.

uint32\_t parityFaultWrite

Indicates the parity fault is caused by read or write.

uint32\_t backdoorAccess

Indicates the LMEM access fault is initiated by core access or backdoor access.

uint32\_t parityFaultSyndrome

Indicates the parity fault syndrome.

uint32\_t overrun

Indicates the number of faultss.

## 2.17 MIPI DSI Driver

void DSI\_Init(MIPI\_DSI\_HOST\_Type \*base, const *dsi\_config\_t* \*config)

Initializes an MIPI DSI host with the user configuration.

This function initializes the MIPI DSI host with the configuration, it should be called first before other MIPI DSI driver functions.

### Parameters

- base – MIPI DSI host peripheral base address.
- config – Pointer to a user-defined configuration structure.

void DSI\_Deinit(MIPI\_DSI\_HOST\_Type \*base)

Deinitializes an MIPI DSI host.

This function should be called after all bother MIPI DSI driver functions.

### Parameters

- base – MIPI DSI host peripheral base address.

void DSI\_GetDefaultConfig(*dsi\_config\_t* \*config)

Get the default configuration to initialize the MIPI DSI host.

The default value is:

```

config->numLanes = 4;
config->enableNonContinuousHsClk = false;
config->enableTxUlps = false;
config->autoInsertEoTp = true;
config->numExtraEoTp = 0;
config->htxTo_ByteClk = 0;
config->lrxHostTo_ByteClk = 0;
config->btaTo_ByteClk = 0;

```

### Parameters

- config – Pointer to a user-defined configuration structure.

```
void DSI_SetDpiConfig(MIPI_DSI_HOST_Type *base, const dsi_dpi_config_t *config, uint8_t
numLanes, uint32_t dpiPixelClkFreq_Hz, uint32_t dsiHsBitClkFreq_Hz)
```

Configure the DPI interface core.

This function sets the DPI interface configuration, it should be used in video mode.

### Parameters

- base – MIPI DSI host peripheral base address.
- config – Pointer to the DPI interface configuration.
- numLanes – Lane number, should be same with the setting in *dsi\_dpi\_config\_t*.
- dpiPixelClkFreq\_Hz – The DPI pixel clock frequency in Hz.
- dsiHsBitClkFreq\_Hz – The DSI high speed bit clock frequency in Hz. It is the same with DPHY PLL output.

```
uint32_t DSI_InitDphy(MIPI_DSI_HOST_Type *base, const dsi_dphy_config_t *config, uint32_t
refClkFreq_Hz)
```

Initializes the D-PHY.

This function configures the D-PHY timing and setups the D-PHY PLL based on user configuration. The configuration structure could be got by the function *DSI\_GetDphyDefaultConfig*.

For some platforms there is not dedicated D-PHY PLL, indicated by the macro *FSL\_FEATURE\_MIPI\_DSI\_NO\_DPHY\_PLL*. For these platforms, the *refClkFreq\_Hz* is useless.

### Parameters

- base – MIPI DSI host peripheral base address.
- config – Pointer to the D-PHY configuration.
- refClkFreq\_Hz – The REFCLK frequency in Hz.

### Returns

The actual D-PHY PLL output frequency. If could not configure the PLL to the target frequency, the return value is 0.

```
void DSI_DeinitDphy(MIPI_DSI_HOST_Type *base)
```

Deinitializes the D-PHY.

Power down the D-PHY PLL and shut down D-PHY.

### Parameters

- base – MIPI DSI host peripheral base address.

```
void DSI_GetDphyDefaultConfig(dsi_dphy_config_t *config, uint32_t txHsBitClk_Hz, uint32_t
txEscClk_Hz)
```

Get the default D-PHY configuration.

Gets the default D-PHY configuration, the timing parameters are set according to D-PHY specification. User could use the configuration directly, or change some parameters according to the special device.

#### Parameters

- config – Pointer to the D-PHY configuration.
- txHsBitClk\_Hz – High speed bit clock in Hz.
- txEscClk\_Hz – Esc clock in Hz.

```
static inline void DSI_EnableInterrupts(MIPI_DSI_HOST_Type *base, uint32_t intGroup1,  
                                       uint32_t intGroup2)
```

Enable the interrupts.

The interrupts to enable are passed in as OR'ed mask value of \_dsi\_interrupt.

#### Parameters

- base – MIPI DSI host peripheral base address.
- intGroup1 – Interrupts to enable in group 1.
- intGroup2 – Interrupts to enable in group 2.

```
static inline void DSI_DisableInterrupts(MIPI_DSI_HOST_Type *base, uint32_t intGroup1,  
                                       uint32_t intGroup2)
```

Disable the interrupts.

The interrupts to disable are passed in as OR'ed mask value of \_dsi\_interrupt.

#### Parameters

- base – MIPI DSI host peripheral base address.
- intGroup1 – Interrupts to disable in group 1.
- intGroup2 – Interrupts to disable in group 2.

```
static inline void DSI_GetAndClearInterruptStatus(MIPI_DSI_HOST_Type *base, uint32_t  
                                                 *intGroup1, uint32_t *intGroup2)
```

Get and clear the interrupt status.

#### Parameters

- base – MIPI DSI host peripheral base address.
- intGroup1 – Group 1 interrupt status.
- intGroup2 – Group 2 interrupt status.

```
static inline void DSI_SetDbiPixelFifoSendLevel(MIPI_DSI_HOST_Type *base, uint16_t  
                                              sendLevel)
```

Configure the DBI pixel FIFO send level.

This controls the level at which the DBI Host bridge begins sending pixels

#### Parameters

- base – MIPI DSI host peripheral base address.
- sendLevel – Send level value set to register.

```
static inline void DSI_SetDbiPixelPayloadSize(MIPI_DSI_HOST_Type *base, uint16_t payloadSize)
```

Configure the DBI pixel payload size.

Configures maximum number of pixels that should be sent as one DSI packet. Recommended to be evenly divisible by the line size (in pixels).

#### Parameters

- base – MIPI DSI host peripheral base address.
- payloadSize – Payload size value set to register.

```
void DSI_SetApbPacketControl(MIPI_DSI_HOST_Type *base, uint16_t wordCount, uint8_t  
virtualChannel, dsi_tx_data_type_t dataType, uint8_t flags)
```

Configure the APB packet to send.

This function configures the next APB packet transfer. After configuration, the packet transfer could be started with function DSI\_SendApbPacket. If the packet is long packet, Use DSI\_WriteApbTxPayload to fill the payload before start transfer.

#### Parameters

- base – MIPI DSI host peripheral base address.
- wordCount – For long packet, this is the byte count of the payload. For short packet, this is (data1 « 8) | data0.
- virtualChannel – Virtual channel.
- dataType – The packet data type, (DI).
- flags – The transfer control flags, see `_dsi_transfer_flags`.

```
void DSI_WriteApbTxPayload(MIPI_DSI_HOST_Type *base, const uint8_t *payload, uint16_t  
payloadSize)
```

Fill the long APB packet payload.

Write the long packet payload to TX FIFO.

#### Parameters

- base – MIPI DSI host peripheral base address.
- payload – Pointer to the payload.
- payloadSize – Payload size in byte.

```
void DSI_WriteApbTxPayloadExt(MIPI_DSI_HOST_Type *base, const uint8_t *payload, uint16_t  
payloadSize, bool sendDcsCmd, uint8_t dcsCmd)
```

Extended function to fill the payload to TX FIFO.

Write the long packet payload to TX FIFO. This function could be used in two ways

- Include the DCS command in parameter `payload`. In this case, the DCS command is the first byte of `payload`. The parameter `sendDcsCmd` is set to false, the `dcsCmd` is not used. This function is the same as `DSI_WriteApbTxPayload` when used in this way.
- The DCS command is not in parameter `payload`, but specified by parameter `dcsCmd`. In this case, the parameter `sendDcsCmd` is set to true, the `dcsCmd` is the DCS command to send. The payload is sent after `dcsCmd`.

#### Parameters

- base – MIPI DSI host peripheral base address.
- payload – Pointer to the payload.
- payloadSize – Payload size in byte.

- `sendDcsCmd` – If set to true, the DCS command is specified by `dcsCmd`, otherwise the DCS command is included in the `payload`.
- `dcsCmd` – The DCS command to send, only used when `sendDCSCmd` is true.

`void DSI_ReadApbRxPayload(MIPI_DSI_HOST_Type *base, uint8_t *payload, uint16_t payloadSize)`

Read the long APB packet payload.

Read the long packet payload from RX FIFO. This function reads directly but does not check the RX FIFO status. Upper layer should make sure there are available data.

#### Parameters

- `base` – MIPI DSI host peripheral base address.
- `payload` – Pointer to the payload.
- `payloadSize` – Payload size in byte.

`static inline void DSI_SendApbPacket(MIPI_DSI_HOST_Type *base)`

Trigger the controller to send out APB packet.

Send the packet set by `DSI_SetApbPacketControl`.

#### Parameters

- `base` – MIPI DSI host peripheral base address.

`static inline uint32_t DSI_GetApbStatus(MIPI_DSI_HOST_Type *base)`

Get the APB status.

The return value is OR'ed value of `_dsi_apb_status`.

#### Parameters

- `base` – MIPI DSI host peripheral base address.

#### Returns

The APB status.

`static inline uint32_t DSI_GetRxErrorStatus(MIPI_DSI_HOST_Type *base)`

Get the error status during data transfer.

The return value is OR'ed value of `_dsi_rx_error_status`.

#### Parameters

- `base` – MIPI DSI host peripheral base address.

#### Returns

The error status.

`static inline uint8_t DSI_GetEccRxErrorPosition(uint32_t rxErrorStatus)`

Get the one-bit RX ECC error position.

When one-bit ECC RX error detected using `DSI_GetRxErrorStatus`, this function could be used to get the error bit position.

```
uint8_t eccErrorPos;
uint32_t rxErrorStatus = DSI_GetRxErrorStatus(MIPI_DSI);
if (kDSI_RxErrorEccOneBit & rxErrorStatus)
{
    eccErrorPos = DSI_GetEccRxErrorPosition(rxErrorStatus);
}
```

#### Parameters

- `rxErrorStatus` – The error status returned by `DSI_GetRxErrorStatus`.



**Returns**

The 1-bit ECC error position.

```
static inline uint32_t DSI_GetAndClearHostStatus(MIPI_DSI_HOST_Type *base)
```

Get and clear the DSI host status.

The host status are returned as mask value of `_dsi_host_status`.

**Parameters**

- `base` – MIPI DSI host peripheral base address.

**Returns**

The DSI host status.

```
static inline uint32_t DSI_GetRxPacketHeader(MIPI_DSI_HOST_Type *base)
```

Get the RX packet header.

**Parameters**

- `base` – MIPI DSI host peripheral base address.

**Returns**

The RX packet header.

```
static inline dsi_rx_data_type_t DSI_GetRxPacketType(uint32_t rxPktHeader)
```

Extract the RX packet type from the packet header.

Extract the RX packet type from the packet header get by `DSI_GetRxPacketHeader`.

**Parameters**

- `rxPktHeader` – The RX packet header get by `DSI_GetRxPacketHeader`.

**Returns**

The RX packet type.

```
static inline uint16_t DSI_GetRxPacketWordCount(uint32_t rxPktHeader)
```

Extract the RX packet word count from the packet header.

Extract the RX packet word count from the packet header get by `DSI_GetRxPacketHeader`.

**Parameters**

- `rxPktHeader` – The RX packet header get by `DSI_GetRxPacketHeader`.

**Returns**

For long packet, return the payload word count (byte). For short packet, return the  $(data0 \ll 8) | data1$ .

```
static inline uint8_t DSI_GetRxPacketVirtualChannel(uint32_t rxPktHeader)
```

Extract the RX packet virtual channel from the packet header.

Extract the RX packet virtual channel from the packet header get by `DSI_GetRxPacketHeader`.

**Parameters**

- `rxPktHeader` – The RX packet header get by `DSI_GetRxPacketHeader`.

**Returns**

The virtual channel.

```
status_t DSI_TransferBlocking(MIPI_DSI_HOST_Type *base, dsi_transfer_t *xfer)
```

APB data transfer using blocking method.

Perform APB data transfer using blocking method. This function waits until all data send or received, or timeout happens.

When using this API to read data, the actually read data count could be got from `xfer->rxDataSize`.

#### Parameters

- `base` – MIPI DSI host peripheral base address.
- `xfer` – Pointer to the transfer structure.

#### Return values

- `kStatus__Success` – Data transfer finished with no error.
- `kStatus__Timeout` – Transfer failed because of timeout.
- `kStatus_DSI_RxDataError` – RX data error, user could use `DSI_GetRxErrorStatus` to check the error details.
- `kStatus_DSI_ErrorReportReceived` – Error Report packet received, user could use `DSI_GetAndClearHostStatus` to check the error report status.
- `kStatus_DSI_NotSupported` – Transfer format not supported.
- `kStatus_DSI_Fail` – Transfer failed for other reasons.

```
status_t DSI_TransferCreateHandle(MIPI_DSI_HOST_Type *base, dsi_handle_t *handle,  
                                dsi_callback_t callback, void *userData)
```

Create the MIPI DSI handle.

This function initializes the MIPI DSI handle which can be used for other transactional APIs.

#### Parameters

- `base` – MIPI DSI host peripheral base address.
- `handle` – Handle pointer.
- `callback` – Callback function.
- `userData` – User data.

```
status_t DSI_TransferNonBlocking(MIPI_DSI_HOST_Type *base, dsi_handle_t *handle,  
                                dsi_transfer_t *xfer)
```

APB data transfer using interrupt method.

Perform APB data transfer using interrupt method, when transfer finished, upper layer could be informed through callback function.

When using this API to read data, the actually read data count could be got from `handle->xfer->rxDataSize` after read finished.

#### Parameters

- `base` – MIPI DSI host peripheral base address.
- `handle` – pointer to `dsi_handle_t` structure which stores the transfer state.
- `xfer` – Pointer to the transfer structure.

#### Return values

- `kStatus__Success` – Data transfer started successfully.
- `kStatus_DSI_Busy` – Failed to start transfer because DSI is busy with previous transfer.
- `kStatus_DSI_NotSupported` – Transfer format not supported.

```
void DSI_TransferAbort(MIPI_DSI_HOST_Type *base, dsi_handle_t *handle)
```

Abort current APB data transfer.

#### Parameters

- base – MIPI DSI host peripheral base address.
- handle – pointer to dsi\_handle\_t structure which stores the transfer state.

void DSI\_TransferHandleIRQ(MIPI\_DSI\_HOST\_Type \*base, dsi\_handle\_t \*handle)

Interrupt handler for the DSI.

#### Parameters

- base – MIPI DSI host peripheral base address.
- handle – pointer to dsi\_handle\_t structure which stores the transfer state.

FSL\_MIPI\_DSI\_DRIVER\_VERSION

Error codes for the MIPI DSI driver.

*Values:*

enumerator kStatus\_DSI\_Busy

DSI is busy.

enumerator kStatus\_DSI\_RxDataError

Read data error.

enumerator kStatus\_DSI\_ErrorReportReceived

Error report package received.

enumerator kStatus\_DSI\_NotSupported

The transfer type not supported.

enum \_dsi\_dpi\_color\_coding

MIPI DPI interface color coding.

*Values:*

enumerator kDSI\_Dpi16BitConfig1

16-bit configuration 1. RGB565: XXXXXXXX\_RRRRRGGG\_GGGBBBBB.

enumerator kDSI\_Dpi16BitConfig2

16-bit configuration 2. RGB565: XXXRRRRR\_XXGGGGGG\_XXXBBBBB.

enumerator kDSI\_Dpi16BitConfig3

16-bit configuration 3. RGB565: XXRRRRRX\_XXGGGGGG\_XXBBBBBX.

enumerator kDSI\_Dpi18BitConfig1

18-bit configuration 1. RGB666: XXXXXXRR\_RRRRGGGG\_GGBBBBBB.

enumerator kDSI\_Dpi18BitConfig2

18-bit configuration 2. RGB666: XXRRRRRR\_XXGGGGGG\_XXBBBBBB.

enumerator kDSI\_Dpi24Bit

24-bit.

enum \_dsi\_dpi\_pixel\_packet

MIPI DSI pixel packet type send through DPI interface.

*Values:*

enumerator kDSI\_PixelPacket16Bit

16 bit RGB565.

enumerator kDSI\_PixelPacket18Bit

18 bit RGB666 packed.

enumerator kDSI\_PixelPacket18BitLoosely  
18 bit RGB666 loosely packed into three bytes.

enumerator kDSI\_PixelPacket24Bit  
24 bit RGB888, each pixel uses three bytes.

\_dsi\_dpi\_polarity\_flag DPI signal polarity.

*Values:*

enumerator kDSI\_DpiVsyncActiveLow  
VSYNC active low.

enumerator kDSI\_DpiHsyncActiveLow  
HSYNC active low.

enumerator kDSI\_DpiVsyncActiveHigh  
VSYNC active high.

enumerator kDSI\_DpiHsyncActiveHigh  
HSYNC active high.

enum \_dsi\_dpi\_video\_mode  
DPI video mode.

*Values:*

enumerator kDSI\_DpiNonBurstWithSyncPulse  
Non-Burst mode with Sync Pulses.

enumerator kDSI\_DpiNonBurstWithSyncEvent  
Non-Burst mode with Sync Events.

enumerator kDSI\_DpiBurst  
Burst mode.

enum \_dsi\_dpi\_bllp\_mode  
Behavior in BLLP (Blanking or Low-Power Interval).

*Values:*

enumerator kDSI\_DpiBllpLowPower  
LP mode used in BLLP periods.

enumerator kDSI\_DpiBllpBlanking  
Blanking packets used in BLLP periods.

enumerator kDSI\_DpiBllpNull  
Null packets used in BLLP periods.

\_dsi\_apb\_status Status of APB to packet interface.

*Values:*

enumerator kDSI\_ApbNotIdle  
State machine not idle

enumerator kDSI\_ApbTxDone  
Tx packet done

enumerator kDSI\_ApbRxControl  
DPHY direction 0 - tx had control, 1 - rx has control

enumerator kDSI\_ApbTxOverflow  
TX fifo overflow

enumerator kDSI\_ApbTxUnderflow  
TX fifo underflow

enumerator kDSI\_ApbRxOverflow  
RX fifo overflow

enumerator kDSI\_ApbRxUnderflow  
RX fifo underflow

enumerator kDSI\_ApbRxHeaderReceived  
RX packet header has been received

enumerator kDSI\_ApbRxPacketReceived  
All RX packet payload data has been received

\_dsi\_rx\_error\_status Host receive error status.

*Values:*

enumerator kDSI\_RxErrorEccOneBit  
ECC single bit error detected.

enumerator kDSI\_RxErrorEccMultiBit  
ECC multi bit error detected.

enumerator kDSI\_RxErrorCrc  
CRC error detected.

enumerator kDSI\_RxErrorHtxTo  
High Speed forward TX timeout detected.

enumerator kDSI\_RxErrorLrxTo  
Reverse Low power data receive timeout detected.

enumerator kDSI\_RxErrorBtaTo  
BTA timeout detected.

enum \_dsi\_host\_status

DSI host controller status (status\_out)

*Values:*

enumerator kDSI\_HostSoTError  
SoT error from peripheral error report.

enumerator kDSI\_HostSoTSyncError  
SoT Sync error from peripheral error report.

enumerator kDSI\_HostEoTSyncError  
EoT Sync error from peripheral error report.

enumerator kDSI\_HostEscEntryCmdError  
Escape Mode Entry Command Error from peripheral error report.

enumerator kDSI\_HostLpTxSyncError  
Low-power transmit Sync Error from peripheral error report.

enumerator kDSI\_HostPeriphToError  
Peripheral timeout error from peripheral error report.

enumerator kDSI\_HostFalseControlError

False control error from peripheral error report.

enumerator kDSI\_HostContentionDetected

Contention detected from peripheral error report.

enumerator kDSI\_HostEccErrorOneBit

Single bit ECC error (corrected) from peripheral error report.

enumerator kDSI\_HostEccErrorMultiBit

Multi bit ECC error (not corrected) from peripheral error report.

enumerator kDSI\_HostChecksumError

Checksum error from peripheral error report.

enumerator kDSI\_HostInvalidDataType

DSI data type not recognized.

enumerator kDSI\_HostInvalidVcId

DSI VC ID invalid.

enumerator kDSI\_HostInvalidTxLength

Invalid transmission length.

enumerator kDSI\_HostProtocolViolation

DSI protocol violation.

enumerator kDSI\_HostResetTriggerReceived

Reset trigger received.

enumerator kDSI\_HostTearTriggerReceived

Tear effect trigger receive.

enumerator kDSI\_HostAckTriggerReceived

Acknowledge trigger message received.

\_dsi\_interrupt DSI interrupt.

*Values:*

enumerator kDSI\_InterruptGroup1ApbNotIdle

State machine not idle

enumerator kDSI\_InterruptGroup1ApbTxDone

Tx packet done

enumerator kDSI\_InterruptGroup1ApbRxControl

DPHY direction 0 - tx control, 1 - rx control

enumerator kDSI\_InterruptGroup1ApbTxOverflow

TX fifo overflow

enumerator kDSI\_InterruptGroup1ApbTxUnderflow

TX fifo underflow

enumerator kDSI\_InterruptGroup1ApbRxOverflow

RX fifo overflow

enumerator kDSI\_InterruptGroup1ApbRxUnderflow

RX fifo underflow

enumerator kDSI\_InterruptGroup1ApbRxHeaderReceived  
RX packet header has been received

enumerator kDSI\_InterruptGroup1ApbRxPacketReceived  
All RX packet payload data has been received

enumerator kDSI\_InterruptGroup1SoTError  
SoT error from peripheral error report.

enumerator kDSI\_InterruptGroup1SoTSyncError  
SoT Sync error from peripheral error report.

enumerator kDSI\_InterruptGroup1EoTSyncError  
EoT Sync error from peripheral error report.

enumerator kDSI\_InterruptGroup1EscEntryCmdError  
Escape Mode Entry Command Error from peripheral error report.

enumerator kDSI\_InterruptGroup1LpTxSyncError  
Low-power transmit Sync Error from peripheral error report.

enumerator kDSI\_InterruptGroup1PeriphToError  
Peripheral timeout error from peripheral error report.

enumerator kDSI\_InterruptGroup1FalseControlError  
False control error from peripheral error report.

enumerator kDSI\_InterruptGroup1ContentionDetected  
Contention detected from peripheral error report.

enumerator kDSI\_InterruptGroup1EccErrorOneBit  
Single bit ECC error (corrected) from peripheral error report.

enumerator kDSI\_InterruptGroup1EccErrorMultiBit  
Multi bit ECC error (not corrected) from peripheral error report.

enumerator kDSI\_InterruptGroup1ChecksumError  
Checksum error from peripheral error report.

enumerator kDSI\_InterruptGroup1InvalidDataType  
DSI data type not recognized.

enumerator kDSI\_InterruptGroup1InvalidVcId  
DSI VC ID invalid.

enumerator kDSI\_InterruptGroup1InvalidTxLength  
Invalid transmission length.

enumerator kDSI\_InterruptGroup1ProtocalViolation  
DSI protocol violation.

enumerator kDSI\_InterruptGroup1ResetTriggerReceived  
Reset trigger received.

enumerator kDSI\_InterruptGroup1TearTriggerReceived  
Tear effect trigger receive.

enumerator kDSI\_InterruptGroup1AckTriggerReceived  
Acknowledge trigger message received.

enumerator kDSI\_InterruptGroup1HtxTo  
High speed TX timeout.

enumerator kDSI\_InterruptGroup1LrxTo  
Low power RX timeout.

enumerator kDSI\_InterruptGroup1BtaTo  
Host BTA timeout.

enumerator kDSI\_InterruptGroup2EccOneBit  
Sinle bit ECC error.

enumerator kDSI\_InterruptGroup2EccMultiBit  
Multi bit ECC error.

enumerator kDSI\_InterruptGroup2CrcError  
CRC error.

enum \_dsi\_tx\_data\_type  
DSI TX data type.

*Values:*

enumerator kDSI\_TxDataVsyncStart  
V Sync start.

enumerator kDSI\_TxDataVsyncEnd  
V Sync end.

enumerator kDSI\_TxDataHsyncStart  
H Sync start.

enumerator kDSI\_TxDataHsyncEnd  
H Sync end.

enumerator kDSI\_TxDataEoTp  
End of transmission packet.

enumerator kDSI\_TxDataCmOff  
Color mode off.

enumerator kDSI\_TxDataCmOn  
Color mode on.

enumerator kDSI\_TxDataShutDownPeriph  
Shut down peripheral.

enumerator kDSI\_TxDataTurnOnPeriph  
Turn on peripheral.

enumerator kDSI\_TxDataGenShortWrNoParam  
Generic Short WRITE, no parameters.

enumerator kDSI\_TxDataGenShortWrOneParam  
Generic Short WRITE, one parameter.

enumerator kDSI\_TxDataGenShortWrTwoParam  
Generic Short WRITE, two parameter.

enumerator kDSI\_TxDataGenShortRdNoParam  
Generic Short READ, no parameters.

enumerator kDSI\_TxDataGenShortRdOneParam  
Generic Short READ, one parameter.



enumerator kDSI\_TxDataGenShortRdTwoParam

Generic Short READ, two parameter.

enumerator kDSI\_TxDataDcsShortWrNoParam

DCS Short WRITE, no parameters.

enumerator kDSI\_TxDataDcsShortWrOneParam

DCS Short WRITE, one parameter.

enumerator kDSI\_TxDataDcsShortRdNoParam

DCS Short READ, no parameters.

enumerator kDSI\_TxDataSetMaxReturnPktSize

Set the Maximum Return Packet Size.

enumerator kDSI\_TxDataNull

Null Packet, no data.

enumerator kDSI\_TxDataBlanking

Blanking Packet, no data.

enumerator kDSI\_TxDataGenLongWr

Generic long write.

enumerator kDSI\_TxDataDcsLongWr

DCS Long Write/write\_LUT Command Packet.

enumerator kDSI\_TxDataLooselyPackedPixel20BitYCbCr

Loosely Packed Pixel Stream, 20-bit YCbCr, 4:2:2 Format.

enumerator kDSI\_TxDataPackedPixel24BitYCbCr

Packed Pixel Stream, 24-bit YCbCr, 4:2:2 Format.

enumerator kDSI\_TxDataPackedPixel16BitYCbCr

Packed Pixel Stream, 16-bit YCbCr, 4:2:2 Format.

enumerator kDSI\_TxDataPackedPixel30BitRGB

Packed Pixel Stream, 30-bit RGB, 10-10-10 Format.

enumerator kDSI\_TxDataPackedPixel36BitRGB

Packed Pixel Stream, 36-bit RGB, 12-12-12 Format.

enumerator kDSI\_TxDataPackedPixel12BitYCrCb

Packed Pixel Stream, 12-bit YCbCr, 4:2:0 Format.

enumerator kDSI\_TxDataPackedPixel16BitRGB

Packed Pixel Stream, 16-bit RGB, 5-6-5 Format.

enumerator kDSI\_TxDataPackedPixel18BitRGB

Packed Pixel Stream, 18-bit RGB, 6-6-6 Format.

enumerator kDSI\_TxDataLooselyPackedPixel18BitRGB

Loosely Packed Pixel Stream, 18-bit RGB, 6-6-6 Format.

enumerator kDSI\_TxDataPackedPixel24BitRGB

Packed Pixel Stream, 24-bit RGB, 8-8-8 Format.

enum \_\_dsi\_rx\_data\_type

DSI RX data type.

*Values:*

enumerator `kDSI_RxDataAckAndErrorReport`  
Acknowledge and Error Report

enumerator `kDSI_RxDataEoTp`  
End of Transmission packet.

enumerator `kDSI_RxDataGenShortRdResponseOneByte`  
Generic Short READ Response, 1 byte returned.

enumerator `kDSI_RxDataGenShortRdResponseTwoByte`  
Generic Short READ Response, 2 byte returned.

enumerator `kDSI_RxDataGenLongRdResponse`  
Generic Long READ Response.

enumerator `kDSI_RxDataDcsLongRdResponse`  
DCS Long READ Response.

enumerator `kDSI_RxDataDcsShortRdResponseOneByte`  
DCS Short READ Response, 1 byte returned.

enumerator `kDSI_RxDataDcsShortRdResponseTwoByte`  
DCS Short READ Response, 2 byte returned.

`_dsi_transfer_flags` DSI transfer control flags.

*Values:*

enumerator `kDSI_TransferUseHighSpeed`  
Use high speed mode or not.

enumerator `kDSI_TransferPerformBTA`  
Perform BTA or not.

typedef struct `_dsi_config` `dsi_config_t`  
MIPI DSI controller configuration.

typedef enum `_dsi_dpi_color_coding` `dsi_dpi_color_coding_t`  
MIPI DPI interface color coding.

typedef enum `_dsi_dpi_pixel_packet` `dsi_dpi_pixel_packet_t`  
MIPI DSI pixel packet type send through DPI interface.

typedef enum `_dsi_dpi_video_mode` `dsi_dpi_video_mode_t`  
DPI video mode.

typedef enum `_dsi_dpi_bllp_mode` `dsi_dpi_bllp_mode_t`  
Behavior in BLLP (Blanking or Low-Power Interval).

typedef struct `_dsi_dpi_config` `dsi_dpi_config_t`  
MIPI DSI controller DPI interface configuration.

typedef struct `_dsi_dphy_config` `dsi_dphy_config_t`  
MIPI DSI D-PHY configuration.

typedef enum `_dsi_tx_data_type` `dsi_tx_data_type_t`  
DSI TX data type.

typedef enum `_dsi_rx_data_type` `dsi_rx_data_type_t`  
DSI RX data type.

```
typedef struct _dsi_transfer dsi_transfer_t
```

Structure for the data transfer.

```
typedef struct _dsi_handle dsi_handle_t
```

MIPI DSI transfer handle.

```
typedef void (*dsi_callback_t)(MIPI_DSI_HOST_Type *base, dsi_handle_t *handle, status_t status, void *userData)
```

MIPI DSI callback for finished transfer.

When transfer finished, one of these status values will be passed to the user:

- `kStatus_Success` Data transfer finished with no error.
- `kStatus_Timeout` Transfer failed because of timeout.
- `kStatus_DSI_RxDataError` RX data error, user could use `DSI_GetRxErrorStatus` to check the error details.
- `kStatus_DSI_ErrorReportReceived` Error Report packet received, user could use `DSI_GetAndClearHostStatus` to check the error report status.
- `kStatus_Fail` Transfer failed for other reasons.

```
FSL_DSI_TX_MAX_PAYLOAD_BYTE
```

```
FSL_DSI_RX_MAX_PAYLOAD_BYTE
```

```
struct _dsi_config
```

*#include <fsl\_mipi\_dsi.h>* MIPI DSI controller configuration.

### Public Members

```
uint8_t numLanes
```

Number of lanes.

```
bool enableNonContinuousHsClk
```

In enabled, the high speed clock will enter low power mode between transmissions.

```
bool autoInsertEoTp
```

Insert an EoTp short package when switching from HS to LP.

```
uint8_t numExtraEoTp
```

How many extra EoTp to send after the end of a packet.

```
uint32_t htXTo_ByteClk
```

HS TX timeout count (HTX\_TO) in byte clock.

```
uint32_t lrxHostTo_ByteClk
```

LP RX host timeout count (LRX-H\_TO) in byte clock.

```
uint32_t btaTo_ByteClk
```

Bus turn around timeout count (TA\_TO) in byte clock.

```
struct _dsi_dpi_config
```

*#include <fsl\_mipi\_dsi.h>* MIPI DSI controller DPI interface configuration.

### Public Members

```
uint16_t pixelPayloadSize
```

Maximum number of pixels that should be sent as one DSI packet. Recommended that the line size (in pixels) is evenly divisible by this parameter.

*dsi\_dpi\_color\_coding\_t* dpiColorCoding

DPI color coding.

*dsi\_dpi\_pixel\_packet\_t* pixelPacket

Pixel packet format.

*dsi\_dpi\_video\_mode\_t* videoMode

Video mode.

*dsi\_dpi\_bllp\_mode\_t* bllpMode

Behavior in BLLP.

*uint8\_t* polarityFlags

OR'ed value of *\_dsi\_dpi\_polarity\_flag* controls signal polarity.

*uint16\_t* hfp

Horizontal front porch, in dpi pixel clock.

*uint16\_t* hbp

Horizontal back porch, in dpi pixel clock.

*uint16\_t* hsw

Horizontal sync width, in dpi pixel clock.

*uint8\_t* vfp

Number of lines in vertical front porch.

*uint8\_t* vbp

Number of lines in vertical back porch.

*uint16\_t* panelHeight

Line number in vertical active area.

*uint8\_t* virtualChannel

Virtual channel.

struct *\_dsi\_dphy\_config*

*#include <fsl\_mipi\_dsi.h>* MIPI DSI D-PHY configuration.

## Public Members

*uint32\_t* txHsBitClk\_Hz

The generated HS TX bit clock in Hz.

*uint8\_t* tClkPre\_ByteClk

TLPX + TCLK-PREPARE + TCLK-ZERO + TCLK-PRE in byte clock. Set how long the controller will wait after enabling clock lane for HS before enabling data lanes for HS.

*uint8\_t* tClkPost\_ByteClk

TCLK-POST + T\_CLK-TRAIL in byte clock. Set how long the controller will wait before putting clock lane into LP mode after data lanes detected in stop state.

*uint8\_t* tHsExit\_ByteClk

THS-EXIT in byte clock. Set how long the controller will wait after the clock lane has been put into LP mode before enabling clock lane for HS again.

*uint8\_t* tHsPrepare\_HalfEscClk

THS-PREPARE in *clk\_esc/2*. Set how long to drive the LP-00 state before HS transmissions, available values are 2, 3, 4, 5.

uint8\_t tClkPrepare\_HalfEscClk

TCLK-PREPARE in clk\_esc/2. Set how long to drive the LP-00 state before HS transmissions, available values are 2, 3.

uint8\_t tHsZero\_ByteClk

THS-ZERO in clk\_byte. Set how long that controller drives data lane HS-0 state before transmit the Sync sequence. Available values are 6, 7, ..., 37.

uint8\_t tClkZero\_ByteClk

TCLK-ZERO in clk\_byte. Set how long that controller drives clock lane HS-0 state before transmit the Sync sequence. Available values are 3, 4, ..., 66.

uint8\_t tHsTrail\_ByteClk

THS-TRAIL + 4\*UI in clk\_byte. Set the time of the flipped differential state after last payload data bit of HS transmission burst. Available values are 0, 1, ..., 15.

uint8\_t tClkTrail\_ByteClk

TCLK-TRAIL + 4\*UI in clk\_byte. Set the time of the flipped differential state after last payload data bit of HS transmission burst. Available values are 0, 1, ..., 15.

struct \_dsi\_transfer

*#include <fsl\_mipi\_dsi.h>* Structure for the data transfer.

### Public Members

uint8\_t virtualChannel

Virtual channel.

dsi\_tx\_data\_type\_t txDataType

TX data type.

uint8\_t flags

Flags to control the transfer, see \_dsi\_transfer\_flags.

const uint8\_t \*txData

The TX data buffer.

uint8\_t \*rxData

The RX data buffer.

uint16\_t txDataSize

Size of the TX data.

uint16\_t rxDataSize

Size of the RX data.

bool sendDcsCmd

If set to true, the DCS command is specified by dcsCmd, otherwise the DCS command is included in the txData.

uint8\_t dcsCmd

The DCS command to send, only valid when sendDcsCmd is true.

struct \_dsi\_handle

*#include <fsl\_mipi\_dsi.h>* MIPI DSI transfer handle structure.

### Public Members

volatile bool isBusy

MIPI DSI is busy with APB data transfer.

*dsi\_transfer\_t* xfer  
Transfer information.

*dsi\_callback\_t* callback  
DSI callback

void \*userData  
Callback parameter

## 2.18 MIPI\_DSI: MIPI DSI Host Controller

## 2.19 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.

*status\_t* 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. If `MU_BUSY_POLL_COUNT` is defined and non-zero, the function will timeout after the specified number of polling iterations and returns `kStatus_Timeout`.

### Parameters

- base – MU peripheral base address.
- regIndex – MU message register, see `mu_msg_reg_index_t`.
- msg – Message to send.

**Return values**

- `kStatus_Success` – Message sent successfully.
- `kStatus_Timeout` – Timeout occurred while waiting for TX register to be empty.

**Returns**

`status_t`

`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.

`status_t MU_ReceiveMsgTimeout(MU_Type *base, uint32_t regIndex, uint32_t *readValue)`

Blocks to receive a message with timeout protection.

This function waits until the RX register is full and receives the message. If `MU_BUSY_POLL_COUNT` is defined and non-zero, the function will timeout after the specified number of polling iterations and return `kStatus_Timeout`.

This function provides the same blocking behavior as `MU_ReceiveMsg()` but with additional timeout protection to prevent system hangs if the other core becomes unresponsive or if hardware issues occur.

---

**Note:** Both `MU_ReceiveMsg()` and `MU_ReceiveMsgTimeout()` are blocking functions. The difference is that this function includes timeout protection while `MU_ReceiveMsg()` waits indefinitely.

---

**Parameters**

- `base` – MU peripheral base address.
- `regIndex` – RX register index, see `mu_msg_reg_index_t`.
- `readValue` – Pointer to store the received message.

**Return values**

- `kStatus_Success` – Message received successfully.
- `kStatus_InvalidArgument` – Invalid `readValue` pointer.
- `kStatus_Timeout` – Timeout occurred while waiting for RX register to be full.

**Returns**

status\_t

uint32\_t MU\_ReceiveMsg(MU\_Type \*base, uint32\_t regIndex)

Blocks to receive a message (infinite wait, no timeout protection).

This function waits until the RX register is full and receives the message. This function will wait indefinitely until a message is received.

---

**Note:** Both MU\_ReceiveMsg() and MU\_ReceiveMsgTimeout() are blocking functions. The difference is that MU\_ReceiveMsgTimeout() includes timeout protection while this function waits indefinitely.

---

**Warning:** This function does not include timeout protection and may cause system hangs if the other core becomes unresponsive. For applications requiring timeout protection, use MU\_ReceiveMsgTimeout() instead.

**Parameters**

- base – MU peripheral base address.
- regIndex – RX register index, 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.

status\_t 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.

If MU\_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 – MU peripheral base address.
- flags – The 3-bit MU flags to set.

**Return values**

- kStatus\_Success – Flags were set successfully.
- kStatus\_Timeout – Timeout occurred while waiting for flags to update.

**Returns**

status\_t

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\_ClearNmi(MU\_Type \*base)

Clear non-maskable interrupt (NMI) sent by the other core.

This function clears non-maskable interrupt (NMI) sent by the other core.

#### Parameters

- base – MU peripheral base address.

void MU\_BootCoreB(MU\_Type \*base, mu\_core\_boot\_mode\_t mode)

Boots the core at B side.

This function sets the B side core's boot configuration and releases the core from reset.

---

**Note:** Only MU side A can use this function.

---

#### Parameters

- base – MU peripheral base address.
- mode – Core B boot mode.

static inline void MU\_HoldCoreBReset(MU\_Type \*base)

Holds the core reset of B side.

This function causes the core of B side to be held in reset following any reset event.

---

**Note:** Only A side could call this function.

---

#### Parameters

- base – MU peripheral base address.

void MU\_BootOtherCore(MU\_Type \*base, mu\_core\_boot\_mode\_t mode)

Boots the other core.

This function boots the other core with a boot configuration.

#### Parameters

- base – MU peripheral base address.
- mode – The other core boot mode.

static inline void MU\_HoldOtherCoreReset(MU\_Type \*base)

Holds the other core reset.

This function causes the other core to be held in reset following any reset event.

#### Parameters

- base – MU peripheral base address.

static inline *status\_t* MU\_ResetBothSides(MU\_Type \*base)

Resets the MU for both A side and B side.

This function resets the MU for both A side and B side. Before reset, it is recommended to interrupt processor B, because this function may affect the ongoing processor B programs.

If MU\_BUSY\_POLL\_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations if waiting for the other side to come out of reset takes too long.

---

**Note:** For some platforms, only MU side A could use this function, check reference manual for details.

---

#### Parameters

- base – MU peripheral base address.

#### Return values

- kStatus\_Success – The MU was reset successfully.
- kStatus\_Timeout – Timeout occurred while waiting for the other side to come out of reset.

#### Returns

*status\_t*

*status\_t* MU\_HardwareResetOtherCore(MU\_Type \*base, bool waitReset, bool holdReset, mu\_core\_boot\_mode\_t bootMode)

Hardware reset the other core.

This function resets the other core, the other core could mask the hardware reset by calling `MU_MaskHardwareReset`. The hardware reset mask feature is only available for some platforms. This function could be used together with `MU_BootOtherCore` to control the other core reset workflow.

If `MU_BUSY_POLL_COUNT` is defined and non-zero, the function will timeout after the specified number of polling iterations and return `kStatus_Timeout` if waiting for the other core to enter or exit reset takes too long.

Example 1: Reset the other core, and no hold reset

```
MU_HardwareResetOtherCore(MU_A, true, false, bootMode);
```

In this example, the core at MU side B will reset with the specified boot mode.

Example 2: Reset the other core and hold it, then boot the other core later. Here the other core enters reset, and the reset is hold

```
MU_HardwareResetOtherCore(MU_A, true, true, modeDontCare);
```

Current core boot the other core when necessary.

```
MU_BootOtherCore(MU_A, bootMode);
```

### Parameters

- `base` – MU peripheral base address.
- `waitReset` – Wait the other core enters reset.
  - `true`: Wait until the other core enters reset, if the other core has masked the hardware reset, then this function will be blocked.
  - `false`: Don't wait the reset.
- `holdReset` – Hold the other core reset or not.
  - `true`: Hold the other core in reset, this function returns directly when the other core enters reset.
  - `false`: Don't hold the other core in reset, this function waits until the other core out of reset.
- `bootMode` – Boot mode of the other core, if `holdReset` is true, this parameter is useless.

### Return values

- `kStatus_Success` – The other core was reset successfully.
- `kStatus_Timeout` – Timeout occurred while waiting for the other core to enter or exit reset.

### Returns

`status_t`

```
static inline void MU_SetClockOnOtherCoreEnable(MU_Type *base, bool enable)
```

Enables or disables the clock on the other core.

This function enables or disables the platform clock on the other core when that core enters a stop mode. If disabled, the platform clock for the other core is disabled when it enters stop mode. If enabled, the platform clock keeps running on the other core in stop mode, until this core also enters stop mode.

### Parameters

- `base` – MU peripheral base address.

- enable – Enable or disable the clock on the other core.

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.

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.

enum `_mu_interrupt_trigger`

MU interrupt that could be triggered to the other core.

*Values:*

enumerator `kMU_NmiInterruptTrigger`  
NMI interrupt.

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

MU\_BUSY\_POLL\_COUNT

Maximum polling iterations for MU waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the MU code before timing out and returning an error.

It applies to all waiting loops in MU driver, such as waiting for TX register to be empty or waiting for RX register to be full.

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 a core becomes unresponsive.

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)

## 2.20 OCOTP: On Chip One-Time Programmable controller.

FSL\_OCOTP\_DRIVER\_VERSION

OCOTP driver version.

*\_ocotp\_status* Error codes for the OCOTP driver.

*Values:*

enumerator kStatus\_OCOTP\_AccessError

eFuse and shadow register access error.

enumerator kStatus\_OCOTP\_CrcFail

CRC check failed.

enumerator kStatus\_OCOTP\_ReloadError

Error happens during reload shadow register.

enumerator kStatus\_OCOTP\_ProgramFail

Fuse programming failed.

enumerator kStatus\_OCOTP\_Locked

Fuse is locked and cannot be programmed.



`void OCOTP_Init(OCOTP_Type *base, uint32_t srcClock_Hz)`

Initializes OCOTP controller.

#### Parameters

- `base` – OCOTP peripheral base address.
- `srcClock_Hz` – source clock frequency in unit of Hz. When the macro `FSL_FEATURE_OCOTP_HAS_TIMING_CTRL` is defined as 0, this parameter is not used, application could pass in 0 in this case.

`void OCOTP_Deinit(OCOTP_Type *base)`

De-initializes OCOTP controller.

#### Return values

`kStatus_Success` – upon successful execution, error status otherwise.

`static inline bool OCOTP_CheckBusyStatus(OCOTP_Type *base)`

Checking the BUSY bit in CTRL register. Checking this BUSY bit will help confirm if the OCOTP controller is ready for access.

#### Parameters

- `base` – OCOTP peripheral base address.

#### Return values

`true` – for bit set and `false` for cleared.

`static inline bool OCOTP_CheckErrorStatus(OCOTP_Type *base)`

Checking the ERROR bit in CTRL register.

#### Parameters

- `base` – OCOTP peripheral base address.

#### Return values

`true` – for bit set and `false` for cleared.

`static inline void OCOTP_ClearErrorStatus(OCOTP_Type *base)`

Clear the error bit if this bit is set.

#### Parameters

- `base` – OCOTP peripheral base address.

`status_t OCOTP_ReloadShadowRegister(OCOTP_Type *base)`

Reload the shadow register. This function will help reload the shadow register without resetting the OCOTP module. Please make sure the OCOTP has been initialized before calling this API.

#### Parameters

- `base` – OCOTP peripheral base address.

#### Return values

- `kStatus_Success` – Reload success.
- `kStatus_OCOTP_ReloadError` – Reload failed.

`uint32_t OCOTP_ReadFuseShadowRegister(OCOTP_Type *base, uint32_t address)`

Read the fuse shadow register with the fuse address.

*Deprecated:*

Use `OCOTP_ReadFuseShadowRegisterExt` instead of this function.

#### Parameters

- `base` – OCOTP peripheral base address.
- `address` – the fuse address to be read from.

**Returns**

The read out data.

`status_t` OCOTP\_ReadFuseShadowRegisterExt(OCOTP\_Type \*base, uint32\_t address, uint32\_t \*data, uint8\_t fuseWords)

Read the fuse shadow register from the fuse address.

This function reads fuse from `address`, how many words to read is specified by the parameter `fuseWords`. This function could read at most OCOTP\_READ\_FUSE\_DATA\_COUNT fuse word one time.

**Parameters**

- `base` – OCOTP peripheral base address.
- `address` – the fuse address to be read from.
- `data` – Data array to save the readout fuse value.
- `fuseWords` – How many words to read.

**Return values**

- `kStatus_Success` – Read success.
- `kStatus_Fail` – Error occurs during read.

`status_t` OCOTP\_WriteFuseShadowRegister(OCOTP\_Type \*base, uint32\_t address, uint32\_t data)

Write the fuse shadow register with the fuse address and data. Please make sure the write address is not locked while calling this API.

**Parameters**

- `base` – OCOTP peripheral base address.
- `address` – the fuse address to be written.
- `data` – the value will be written to fuse address.

**Return values**

`write` – status, `kStatus_Success` for success and `kStatus_Fail` for failed.

`status_t` OCOTP\_WriteFuseShadowRegisterWithLock(OCOTP\_Type \*base, uint32\_t address, uint32\_t data, bool lock)

Write the fuse shadow register and lock it.

Please make sure the write address is not locked while calling this API.

Some OCOTP controller supports ECC mode and redundancy mode (see reference manual for more details). OCOTP controller will auto select ECC or redundancy mode to program the fuse word according to fuse map definition. In ECC mode, the 32 fuse bits in one word can only be written once. In redundancy mode, the word can be written more than once as long as they are different fuse bits. Set parameter `lock` as true to force use ECC mode.

**Parameters**

- `base` – OCOTP peripheral base address.
- `address` – The fuse address to be written.
- `data` – The value will be written to fuse address.
- `lock` – Lock or unlock write fuse shadow register operation.

**Return values**

- `kStatus_Success` – Program and reload success.

- `kStatus_OCOTP_Locked` – The eFuse word is locked and cannot be programmed.
- `kStatus_OCOTP_ProgramFail` – eFuse word programming failed.
- `kStatus_OCOTP_ReloadError` – eFuse word programming success, but error happens during reload the values.
- `kStatus_OCOTP_AccessError` – Cannot access eFuse word.

`static inline uint32_t OCOTP_GetVersion(OCOTP_Type *base)`

Get the OCOTP controller version from the register.

#### Parameters

- `base` – OCOTP peripheral base address.

#### Return values

`return` – the version value.

`OCOTP_READ_FUSE_DATA_COUNT`

## 2.21 PWM: Pulse Width Modulation Driver

`status_t PWM_Init(PWM_Type *base, const pwm_config_t *config)`

Un gates the PWM clock and configures the peripheral for basic operation.

---

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

---

#### Parameters

- `base` – PWM peripheral base address
- `config` – Pointer to user's PWM config structure.

#### Returns

`kStatus_Success` means success; else failed.

`void PWM_Deinit(PWM_Type *base)`

Gate the PWM submodule clock.

#### Parameters

- `base` – PWM peripheral base address

`void PWM_GetDefaultConfig(pwm_config_t *config)`

Fill in the PWM config struct with the default settings.

The default values are:

```
config->enableStopMode = false;
config->enableDozeMode = false;
config->enableWaitMode = false;
config->enableDozeMode = false;
config->clockSource = kPWM_LowFrequencyClock;
config->prescale = 0U;
config->outputConfig = kPWM_SetAtRolloverAndClearAtcomparison;
config->fifoWater = kPWM_FIFOWaterMark_2;
config->sampleRepeat = kPWM_EachSampleOnce;
config->byteSwap = kPWM_ByteNoSwap;
config->halfWordSwap = kPWM_HalfWordNoSwap;
```

**Parameters**

- config – Pointer to user's PWM config structure.

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

Starts the PWM counter when the PWM is enabled.

When the PWM is enabled, it begins a new period, the output pin is set to start a new period while the prescaler and counter are released and counting begins.

**Parameters**

- base – PWM peripheral base address

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

Stops the PWM counter when the pwm is disabled.

**Parameters**

- base – PWM peripheral base address

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

Enables the selected PWM interrupts.

**Parameters**

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

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

Disables the selected PWM interrupts.

**Parameters**

- base – PWM peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration pwm\_interrupt\_enable\_t

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

Gets the enabled PWM interrupts.

**Parameters**

- base – PWM peripheral base address

**Returns**

The enabled interrupts. This is the logical OR of members of the enumeration pwm\_interrupt\_enable\_t

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

Gets the PWM status flags.

**Parameters**

- base – PWM peripheral base address

**Returns**

The status flags. This is the logical OR of members of the enumeration pwm\_status\_flags\_t

static inline void PWM\_clearStatusFlags(PWM\_Type \*base, uint32\_t mask)

Clears the PWM status flags.

**Parameters**

- base – PWM peripheral base address

- mask – The status flags to clear. This is a logical OR of members of the enumeration `pwm_status_flags_t`

static inline uint32\_t PWM\_GetFIFOAvailable(PWM\_Type \*base)

Gets the PWM FIFO available.

#### Parameters

- base – PWM peripheral base address

#### Returns

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

static inline void PWM\_SetSampleValue(PWM\_Type \*base, uint32\_t value)

Sets the PWM sample value.

#### Parameters

- base – PWM peripheral base address
- value – The sample value. This is the input to the 4x16 FIFO. The value in this register denotes the value of the sample being currently used.

static inline uint32\_t PWM\_GetSampleValue(PWM\_Type \*base)

Gets the PWM sample value.

#### Parameters

- base – PWM peripheral base address

#### Returns

The sample value. It can be read only when the PWM is enable.

FSL\_PWM\_DRIVER\_VERSION

enum \_\_pwm\_clock\_source

PWM clock source select.

*Values:*

enumerator kPWM\_PeripheralClock

The Peripheral clock is used as the clock

enumerator kPWM\_HighFrequencyClock

High-frequency reference clock is used as the clock

enumerator kPWM\_LowFrequencyClock

Low-frequency reference clock(32KHz) is used as the clock

enum \_\_pwm\_fifo\_water\_mark

PWM FIFO water mark select. Sets the data level at which the FIFO empty flag will be set.

*Values:*

enumerator kPWM\_FIFOWaterMark\_1

FIFO empty flag is set when there are more than or equal to 1 empty slots

enumerator kPWM\_FIFOWaterMark\_2

FIFO empty flag is set when there are more than or equal to 2 empty slots

enumerator kPWM\_FIFOWaterMark\_3

FIFO empty flag is set when there are more than or equal to 3 empty slots

enumerator kPWM\_FIFOWaterMark\_4

FIFO empty flag is set when there are more than or equal to 4 empty slots

enum \_\_pwm\_byte\_data\_swap

PWM byte data swap select. It determines the byte ordering of the 16-bit data when it goes into the FIFO from the sample register.

*Values:*

enumerator kPWM\_ByteNoSwap

byte ordering remains the same

enumerator kPWM\_ByteSwap

byte ordering is reversed

enum \_\_pwm\_half\_word\_data\_swap

PWM half-word data swap select.

*Values:*

enumerator kPWM\_HalfWordNoSwap

Half word swapping does not take place

enumerator kPWM\_HalfWordSwap

Half word from write data bus are swapped

enum \_\_pwm\_output\_configuration

PWM Output Configuration.

*Values:*

enumerator kPWM\_SetAtRolloverAndClearAtcomparison

Output pin is set at rollover and cleared at comparison

enumerator kPWM\_ClearAtRolloverAndSetAtcomparison

Output pin is cleared at rollover and set at comparison

enumerator kPWM\_NoConfigure

PWM output is disconnected

enum \_\_pwm\_sample\_repeat

PWM FIFO sample repeat It determines the number of times each sample from the FIFO is to be used.

*Values:*

enumerator kPWM\_EachSampleOnce

Use each sample once

enumerator kPWM\_EachSampletwice

Use each sample twice

enumerator kPWM\_EachSampleFourTimes

Use each sample four times

enumerator kPWM\_EachSampleEightTimes

Use each sample eight times

enum \_\_pwm\_interrupt\_enable

List of PWM interrupt options.

*Values:*

enumerator kPWM\_FIFOEmptyInterruptEnable

This bit controls the generation of the FIFO Empty interrupt.

enumerator kPWM\_RolloverInterruptEnable

This bit controls the generation of the Rollover interrupt.

enumerator kPWM\_CompareInterruptEnable

This bit controls the generation of the Compare interrupt

enum \_\_pwm\_status\_flags

List of PWM status flags.

*Values:*

enumerator kPWM\_FIFOEmptyFlag

This bit indicates the FIFO data level in comparison to the water level set by FWM field in the control register.

enumerator kPWM\_RolloverFlag

This bit shows that a roll-over event has occurred.

enumerator kPWM\_CompareFlag

This bit shows that a compare event has occurred.

enumerator kPWM\_FIFOWriteErrorFlag

This bit shows that an attempt has been made to write FIFO when it is full.

enum \_\_pwm\_fifo\_available

List of PWM FIFO available.

*Values:*

enumerator kPWM\_NoDataInFIFOFlag

No data available

enumerator kPWM\_OneWordInFIFOFlag

1 word of data in FIFO

enumerator kPWM\_TwoWordsInFIFOFlag

2 word of data in FIFO

enumerator kPWM\_ThreeWordsInFIFOFlag

3 word of data in FIFO

enumerator kPWM\_FourWordsInFIFOFlag

4 word of data in FIFO

typedef enum \_\_pwm\_clock\_source pwm\_clock\_source\_t

PWM clock source select.

typedef enum \_\_pwm\_fifo\_water\_mark pwm\_fifo\_water\_mark\_t

PWM FIFO water mark select. Sets the data level at which the FIFO empty flag will be set.

typedef enum \_\_pwm\_byte\_data\_swap pwm\_byte\_data\_swap\_t

PWM byte data swap select. It determines the byte ordering of the 16-bit data when it goes into the FIFO from the sample register.

typedef enum \_\_pwm\_half\_word\_data\_swap pwm\_half\_word\_data\_swap\_t

PWM half-word data swap select.

typedef enum \_\_pwm\_output\_configuration pwm\_output\_configuration\_t

PWM Output Configuration.

typedef enum \_\_pwm\_sample\_repeat pwm\_sample\_repeat\_t

PWM FIFO sample repeat It determines the number of times each sample from the FIFO is to be used.

```
typedef enum _pwm_interrupt_enable pwm_interrupt_enable_t
```

List of PWM interrupt options.

```
typedef enum _pwm_status_flags pwm_status_flags_t
```

List of PWM status flags.

```
typedef enum _pwm_fifo_available pwm_fifo_available_t
```

List of PWM FIFO available.

```
typedef struct _pwm_config pwm_config_t
```

```
static inline void PWM_SoftwareReset(PWM_Type *base)
```

Software reset.

PWM is reset when this bit is set to 1. It is a self clearing bit. Setting this bit resets all the registers to their reset values except for the STOPEN, DOZEN, WAITEN, and DBGEN bits in this control register.

#### Parameters

- base – PWM peripheral base address

```
static inline void PWM_SetPeriodValue(PWM_Type *base, uint32_t value)
```

Sets the PWM period value.

#### Parameters

- base – PWM peripheral base address
- value – The period value. The PWM period register (PWM\_PWMPR) determines the period of the PWM output signal. Writing 0xFFFF to this register will achieve the same result as writing 0xFFFE.  $PWMO(Hz) = PCLK(Hz) / (period + 2)$

```
static inline uint32_t PWM_GetPeriodValue(PWM_Type *base)
```

Gets the PWM period value.

#### Parameters

- base – PWM peripheral base address

#### Returns

The period value. The PWM period register (PWM\_PWMPR) determines the period of the PWM output signal.

```
static inline uint32_t PWM_GetCounterValue(PWM_Type *base)
```

Gets the PWM counter value.

#### Parameters

- base – PWM peripheral base address

#### Returns

The counter value. The current count value.

```
struct _pwm_config
```

```
#include <fsl_pwm.h>
```

### Public Members

```
bool enableStopMode
```

True: PWM continues to run in stop mode; False: PWM is paused in stop mode.

```
bool enableDozeMode
```

True: PWM continues to run in doze mode; False: PWM is paused in doze mode.



`bool enableWaitMode`

True: PWM continues to run in wait mode; False: PWM is paused in wait mode.

`bool enableDebugMode`

True: PWM continues to run in debug mode; False: PWM is paused in debug mode.

`uint16_t prescale`

Pre-scaler to divide down the clock The prescaler value is not more than 0xFF. Divide by (value + 1)

`pwm_clock_source_t clockSource`

Clock source for the counter

`pwm_output_configuration_t outputConfig`

Set the mode of the PWM output on the output pin.

`pwm_fifo_water_mark_t fifoWater`

Set the data level for FIFO.

`pwm_sample_repeat_t sampleRepeat`

The number of times each sample from the FIFO is to be used.

`pwm_byte_data_swap_t byteSwap`

It determines the byte ordering of the 16-bit data when it goes into the FIFO from the sample register.

`pwm_half_word_data_swap_t halfWordSwap`

It determines which half word data from the 32-bit IP Bus interface is written into the lower 16 bits of the sample register.

## 2.22 QSPI: Quad Serial Peripheral Interface

### 2.23 Quad Serial Peripheral Interface Driver

`uint32_t QSPI_GetInstance(QuadSPI_Type *base)`

Get the instance number for QSPI.

#### Parameters

- `base` – QSPI base pointer.

`void QSPI_Init(QuadSPI_Type *base, qspi_config_t *config, uint32_t srcClock_Hz)`

Initializes the QSPI module and internal state.

This function enables the clock for QSPI and also configures the QSPI with the input configuration parameters. Users should call this function before any QSPI operations.

#### Parameters

- `base` – Pointer to QuadSPI Type.
- `config` – QSPI configure structure.
- `srcClock_Hz` – QSPI source clock frequency in Hz.

`void QSPI_GetDefaultQspiConfig(qspi_config_t *config)`

Gets default settings for QSPI.

#### Parameters

- `config` – QSPI configuration structure.

void QSPI\_Deinit(QuadSPI\_Type \*base)

Deinitializes the QSPI module.

Clears the QSPI state and QSPI module registers.

**Parameters**

- base – Pointer to QuadSPI Type.

void QSPI\_SetFlashConfig(QuadSPI\_Type \*base, *qspi\_flash\_config\_t* \*config)

Configures the serial flash parameter.

This function configures the serial flash 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 QSPI features.

**Parameters**

- base – Pointer to QuadSPI Type.
- config – Flash configuration parameters.

void QSPI\_SetDqsConfig(QuadSPI\_Type \*base, *qspi\_dqs\_config\_t* \*config)

Configures the serial flash DQS parameter.

This function configures the serial flash DQS relevant parameters, such as the delay chain tap number, . DQS shift phase, whether need to inverse and the rxc sample clock selection.

**Parameters**

- base – Pointer to QuadSPI Type.
- config – Dqs configuration parameters.

void QSPI\_SoftwareReset(QuadSPI\_Type \*base)

Software reset for the QSPI 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 – Pointer to QuadSPI Type.

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

Enables or disables the QSPI module.

**Parameters**

- base – Pointer to QuadSPI Type.
- enable – True means enable QSPI, false means disable.

static inline uint32\_t QSPI\_GetStatusFlags(QuadSPI\_Type \*base)

Gets the state value of QSPI.

**Parameters**

- base – Pointer to QuadSPI Type.

**Returns**

status flag, use status flag to AND `_qspi_flags` could get the related status.

static inline uint32\_t QSPI\_GetErrorStatusFlags(QuadSPI\_Type \*base)

Gets QSPI error status flags.

**Parameters**

- base – Pointer to QuadSPI Type.

**Returns**

status flag, use status flag to AND `_qspi_error_flags` could get the related status.

```
static inline void QSPI_ClearErrorFlag(QuadSPI_Type *base, uint32_t mask)
```

Clears the QSPI error flags.

**Parameters**

- `base` – Pointer to QuadSPI Type.
- `mask` – Which kind of QSPI flags to be cleared, a combination of `_qspi_error_flags`.

```
static inline void QSPI_EnableInterrupts(QuadSPI_Type *base, uint32_t mask)
```

Enables the QSPI interrupts.

**Parameters**

- `base` – Pointer to QuadSPI Type.
- `mask` – QSPI interrupt source.

```
static inline void QSPI_DisableInterrupts(QuadSPI_Type *base, uint32_t mask)
```

Disables the QSPI interrupts.

**Parameters**

- `base` – Pointer to QuadSPI Type.
- `mask` – QSPI interrupt source.

```
static inline void QSPI_EnableDMA(QuadSPI_Type *base, uint32_t mask, bool enable)
```

Enables the QSPI DMA source.

**Parameters**

- `base` – Pointer to QuadSPI Type.
- `mask` – QSPI DMA source.
- `enable` – True means enable DMA, false means disable.

```
static inline uint32_t QSPI_GetTxDataRegisterAddress(QuadSPI_Type *base)
```

Gets the Tx data register address. It is used for DMA operation.

**Parameters**

- `base` – Pointer to QuadSPI Type.

**Returns**

QSPI Tx data register address.

```
uint32_t QSPI_GetRxDataRegisterAddress(QuadSPI_Type *base)
```

Gets the Rx data register address used for DMA operation.

This function returns the Rx data register address or Rx buffer address according to the Rx read area settings.

**Parameters**

- `base` – Pointer to QuadSPI Type.

**Returns**

QSPI Rx data register address.

```
static inline void QSPI_SetIPCommandAddress(QuadSPI_Type *base, uint32_t addr)
```

Sets the IP command address.

**Parameters**

- `base` – Pointer to QuadSPI Type.

- `addr` – IP command address.

static inline void QSPI\_SetIPCommandSize(QuadSPI\_Type \*base, uint32\_t size)

Sets the IP command size.

#### Parameters

- `base` – Pointer to QuadSPI Type.
- `size` – IP command size.

void QSPI\_ExecuteIPCommand(QuadSPI\_Type \*base, uint32\_t index)

Executes IP commands located in LUT table.

#### Parameters

- `base` – Pointer to QuadSPI Type.
- `index` – IP command located in which LUT table index.

void QSPI\_ExecuteAHBCommand(QuadSPI\_Type \*base, uint32\_t index)

Executes AHB commands located in LUT table.

#### Parameters

- `base` – Pointer to QuadSPI Type.
- `index` – AHB command located in which LUT table index.

void QSPI\_UpdateLUT(QuadSPI\_Type \*base, uint32\_t index, uint32\_t \*cmd)

Updates the LUT table.

#### Parameters

- `base` – Pointer to QuadSPI Type.
- `index` – Which LUT index needs to be located. It should be an integer divided by 4.
- `cmd` – Command sequence array.

static inline void QSPI\_ClearFifo(QuadSPI\_Type \*base, uint32\_t mask)

Clears the QSPI FIFO logic.

#### Parameters

- `base` – Pointer to QuadSPI Type.
- `mask` – Which kind of QSPI FIFO to be cleared.

static inline void QSPI\_ClearCommandSequence(QuadSPI\_Type \*base, *qspi\_command\_seq\_t* seq)

@ brief Clears the command sequence for the IP/buffer command.

This function can reset the command sequence.

#### Parameters

- `base` – QSPI base address.
- `seq` – Which command sequence need to reset, IP command, buffer command or both.

void QSPI\_SetReadDataArea(QuadSPI\_Type \*base, *qspi\_read\_area\_t* area)

@ brief Set the RX buffer readout area.

This function can set the RX buffer readout, from AHB bus or IP Bus.

#### Parameters

- `base` – QSPI base address.
- `area` – QSPI Rx buffer readout area. AHB bus buffer or IP bus buffer.

void QSPI\_WriteBlocking(QuadSPI\_Type \*base, const uint32\_t \*buffer, size\_t size)

Sends a buffer of data bytes using a blocking method.

---

**Note:** This function blocks via polling until all bytes have been sent.

---

#### Parameters

- base – QSPI base pointer
- buffer – The data bytes to send
- size – The number of data bytes to send

static inline void QSPI\_WriteData(QuadSPI\_Type \*base, uint32\_t data)

Writes data into FIFO.

#### Parameters

- base – QSPI base pointer
- data – The data bytes to send

void QSPI\_ReadBlocking(QuadSPI\_Type \*base, uint32\_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. Users shall notice that this receive size shall not bigger than 64 bytes. As this interface is used to read flash status registers. For flash contents read, please use AHB bus read, this is much more efficiency.

---

#### Parameters

- base – QSPI base pointer
- buffer – The data bytes to send
- size – The number of data bytes to receive

uint32\_t QSPI\_ReadData(QuadSPI\_Type \*base)

Receives data from data FIFO.

#### Parameters

- base – QSPI base pointer

#### Returns

The data in the FIFO.

static inline void QSPI\_TransferSendBlocking(QuadSPI\_Type \*base, *qspi\_transfer\_t* \*xfer)

Writes data to the QSPI transmit buffer.

This function writes a continuous data to the QSPI transmit FIFO. This function is a block function and can return only when finished. This function uses polling methods.

#### Parameters

- base – Pointer to QuadSPI Type.
- xfer – QSPI transfer structure.

static inline void QSPI\_TransferReceiveBlocking(QuadSPI\_Type \*base, *qspi\_transfer\_t* \*xfer)

Reads data from the QSPI receive buffer in polling way.

This function reads continuous data from the QSPI receive buffer/FIFO. This function is a blocking function and can return only when finished. This function uses polling methods. Users shall notice that this receive size shall not bigger than 64 bytes. As this interface is

used to read flash status registers. For flash contents read, please use AHB bus read, this is much more efficiency.

#### Parameters

- base – Pointer to QuadSPI Type.
- xfer – QSPI transfer structure.

FSL\_QSPI\_DRIVER\_VERSION  
QSPI driver version.

Status structure of QSPI.

*Values:*

enumerator kStatus\_QSPI\_Idle  
QSPI is in idle state

enumerator kStatus\_QSPI\_Busy  
QSPI is busy

enumerator kStatus\_QSPI\_Error  
Error occurred during QSPI transfer

enum \_qspi\_read\_area  
QSPI read data area, from IP FIFO or AHB buffer.

*Values:*

enumerator kQSPI\_ReadAHB  
QSPI read from AHB buffer.

enumerator kQSPI\_ReadIP  
QSPI read from IP FIFO.

enum \_qspi\_command\_seq  
QSPI command sequence type.

*Values:*

enumerator kQSPI\_IPSeq  
IP command sequence

enumerator kQSPI\_BufferSeq  
Buffer command sequence

enumerator kQSPI\_AllSeq

enum \_qspi\_fifo  
QSPI buffer type.

*Values:*

enumerator kQSPI\_TxFifo  
QSPI Tx FIFO

enumerator kQSPI\_RxFifo  
QSPI Rx FIFO

enumerator kQSPI\_AllFifo  
QSPI all FIFO, including Tx and Rx

enum \_qspi\_endianness

QSPI transfer endianness.

*Values:*

enumerator kQSPI\_64BigEndian

64 bits big endian

enumerator kQSPI\_32LittleEndian

32 bit little endian

enumerator kQSPI\_32BigEndian

32 bit big endian

enumerator kQSPI\_64LittleEndian

64 bit little endian

enum \_qspi\_error\_flags

QSPI error flags.

*Values:*

enumerator kQSPI\_TxBufferFill

Tx buffer fill flag

enumerator kQSPI\_TxBufferUnderrun

Tx buffer underrun flag

enumerator kQSPI\_IllegalInstruction

Illegal instruction error flag

enumerator kQSPI\_RxBufferOverflow

Rx buffer overflow flag

enumerator kQSPI\_RxBufferDrain

Rx buffer drain flag

enumerator kQSPI\_AHBIllegalTransaction

AHB illegal transaction error flag

enumerator kQSPI\_AHBIllegalBurstSize

AHB illegal burst error flag

enumerator kQSPI\_AHBBufferOverflow

AHB buffer overflow flag

enumerator kQSPI\_IPCommandTriggerDuringAHBAccess

IP command trigger during AHB access error

enumerator kQSPI\_IPCommandTriggerDuringIPAccess

IP command trigger cannot be executed

enumerator kQSPI\_IPCommandTransactionFinished

IP command transaction finished flag

enumerator kQSPI\_FlagAll

All error flag

enum \_qspi\_flags

QSPI state bit.

*Values:*

enumerator kQSPI\_TxBufferFull  
Tx buffer full flag

enumerator kQSPI\_TxDMA  
Tx DMA is requested or running

enumerator kQSPI\_TxWatermark  
Tx buffer watermark available

enumerator kQSPI\_RxDMA  
Rx DMA is requesting or running

enumerator kQSPI\_RxBufferFull  
Rx buffer full

enumerator kQSPI\_RxWatermark  
Rx buffer watermark exceeded

enumerator kQSPI\_AHB3BufferFull  
AHB buffer 3 full

enumerator kQSPI\_AHB2BufferFull  
AHB buffer 2 full

enumerator kQSPI\_AHB1BufferFull  
AHB buffer 1 full

enumerator kQSPI\_AHB0BufferFull  
AHB buffer 0 full

enumerator kQSPI\_AHB3BufferNotEmpty  
AHB buffer 3 not empty

enumerator kQSPI\_AHB2BufferNotEmpty  
AHB buffer 2 not empty

enumerator kQSPI\_AHB1BufferNotEmpty  
AHB buffer 1 not empty

enumerator kQSPI\_AHB0BufferNotEmpty  
AHB buffer 0 not empty

enumerator kQSPI\_AHBTransactionPending  
AHB access transaction pending

enumerator kQSPI\_AHBAccess  
AHB access

enumerator kQSPI\_IPAccess  
IP access

enumerator kQSPI\_Busy  
Module busy

enumerator kQSPI\_StateAll  
All flags

enum \_\_qspi\_interrupt\_enable  
QSPI interrupt enable.

*Values:*



enumerator kQSPI\_TxBufferFillInterruptEnable  
Tx buffer fill interrupt enable

enumerator kQSPI\_TxBufferUnderrunInterruptEnable  
Tx buffer underrun interrupt enable

enumerator kQSPI\_IllegalInstructionInterruptEnable  
Illegal instruction error interrupt enable

enumerator kQSPI\_RxBufferOverflowInterruptEnable  
Rx buffer overflow interrupt enable

enumerator kQSPI\_RxBufferDrainInterruptEnable  
Rx buffer drain interrupt enable

enumerator kQSPI\_AHBIllegalTransactionInterruptEnable  
AHB illegal transaction error interrupt enable

enumerator kQSPI\_AHBIllegalBurstSizeInterruptEnable  
AHB illegal burst error interrupt enable

enumerator kQSPI\_AHBBufferOverflowInterruptEnable  
AHB buffer overflow interrupt enable

enumerator kQSPI\_IPCommandTriggerDuringAHBAccessInterruptEnable  
IP command trigger during AHB access error

enumerator kQSPI\_IPCommandTriggerDuringIPAccessInterruptEnable  
IP command trigger cannot be executed

enumerator kQSPI\_IPCommandTransactionFinishedInterruptEnable  
IP command transaction finished interrupt enable

enumerator kQSPI\_AllInterruptEnable  
All error interrupt enable

enum \_\_qspi\_dma\_enable  
QSPI DMA request flag.

*Values:*

enumerator kQSPI\_TxBufferFillDMAEnable  
Tx buffer fill DMA

enumerator kQSPI\_RxBufferDrainDMAEnable  
Rx buffer drain DMA

enumerator kQSPI\_AllDDMAEnable  
All DMA source

enum \_\_qspi\_dqs\_phrase\_shift  
Phrase shift number for DQS mode.

*Values:*

enumerator kQSPI\_DQSNoPhraseShift  
No phase shift

enumerator kQSPI\_DQSPhraseShift45Degree  
Select 45 degree phase shift

enumerator kQSPI\_DQSPhraseShift90Degree  
Select 90 degree phase shift

enumerator kQSPI\_DQSPhraseShift135Degree  
Select 135 degree phase shift

enum \_qspi\_dqs\_read\_sample\_clock  
Qspi read sampling option.

Values:

enumerator kQSPI\_ReadSampleClkInternalLoopback  
Read sample clock adopts internal loopback mode.

enumerator kQSPI\_ReadSampleClkLoopbackFromDqsPad  
Dummy Read strobe generated by QSPI Controller and loopback from DQS pad.

enumerator kQSPI\_ReadSampleClkExternalInputFromDqsPad  
Flash provided Read strobe and input from DQS pad.

typedef enum \_qspi\_read\_area qspi\_read\_area\_t  
QSPI read data area, from IP FIFO or AHB buffer.

typedef enum \_qspi\_command\_seq qspi\_command\_seq\_t  
QSPI command sequence type.

typedef enum \_qspi\_fifo qspi\_fifo\_t  
QSPI buffer type.

typedef enum \_qspi\_endianness qspi\_endianness\_t  
QSPI transfer endianness.

typedef enum \_qspi\_dqs\_phrase\_shift qspi\_dqs\_phrase\_shift\_t  
Phrase shift number for DQS mode.

typedef enum \_qspi\_dqs\_read\_sample\_clock qspi\_dqs\_read\_sample\_clock\_t  
Qspi read sampling option.

typedef struct QspiDQSConfig qspi\_dqs\_config\_t  
DQS configure features.

typedef struct QspiFlashTiming qspi\_flash\_timing\_t  
Flash timing configuration.

typedef struct QspiConfig qspi\_config\_t  
QSPI configuration structure.

typedef struct \_qspi\_flash\_config qspi\_flash\_config\_t  
External flash configuration items.

typedef struct \_qspi\_transfer qspi\_transfer\_t  
Transfer structure for QSPI.

typedef struct \_ip\_command\_config ip\_command\_config\_t  
16-bit access reg for IPCR register

typedef struct \_qspi\_delay\_chain\_config qspi\_delay\_chain\_config\_t  
Slave delay chain configuration items.

QSPI\_LUT\_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)  
Macro functions for LUT table.

FSL\_FEATURE\_QSPI\_LUT\_SEQ\_UNIT

QSPI\_CMD  
Macro for QSPI LUT command.

QSPI\_ADDR  
QSPI\_DUMMY  
QSPI\_MODE  
QSPI\_MODE2  
QSPI\_MODE4  
QSPI\_READ  
QSPI\_WRITE  
QSPI\_JMP\_ON\_CS  
QSPI\_ADDR\_DDR  
QSPI\_MODE\_DDR  
QSPI\_MODE2\_DDR  
QSPI\_MODE4\_DDR  
QSPI\_READ\_DDR  
QSPI\_WRITE\_DDR  
QSPI\_DATA\_LEARN  
QSPI\_CMD\_DDR  
QSPI\_CADDR  
QSPI\_CADDR\_DDR  
QSPI\_STOP  
QSPI\_PAD\_1  
Macro for QSPI PAD.  
QSPI\_PAD\_2  
QSPI\_PAD\_4  
QSPI\_PAD\_8

struct QspiDQSConfig  
    *#include <fsl\_qspi.h>* DQS configure features.

### Public Members

uint32\_t portADelayTapNum  
    Delay chain tap number selection for QSPI port A DQS  
*qspi\_dqs\_phrase\_shift\_t* shift  
    Phase shift for internal DQS generation  
*qspi\_dqs\_read\_sample\_clock\_t* rxSampleClock  
    Read sample clock for Dqs.  
bool enableDQSClkInverse  
    Enable inverse clock for internal DQS generation

struct QspiFlashTiming  
    *#include <fsl\_qspi.h>* Flash timing configuration.

**Public Members**

uint32\_t dataHoldTime

Serial flash data in hold time

uint32\_t CSHoldTime

Serial flash CS hold time in terms of serial flash clock cycles

uint32\_t CSSetupTime

Serial flash CS setup time in terms of serial flash clock cycles

struct QspiConfig

*#include <fsl\_qspi.h>* QSPI configuration structure.

**Public Members**

uint32\_t clockSource

Clock source for QSPI module

uint32\_t baudRate

Serial flash clock baud rate

uint8\_t txWatermark

QSPI transmit watermark value

uint8\_t rxWatermark

QSPI receive watermark value.

uint32\_t AHBbufferSize[FSL\_FEATURE\_QSPI\_AHB\_BUFFER\_COUNT]

AHB buffer size.

uint8\_t AHBbufferMaster[FSL\_FEATURE\_QSPI\_AHB\_BUFFER\_COUNT]

AHB buffer master.

bool enableAHBbuffer3AllMaster

Is AHB buffer3 for all master.

*qspi\_read\_area\_t* area

Which area Rx data readout

bool enableQspi

Enable QSPI after initialization

struct \_qspi\_flash\_config

*#include <fsl\_qspi.h>* External flash configuration items.

**Public Members**

uint32\_t flashA1Size

Flash A1 size

uint32\_t flashA2Size

Flash A2 size

uint32\_t lookupable[FSL\_FEATURE\_QSPI\_LUT\_DEPTH]

Flash command in LUT

uint32\_t dataHoldTime

Data line hold time.

```

uint32_t CSHoldTime
    CS line hold time
uint32_t CSSetupTime
    CS line setup time
uint32_t cloumnspace
    Column space size
uint32_t dataLearnValue
    Data Learn value if enable data learn
qspi_endianness_t endian
    Flash data endianness.
bool enableWordAddress
    If enable word address.
struct __qspi_transfer
    #include <fsl_qspi.h> Transfer structure for QSPI.

```

### Public Members

```

uint32_t *data
    Pointer to data to transmit
size_t dataSize
    Bytes to be transmit
struct __ip_command_config
    #include <fsl_qspi.h> 16-bit access reg for IPCR register
struct __qspi_delay_chain_config
    #include <fsl_qspi.h> Slave delay chain configuration items.

```

### Public Members

```

bool highFreqDelay
    Selects delay chain for low/high frequency of operation.
union IPCR_REG

```

### Public Members

```

__IO uint32_t IPCR
    IP Configuration Register
struct __ip_command_config BITFIELD
struct BITFIELD

```

### Public Members

```

__IO uint16_t IDATZ
    16-bit access for IDATZ field in IPCR register

```

\_\_IO uint8\_t RESERVED\_0  
8-bit access for RESERVED\_0 field in IPCR register

\_\_IO uint8\_t SEQID  
8-bit access for SEQID field in IPCR register

## 2.24 RDC: Resource Domain Controller

enum \_rdc\_interrupts

RDC interrupts.

*Values:*

enumerator kRDC\_RestoreCompleteInterrupt

Interrupt generated when the RDC has completed restoring state to a recently re-powered memory regions.

enum \_rdc\_flags

RDC status.

*Values:*

enumerator kRDC\_PowerDownDomainOn

Power down domain is ON.

enum \_rdc\_access\_policy

Access permission policy.

*Values:*

enumerator kRDC\_NoAccess

Could not read or write.

enumerator kRDC\_WriteOnly

Write only.

enumerator kRDC\_ReadOnly

Read only.

enumerator kRDC\_ReadWrite

Read and write.

typedef struct \_rdc\_hardware\_config rdc\_hardware\_config\_t

RDC hardware configuration.

typedef struct \_rdc\_domain\_assignment rdc\_domain\_assignment\_t

Master domain assignment.

typedef struct \_rdc\_periph\_access\_config rdc\_periph\_access\_config\_t

Peripheral domain access permission configuration.

typedef struct \_rdc\_mem\_access\_config rdc\_mem\_access\_config\_t

Memory region domain access control configuration.

Note that when setting the `rdc_mem_access_config_t::baseAddress` and `rdc_mem_access_config_t::endAddress`, should be aligned to the region resolution, see `rdc_mem_t` definitions.

typedef struct \_rdc\_mem\_status rdc\_mem\_status\_t

Memory region access violation status.

`void RDC_Init(RDC_Type *base)`

Initializes the RDC module.

This function enables the RDC clock.

**Parameters**

- `base` – RDC peripheral base address.

`void RDC_Deinit(RDC_Type *base)`

De-initializes the RDC module.

This function disables the RDC clock.

**Parameters**

- `base` – RDC peripheral base address.

`void RDC_GetHardwareConfig(RDC_Type *base, rdc_hardware_config_t *config)`

Gets the RDC hardware configuration.

This function gets the RDC hardware configurations, including number of bus masters, number of domains, number of memory regions and number of peripherals.

**Parameters**

- `base` – RDC peripheral base address.
- `config` – Pointer to the structure to get the configuration.

`static inline void RDC_EnableInterrupts(RDC_Type *base, uint32_t mask)`

Enable interrupts.

**Parameters**

- `base` – RDC peripheral base address.
- `mask` – Interrupts to enable, it is OR'ed value of `enum _rdc_interrupts`.

`static inline void RDC_DisableInterrupts(RDC_Type *base, uint32_t mask)`

Disable interrupts.

**Parameters**

- `base` – RDC peripheral base address.
- `mask` – Interrupts to disable, it is OR'ed value of `enum _rdc_interrupts`.

`static inline uint32_t RDC_GetInterruptStatus(RDC_Type *base)`

Get the interrupt pending status.

**Parameters**

- `base` – RDC peripheral base address.

**Returns**

Interrupts pending status, it is OR'ed value of `enum _rdc_interrupts`.

`static inline void RDC_ClearInterruptStatus(RDC_Type *base, uint32_t mask)`

Clear interrupt pending status.

**Parameters**

- `base` – RDC peripheral base address.
- `mask` – Status to clear, it is OR'ed value of `enum _rdc_interrupts`.

```
static inline uint32_t RDC_GetStatus(RDC_Type *base)
```

Get RDC status.

**Parameters**

- base – RDC peripheral base address.

**Returns**

mask RDC status, it is OR'ed value of enum `_rdc_flags`.

```
static inline void RDC_ClearStatus(RDC_Type *base, uint32_t mask)
```

Clear RDC status.

**Parameters**

- base – RDC peripheral base address.
- mask – RDC status to clear, it is OR'ed value of enum `_rdc_flags`.

```
void RDC_SetMasterDomainAssignment(RDC_Type *base, rdc_master_t master, const  
                                   rdc_domain_assignment_t *domainAssignment)
```

Set master domain assignment.

**Parameters**

- base – RDC peripheral base address.
- master – Which master to set.
- domainAssignment – Pointer to the assignment.

```
void RDC_GetDefaultMasterDomainAssignment(rdc_domain_assignment_t *domainAssignment)
```

Get default master domain assignment.

The default configuration is:

```
assignment->domainId = 0U;  
assignment->lock = 0U;
```

**Parameters**

- domainAssignment – Pointer to the assignment.

```
static inline void RDC_LockMasterDomainAssignment(RDC_Type *base, rdc_master_t master)
```

Lock master domain assignment.

Once locked, it could not be unlocked until next reset.

**Parameters**

- base – RDC peripheral base address.
- master – Which master to lock.

```
void RDC_SetPeriphAccessConfig(RDC_Type *base, const rdc_periph_access_config_t *config)
```

Set peripheral access policy.

**Parameters**

- base – RDC peripheral base address.
- config – Pointer to the policy configuration.

```
void RDC_GetDefaultPeriphAccessConfig(rdc_periph_access_config_t *config)
```

Get default peripheral access policy.

The default configuration is:



```
config->lock = false;
config->enableSema = false;
config->policy = RDC_ACCESS_POLICY(0, kRDC_ReadWrite) |
                RDC_ACCESS_POLICY(1, kRDC_ReadWrite) |
                RDC_ACCESS_POLICY(2, kRDC_ReadWrite) |
                RDC_ACCESS_POLICY(3, kRDC_ReadWrite);
```

### Parameters

- config – Pointer to the policy configuration.

static inline void RDC\_LockPeriphAccessConfig(RDC\_Type \*base, rdc\_periph\_t periph)

Lock peripheral access policy configuration.

Once locked, it could not be unlocked until reset.

### Parameters

- base – RDC peripheral base address.
- periph – Which peripheral to lock.

static inline uint8\_t RDC\_GetPeriphAccessPolicy(RDC\_Type \*base, rdc\_periph\_t periph, uint8\_t domainId)

Get the peripheral access policy for specific domain.

### Parameters

- base – RDC peripheral base address.
- periph – Which peripheral to get.
- domainId – Get policy for which domain.

### Returns

Access policy, see `_rdc_access_policy`.

void RDC\_SetMemAccessConfig(RDC\_Type \*base, const *rdc\_mem\_access\_config\_t* \*config)

Set memory region access policy.

Note that when setting the baseAddress and endAddress in config, should be aligned to the region resolution, see `rdc_mem_t` definitions.

### Parameters

- base – RDC peripheral base address.
- config – Pointer to the policy configuration.

void RDC\_GetDefaultMemAccessConfig(*rdc\_mem\_access\_config\_t* \*config)

Get default memory region access policy.

The default configuration is:

```
config->lock = false;
config->baseAddress = 0;
config->endAddress = 0;
config->policy = RDC_ACCESS_POLICY(0, kRDC_ReadWrite) |
                RDC_ACCESS_POLICY(1, kRDC_ReadWrite) |
                RDC_ACCESS_POLICY(2, kRDC_ReadWrite) |
                RDC_ACCESS_POLICY(3, kRDC_ReadWrite);
```

### Parameters

- config – Pointer to the policy configuration.

static inline void RDC\_LockMemAccessConfig(RDC\_Type \*base, rdc\_mem\_t mem)

Lock memory access policy configuration.

Once locked, it could not be unlocked until reset. After locked, you can only call RDC\_SetMemAccessValid to enable the configuration, but can not disable it or change other settings.

**Parameters**

- base – RDC peripheral base address.
- mem – Which memory region to lock.

static inline void RDC\_SetMemAccessValid(RDC\_Type \*base, rdc\_mem\_t mem, bool valid)

Enable or disable memory access policy configuration.

**Parameters**

- base – RDC peripheral base address.
- mem – Which memory region to operate.
- valid – Pass in true to valid, false to invalid.

void RDC\_GetMemViolationStatus(RDC\_Type \*base, rdc\_mem\_t mem, *rdc\_mem\_status\_t* \*status)

Get the memory region violation status.

The first access violation is captured. Subsequent violations are ignored until the status register is cleared. Contents are cleared upon reading the register. Clearing of contents occurs only when the status is read by the memory region's associated domain ID(s).

**Parameters**

- base – RDC peripheral base address.
- mem – Which memory region to get.
- status – The returned status.

static inline void RDC\_ClearMemViolationFlag(RDC\_Type \*base, rdc\_mem\_t mem)

Clear the memory region violation flag.

**Parameters**

- base – RDC peripheral base address.
- mem – Which memory region to clear.

static inline uint8\_t RDC\_GetMemAccessPolicy(RDC\_Type \*base, rdc\_mem\_t mem, uint8\_t domainId)

Get the memory region access policy for specific domain.

**Parameters**

- base – RDC peripheral base address.
- mem – Which memory region to get.
- domainId – Get policy for which domain.

**Returns**

Access policy, see `_rdc_access_policy`.

static inline uint8\_t RDC\_GetCurrentMasterDomainId(RDC\_Type \*base)

Gets the domain ID of the current bus master.

This function returns the domain ID of the current bus master.

**Parameters**

- base – RDC peripheral base address.

**Returns**

Domain ID of current bus master.

FSL\_RDC\_DRIVER\_VERSION

RDC\_ACCESS\_POLICY(domainID, policy)

struct \_rdc\_hardware\_config

*#include <fsl\_rdc.h>* RDC hardware configuration.

**Public Members**

uint32\_t domainNumber

Number of domains.

uint32\_t masterNumber

Number of bus masters.

uint32\_t periphNumber

Number of peripherals.

uint32\_t memNumber

Number of memory regions.

struct \_rdc\_domain\_assignment

*#include <fsl\_rdc.h>* Master domain assignment.

**Public Members**

uint32\_t domainId

Domain ID.

uint32\_t \_\_pad0\_\_

Reserved.

uint32\_t lock

Lock the domain assignment.

struct \_rdc\_periph\_access\_config

*#include <fsl\_rdc.h>* Peripheral domain access permission configuration.

**Public Members**

rdc\_periph\_t periph

Peripheral name.

bool lock

Lock the permission until reset.

bool enableSema

Enable semaphore or not, when enabled, master should call RDC\_SEMA42\_Lock to lock the semaphore gate accordingly before access the peripheral.

uint16\_t policy

Access policy.

struct \_rdc\_mem\_access\_config

*#include <fsl\_rdc.h>* Memory region domain access control configuration.

Note that when setting the `rdc_mem_access_config_t::baseAddress` and `rdc_mem_access_config_t::endAddress`, should be aligned to the region resolution, see `rdc_mem_t` definitions.

### Public Members

`rdc_mem_t mem`

Memory region descriptor name.

`bool lock`

Lock the configuration.

`uint64_t baseAddress`

Start address of the memory region.

`uint64_t endAddress`

End address of the memory region.

`uint16_t policy`

Access policy.

struct \_rdc\_mem\_status

*#include <fsl\_rdc.h>* Memory region access violation status.

### Public Members

`bool hasViolation`

Violating happens or not.

`uint8_t domainID`

Violating Domain ID.

`uint64_t address`

Violating Address.

## 2.25 RDC\_SEMA42: Hardware Semaphores Driver

`FSL_RDC_SEMA42_DRIVER_VERSION`

RDC\_SEMA42 driver version.

`void RDC_SEMA42_Init(RDC_SEMAPHORE_Type *base)`

Initializes the RDC\_SEMA42 module.

This function initializes the RDC\_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 `RDC_SEMA42_ResetGate` or `RDC_SEMA42_ResetAllGates` function.

### Parameters

- `base` – RDC\_SEMA42 peripheral base address.

`void RDC_SEMA42_Deinit(RDC_SEMAPHORE_Type *base)`

De-initializes the RDC\_SEMA42 module.

This function de-initializes the RDC\_SEMA42 module. It only disables the clock.

**Parameters**

- base – RDC\_SEMA42 peripheral base address.

*status\_t* RDC\_SEMA42\_TryLock(RDC\_SEMAPHORE\_Type \*base, uint8\_t gateNum, uint8\_t masterIndex, uint8\_t domainId)

Tries to lock the RDC\_SEMA42 gate.

This function tries to lock the specific RDC\_SEMA42 gate. If the gate has been locked by another processor, this function returns an error code.

**Parameters**

- base – RDC\_SEMA42 peripheral base address.
- gateNum – Gate number to lock.
- masterIndex – Current processor master index.
- domainId – Current processor domain ID.

**Return values**

- kStatus\_Success – Lock the sema42 gate successfully.
- kStatus\_Failed – Sema42 gate has been locked by another processor.

*void* RDC\_SEMA42\_Lock(RDC\_SEMAPHORE\_Type \*base, uint8\_t gateNum, uint8\_t masterIndex, uint8\_t domainId)

Locks the RDC\_SEMA42 gate.

This function locks the specific RDC\_SEMA42 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

**Parameters**

- base – RDC\_SEMA42 peripheral base address.
- gateNum – Gate number to lock.
- masterIndex – Current processor master index.
- domainId – Current processor domain ID.

*static inline void* RDC\_SEMA42\_Unlock(RDC\_SEMAPHORE\_Type \*base, uint8\_t gateNum)

Unlocks the RDC\_SEMA42 gate.

This function unlocks the specific RDC\_SEMA42 gate. It only writes unlock value to the RDC\_SEMA42 gate register. However, it does not check whether the RDC\_SEMA42 gate is locked by the current processor or not. As a result, if the RDC\_SEMA42 gate is not locked by the current processor, this function has no effect.

**Parameters**

- base – RDC\_SEMA42 peripheral base address.
- gateNum – Gate number to unlock.

*static inline int32\_t* RDC\_SEMA42\_GetLockMasterIndex(RDC\_SEMAPHORE\_Type \*base, uint8\_t gateNum)

Gets which master has currently locked the gate.

**Parameters**

- base – RDC\_SEMA42 peripheral base address.
- gateNum – Gate number.

**Returns**

Return -1 if the gate is not locked by any master, otherwise return the master index.

`int32_t RDC_SEMA42_GetLockDomainID(RDC_SEMAPHORE_Type *base, uint8_t gateNum)`

Gets which domain has currently locked the gate.

**Parameters**

- `base` – RDC\_SEMA42 peripheral base address.
- `gateNum` – Gate number.

**Returns**

Return -1 if the gate is not locked by any domain, otherwise return the domain ID.

`status_t RDC_SEMA42_ResetGate(RDC_SEMAPHORE_Type *base, uint8_t gateNum)`

Resets the RDC\_SEMA42 gate to an unlocked status.

This function resets a RDC\_SEMA42 gate to an unlocked status.

**Parameters**

- `base` – RDC\_SEMA42 peripheral base address.
- `gateNum` – Gate number.

**Return values**

- `kStatus_Success` – RDC\_SEMA42 gate is reset successfully.
- `kStatus_Failed` – Some other reset process is ongoing.

`static inline status_t RDC_SEMA42_ResetAllGates(RDC_SEMAPHORE_Type *base)`

Resets all RDC\_SEMA42 gates to an unlocked status.

This function resets all RDC\_SEMA42 gate to an unlocked status.

**Parameters**

- `base` – RDC\_SEMA42 peripheral base address.

**Return values**

- `kStatus_Success` – RDC\_SEMA42 is reset successfully.
- `kStatus_RDC_SEMA42_Reseting` – Some other reset process is ongoing.

`RDC_SEMA42_GATE_NUM_RESET_ALL`

The number to reset all RDC\_SEMA42 gates.

`RDC_SEMA42_GATEn(base, n)`

RDC\_SEMA42 gate n register address.

`RDC_SEMA42_GATE_COUNT`

RDC\_SEMA42 gate count.

`RDC_SEMAPHORE_GATE_GTFSM_MASK`

## 2.26 SAI: Serial Audio Interface

### 2.27 SAI Driver

`void SAI_Init(I2S_Type *base)`

Initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless `SAI_Init` is called to enable the clock.

**Parameters**

- base – SAI base pointer.

void SAI\_Deinit(I2S\_Type \*base)

De-initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless SAI\_TxInit or SAI\_RxInit is called to enable the clock.

**Parameters**

- base – SAI base pointer.

void SAI\_TxReset(I2S\_Type \*base)

Resets the SAI Tx.

This function enables the software reset and FIFO reset of SAI Tx. After reset, clear the reset bit.

**Parameters**

- base – SAI base pointer

void SAI\_RxReset(I2S\_Type \*base)

Resets the SAI Rx.

This function enables the software reset and FIFO reset of SAI Rx. After reset, clear the reset bit.

**Parameters**

- base – SAI base pointer

void SAI\_TxEnable(I2S\_Type \*base, bool enable)

Enables/disables the SAI Tx.

**Parameters**

- base – SAI base pointer.
- enable – True means enable SAI Tx, false means disable.

void SAI\_RxEnable(I2S\_Type \*base, bool enable)

Enables/disables the SAI Rx.

**Parameters**

- base – SAI base pointer.
- enable – True means enable SAI Rx, false means disable.

static inline void SAI\_TxSetBitClockDirection(I2S\_Type \*base, sai\_master\_slave\_t masterSlave)

Set Rx bit clock direction.

Select bit clock direction, master or slave.

**Parameters**

- base – SAI base pointer.
- masterSlave – reference sai\_master\_slave\_t.

static inline void SAI\_RxSetBitClockDirection(I2S\_Type \*base, sai\_master\_slave\_t masterSlave)

Set Rx bit clock direction.

Select bit clock direction, master or slave.

**Parameters**

- base – SAI base pointer.

- masterSlave – reference sai\_master\_slave\_t.

```
static inline void SAI_RxSetFrameSyncDirection(I2S_Type *base, sai_master_slave_t
                                              masterSlave)
```

Set Rx frame sync direction.

Select frame sync direction, master or slave.

#### Parameters

- base – SAI base pointer.
- masterSlave – reference sai\_master\_slave\_t.

```
static inline void SAI_TxSetFrameSyncDirection(I2S_Type *base, sai_master_slave_t masterSlave)
```

Set Tx frame sync direction.

Select frame sync direction, master or slave.

#### Parameters

- base – SAI base pointer.
- masterSlave – reference sai\_master\_slave\_t.

```
void SAI_TxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate,
                           uint32_t bitWidth, uint32_t channelNumbers)
```

Transmitter 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 SAI_RxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate,
                           uint32_t bitWidth, uint32_t channelNumbers)
```

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 SAI_TxSetBitclockConfig(I2S_Type *base, sai_master_slave_t masterSlave, sai_bit_clock_t
                             *config)
```

Transmitter Bit clock configurations.

#### Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.



```
void SAI_RxSetBitclockConfig(I2S_Type *base, sai_master_slave_t masterSlave, sai_bit_clock_t *config)
```

Receiver Bit clock configurations.

#### Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

```
void SAI_TxSetFrameSyncConfig(I2S_Type *base, sai_master_slave_t masterSlave, sai_frame_sync_t *config)
```

SAI transmitter Frame sync configurations.

#### Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

```
void SAI_RxSetFrameSyncConfig(I2S_Type *base, sai_master_slave_t masterSlave, sai_frame_sync_t *config)
```

SAI receiver Frame sync configurations.

#### Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

```
void SAI_TxSetSerialDataConfig(I2S_Type *base, sai_serial_data_t *config)
```

SAI transmitter Serial data configurations.

#### Parameters

- base – SAI base pointer.
- config – serial data configurations.

```
void SAI_RxSetSerialDataConfig(I2S_Type *base, sai_serial_data_t *config)
```

SAI receiver Serial data configurations.

#### Parameters

- base – SAI base pointer.
- config – serial data configurations.

```
void SAI_TxSetConfig(I2S_Type *base, sai_transceiver_t *config)
```

SAI transmitter configurations.

#### Parameters

- base – SAI base pointer.
- config – transmitter configurations.

```
void SAI_RxSetConfig(I2S_Type *base, sai_transceiver_t *config)
```

SAI receiver configurations.

#### Parameters

- base – SAI base pointer.
- config – receiver configurations.

```
void SAI_GetClassicI2SConfig(sai_transceiver_t *config, sai_word_width_t bitWidth,  
                             sai_mono_stereo_t mode, uint32_t saiChannelMask)
```

Get classic I2S mode configurations.

#### Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

```
void SAI_GetLeftJustifiedConfig(sai_transceiver_t *config, sai_word_width_t bitWidth,  
                                sai_mono_stereo_t mode, uint32_t saiChannelMask)
```

Get left justified mode configurations.

#### Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

```
void SAI_GetRightJustifiedConfig(sai_transceiver_t *config, sai_word_width_t bitWidth,  
                                  sai_mono_stereo_t mode, uint32_t saiChannelMask)
```

Get right justified mode configurations.

#### Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

```
void SAI_GetTDMConfig(sai_transceiver_t *config, sai_frame_sync_len_t frameSyncWidth,  
                      sai_word_width_t bitWidth, uint32_t dataWordNum, uint32_t  
                      saiChannelMask)
```

Get TDM mode configurations.

#### Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data word width.
- dataWordNum – word number in one frame.
- saiChannelMask – mask value of the channel to be enable.

```
void SAI_GetDSPConfig(sai_transceiver_t *config, sai_frame_sync_len_t frameSyncWidth,  
                      sai_word_width_t bitWidth, sai_mono_stereo_t mode, uint32_t  
                      saiChannelMask)
```

Get DSP mode configurations.

DSP/PCM MODE B configuration flow for TX. RX is similiar but uses SAI\_RxSetConfig instead of SAI\_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
SAI_TxSetConfig(base, config)
```

**Note:** DSP mode is also called PCM mode which support MODE A and MODE B, DSP/PCM MODE A configuration flow. RX is similar but uses SAI\_RxSetConfig instead of SAI\_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
config->frameSync.frameSyncEarly = true;
SAI_TxSetConfig(base, config)
```

### Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to enable.

```
static inline uint32_t SAI_TxGetStatusFlag(I2S_Type *base)
```

Gets the SAI Tx status flag state.

### Parameters

- base – SAI base pointer

### Returns

SAI Tx status flag value. Use the Status Mask to get the status value needed.

```
static inline void SAI_TxClearStatusFlags(I2S_Type *base, uint32_t mask)
```

Clears the SAI Tx status flag state.

### Parameters

- base – SAI base pointer
- mask – State mask. It can be a combination of the following source if defined:
  - kSAI\_WordStartFlag
  - kSAI\_SyncErrorFlag
  - kSAI\_FIFOErrorFlag

```
static inline uint32_t SAI_RxGetStatusFlag(I2S_Type *base)
```

Gets the SAI Tx status flag state.

### Parameters

- base – SAI base pointer

### Returns

SAI Rx status flag value. Use the Status Mask to get the status value needed.

```
static inline void SAI_RxClearStatusFlags(I2S_Type *base, uint32_t mask)
```

Clears the SAI Rx status flag state.

### Parameters

- base – SAI base pointer

- mask – State mask. It can be a combination of the following sources if defined.
  - kSAI\_WordStartFlag
  - kSAI\_SyncErrorFlag
  - kSAI\_FIFOErrorFlag

void SAI\_TxSoftwareReset(I2S\_Type \*base, *sai\_reset\_type\_t* resetType)

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like TCR1~TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

#### Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

void SAI\_RxSoftwareReset(I2S\_Type \*base, *sai\_reset\_type\_t* resetType)

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1~RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

#### Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

void SAI\_TxSetChannelFIFOMask(I2S\_Type \*base, uint8\_t mask)

Set the Tx channel FIFO enable mask.

#### Parameters

- base – SAI base pointer
- mask – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

void SAI\_RxSetChannelFIFOMask(I2S\_Type \*base, uint8\_t mask)

Set the Rx channel FIFO enable mask.

#### Parameters

- base – SAI base pointer
- mask – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

void SAI\_TxSetDataOrder(I2S\_Type \*base, *sai\_data\_order\_t* order)

Set the Tx data order.

#### Parameters

- base – SAI base pointer
- order – Data order MSB or LSB

void SAI\_RxSetDataOrder(I2S\_Type \*base, *sai\_data\_order\_t* order)

Set the Rx data order.

#### Parameters

- base – SAI base pointer
- order – Data order MSB or LSB

void SAI\_TxSetBitClockPolarity(I2S\_Type \*base, *sai\_clock\_polarity\_t* polarity)

Set the Tx data order.

**Parameters**

- base – SAI base pointer
- polarity –

void SAI\_RxSetBitClockPolarity(I2S\_Type \*base, *sai\_clock\_polarity\_t* polarity)

Set the Rx data order.

**Parameters**

- base – SAI base pointer
- polarity –

void SAI\_TxSetFrameSyncPolarity(I2S\_Type \*base, *sai\_clock\_polarity\_t* polarity)

Set the Tx data order.

**Parameters**

- base – SAI base pointer
- polarity –

void SAI\_RxSetFrameSyncPolarity(I2S\_Type \*base, *sai\_clock\_polarity\_t* polarity)

Set the Rx data order.

**Parameters**

- base – SAI base pointer
- polarity –

static inline void SAI\_TxEnableInterrupts(I2S\_Type \*base, uint32\_t mask)

Enables the SAI Tx interrupt requests.

**Parameters**

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI\_WordStartInterruptEnable
  - kSAI\_SyncErrorInterruptEnable
  - kSAI\_FIFOWarningInterruptEnable
  - kSAI\_FIFORequestInterruptEnable
  - kSAI\_FIFOErrorInterruptEnable

static inline void SAI\_RxEnableInterrupts(I2S\_Type \*base, uint32\_t mask)

Enables the SAI Rx interrupt requests.

**Parameters**

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI\_WordStartInterruptEnable
  - kSAI\_SyncErrorInterruptEnable

- kSAI\_FIFOWarningInterruptEnable
- kSAI\_FIFORequestInterruptEnable
- kSAI\_FIFOErrorInterruptEnable

static inline void SAI\_TxDisableInterrupts(I2S\_Type \*base, uint32\_t mask)

Disables the SAI Tx interrupt requests.

#### Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI\_WordStartInterruptEnable
  - kSAI\_SyncErrorInterruptEnable
  - kSAI\_FIFOWarningInterruptEnable
  - kSAI\_FIFORequestInterruptEnable
  - kSAI\_FIFOErrorInterruptEnable

static inline void SAI\_RxDisableInterrupts(I2S\_Type \*base, uint32\_t mask)

Disables the SAI Rx interrupt requests.

#### Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI\_WordStartInterruptEnable
  - kSAI\_SyncErrorInterruptEnable
  - kSAI\_FIFOWarningInterruptEnable
  - kSAI\_FIFORequestInterruptEnable
  - kSAI\_FIFOErrorInterruptEnable

static inline void SAI\_TxEnableDMA(I2S\_Type \*base, uint32\_t mask, bool enable)

Enables/disables the SAI Tx DMA requests.

#### Parameters

- base – SAI base pointer
- mask – DMA source The parameter can be combination of the following sources if defined.
  - kSAI\_FIFOWarningDMAEnable
  - kSAI\_FIFORequestDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline void SAI\_RxEnableDMA(I2S\_Type \*base, uint32\_t mask, bool enable)

Enables/disables the SAI Rx DMA requests.

#### Parameters

- base – SAI base pointer
- mask – DMA source The parameter can be a combination of the following sources if defined.
  - kSAI\_FIFOWarningDMAEnable

- kSAI\_FIFORequestDMAEnable

- enable – True means enable DMA, false means disable DMA.

static inline uintptr\_t SAI\_TxGetDataRegisterAddress(I2S\_Type \*base, uint32\_t channel)

Gets the SAI Tx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

**Parameters**

- base – SAI base pointer.
- channel – Which data channel used.

**Returns**

data register address.

static inline uintptr\_t SAI\_RxGetDataRegisterAddress(I2S\_Type \*base, uint32\_t channel)

Gets the SAI Rx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

**Parameters**

- base – SAI base pointer.
- channel – Which data channel used.

**Returns**

data register address.

void SAI\_WriteBlocking(I2S\_Type \*base, uint32\_t channel, uint32\_t bitWidth, uint8\_t \*buffer,  
uint32\_t size)

Sends data using a blocking method.

---

**Note:** This function blocks by polling until data is ready to be sent.

---

**Parameters**

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

void SAI\_WriteMultiChannelBlocking(I2S\_Type \*base, uint32\_t channel, uint32\_t channelMask,  
uint32\_t bitWidth, uint8\_t \*buffer, uint32\_t size)

Sends data to multi channel using a blocking method.

---

**Note:** This function blocks by polling until data is ready to be sent.

---

**Parameters**

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.

- buffer – Pointer to the data to be written.
- size – Bytes to be written.

```
static inline void SAI_WriteData(I2S_Type *base, uint32_t channel, uint32_t data)
```

Writes data into SAI FIFO.

#### Parameters

- base – SAI base pointer.
- channel – Data channel used.
- data – Data needs to be written.

```
void SAI_ReadBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer,  
                     uint32_t size)
```

Receives data using a blocking method.

---

**Note:** This function blocks by polling until data is ready to be sent.

---

#### Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

```
void SAI_ReadMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask,  
                                 uint32_t bitWidth, uint8_t *buffer, uint32_t size)
```

Receives multi channel data using a blocking method.

---

**Note:** This function blocks by polling until data is ready to be sent.

---

#### Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

```
static inline uint32_t SAI_ReadData(I2S_Type *base, uint32_t channel)
```

Reads data from the SAI FIFO.

#### Parameters

- base – SAI base pointer.
- channel – Data channel used.

#### Returns

Data in SAI FIFO.



```
void SAI_TransferTxCreateHandle(I2S_Type *base, sai_handle_t *handle, sai_transfer_callback_t  
                                callback, void *userData)
```

Initializes the SAI Tx handle.

This function initializes the Tx handle for the SAI Tx transactional APIs. Call this function once to get the handle initialized.

#### Parameters

- base – SAI base pointer
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function

```
void SAI_TransferRxCreateHandle(I2S_Type *base, sai_handle_t *handle, sai_transfer_callback_t  
                                callback, void *userData)
```

Initializes the SAI Rx handle.

This function initializes the Rx handle for the SAI Rx transactional APIs. Call this function once to get the handle initialized.

#### Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function.

```
void SAI_TransferTxSetConfig(I2S_Type *base, sai_handle_t *handle, sai_transceiver_t *config)  
SAI transmitter transfer configurations.
```

This function initializes the Tx, include bit clock, frame sync, master clock, serial data and fifo configurations.

#### Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- config – transmitter configurations.

```
void SAI_TransferRxSetConfig(I2S_Type *base, sai_handle_t *handle, sai_transceiver_t *config)  
SAI receiver transfer configurations.
```

This function initializes the Rx, include bit clock, frame sync, master clock, serial data and fifo configurations.

#### Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- config – receiver configurations.

```
status_t SAI_TransferSendNonBlocking(I2S_Type *base, sai_handle_t *handle, sai_transfer_t  
                                      *xfer)
```

Performs an interrupt non-blocking send transfer on SAI.

---

**Note:** This API returns immediately after the transfer initiates. Call the `SAI_TxGetTransferStatusIRQ` to poll the transfer status and check whether the transfer is finished. If the return status is not `kStatus_SAI_Busy`, the transfer is finished.

---

#### Parameters

- `base` – SAI base pointer.
- `handle` – Pointer to the `sai_handle_t` structure which stores the transfer state.
- `xfer` – Pointer to the `sai_transfer_t` structure.

#### Return values

- `kStatus_Success` – Successfully started the data receive.
- `kStatus_SAI_TxBusy` – Previous receive still not finished.
- `kStatus_InvalidArgument` – The input parameter is invalid.

`status_t` `SAI_TransferReceiveNonBlocking(I2S_Type *base, sai_handle_t *handle, sai_transfer_t *xfer)`

Performs an interrupt non-blocking receive transfer on SAI.

---

**Note:** This API returns immediately after the transfer initiates. Call the `SAI_RxGetTransferStatusIRQ` to poll the transfer status and check whether the transfer is finished. If the return status is not `kStatus_SAI_Busy`, the transfer is finished.

---

#### Parameters

- `base` – SAI base pointer
- `handle` – Pointer to the `sai_handle_t` structure which stores the transfer state.
- `xfer` – Pointer to the `sai_transfer_t` structure.

#### Return values

- `kStatus_Success` – Successfully started the data receive.
- `kStatus_SAI_RxBusy` – Previous receive still not finished.
- `kStatus_InvalidArgument` – The input parameter is invalid.

`status_t` `SAI_TransferGetSendCount(I2S_Type *base, sai_handle_t *handle, size_t *count)`

Gets a set byte count.

#### Parameters

- `base` – SAI base pointer.
- `handle` – Pointer to the `sai_handle_t` structure which stores the transfer state.
- `count` – Bytes count sent.

#### Return values

- `kStatus_Success` – Succeed get the transfer count.
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

*status\_t* SAI\_TransferGetReceiveCount(I2S\_Type \*base, *sai\_handle\_t* \*handle, *size\_t* \*count)

Gets a received byte count.

#### Parameters

- base – SAI base pointer.
- handle – Pointer to the *sai\_handle\_t* structure which stores the transfer state.
- count – Bytes count received.

#### Return values

- kStatus\_Success – Succeed get the transfer count.
- kStatus\_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void SAI\_TransferAbortSend(I2S\_Type \*base, *sai\_handle\_t* \*handle)

Aborts the current send.

---

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

---

#### Parameters

- base – SAI base pointer.
- handle – Pointer to the *sai\_handle\_t* structure which stores the transfer state.

void SAI\_TransferAbortReceive(I2S\_Type \*base, *sai\_handle\_t* \*handle)

Aborts the current IRQ receive.

---

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

---

#### Parameters

- base – SAI base pointer
- handle – Pointer to the *sai\_handle\_t* structure which stores the transfer state.

void SAI\_TransferTerminateSend(I2S\_Type \*base, *sai\_handle\_t* \*handle)

Terminate all SAI send.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI\_TransferAbortSend.

#### Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI\_TransferTerminateReceive(I2S\_Type \*base, *sai\_handle\_t* \*handle)

Terminate all SAI receive.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI\_TransferAbortReceive.

#### Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI\_TransferTxHandleIRQ(I2S\_Type \*base, sai\_handle\_t \*handle)

Tx interrupt handler.

#### Parameters

- base – SAI base pointer.
- handle – Pointer to the sai\_handle\_t structure.

void SAI\_TransferRxHandleIRQ(I2S\_Type \*base, sai\_handle\_t \*handle)

Tx interrupt handler.

#### Parameters

- base – SAI base pointer.
- handle – Pointer to the sai\_handle\_t structure.

void SAI\_DriverIRQHandler(uint32\_t instance)

SAI driver IRQ handler common entry.

This function provides the common IRQ request entry for SAI.

#### Parameters

- instance – SAI instance.

FSL\_SAI\_DRIVER\_VERSION

Version 2.4.10

\_sai\_status\_t, SAI return status.

*Values:*

enumerator kStatus\_SAI\_TxBusy  
SAI Tx is busy.

enumerator kStatus\_SAI\_RxBusy  
SAI Rx is busy.

enumerator kStatus\_SAI\_TxError  
SAI Tx FIFO error.

enumerator kStatus\_SAI\_RxError  
SAI Rx FIFO error.

enumerator kStatus\_SAI\_QueueFull  
SAI transfer queue is full.

enumerator kStatus\_SAI\_TxIdle  
SAI Tx is idle

enumerator kStatus\_SAI\_RxIdle  
SAI Rx is idle

\_sai\_channel\_mask, sai channel mask value, actual channel numbers is depend soc specific

*Values:*

enumerator kSAI\_Channel0Mask  
channel 0 mask value

enumerator kSAI\_Channel1Mask  
channel 1 mask value

enumerator kSAI\_Channel2Mask  
channel 2 mask value

enumerator kSAI\_Channel3Mask  
channel 3 mask value

enumerator kSAI\_Channel4Mask  
channel 4 mask value

enumerator kSAI\_Channel5Mask  
channel 5 mask value

enumerator kSAI\_Channel6Mask  
channel 6 mask value

enumerator kSAI\_Channel7Mask  
channel 7 mask value

enum \_\_sai\_protocol  
Define the SAI bus type.

*Values:*

enumerator kSAI\_BusLeftJustified  
Uses left justified format.

enumerator kSAI\_BusRightJustified  
Uses right justified format.

enumerator kSAI\_BusI2S  
Uses I2S format.

enumerator kSAI\_BusPCMA  
Uses I2S PCM A format.

enumerator kSAI\_BusPCMB  
Uses I2S PCM B format.

enum \_\_sai\_master\_slave  
Master or slave mode.

*Values:*

enumerator kSAI\_Master  
Master mode include bclk and frame sync

enumerator kSAI\_Slave  
Slave mode include bclk and frame sync

enumerator kSAI\_Bclk\_Master\_FrameSync\_Slave  
bclk in master mode, frame sync in slave mode

enumerator kSAI\_Bclk\_Slave\_FrameSync\_Master  
bclk in slave mode, frame sync in master mode

enum \_\_sai\_mono\_stereo  
Mono or stereo audio format.

*Values:*

enumerator kSAI\_Stereo

Stereo sound.

enumerator kSAI\_MonoRight

Only Right channel have sound.

enumerator kSAI\_MonoLeft

Only left channel have sound.

enum \_sai\_data\_order

SAI data order, MSB or LSB.

*Values:*

enumerator kSAI\_DataLSB

LSB bit transferred first

enumerator kSAI\_DataMSB

MSB bit transferred first

enum \_sai\_clock\_polarity

SAI clock polarity, active high or low.

*Values:*

enumerator kSAI\_PolarityActiveHigh

Drive outputs on rising edge

enumerator kSAI\_PolarityActiveLow

Drive outputs on falling edge

enumerator kSAI\_SampleOnFallingEdge

Sample inputs on falling edge

enumerator kSAI\_SampleOnRisingEdge

Sample inputs on rising edge

enum \_sai\_sync\_mode

Synchronous or asynchronous mode.

*Values:*

enumerator kSAI\_ModeAsync

Asynchronous mode

enumerator kSAI\_ModeSync

Synchronous mode (with receiver or transmit)

enum \_sai\_mclk\_source

Master clock source.

*Values:*

enumerator kSAI\_MclkSourceSysclk

Master clock from the system clock

enumerator kSAI\_MclkSourceSelect1

Master clock from source 1

enumerator kSAI\_MclkSourceSelect2

Master clock from source 2

enumerator kSAI\_MclkSourceSelect3

Master clock from source 3

enum \_sai\_bclk\_source

Bit clock source.

*Values:*

enumerator kSAI\_BclkSourceBusclk

Bit clock using bus clock

enumerator kSAI\_BclkSourceMclkOption1

Bit clock MCLK option 1

enumerator kSAI\_BclkSourceMclkOption2

Bit clock MCLK option2

enumerator kSAI\_BclkSourceMclkOption3

Bit clock MCLK option3

enumerator kSAI\_BclkSourceMclkDiv

Bit clock using master clock divider

enumerator kSAI\_BclkSourceOtherSai0

Bit clock from other SAI device

enumerator kSAI\_BclkSourceOtherSai1

Bit clock from other SAI device

\_sai\_interrupt\_enable\_t, The SAI interrupt enable flag

*Values:*

enumerator kSAI\_WordStartInterruptEnable

Word start flag, means the first word in a frame detected

enumerator kSAI\_SyncErrorInterruptEnable

Sync error flag, means the sync error is detected

enumerator kSAI\_FIFOWarningInterruptEnable

FIFO warning flag, means the FIFO is empty

enumerator kSAI\_FIFOErrorInterruptEnable

FIFO error flag

\_sai\_dma\_enable\_t, The DMA request sources

*Values:*

enumerator kSAI\_FIFOWarningDMAEnable

FIFO warning caused by the DMA request

\_sai\_flags, The SAI status flag

*Values:*

enumerator kSAI\_WordStartFlag

Word start flag, means the first word in a frame detected

enumerator kSAI\_SyncErrorFlag

Sync error flag, means the sync error is detected

enumerator kSAI\_FIFOErrorFlag

FIFO error flag

enumerator kSAI\_FIFOWarningFlag

FIFO warning flag

enum \_sai\_reset\_type

The reset type.

*Values:*

enumerator kSAI\_ResetTypeSoftware

Software reset, reset the logic state

enumerator kSAI\_ResetTypeFIFO

FIFO reset, reset the FIFO read and write pointer

enumerator kSAI\_ResetAll

All reset.

enum \_sai\_sample\_rate

Audio sample rate.

*Values:*

enumerator kSAI\_SampleRate8KHz

Sample rate 8000 Hz

enumerator kSAI\_SampleRate11025Hz

Sample rate 11025 Hz

enumerator kSAI\_SampleRate12KHz

Sample rate 12000 Hz

enumerator kSAI\_SampleRate16KHz

Sample rate 16000 Hz

enumerator kSAI\_SampleRate22050Hz

Sample rate 22050 Hz

enumerator kSAI\_SampleRate24KHz

Sample rate 24000 Hz

enumerator kSAI\_SampleRate32KHz

Sample rate 32000 Hz

enumerator kSAI\_SampleRate44100Hz

Sample rate 44100 Hz

enumerator kSAI\_SampleRate48KHz

Sample rate 48000 Hz

enumerator kSAI\_SampleRate96KHz

Sample rate 96000 Hz

enumerator kSAI\_SampleRate192KHz

Sample rate 192000 Hz

enumerator kSAI\_SampleRate384KHz

Sample rate 384000 Hz

enum \_sai\_word\_width

Audio word width.

*Values:*



enumerator kSAI\_WordWidth8bits

Audio data width 8 bits

enumerator kSAI\_WordWidth16bits

Audio data width 16 bits

enumerator kSAI\_WordWidth24bits

Audio data width 24 bits

enumerator kSAI\_WordWidth32bits

Audio data width 32 bits

enum \_\_sai\_transceiver\_type

sai transceiver type

*Values:*

enumerator kSAI\_Transmitter

sai transmitter

enumerator kSAI\_Receiver

sai receiver

enum \_\_sai\_frame\_sync\_len

sai frame sync len

*Values:*

enumerator kSAI\_FrameSyncLenOneBitClk

1 bit clock frame sync len for DSP mode

enumerator kSAI\_FrameSyncLenPerWordWidth

Frame sync length decided by word width

typedef enum \_\_sai\_protocol sai\_protocol\_t

Define the SAI bus type.

typedef enum \_\_sai\_master\_slave sai\_master\_slave\_t

Master or slave mode.

typedef enum \_\_sai\_mono\_stereo sai\_mono\_stereo\_t

Mono or stereo audio format.

typedef enum \_\_sai\_data\_order sai\_data\_order\_t

SAI data order, MSB or LSB.

typedef enum \_\_sai\_clock\_polarity sai\_clock\_polarity\_t

SAI clock polarity, active high or low.

typedef enum \_\_sai\_sync\_mode sai\_sync\_mode\_t

Synchronous or asynchronous mode.

typedef enum \_\_sai\_mclk\_source sai\_mclk\_source\_t

Master clock source.

typedef enum \_\_sai\_bclk\_source sai\_bclk\_source\_t

Bit clock source.

typedef enum \_\_sai\_reset\_type sai\_reset\_type\_t

The reset type.

typedef struct \_\_sai\_config sai\_config\_t

SAI user configuration structure.

```
typedef enum _sai_sample_rate sai_sample_rate_t
```

Audio sample rate.

```
typedef enum _sai_word_width sai_word_width_t
```

Audio word width.

```
typedef enum _sai_transceiver_type sai_transceiver_type_t
```

sai transceiver type

```
typedef enum _sai_frame_sync_len sai_frame_sync_len_t
```

sai frame sync len

```
typedef struct _sai_transfer_format sai_transfer_format_t
```

sai transfer format

```
typedef struct _sai_bit_clock sai_bit_clock_t
```

sai bit clock configurations

```
typedef struct _sai_frame_sync sai_frame_sync_t
```

sai frame sync configurations

```
typedef struct _sai_serial_data sai_serial_data_t
```

sai serial data configurations

```
typedef struct _sai_transceiver sai_transceiver_t
```

sai transceiver configurations

```
typedef struct _sai_transfer sai_transfer_t
```

SAI transfer structure.

```
typedef struct _sai_handle sai_handle_t
```

```
typedef void (*sai_transfer_callback_t)(I2S_Type *base, sai_handle_t *handle, status_t status, void *userData)
```

SAI transfer callback prototype.

MCUX\_SDK\_SAI\_ALLOW\_NULL\_FIFO\_WATERMARK

Used to control whether SAI\_RxSetFifoConfig()/SAI\_TxSetFifoConfig() allows a NULL FIFO watermark.

If this macro is set to 0 then SAI\_RxSetFifoConfig()/SAI\_TxSetFifoConfig() will set the watermark to half of the FIFO's depth if passed a NULL watermark.

MCUX\_SDK\_SAI\_DISABLE\_IMPLICIT\_CHAN\_CONFIG

Disable implicit channel data configuration within SAI\_TxSetConfig()/SAI\_RxSetConfig().

Use this macro to control whether SAI\_RxSetConfig()/SAI\_TxSetConfig() will attempt to implicitly configure the channel data. By channel data we mean the startChannel, channelMask, endChannel, and channelNums fields from the sai\_transceiver\_t structure. By default, SAI\_TxSetConfig()/SAI\_RxSetConfig() will attempt to compute these fields, which may not be desired in cases where the user wants to set them before the call to said functions.

SAI\_XFER\_QUEUE\_SIZE

SAI transfer queue size, user can refine it according to use case.

FSL\_SAI\_HAS\_FIFO\_EXTEND\_FEATURE

sai fifo feature

```
struct _sai_config
```

*#include <fsl\_sai.h>* SAI user configuration structure.

**Public Members**

*sai\_protocol\_t* protocol  
Audio bus protocol in SAI

*sai\_sync\_mode\_t* syncMode  
SAI sync mode, control Tx/Rx clock sync

*sai\_bclk\_source\_t* bclkSource  
Bit Clock source

*sai\_master\_slave\_t* masterSlave  
Master or slave

struct *\_sai\_transfer\_format*  
*#include <fsl\_sai.h>* sai transfer format

**Public Members**

uint32\_t sampleRate\_Hz  
Sample rate of audio data

uint32\_t bitWidth  
Data length of audio data, usually 8/16/24/32 bits

*sai\_mono\_stereo\_t* stereo  
Mono or stereo

uint8\_t channel  
Transfer start channel

uint8\_t channelMask  
enabled channel mask value, reference *\_sai\_channel\_mask*

uint8\_t endChannel  
end channel number

uint8\_t channelNums  
Total enabled channel numbers

*sai\_protocol\_t* protocol  
Which audio protocol used

bool isFrameSyncCompact  
True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.

struct *\_sai\_bit\_clock*  
*#include <fsl\_sai.h>* sai bit clock configurations

**Public Members**

bool bclkInputDelay  
bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time  
.

*sai\_clock\_polarity\_t* bclkPolarity  
bit clock polarity

*sai\_bclk\_source\_t* bclkSource  
bit Clock source

struct *\_sai\_frame\_sync*  
*#include <fsl\_sai.h>* sai frame sync configurations

### Public Members

uint8\_t frameSyncWidth  
frame sync width in number of bit clocks

bool frameSyncEarly  
TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame

*sai\_clock\_polarity\_t* frameSyncPolarity  
frame sync polarity

struct *\_sai\_serial\_data*  
*#include <fsl\_sai.h>* sai serial data configurations

### Public Members

*sai\_data\_order\_t* dataOrder  
configure whether the LSB or MSB is transmitted first

uint8\_t dataWord0Length  
configure the number of bits in the first word in each frame

uint8\_t dataWordNLength  
configure the number of bits in the each word in each frame, except the first word

uint8\_t dataWordLength  
used to record the data length for dma transfer

uint8\_t dataFirstBitShifted  
Configure the bit index for the first bit transmitted for each word in the frame

uint8\_t dataWordNum  
configure the number of words in each frame

uint32\_t dataMaskedWord  
configure whether the transmit word is masked

struct *\_sai\_transceiver*  
*#include <fsl\_sai.h>* sai transceiver configurations

### Public Members

*sai\_serial\_data\_t* serialData  
serial data configurations

*sai\_frame\_sync\_t* frameSync  
ws configurations

*sai\_bit\_clock\_t* bitClock  
bit clock configurations

*sai\_master\_slave\_t* masterSlave  
transceiver is master or slave

*sai\_sync\_mode\_t* syncMode  
transceiver sync mode

*uint8\_t* startChannel  
Transfer start channel

*uint8\_t* channelMask  
enabled channel mask value, reference *\_sai\_channel\_mask*

*uint8\_t* endChannel  
end channel number

*uint8\_t* channelNums  
Total enabled channel numbers

struct *\_sai\_transfer*  
*#include <fsl\_sai.h>* SAI transfer structure.

### Public Members

*uint8\_t* \*data  
Data start address to transfer.

*size\_t* dataSize  
Transfer size.

struct *\_sai\_handle*  
*#include <fsl\_sai.h>* SAI handle structure.

### Public Members

*I2S\_Type* \*base  
base address

*uint32\_t* state  
Transfer status

*sai\_transfer\_callback\_t* callback  
Callback function called at transfer event

*void* \*userData  
Callback parameter passed to callback function

*uint8\_t* bitWidth  
Bit width for transfer, 8/16/24/32 bits

*uint8\_t* channel  
Transfer start channel

*uint8\_t* channelMask  
enabled channel mask value, refernece *\_sai\_channel\_mask*

*uint8\_t* endChannel  
end channel number

*uint8\_t* channelNums  
Total enabled channel numbers

*sai\_transfer\_t* saiQueue[(4U)]  
Transfer queue storing queued transfer

*size\_t* transferSize[(4U)]  
Data bytes need to transfer

*volatile uint8\_t* queueUser  
Index for user to queue transfer

*volatile uint8\_t* queueDriver  
Index for driver to get the transfer data and size

## 2.28 SAI SDMA Driver

```
void SAI_TransferTxCreateHandleSDMA(I2S_Type *base, sai_sdma_handle_t *handle,  
                                   sai_sdma_callback_t callback, void *userData,  
                                   sdma_handle_t *dmaHandle, uint32_t eventSource)
```

Initializes the SAI SDMA handle.

This function initializes the SAI master DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

### Parameters

- *base* – SAI base pointer.
- *handle* – SAI SDMA handle pointer.
- *base* – SAI peripheral base address.
- *callback* – Pointer to user callback function.
- *userData* – User parameter passed to the callback function.
- *dmaHandle* – SDMA handle pointer, this handle shall be static allocated by users.
- *eventSource* – SAI event source number.

```
void SAI_TransferRxCreateHandleSDMA(I2S_Type *base, sai_sdma_handle_t *handle,  
                                   sai_sdma_callback_t callback, void *userData,  
                                   sdma_handle_t *dmaHandle, uint32_t eventSource)
```

Initializes the SAI Rx SDMA handle.

This function initializes the SAI slave DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

### Parameters

- *base* – SAI base pointer.
- *handle* – SAI SDMA handle pointer.
- *base* – SAI peripheral base address.
- *callback* – Pointer to user callback function.
- *userData* – User parameter passed to the callback function.
- *dmaHandle* – SDMA handle pointer, this handle shall be static allocated by users.
- *eventSource* – SAI event source number.

---

```
status_t SAI_TransferSendSDMA(I2S_Type *base, sai_sdma_handle_t *handle, sai_transfer_t
                             *xfer)
```

Performs a non-blocking SAI transfer using DMA.

---

**Note:** This interface returns immediately after the transfer initiates. Call SAI\_GetTransferStatus to poll the transfer status and check whether the SAI transfer is finished.

---

#### Parameters

- base – SAI base pointer.
- handle – SAI SDMA handle pointer.
- xfer – Pointer to the DMA transfer structure.

#### Return values

- kStatus\_\_Success – Start a SAI SDMA send successfully.
- kStatus\_\_InvalidArgument – The input argument is invalid.
- kStatus\_\_TxBusy – SAI is busy sending data.

```
status_t SAI_TransferReceiveSDMA(I2S_Type *base, sai_sdma_handle_t *handle, sai_transfer_t
                                *xfer)
```

Performs a non-blocking SAI receive using SDMA.

---

**Note:** This interface returns immediately after the transfer initiates. Call the SAI\_GetReceiveRemainingBytes to poll the transfer status and check whether the SAI transfer is finished.

---

#### Parameters

- base – SAI base pointer
- handle – SAI SDMA handle pointer.
- xfer – Pointer to DMA transfer structure.

#### Return values

- kStatus\_\_Success – Start a SAI SDMA receive successfully.
- kStatus\_\_InvalidArgument – The input argument is invalid.
- kStatus\_\_RxBusy – SAI is busy receiving data.

```
void SAI_TransferAbortSendSDMA(I2S_Type *base, sai_sdma_handle_t *handle)
```

Aborts a SAI transfer using SDMA.

#### Parameters

- base – SAI base pointer.
- handle – SAI SDMA handle pointer.

```
void SAI_TransferAbortReceiveSDMA(I2S_Type *base, sai_sdma_handle_t *handle)
```

Aborts a SAI receive using SDMA.

#### Parameters

- base – SAI base pointer
- handle – SAI SDMA handle pointer.

```
void SAI_TransferTerminateReceiveSDMA(I2S_Type *base, sai_sdma_handle_t *handle)
```

Terminate all the SAI sdma receive transfer.

#### Parameters

- base – SAI base pointer.
- handle – SAI SDMA handle pointer.

```
void SAI_TransferTerminateSendSDMA(I2S_Type *base, sai_sdma_handle_t *handle)
```

Terminate all the SAI sdma send transfer.

#### Parameters

- base – SAI base pointer.
- handle – SAI SDMA handle pointer.

```
void SAI_TransferRxSetConfigSDMA(I2S_Type *base, sai_sdma_handle_t *handle,  
                                sai_transceiver_t *saiConfig)
```

brief Configures the SAI RX.

param base SAI base pointer. param handle SAI SDMA handle pointer. param saiConig sai configurations.

```
void SAI_TransferTxSetConfigSDMA(I2S_Type *base, sai_sdma_handle_t *handle,  
                                sai_transceiver_t *saiConfig)
```

brief Configures the SAI Tx.

param base SAI base pointer. param handle SAI SDMA handle pointer. param saiConig sai configurations.

FSL\_SAI\_SDMA\_DRIVER\_VERSION

Version 2.6.0

```
typedef struct _sai_sdma_handle sai_sdma_handle_t
```

```
typedef void (*sai_sdma_callback_t)(I2S_Type *base, sai_sdma_handle_t *handle, status_t status,  
void *userData)
```

SAI SDMA transfer callback function for finish and error.

```
struct _sai_sdma_handle
```

*#include <fsl\_sai\_sdma.h>* SAI DMA transfer handle, users should not touch the content of the handle.

#### Public Members

*sdma\_handle\_t* \*dmaHandle

DMA handler for SAI send

uint8\_t bytesPerFrame

Bytes in a frame

uint8\_t channel

start data channel

uint8\_t channelNums

total transfer channel numbers, used for multifo

uint8\_t channelMask

enabled channel mask value, refernece *\_sai\_channel\_mask*

uint8\_t fifoOffset

fifo address offset between multifo



uint32\_t count

The transfer data count in a DMA request

uint32\_t state

Internal state for SAI SDMA transfer

uint32\_t eventSource

SAI event source number

sai\_sdma\_callback\_t callback

Callback for users while transfer finish or error occurs

void \*userData

User callback parameter

sdma\_buffer\_descriptor\_t bdPool[(4U)]

BD pool for SDMA transfer.

sai\_transfer\_t saiQueue[(4U)]

Transfer queue storing queued transfer.

size\_t transferSize[(4U)]

Data bytes need to transfer

volatile uint8\_t queueUser

Index for user to queue transfer.

volatile uint8\_t queueDriver

Index for driver to get the transfer data and size

## 2.29 SDMA: Smart Direct Memory Access (SDMA) Controller Driver

void SDMA\_Init(SDMAARM\_Type \*base, const sdma\_config\_t \*config)

Initializes the SDMA peripheral.

This function ungates the SDMA clock and configures the SDMA peripheral according to the configuration structure.

---

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

---

### Parameters

- base – SDMA peripheral base address.
- config – A pointer to the configuration structure, see “sdma\_config\_t”.

void SDMA\_Deinit(SDMAARM\_Type \*base)

Deinitializes the SDMA peripheral.

This function gates the SDMA clock.

### Parameters

- base – SDMA peripheral base address.

void SDMA\_GetDefaultConfig(*sdma\_config\_t* \*config)

Gets the SDMA default configuration structure.

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

```
config.enableRealTimeDebugPin = false;
config.isSoftwareResetClearLock = true;
config.ratio = kSDMA_HalfARMClockFreq;
```

#### Parameters

- config – A pointer to the SDMA configuration structure.

void SDMA\_ResetModule(SDMAARM\_Type \*base)

Sets all SDMA core register to reset status.

If only reset ARM core, SDMA register cannot return to reset value, shall call this function to reset all SDMA register to reset value. But the internal status cannot be reset.

#### Parameters

- base – SDMA peripheral base address.

static inline void SDMA\_EnableChannelErrorInterrupts(SDMAARM\_Type \*base, uint32\_t channel)

Enables the interrupt source for the SDMA error.

Enable this will trigger an interrupt while SDMA occurs error while executing scripts.

#### Parameters

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

static inline void SDMA\_DisableChannelErrorInterrupts(SDMAARM\_Type \*base, uint32\_t channel)

Disables the interrupt source for the SDMA error.

#### Parameters

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

void SDMA\_ConfigBufferDescriptor(*sdma\_buffer\_descriptor\_t* \*bd, uint32\_t srcAddr, uint32\_t destAddr, *sdma\_transfer\_size\_t* busWidth, size\_t bufferSize, bool isLast, bool enableInterrupt, bool isWrap, *sdma\_transfer\_type\_t* type)

Sets buffer descriptor contents.

This function sets the descriptor contents such as source, dest address and status bits.

#### Parameters

- bd – Pointer to the buffer descriptor structure.
- srcAddr – Source address for the buffer descriptor.
- destAddr – Destination address for the buffer descriptor.
- busWidth – The transfer width, it only can be a member of *sdma\_transfer\_size\_t*.
- bufferSize – Buffer size for this descriptor, this number shall less than 0xFFFF. If need to transfer a big size, shall divide into several buffer descriptors.

- `isLast` – Is the buffer descriptor the last one for the channel to transfer. If only one descriptor used for the channel, this bit shall set to TRUE.
- `enableInterrupt` – If trigger an interrupt while this buffer descriptor transfer finished.
- `isWrap` – Is the buffer descriptor need to be wrapped. While this bit set to true, it will automatically wrap to the first buffer descriptor to do transfer.
- `type` – Transfer type, memory to memory, peripheral to memory or memory to peripheral.

```
static inline void SDMA_SetChannelPriority(SDMAARM_Type *base, uint32_t channel, uint8_t priority)
```

Set SDMA channel priority.

This function sets the channel priority. The default value is 0 for all channels, priority 0 will prevent channel from starting, so the priority must be set before start a channel.

#### Parameters

- `base` – SDMA peripheral base address.
- `channel` – SDMA channel number.
- `priority` – SDMA channel priority.

```
static inline void SDMA_SetSourceChannel(SDMAARM_Type *base, uint32_t source, uint32_t channelMask)
```

Set SDMA request source mapping channel.

This function sets which channel will be triggered by the dma request source.

#### Parameters

- `base` – SDMA peripheral base address.
- `source` – SDMA dma request source number.
- `channelMask` – SDMA channel mask. 1 means channel 0, 2 means channel 1, 4 means channel 3. SDMA supports an event trigger multi-channel. A channel can also be triggered by several source events.

```
static inline void SDMA_StartChannelSoftware(SDMAARM_Type *base, uint32_t channel)
```

Start a SDMA channel by software trigger.

This function start a channel.

#### Parameters

- `base` – SDMA peripheral base address.
- `channel` – SDMA channel number.

```
static inline void SDMA_StartChannelEvents(SDMAARM_Type *base, uint32_t channel)
```

Start a SDMA channel by hardware events.

This function start a channel.

#### Parameters

- `base` – SDMA peripheral base address.
- `channel` – SDMA channel number.

```
static inline void SDMA_StopChannel(SDMAARM_Type *base, uint32_t channel)
```

Stop a SDMA channel.

This function stops a channel.

#### Parameters

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

void SDMA\_SetContextSwitchMode(SDMAARM\_Type \*base, *sdma\_context\_switch\_mode\_t* mode)

Set the SDMA context switch mode.

#### Parameters

- base – SDMA peripheral base address.
- mode – SDMA context switch mode.

static inline uint32\_t SDMA\_GetChannelInterruptStatus(SDMAARM\_Type \*base)

Gets the SDMA interrupt status of all channels.

#### Parameters

- base – SDMA peripheral base address.

#### Returns

The interrupt status for all channels. Check the relevant bits for specific channel.

static inline void SDMA\_ClearChannelInterruptStatus(SDMAARM\_Type \*base, uint32\_t mask)

Clear the SDMA channel interrupt status of specific channels.

#### Parameters

- base – SDMA peripheral base address.
- mask – The interrupt status need to be cleared.

static inline uint32\_t SDMA\_GetChannelStopStatus(SDMAARM\_Type \*base)

Gets the SDMA stop status of all channels.

#### Parameters

- base – SDMA peripheral base address.

#### Returns

The stop status for all channels. Check the relevant bits for specific channel.

static inline void SDMA\_ClearChannelStopStatus(SDMAARM\_Type \*base, uint32\_t mask)

Clear the SDMA channel stop status of specific channels.

#### Parameters

- base – SDMA peripheral base address.
- mask – The stop status need to be cleared.

static inline uint32\_t SDMA\_GetChannelPendStatus(SDMAARM\_Type \*base)

Gets the SDMA channel pending status of all channels.

#### Parameters

- base – SDMA peripheral base address.

#### Returns

The pending status for all channels. Check the relevant bits for specific channel.

static inline void SDMA\_ClearChannelPendStatus(SDMAARM\_Type \*base, uint32\_t mask)

Clear the SDMA channel pending status of specific channels.

#### Parameters

- base – SDMA peripheral base address.
- mask – The pending status need to be cleared.

```
static inline uint32_t SDMA_GetErrorStatus(SDMAARM_Type *base)
```

Gets the SDMA channel error status.

SDMA channel error flag is asserted while an incoming DMA request was detected and it triggers a channel that is already pending or being serviced. This probably means there is an overflow of data for that channel.

#### Parameters

- `base` – SDMA peripheral base address.

#### Returns

The error status for all channels. Check the relevant bits for specific channel.

```
bool SDMA_GetRequestSourceStatus(SDMAARM_Type *base, uint32_t source)
```

Gets the SDMA request source pending status.

#### Parameters

- `base` – SDMA peripheral base address.
- `source` – DMA request source number.

#### Returns

True means the request source is pending, otherwise not pending.

```
void SDMA_CreateHandle(sdma_handle_t *handle, SDMAARM_Type *base, uint32_t channel,  
                      sdma_context_data_t *context)
```

Creates the SDMA handle.

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

#### Parameters

- `handle` – SDMA handle pointer. The SDMA handle stores callback function and parameters.
- `base` – SDMA peripheral base address.
- `channel` – SDMA channel number.
- `context` – Context structure for the channel to download into SDMA. Users shall make sure the context located in a non-cacheable memory, or it will cause SDMA run fail. Users shall not touch the context contents, it only be filled by SDMA driver in `SDMA_SubmitTransfer` function.

```
void SDMA_InstallBDMemory(sdma_handle_t *handle, sdma_buffer_descriptor_t *BDPool,  
                        uint32_t BDCount)
```

Installs the BDs memory pool into the SDMA handle.

This function is called after the `SDMA_CreateHandle` to use multi-buffer feature.

#### Parameters

- `handle` – SDMA handle pointer.
- `BDPool` – A memory pool to store BDs. It must be located in non-cacheable address.
- `BDCount` – The number of BD slots.

```
void SDMA_SetCallback(sdma_handle_t *handle, sdma_callback callback, void *userData)
```

Installs a callback function for the SDMA transfer.

This callback is called in the SDMA IRQ handler. Use the callback to do something after the current major loop transfer completes.

#### Parameters

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

```
void SDMA__SetMultiFifoConfig(sdma_transfer_config_t *config, uint32_t fifoNums, uint32_t  
                             fifoOffset)
```

multi fifo configurations.

This api is used to support multi fifo for SDMA, if user want to get multi fifo data, then this api should be called before submit transfer.

#### Parameters

- config – transfer configurations.
- fifoNums – fifo numbers that multi fifo operation perform, support up to 15 fifo numbers.
- fifoOffset – fifoOffset = fifo address offset / sizeof(uint32\_t) - 1.

```
void SDMA__EnableSwDone(SDMAARM_Type *base, sdma_transfer_config_t *config, uint8_t sel,  
                        sdma_peripheral_t type)
```

enable sdma sw done feature.

#### Deprecated:

Do not use this function. It has been superceded by SDMA\_SetDoneConfig.

#### Parameters

- base – SDMA base.
- config – transfer configurations.
- sel – sw done selector.
- type – peripheral type is used to determine the corresponding peripheral sw done selector bit.

```
void SDMA__SetDoneConfig(SDMAARM_Type *base, sdma_transfer_config_t *config,  
                         sdma_peripheral_t type, sdma_done_src_t doneSrc)
```

sdma channel done configurations.

#### Parameters

- base – SDMA base.
- config – transfer configurations.
- type – peripheral type.
- doneSrc – reference sdma\_done\_src\_t.

```
void SDMA__LoadScript(SDMAARM_Type *base, uint32_t destAddr, void *srcAddr, size_t  
                     bufferSizeBytes)
```

load script to sdma program memory.

#### Parameters

- base – SDMA base.
- destAddr – dest script address, should be SDMA program memory address.
- srcAddr – source address of target script.
- bufferSizeBytes – bytes size of script.

```
void SDMA_DumpScript(SDMAARM_Type *base, uint32_t srcAddr, void *destAddr, size_t  
                    bufferSizeBytes)
```

dump script from sdma program memory.

#### Parameters

- base – SDMA base.
- srcAddr – should be SDMA program memory address.
- destAddr – address to store scripts.
- bufferSizeBytes – bytes size of script.

```
static inline const char *SDMA_GetRamScriptVersion(SDMAARM_Type *base)
```

Get RAM script version.

#### Parameters

- base – SDMA base.

#### Returns

The script version of RAM.

```
void SDMA__PrepareTransfer(sdma_transfer_config_t *config, uint32_t srcAddr, uint32_t  
                          destAddr, uint32_t srcWidth, uint32_t destWidth, uint32_t  
                          bytesEachRequest, uint32_t transferSize, uint32_t eventSource,  
                          sdma_peripheral_t peripheral, sdma_transfer_type_t type)
```

Prepares the SDMA transfer structure.

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

---

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

---

#### Parameters

- config – The user configuration structure of type *sdma\_transfer\_t*.
- srcAddr – SDMA transfer source address.
- destAddr – SDMA transfer destination address.
- srcWidth – SDMA transfer source address width(bytes).
- destWidth – SDMA transfer destination address width(bytes).
- bytesEachRequest – SDMA transfer bytes per channel request.
- transferSize – SDMA transfer bytes to be transferred.
- eventSource – Event source number for the transfer, if use software trigger, just write 0.
- peripheral – Peripheral type, used to decide if need to use some special scripts.
- type – SDMA transfer type. Used to decide the correct SDMA script address in SDMA ROM.

```
void SDMA__PrepareP2PTransfer(sdma_transfer_config_t *config, uint32_t srcAddr, uint32_t  
                             destAddr, uint32_t srcWidth, uint32_t destWidth, uint32_t  
                             bytesEachRequest, uint32_t transferSize, uint32_t eventSource,  
                             uint32_t eventSource1, sdma_peripheral_t peripheral,  
                             sdma_p2p_config_t *p2p)
```

Prepares the SDMA P2P transfer structure.

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

---

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

---

#### Parameters

- `config` – The user configuration structure of type `sdma_transfer_t`.
- `srcAddr` – SDMA transfer source address.
- `destAddr` – SDMA transfer destination address.
- `srcWidth` – SDMA transfer source address width(bytes).
- `destWidth` – SDMA transfer destination address width(bytes).
- `bytesEachRequest` – SDMA transfer bytes per channel request.
- `transferSize` – SDMA transfer bytes to be transferred.
- `eventSource` – Event source number for the transfer.
- `eventSource1` – Event source1 number for the transfer.
- `peripheral` – Peripheral type, used to decide if need to use some special scripts.
- `p2p` – sdma p2p configuration pointer.

`void SDMA_SubmitTransfer(sdma_handle_t *handle, const sdma_transfer_config_t *config)`  
Submits the SDMA transfer request.

This function submits the SDMA transfer request according to the transfer configuration structure.

#### Parameters

- `handle` – SDMA handle pointer.
- `config` – Pointer to SDMA transfer configuration structure.

`void SDMA_StartTransfer(sdma_handle_t *handle)`  
SDMA starts transfer.

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

#### Parameters

- `handle` – SDMA handle pointer.

`void SDMA_StopTransfer(sdma_handle_t *handle)`  
SDMA stops transfer.

This function disables the channel request to pause the transfer. Users can call `SDMA_StartTransfer()` again to resume the transfer.

#### Parameters

- `handle` – SDMA handle pointer.



void SDMA\_AbortTransfer(*sdma\_handle\_t* \*handle)

SDMA aborts transfer.

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

#### Parameters

- handle – DMA handle pointer.

uint32\_t SDMA\_GetTransferredBytes(*sdma\_handle\_t* \*handle)

Get transferred bytes while not using BD pools.

This function returns the buffer descriptor count value if not using buffer descriptor. While do a simple transfer, which only uses one descriptor, the SDMA driver inside handle the buffer descriptor. In uart receive case, it can tell users how many data already received, also it can tells users how many data transfferd while error occurred. Notice, the count would not change while transfer is on-going using default SDMA script.

#### Parameters

- handle – DMA handle pointer.

#### Returns

Transferred bytes.

void SDMA\_HandleIRQ(*sdma\_handle\_t* \*handle)

SDMA IRQ handler for complete a buffer descriptor transfer.

This function clears the interrupt flags and also handle the CCB for the channel.

#### Parameters

- handle – SDMA handle pointer.

FSL\_SDMA\_DRIVER\_VERSION

SDMA driver version.

Version 2.4.2.

enum \_sdma\_transfer\_size

SDMA transfer configuration.

*Values:*

enumerator kSDMA\_TransferSize1Bytes

Source/Destination data transfer size is 1 byte every time

enumerator kSDMA\_TransferSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kSDMA\_TransferSize3Bytes

Source/Destination data transfer size is 3 bytes every time

enumerator kSDMA\_TransferSize4Bytes

Source/Destination data transfer size is 4 bytes every time

enum \_sdma\_bd\_status

SDMA buffer descriptor status.

*Values:*

enumerator kSDMA\_BDStatusDone

BD ownership, 0 means ARM core owns the BD, while 1 means SDMA owns BD.

enumerator kSDMA\_BDStatusWrap

While this BD is last one, the next BD will be the first one

enumerator kSDMA\_BDStatusContinuous

Buffer is allowed to transfer/receive to/from multiple buffers

enumerator kSDMA\_BDStatusInterrupt

While this BD finished, send an interrupt.

enumerator kSDMA\_BDStatusError

Error occurred on buffer descriptor command.

enumerator kSDMA\_BDStatusLast

This BD is the last BD in this array. It means the transfer ended after this buffer

enumerator kSDMA\_BDStatusExtend

Buffer descriptor extend status for SDMA scripts

enum \_sdma\_bd\_command

SDMA buffer descriptor command.

*Values:*

enumerator kSDMA\_BDCommandSETDM

Load SDMA data memory from ARM core memory buffer.

enumerator kSDMA\_BDCommandGETDM

Copy SDMA data memory to ARM core memory buffer.

enumerator kSDMA\_BDCommandSETPM

Load SDMA program memory from ARM core memory buffer.

enumerator kSDMA\_BDCommandGETPM

Copy SDMA program memory to ARM core memory buffer.

enumerator kSDMA\_BDCommandSETCTX

Load context for one channel into SDMA RAM from ARM platform memory buffer.

enumerator kSDMA\_BDCommandGETCTX

Copy context for one channel from SDMA RAM to ARM platform memory buffer.

enum \_sdma\_context\_switch\_mode

SDMA context switch mode.

*Values:*

enumerator kSDMA\_ContextSwitchModeStatic

SDMA context switch mode static

enumerator kSDMA\_ContextSwitchModeDynamicLowPower

SDMA context switch mode dynamic with low power

enumerator kSDMA\_ContextSwitchModeDynamicWithNoLoop

SDMA context switch mode dynamic with no loop

enumerator kSDMA\_ContextSwitchModeDynamic

SDMA context switch mode dynamic

enum \_sdma\_clock\_ratio

SDMA core clock frequency ratio to the ARM DMA interface.

*Values:*

enumerator kSDMA\_HalfARMClockFreq

SDMA core clock frequency half of ARM platform

enumerator kSDMA\_ARMClockFreq  
SDMA core clock frequency equals to ARM platform

enum \_sdma\_transfer\_type  
SDMA transfer type.

*Values:*

enumerator kSDMA\_MemoryToMemory  
Transfer from memory to memory

enumerator kSDMA\_PeripheralToMemory  
Transfer from peripheral to memory

enumerator kSDMA\_MemoryToPeripheral  
Transfer from memory to peripheral

enumerator kSDMA\_PeripheralToPeripheral  
Transfer from peripheral to peripheral

enum sdma\_peripheral  
Peripheral type use SDMA.

*Values:*

enumerator kSDMA\_PeripheralTypeMemory  
Peripheral DDR memory

enumerator kSDMA\_PeripheralTypeUART  
UART use SDMA

enumerator kSDMA\_PeripheralTypeUART\_SP  
UART instance in SPBA use SDMA

enumerator kSDMA\_PeripheralTypeSPDIF  
SPDIF use SDMA

enumerator kSDMA\_PeripheralNormal  
Normal peripheral use SDMA

enumerator kSDMA\_PeripheralNormal\_SP  
Normal peripheral in SPBA use SDMA

enumerator kSDMA\_PeripheralMultiFifoPDM  
multi fifo PDM

enumerator kSDMA\_PeripheralMultiFifoSaiRX  
multi fifo sai rx use SDMA

enumerator kSDMA\_PeripheralMultiFifoSaiTX  
multi fifo sai tx use SDMA

enumerator kSDMA\_PeripheralASRCM2P  
asrc m2p

enumerator kSDMA\_PeripheralASRCP2M  
asrc p2m

enumerator kSDMA\_PeripheralASRCP2P  
asrc p2p

\_sdma\_transfer\_status SDMA transfer status

*Values:*

enumerator kStatus\_SDMA\_ERROR

SDMA context error.

enumerator kStatus\_SDMA\_Busy

Channel is busy and can't handle the transfer request.

\_sdma\_multi\_fifo\_mask SDMA multi fifo mask

*Values:*

enumerator kSDMA\_MultiFifoWatermarkLevelMask

multi fifo watermark level mask

enumerator kSDMA\_MultiFifoNumsMask

multi fifo nums mask

enumerator kSDMA\_MultiFifoOffsetMask

multi fifo offset mask

enumerator kSDMA\_MultiFifoSwDoneMask

multi fifo sw done mask

enumerator kSDMA\_MultiFifoSwDoneSelectorMask

multi fifo sw done selector mask

\_sdma\_multi\_fifo\_shift SDMA multi fifo shift

*Values:*

enumerator kSDMA\_MultiFifoWatermarkLevelShift

multi fifo watermark level shift

enumerator kSDMA\_MultiFifoNumsShift

multi fifo nums shift

enumerator kSDMA\_MultiFifoOffsetShift

multi fifo offset shift

enumerator kSDMA\_MultiFifoSwDoneShift

multi fifo sw done shift

enumerator kSDMA\_MultiFifoSwDoneSelectorShift

multi fifo sw done selector shift

\_sdma\_done\_channel SDMA done channel

*Values:*

enumerator kSDMA\_DoneChannel0

SDMA done channel 0

enumerator kSDMA\_DoneChannel1

SDMA done channel 1

enumerator kSDMA\_DoneChannel2

SDMA done channel 2

enumerator kSDMA\_DoneChannel3

SDMA done channel 3

enumerator kSDMA\_DoneChannel4  
SDMA done channel 4

enumerator kSDMA\_DoneChannel5  
SDMA done channel 5

enumerator kSDMA\_DoneChannel6  
SDMA done channel 6

enumerator kSDMA\_DoneChannel7  
SDMA done channel 7

enum \_sdma\_done\_src  
SDMA done source.

*Values:*

enumerator kSDMA\_DoneSrcSW  
software done

enumerator kSDMA\_DoneSrcHwEvent0U  
HW event 0 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent1U  
HW event 1 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent2U  
HW event 2 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent3U  
HW event 3 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent4U  
HW event 4 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent5U  
HW event 5 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent6U  
HW event 6 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent7U  
HW event 7 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent8U  
HW event 8 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent9U  
HW event 9 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent10U  
HW event 10 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent11U  
HW event 11 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent12U  
HW event 12 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent13U  
HW event 13 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent14U

HW event 14 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent15U

HW event 15 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent16U

HW event 16 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent17U

HW event 17 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent18U

HW event 18 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent19U

HW event 19 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent20U

HW event 20 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent21U

HW event 21 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent22U

HW event 22 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent23U

HW event 23 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent24U

HW event 24 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent25U

HW event 25 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent26U

HW event 26 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent27U

HW event 27 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent28U

HW event 28 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent29U

HW event 29 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent30U

HW event 30 is used for DONE event

enumerator kSDMA\_DoneSrcHwEvent31U

HW event 31 is used for DONE event

typedef enum *\_sdma\_transfer\_size* sdma\_transfer\_size\_t

SDMA transfer configuration.

typedef enum *\_sdma\_bd\_status* sdma\_bd\_status\_t

SDMA buffer descriptor status.

typedef enum *\_sdma\_bd\_command* sdma\_bd\_command\_t

SDMA buffer descriptor command.

```
typedef enum _sdma_context_switch_mode sdma_context_switch_mode_t
    SDMA context switch mode.

typedef enum _sdma_clock_ratio sdma_clock_ratio_t
    SDMA core clock frequency ratio to the ARM DMA interface.

typedef enum _sdma_transfer_type sdma_transfer_type_t
    SDMA transfer type.

typedef enum sdma_peripheral sdma_peripheral_t
    Peripheral type use SDMA.

typedef enum _sdma_done_src sdma_done_src_t
    SDMA done source.

typedef struct _sdma_config sdma_config_t
    SDMA global configuration structure.

typedef struct _sdma_multi_fifo_config sdma_multi_fifo_config_t
    SDMA multi fifo configurations.

typedef struct _sdma_sw_done_config sdma_sw_done_config_t
    SDMA sw done configurations.

typedef struct _sdma_p2p_config sdma_p2p_config_t
    SDMA peripheral to peripheral R7 config.

typedef struct _sdma_transfer_config sdma_transfer_config_t
    SDMA transfer configuration.

    This structure configures the source/destination transfer attribute.

typedef struct _sdma_buffer_descriptor sdma_buffer_descriptor_t
    SDMA buffer descriptor structure.

    This structure is a buffer descriptor, this structure describes the buffer start address and
    other options

typedef struct _sdma_channel_control sdma_channel_control_t
    SDMA channel control descriptor structure.

typedef struct _sdma_context_data sdma_context_data_t
    SDMA context structure for each channel. This structure can be load into SDMA core, with
    this structure, SDMA scripts can start work.

typedef void (*sdma_callback)(struct _sdma_handle *handle, void *userData, bool transferDone,
uint32_t bdIndex)
    Define callback function for SDMA.

typedef struct _sdma_handle sdma_handle_t
    SDMA transfer handle structure.

SDMA_DRIVER_LOAD_RAM_SCRIPT

struct _sdma_config
    #include <fsl_sdma.h> SDMA global configuration structure.
```

### Public Members

```
bool enableRealTimeDebugPin
    If enable real-time debug pin, default is closed to reduce power consumption.
```

bool isSoftwareResetClearLock

If software reset clears the LOCK bit which prevent writing SDMA scripts into SDMA.

sdma\_clock\_ratio\_t ratio

SDMA core clock ratio to ARM platform DMA interface

struct \_\_sdma\_multi\_fifo\_config

*#include <fsl\_sdma.h>* SDMA multi fifo configurations.

### Public Members

uint8\_t fifoNums

fifo numbers

uint8\_t fifoOffset

offset between multi fifo data register address

struct \_\_sdma\_sw\_done\_config

*#include <fsl\_sdma.h>* SDMA sw done configurations.

### Public Members

bool enableSwDone

true is enable sw done, false is disable

uint8\_t swDoneSel

sw done channel number per peripheral type

struct \_\_sdma\_p2p\_config

*#include <fsl\_sdma.h>* SDMA peripheral to peripheral R7 config.

### Public Members

uint8\_t sourceWatermark

lower watermark value

uint8\_t destWatermark

higher water mark value

bool continuousTransfer

0: the amount of samples to be transferred is equal to the cont field of mode word 1: the amount of samples to be transferred is unknown and script will keep on transferring as long as both events are detected and script must be stopped by application.

struct \_\_sdma\_transfer\_config

*#include <fsl\_sdma.h>* SDMA transfer configuration.

This structure configures the source/destination transfer attribute.

### Public Members

uint32\_t srcAddr

Source address of the transfer

uint32\_t destAddr

Destination address of the transfer



*sdma\_transfer\_size\_t* srcTransferSize  
Source data transfer size.

*sdma\_transfer\_size\_t* destTransferSize  
Destination data transfer size.

uint32\_t bytesPerRequest  
Bytes to transfer in a minor loop

uint32\_t transferSzie  
Bytes to transfer for this descriptor

uint32\_t scriptAddr  
SDMA script address located in SDMA ROM.

uint32\_t eventSource  
Event source number for the channel. 0 means no event, use software trigger

uint32\_t eventSource1  
event source 1

bool isEventIgnore  
True means software trigger, false means hardware trigger

bool isSoftTriggerIgnore  
If ignore the HE bit, 1 means use hardware events trigger, 0 means software trigger

*sdma\_transfer\_type\_t* type  
Transfer type, transfer type used to decide the SDMA script.

*sdma\_multi\_fifo\_config\_t* multiFifo  
multi fifo configurations

*sdma\_sw\_done\_config\_t* swDone  
sw done selector

uint32\_t watermarkLevel  
watermark level

uint32\_t eventMask0  
event mask 0

uint32\_t eventMask1  
event mask 1

struct \_sdma\_buffer\_descriptor  
*#include <fsl\_sdma.h>* SDMA buffer descriptor structure.  
This structure is a buffer descriptor, this structure describes the buffer start address and other options

### Public Members

uint32\_t count  
Bytes of the buffer length for this buffer descriptor.

uint32\_t status  
E,R,I,C,W,D status bits stored here

uint32\_t command  
command mostlky used for channel 0

uint32\_t bufferAddr

Buffer start address for this descriptor.

uint32\_t extendBufferAddr

External buffer start address, this is an optional for a transfer.

struct \_\_sdma\_channel\_control

*#include <fsl\_sdma.h>* SDMA channel control descriptor structure.

### Public Members

uint32\_t currentBDAAddr

Address of current buffer descriptor processed

uint32\_t baseBDAAddr

The start address of the buffer descriptor array

uint32\_t channelDesc

Optional for transfer

uint32\_t status

Channel status

struct \_\_sdma\_context\_data

*#include <fsl\_sdma.h>* SDMA context structure for each channel. This structure can be load into SDMA core, with this structure, SDMA scripts can start work.

### Public Members

uint32\_t GeneralReg[8]

8 general registers used for SDMA RISC core

struct \_\_sdma\_handle

*#include <fsl\_sdma.h>* SDMA transfer handle structure.

### Public Members

*sdma\_callback* callback

Callback function for major count exhausted.

void \*userData

Callback function parameter.

SDMAARM\_Type \*base

SDMA peripheral base address.

*sdma\_buffer\_descriptor\_t* \*BDPool

Pointer to memory stored BD arrays.

uint32\_t bdCount

How many buffer descriptor

uint32\_t bdIndex

How many buffer descriptor

uint32\_t eventSource

Event source count for the channel

uint32\_t eventSource1

Event source 1 count for the channel

*sdma\_context\_data\_t* \*context

Channel context to execute in SDMA

uint8\_t channel

SDMA channel number.

uint8\_t priority

SDMA channel priority

uint8\_t flags

The status of the current channel.

## 2.30 SEMA4: Hardware Semaphores Driver

FSL\_SEMA4\_DRIVER\_VERSION

SEMA4 driver version.

void SEMA4\_Init(SEMA4\_Type \*base)

Initializes the SEMA4 module.

This function initializes the SEMA4 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 SEMA4\_ResetGate or SEMA4\_ResetAllGates function.

### Parameters

- base – SEMA4 peripheral base address.

void SEMA4\_Deinit(SEMA4\_Type \*base)

De-initializes the SEMA4 module.

This function de-initializes the SEMA4 module. It only disables the clock.

### Parameters

- base – SEMA4 peripheral base address.

*status\_t* SEMA4\_TryLock(SEMA4\_Type \*base, uint8\_t gateNum, uint8\_t procNum)

Tries to lock the SEMA4 gate.

This function tries to lock the specific SEMA4 gate. If the gate has been locked by another processor, this function returns an error code.

### Parameters

- base – SEMA4 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

### Return values

- kStatus\_Success – Lock the sema4 gate successfully.
- kStatus\_Fail – Sema4 gate has been locked by another processor.

*status\_t* SEMA4\_Lock(SEMA4\_Type \*base, uint8\_t gateNum, uint8\_t procNum)

Locks the SEMA4 gate.

This function locks the specific SEMA4 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

If SEMA4\_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 – SEMA4 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 SEMA4\_Unlock(SEMA4\_Type \*base, uint8\_t gateNum)

Unlocks the SEMA4 gate.

This function unlocks the specific SEMA4 gate. It only writes unlock value to the SEMA4 gate register. However, it does not check whether the SEMA4 gate is locked by the current processor or not. As a result, if the SEMA4 gate is not locked by the current processor, this function has no effect.

#### Parameters

- base – SEMA4 peripheral base address.
- gateNum – Gate number to unlock.

static inline int32\_t SEMA4\_GetLockProc(SEMA4\_Type \*base, uint8\_t gateNum)

Gets the status of the SEMA4 gate.

This function checks the lock status of a specific SEMA4 gate.

#### Parameters

- base – SEMA4 peripheral base address.
- gateNum – Gate number.

#### Returns

Return -1 if the gate is unlocked, otherwise return the processor number which has locked the gate.

*status\_t* SEMA4\_ResetGate(SEMA4\_Type \*base, uint8\_t gateNum)

Resets the SEMA4 gate to an unlocked status.

This function resets a SEMA4 gate to an unlocked status.

#### Parameters

- base – SEMA4 peripheral base address.
- gateNum – Gate number.

#### Return values

- kStatus\_Success – SEMA4 gate is reset successfully.
- kStatus\_Fail – Some other reset process is ongoing.

```
static inline status_t SEMA4_ResetAllGates(SEMA4_Type *base)
```

Resets all SEMA4 gates to an unlocked status.

This function resets all SEMA4 gate to an unlocked status.

**Parameters**

- base – SEMA4 peripheral base address.

**Return values**

- kStatus\_Success – SEMA4 is reset successfully.
- kStatus\_Fail – Some other reset process is ongoing.

```
static inline void SEMA4_EnableGateNotifyInterrupt(SEMA4_Type *base, uint8_t procNum,  
                                                    uint16_t mask)
```

Enable the gate notification interrupt.

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle.

**Parameters**

- base – SEMA4 peripheral base address.
- procNum – Current processor number.
- mask – OR'ed value of the gate index, for example: (1<<0) | (1<<1) means gate 0 and gate 1.

```
static inline void SEMA4_DisableGateNotifyInterrupt(SEMA4_Type *base, uint8_t procNum,  
                                                    uint16_t mask)
```

Disable the gate notification interrupt.

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle.

**Parameters**

- base – SEMA4 peripheral base address.
- procNum – Current processor number.
- mask – OR'ed value of the gate index, for example: (1<<0) | (1<<1) means gate 0 and gate 1.

```
static inline uint32_t SEMA4_GetGateNotifyStatus(SEMA4_Type *base, uint8_t procNum)
```

Get the gate notification flags.

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle. The status flags are cleared automatically when the gate is locked by current core or locked again before the other core.

**Parameters**

- base – SEMA4 peripheral base address.
- procNum – Current processor number.

**Returns**

OR'ed value of the gate index, for example: (1<<0) | (1<<1) means gate 0 and gate 1 flags are pending.

```
status_t SEMA4_ResetGateNotify(SEMA4_Type *base, uint8_t gateNum)
```

Resets the SEMA4 gate IRQ notification.

This function resets a SEMA4 gate IRQ notification.

**Parameters**

- base – SEMA4 peripheral base address.
- gateNum – Gate number.

**Return values**

- kStatus\_Success – Reset successfully.
- kStatus\_Fail – Some other reset process is ongoing.

static inline *status\_t* SEMA4\_ResetAllGateNotify(SEMA4\_Type \*base)

Resets all SEMA4 gates IRQ notification.

This function resets all SEMA4 gate IRQ notifications.

**Parameters**

- base – SEMA4 peripheral base address.

**Return values**

- kStatus\_Success – Reset successfully.
- kStatus\_Fail – Some other reset process is ongoing.

SEMA4\_GATE\_NUM\_RESET\_ALL

The number to reset all SEMA4 gates.

SEMA4\_GATE<sub>n</sub>(base, n)

SEMA4 gate n register address.

SEMA4\_BUSY\_POLL\_COUNT

Maximum polling iterations for SEMA4 waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the SEMA4 driver code before timing out and returning an error.

It applies to all waiting loops in SEMA4 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.

## 2.31 SNVS: Secure Non-Volatile Storage

## 2.32 Secure Non-Volatile Storage High-Power

void SNVS\_HP\_Init(SNVS\_Type \*base)

Initialize the SNVS.

---

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

---

**Parameters**

- base – SNVS peripheral base address

void SNVS\_HP\_Deinit(SNVS\_Type \*base)

Deinitialize the SNVS.

**Parameters**

- base – SNVS peripheral base address

void SNVS\_HP\_RTC\_Init(SNVS\_Type \*base, const *snvs\_hp\_rtc\_config\_t* \*config)

Ungates the SNVS clock and configures the peripheral for basic operation.

---

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

---

**Parameters**

- base – SNVS peripheral base address
- config – Pointer to the user's SNVS configuration structure.

void SNVS\_HP\_RTC\_Deinit(SNVS\_Type \*base)

Stops the RTC and SRTC timers.

**Parameters**

- base – SNVS peripheral base address

void SNVS\_HP\_RTC\_GetDefaultConfig(*snvs\_hp\_rtc\_config\_t* \*config)

Fills in the SNVS config struct with the default settings.

The default values are as follows.

```
config->rtccalenable = false;  
config->rtccalvalue = 0U;  
config->PIFreq = 0U;
```

**Parameters**

- config – Pointer to the user's SNVS configuration structure.

*status\_t* SNVS\_HP\_RTC\_SetDatetime(SNVS\_Type \*base, const *snvs\_hp\_rtc\_datetime\_t* \*datetime)

Sets the SNVS RTC date and time according to the given time structure.

**Parameters**

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

**Returns**

kStatus\_Success: Success in setting the time and starting the SNVS RTC  
kStatus\_InvalidArgument: Error because the datetime format is incorrect

void SNVS\_HP\_RTC\_GetDatetime(SNVS\_Type \*base, *snvs\_hp\_rtc\_datetime\_t* \*datetime)

Gets the SNVS RTC time and stores it in the given time structure.

**Parameters**

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

```
status_t SNVS_HP_RTC_SetAlarm(SNVS_Type *base, const snvs_hp_rtc_datetime_t
                             *alarmTime)
```

Sets the SNVS RTC alarm time.

The function sets the RTC alarm. It also 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 – SNVS peripheral base address
- alarmTime – Pointer to the structure where the alarm time is stored.

#### Returns

kStatus\_Success: success in setting the SNVS RTC alarm

kStatus\_InvalidArgument: Error because the alarm datetime format is incorrect

kStatus\_Fail: Error because the alarm time has already passed

```
void SNVS_HP_RTC_GetAlarm(SNVS_Type *base, snvs_hp_rtc_datetime_t *datetime)
```

Returns the SNVS RTC alarm time.

#### Parameters

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the alarm date and time details are stored.

```
static inline void SNVS_HP_RTC_EnableInterrupts(SNVS_Type *base, uint32_t mask)
```

Enables the selected SNVS interrupts.

#### Parameters

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

```
static inline void SNVS_HP_RTC_DisableInterrupts(SNVS_Type *base, uint32_t mask)
```

Disables the selected SNVS interrupts.

#### Parameters

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

```
uint32_t SNVS_HP_RTC_GetEnabledInterrupts(SNVS_Type *base)
```

Gets the enabled SNVS interrupts.

#### Parameters

- base – SNVS peripheral base address

#### Returns

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

:: `_snvs_hp_interrupts_t`

```
uint32_t SNVS_HP_RTC_GetStatusFlags(SNVS_Type *base)
```

Gets the SNVS status flags.

#### Parameters

- base – SNVS peripheral base address

#### Returns

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

`_snvs_hp_status_flags_t`



```
static inline void SNVS_HP_RTC_ClearStatusFlags(SNVS_Type *base, uint32_t mask)
```

Clears the SNVS status flags.

**Parameters**

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

```
static inline void SNVS_HP_RTC_StartTimer(SNVS_Type *base)
```

Starts the SNVS RTC time counter.

**Parameters**

- base – SNVS peripheral base address

```
static inline void SNVS_HP_RTC_StopTimer(SNVS_Type *base)
```

Stops the SNVS RTC time counter.

**Parameters**

- base – SNVS peripheral base address

```
static inline void SNVS_HP_EnableHighAssuranceCounter(SNVS_Type *base, bool enable)
```

Enable or disable the High Assurance Counter (HAC)

**Parameters**

- base – SNVS peripheral base address
- enable – Pass true to enable, false to disable.

```
static inline void SNVS_HP_StartHighAssuranceCounter(SNVS_Type *base, bool start)
```

Start or stop the High Assurance Counter (HAC)

**Parameters**

- base – SNVS peripheral base address
- start – Pass true to start, false to stop.

```
static inline void SNVS_HP_SetHighAssuranceCounterInitialValue(SNVS_Type *base, uint32_t value)
```

Set the High Assurance Counter (HAC) initialize value.

**Parameters**

- base – SNVS peripheral base address
- value – The initial value to set.

```
static inline void SNVS_HP_LoadHighAssuranceCounter(SNVS_Type *base)
```

Load the High Assurance Counter (HAC)

This function loads the HAC initialize value to counter register.

**Parameters**

- base – SNVS peripheral base address

```
static inline uint32_t SNVS_HP_GetHighAssuranceCounter(SNVS_Type *base)
```

Get the current High Assurance Counter (HAC) value.

**Parameters**

- base – SNVS peripheral base address

**Returns**

HAC current value.

static inline void SNVS\_HP\_ClearHighAssuranceCounter(SNVS\_Type \*base)

Clear the High Assurance Counter (HAC)

This function can be called in a functional or soft fail state. When the HAC is enabled:

- If the HAC is cleared in the soft fail state, the SSM transitions to the hard fail state immediately;
- If the HAC is cleared in functional state, the SSM will transition to hard fail immediately after transitioning to soft fail.

#### Parameters

- base – SNVS peripheral base address

static inline void SNVS\_HP\_LockHighAssuranceCounter(SNVS\_Type \*base)

Lock the High Assurance Counter (HAC)

Once locked, the HAC initialize value could not be changed, the HAC enable status could not be changed. This could only be unlocked by system reset.

#### Parameters

- base – SNVS peripheral base address

FSL\_SNVS\_HP\_DRIVER\_VERSION

Version 2.3.2

enum \_snvs\_hp\_interrupts

List of SNVS interrupts.

*Values:*

enumerator kSNVS\_RTC\_AlarmInterrupt

RTC time alarm

enumerator kSNVS\_RTC\_PeriodicInterrupt

RTC periodic interrupt

enum \_snvs\_hp\_status\_flags

List of SNVS flags.

*Values:*

enumerator kSNVS\_RTC\_AlarmInterruptFlag

RTC time alarm flag

enumerator kSNVS\_RTC\_PeriodicInterruptFlag

RTC periodic interrupt flag

enumerator kSNVS\_ZMK\_ZeroFlag

The ZMK is zero

enumerator kSNVS\_OTPMK\_ZeroFlag

The OTPMK is zero

enum \_snvs\_hp\_sv\_status\_flags

List of SNVS security violation flags.

*Values:*

enumerator kSNVS\_LP\_ViolationFlag

Low Power section Security Violation

enumerator kSNVS\_ZMK\_EccFailFlag

Zeroizable Master Key Error Correcting Code Check Failure

enumerator kSNVS\_LP\_SoftwareViolationFlag

LP Software Security Violation

enumerator kSNVS\_FatalSoftwareViolationFlag

Software Fatal Security Violation

enumerator kSNVS\_SoftwareViolationFlag

Software Security Violation

enumerator kSNVS\_Violation0Flag

Security Violation 0

enumerator kSNVS\_Violation1Flag

Security Violation 1

enumerator kSNVS\_Violation2Flag

Security Violation 2

enumerator kSNVS\_Violation4Flag

Security Violation 4

enumerator kSNVS\_Violation5Flag

Security Violation 5

enum \_snvs\_hp\_ssm\_state

List of SNVS Security State Machine State.

*Values:*

enumerator kSNVS\_SSMInit

Init

enumerator kSNVS\_SSMHardFail

Hard Fail

enumerator kSNVS\_SSMSoftFail

Soft Fail

enumerator kSNVS\_SSMInitInter

Init Intermediate (transition state between Init and Check)

enumerator kSNVS\_SSMCheck

Check

enumerator kSNVS\_SSMNonSecure

Non-Secure

enumerator kSNVS\_SSMTrusted

Trusted

enumerator kSNVS\_SSMSecure

Secure

typedef enum \_snvs\_hp\_interrupts snvs\_hp\_interrupts\_t

List of SNVS interrupts.

typedef enum \_snvs\_hp\_status\_flags snvs\_hp\_status\_flags\_t

List of SNVS flags.

typedef enum \_snvs\_hp\_sv\_status\_flags snvs\_hp\_sv\_status\_flags\_t

List of SNVS security violation flags.

```
typedef struct _snvs_hp_rtc_datetime snvs_hp_rtc_datetime_t
```

Structure is used to hold the date and time.

```
typedef struct _snvs_hp_rtc_config snvs_hp_rtc_config_t
```

SNVS config structure.

This structure holds the configuration settings for the SNVS peripheral. To initialize this structure to reasonable defaults, call the SNVS\_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

```
typedef enum _snvs_hp_ssm_state snvs_hp_ssm_state_t
```

List of SNVS Security State Machine State.

```
static inline void SNVS_HP_EnableMasterKeySelection(SNVS_Type *base, bool enable)
```

Enable or disable master key selection.

#### Parameters

- base – SNVS peripheral base address
- enable – Pass true to enable, false to disable.

```
static inline void SNVS_HP_ProgramZeroizableMasterKey(SNVS_Type *base)
```

Trigger to program Zeroizable Master Key.

#### Parameters

- base – SNVS peripheral base address

```
static inline void SNVS_HP_ChangeSSMState(SNVS_Type *base)
```

Trigger SSM State Transition.

Trigger state transition of the system security monitor (SSM). It results only the following transitions of the SSM:

- Check State -> Non-Secure (when Non-Secure Boot and not in Fab Configuration)
- Check State -> Trusted (when Secure Boot or in Fab Configuration )
- Trusted State -> Secure
- Secure State -> Trusted
- Soft Fail -> Non-Secure

#### Parameters

- base – SNVS peripheral base address

```
static inline void SNVS_HP_SetSoftwareFatalSecurityViolation(SNVS_Type *base)
```

Trigger Software Fatal Security Violation.

The result SSM state transition is:

- Check State -> Soft Fail
- Non-Secure State -> Soft Fail
- Trusted State -> Soft Fail
- Secure State -> Soft Fail

#### Parameters

- base – SNVS peripheral base address

static inline void SNVS\_HP\_SetSoftwareSecurityViolation(SNVS\_Type \*base)

Trigger Software Security Violation.

The result SSM state transition is:

- Check -> Non-Secure
- Trusted -> Soft Fail
- Secure -> Soft Fail

#### Parameters

- base – SNVS peripheral base address

static inline snvs\_hp\_ssm\_state\_t SNVS\_HP\_GetSSMState(SNVS\_Type \*base)

Get current SSM State.

#### Parameters

- base – SNVS peripheral base address

#### Returns

Current SSM state

static inline void SNVS\_HP\_ResetLP(SNVS\_Type \*base)

Reset the SNVS LP section.

Reset the LP section except SRTC and Time alarm.

#### Parameters

- base – SNVS peripheral base address

static inline uint32\_t SNVS\_HP\_GetStatusFlags(SNVS\_Type \*base)

Get the SNVS HP status flags.

The flags are returned as the OR'ed value of the enumeration :: `_snvs_hp_status_flags_t`.

#### Parameters

- base – SNVS peripheral base address

#### Returns

The OR'ed value of status flags.

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

Clear the SNVS HP status flags.

The flags to clear are passed in as the OR'ed value of the enumeration :: `_snvs_hp_status_flags_t`. Only these flags could be cleared using this API.

- `kSNVS_RTC_PeriodicInterruptFlag`
- `kSNVS_RTC_AlarmInterruptFlag`

#### Parameters

- base – SNVS peripheral base address
- mask – OR'ed value of the flags to clear.

static inline uint32\_t SNVS\_HP\_GetSecurityViolationStatusFlags(SNVS\_Type \*base)

Get the SNVS HP security violation status flags.

The flags are returned as the OR'ed value of the enumeration :: `_snvs_hp_sv_status_flags_t`.

#### Parameters

- base – SNVS peripheral base address

### Returns

The OR'ed value of security violation status flags.

static inline void SNVS\_HP\_ClearSecurityViolationStatusFlags(SNVS\_Type \*base, uint32\_t mask)

Clear the SNVS HP security violation status flags.

The flags to clear are passed in as the OR'ed value of the enumeration :: `_snvs_hp_sv_status_flags_t`. Only these flags could be cleared using this API.

- `kSNVS_ZMK_EccFailFlag`
- `kSNVS_Violation0Flag`
- `kSNVS_Violation1Flag`
- `kSNVS_Violation2Flag`
- `kSNVS_Violation3Flag`
- `kSNVS_Violation4Flag`
- `kSNVS_Violation5Flag`

### Parameters

- `base` – SNVS peripheral base address
- `mask` – OR'ed value of the flags to clear.

`SNVS_HPSVSR_SV0_MASK`

`SNVS_HPSVSR_SV1_MASK`

`SNVS_HPSVSR_SV2_MASK`

`SNVS_HPSVSR_SV4_MASK`

`SNVS_HPSVSR_SV5_MASK`

`SNVS_MAKE_HP_SV_FLAG(x)`

Macro to make security violation flag.

Macro help to make security violation flag `kSNVS_Violation0Flag` to `kSNVS_Violation5Flag`, For example, `SNVS_MAKE_HP_SV_FLAG(0)` is `kSNVS_Violation0Flag`.

struct `_snvs_hp_rtc_datetime`

*#include <fsl\_snvs\_hp.h>* Structure is used to hold the date and time.

### Public Members

`uint16_t` year

Range from 1970 to 2099.

`uint8_t` month

Range from 1 to 12.

`uint8_t` day

Range from 1 to 31 (depending on month).

`uint8_t` hour

Range from 0 to 23.

`uint8_t` minute

Range from 0 to 59.

uint8\_t second

Range from 0 to 59.

struct \_\_snvs\_\_hp\_\_rtc\_\_config

#include <fsl\_snvs\_hp.h> SNVS config structure.

This structure holds the configuration settings for the SNVS peripheral. To initialize this structure to reasonable defaults, call the SNVS\_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 rtcCalEnable

true: RTC calibration mechanism is enabled; false: No calibration is used

uint32\_t rtcCalValue

Defines signed calibration value for nonsecure RTC; This is a 5-bit 2's complement value, range from -16 to +15

uint32\_t periodicInterruptFreq

Defines frequency of the periodic interrupt; Range from 0 to 15

## 2.33 Secure Non-Volatile Storage Low-Power

void SNVS\_LP\_Init(SNVS\_Type \*base)

Ungates the SNVS clock and configures the peripheral for basic operation.

---

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

---

### Parameters

- base – SNVS peripheral base address

void SNVS\_LP\_Deinit(SNVS\_Type \*base)

Deinit the SNVS LP section.

### Parameters

- base – SNVS peripheral base address

status\_t SNVS\_LP\_SRTC\_SetDatetime(SNVS\_Type \*base, const snvs\_lp\_srtc\_datetime\_t \*datetime)

Sets the SNVS SRTC date and time according to the given time structure.

### Parameters

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

### Returns

kStatus\_Success: Success in setting the time and starting the SNVS SRTC  
kStatus\_InvalidArgument: Error because the datetime format is incorrect

```
void SNVS_LP_SRTC_GetDatetime(SNVS_Type *base, snvs_lp_srtc_datetime_t *datetime)
```

Gets the SNVS SRTC time and stores it in the given time structure.

#### Parameters

- *base* – SNVS peripheral base address
- *datetime* – Pointer to the structure where the date and time details are stored.

```
status_t SNVS_LP_SRTC_SetAlarm(SNVS_Type *base, const snvs_lp_srtc_datetime_t  
                               *alarmTime)
```

Sets the SNVS SRTC alarm time.

The function sets the SRTC alarm. It also 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. Please note, that SRTC alarm has limited resolution because only 32 most significant bits of SRTC counter are compared to SRTC Alarm register. If the alarm time is beyond SRTC resolution, the function does not set the alarm and returns an error.

#### Parameters

- *base* – SNVS peripheral base address
- *alarmTime* – Pointer to the structure where the alarm time is stored.

#### Returns

*kStatus\_Success*: success in setting the SNVS SRTC alarm  
*kStatus\_InvalidArgument*: Error because the alarm datetime format is incorrect  
*kStatus\_Fail*: Error because the alarm time has already passed or is beyond resolution

```
void SNVS_LP_SRTC_GetAlarm(SNVS_Type *base, snvs_lp_srtc_datetime_t *datetime)
```

Returns the SNVS SRTC alarm time.

#### Parameters

- *base* – SNVS peripheral base address
- *datetime* – Pointer to the structure where the alarm date and time details are stored.

```
static inline void SNVS_LP_SRTC_EnableInterrupts(SNVS_Type *base, uint32_t mask)
```

Enables the selected SNVS interrupts.

#### Parameters

- *base* – SNVS peripheral base address
- *mask* – The interrupts to enable. This is a logical OR of members of the enumeration :: *\_snvs\_lp\_srtc\_interrupts*

```
static inline void SNVS_LP_SRTC_DisableInterrupts(SNVS_Type *base, uint32_t mask)
```

Disables the selected SNVS interrupts.

#### Parameters

- *base* – SNVS peripheral base address
- *mask* – The interrupts to enable. This is a logical OR of members of the enumeration :: *\_snvs\_lp\_srtc\_interrupts*

```
uint32_t SNVS_LP_SRTC_GetEnabledInterrupts(SNVS_Type *base)
```

Gets the enabled SNVS interrupts.

#### Parameters

- *base* – SNVS peripheral base address



**Returns**

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

```
uint32_t SNVS_LP_SRTC_GetStatusFlags(SNVS_Type *base)
```

Gets the SNVS status flags.

**Parameters**

- `base` – SNVS peripheral base address

**Returns**

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

```
static inline void SNVS_LP_SRTC_ClearStatusFlags(SNVS_Type *base, uint32_t mask)
```

Clears the SNVS status flags.

**Parameters**

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

```
static inline void SNVS_LP_SRTC_StartTimer(SNVS_Type *base)
```

Starts the SNVS SRTC time counter.

**Parameters**

- `base` – SNVS peripheral base address

```
static inline void SNVS_LP_SRTC_StopTimer(SNVS_Type *base)
```

Stops the SNVS SRTC time counter.

**Parameters**

- `base` – SNVS peripheral base address

```
void SNVS_LP_PassiveTamperPin_GetDefaultConfig(snvs_lp_passive_tamper_t *config)
```

Fills in the SNVS tamper pin config struct with the default settings.

The default values are as follows. `code config->polarity = 0U`; `config->filterenable = 0U`; if available on SoC `config->filter = 0U`; if available on SoC endcode

**Parameters**

- `config` – Pointer to the user's SNVS configuration structure.

```
static inline void SNVS_LP_EnableMonotonicCounter(SNVS_Type *base, bool enable)
```

Enable or disable the Monotonic Counter.

**Parameters**

- `base` – SNVS peripheral base address
- `enable` – Pass true to enable, false to disable.

```
uint64_t SNVS_LP_GetMonotonicCounter(SNVS_Type *base)
```

Get the current Monotonic Counter.

**Parameters**

- `base` – SNVS peripheral base address

**Returns**

Current Monotonic Counter value.

```
static inline void SNVS_LP_IncreaseMonotonicCounter(SNVS_Type *base)
```

Increase the Monotonic Counter.

Increase the Monotonic Counter by 1.

#### Parameters

- base – SNVS peripheral base address

```
void SNVS_LP_WriteZeroizableMasterKey(SNVS_Type *base, uint32_t ZMKey[8U])
```

Write Zeroizable Master Key (ZMK) to the SNVS registers.

#### Parameters

- base – SNVS peripheral base address
- ZMKey – The ZMK write to the SNVS register.

```
static inline void SNVS_LP_SetZeroizableMasterKeyValid(SNVS_Type *base, bool valid)
```

Set Zeroizable Master Key valid.

This API could only be called when using software programming mode. After writing ZMK using SNVS\_LP\_WriteZeroizableMasterKey, call this API to make the ZMK valid.

#### Parameters

- base – SNVS peripheral base address
- valid – Pass true to set valid, false to set invalid.

```
static inline bool SNVS_LP_GetZeroizableMasterKeyValid(SNVS_Type *base)
```

Get Zeroizable Master Key valid status.

In hardware programming mode, call this API to check whether the ZMK is valid.

#### Parameters

- base – SNVS peripheral base address

#### Returns

true if valid, false if invalid.

```
static inline void SNVS_LP_SetZeroizableMasterKeyProgramMode(SNVS_Type *base,  
                                                             snvs_lp_zmk_program_mode_t  
                                                             mode)
```

Set Zeroizable Master Key programming mode.

#### Parameters

- base – SNVS peripheral base address
- mode – ZMK programming mode.

```
static inline void SNVS_LP_EnableZeroizableMasterKeyECC(SNVS_Type *base, bool enable)
```

Enable or disable Zeroizable Master Key ECC.

#### Parameters

- base – SNVS peripheral base address
- enable – Pass true to enable, false to disable.

```
static inline void SNVS_LP_SetMasterKeyMode(SNVS_Type *base, snvs_lp_master_key_mode_t  
                                             mode)
```

Set SNVS Master Key mode.

---

**Note:** When kSNVS\_ZMK or kSNVS\_CMK used, the SNVS\_HP must be configured to enable the master key selection.

---

**Parameters**

- base – SNVS peripheral base address
- mode – Master Key mode.

FSL\_SNVS\_LP\_DRIVER\_VERSION

Version 2.4.6

enum \_snvs\_lp\_srtc\_interrupts

List of SNVS\_LP interrupts.

*Values:*

enumerator kSNVS\_SRTC\_AlarmInterrupt

SRTC time alarm.

enum \_snvs\_lp\_srtc\_status\_flags

List of SNVS\_LP flags.

*Values:*

enumerator kSNVS\_SRTC\_AlarmInterruptFlag

SRTC time alarm flag

enum \_snvs\_lp\_external\_tamper\_status

List of SNVS\_LP external tampers status.

*Values:*

enumerator kSNVS\_TamperNotDetected

enumerator kSNVS\_TamperDetected

enum \_snvs\_lp\_external\_tamper\_polarity

SNVS\_LP external tamper polarity.

*Values:*

enumerator kSNVS\_ExternalTamperActiveLow

enumerator kSNVS\_ExternalTamperActiveHigh

enum \_snvs\_lp\_zmk\_program\_mode

SNVS\_LP Zeroizable Master Key programming mode.

*Values:*

enumerator kSNVS\_ZMKSoftwareProgram

Software programming mode.

enumerator kSNVS\_ZMKHardwareProgram

Hardware programming mode.

enum \_snvs\_lp\_master\_key\_mode

SNVS\_LP Master Key mode.

*Values:*

enumerator kSNVS\_OTPMK

One Time Programmable Master Key.

enumerator kSNVS\_ZMK

Zeroizable Master Key.

enumerator `kSNVS_CMK`

Combined Master Key, it is XOR of OPTMK and ZMK.

typedef enum `_snvs_lp_srtc_interrupts` `snvs_lp_srtc_interrupts_t`

List of SNVS\_LP interrupts.

typedef enum `_snvs_lp_srtc_status_flags` `snvs_lp_srtc_status_flags_t`

List of SNVS\_LP flags.

typedef enum `_snvs_lp_external_tamper_status` `snvs_lp_external_tamper_status_t`

List of SNVS\_LP external tamper status.

typedef enum `_snvs_lp_external_tamper_polarity` `snvs_lp_external_tamper_polarity_t`

SNVS\_LP external tamper polarity.

typedef struct `_snvs_lp_srtc_datetime` `snvs_lp_srtc_datetime_t`

Structure is used to hold the date and time.

typedef struct `_snvs_lp_srtc_config` `snvs_lp_srtc_config_t`

SNVS\_LP config structure.

This structure holds the configuration settings for the SNVS\_LP peripheral. To initialize this structure to reasonable defaults, call the `SNVS_LP_GetDefaultConfig()` function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

typedef enum `_snvs_lp_zmk_program_mode` `snvs_lp_zmk_program_mode_t`

SNVS\_LP Zeroizable Master Key programming mode.

typedef enum `_snvs_lp_master_key_mode` `snvs_lp_master_key_mode_t`

SNVS\_LP Master Key mode.

void `SNVS_LP_SRTC_Init(SNVS_Type *base, const snvs_lp_srtc_config_t *config)`

Ungates the SNVS clock and configures the peripheral for basic operation.

---

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

---

### Parameters

- `base` – SNVS peripheral base address
- `config` – Pointer to the user's SNVS configuration structure.

void `SNVS_LP_SRTC_Deinit(SNVS_Type *base)`

Stops the SRTC timer.

### Parameters

- `base` – SNVS peripheral base address

void `SNVS_LP_SRTC_GetDefaultConfig(snvs_lp_srtc_config_t *config)`

Fills in the SNVS\_LP config struct with the default settings.

The default values are as follows.

```
config->srtccalenable = false;  
config->srtccalvalue = 0U;
```

### Parameters

- `config` – Pointer to the user's SNVS configuration structure.

SNVS\_ZMK\_REG\_COUNT

Define of SNVS\_LP Zeroizable Master Key registers.

SNVS\_LP\_MAX\_TAMPER

Define of SNVS\_LP Max possible tamper.

struct snvs\_lp\_passive\_tamper\_t

*#include <fsl\_snvs\_lp.h>* Structure is used to configure SNVS LP passive tamper pins.

struct \_\_snvs\_lp\_srtc\_datetime

*#include <fsl\_snvs\_lp.h>* Structure is used to hold the date and time.

### Public Members

uint16\_t year

Range from 1970 to 2099.

uint8\_t month

Range from 1 to 12.

uint8\_t day

Range from 1 to 31 (depending on month).

uint8\_t hour

Range from 0 to 23.

uint8\_t minute

Range from 0 to 59.

uint8\_t second

Range from 0 to 59.

struct \_\_snvs\_lp\_srtc\_config

*#include <fsl\_snvs\_lp.h>* SNVS\_LP config structure.

This structure holds the configuration settings for the SNVS\_LP peripheral. To initialize this structure to reasonable defaults, call the SNVS\_LP\_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 srtcCalEnable

true: SRTC calibration mechanism is enabled; false: No calibration is used

uint32\_t srtcCalValue

Defines signed calibration value for SRTC; This is a 5-bit 2's complement value, range from -16 to +15

## 2.34 SPDIF: Sony/Philips Digital Interface

void SPDIF\_Init(SPDIF\_Type \*base, const *spdif\_config\_t* \*config)

Initializes the SPDIF peripheral.

Ungates the SPDIF clock, resets the module, and configures SPDIF with a configuration structure. The configuration structure can be custom filled or set with default values by SPDIF\_GetDefaultConfig().

---

**Note:** This API should be called at the beginning of the application to use the SPDIF driver. Otherwise, accessing the SPDIF module can cause a hard fault because the clock is not enabled.

---

### Parameters

- base – SPDIF base pointer
- config – SPDIF configuration structure.

void SPDIF\_GetDefaultConfig(*spdif\_config\_t* \*config)

Sets the SPDIF configuration structure to default values.

This API initializes the configuration structure for use in SPDIF\_Init. The initialized structure can remain unchanged in SPDIF\_Init, or it can be modified before calling SPDIF\_Init. This is an example.

```
spdif_config_t config;  
SPDIF_GetDefaultConfig(&config);
```

### Parameters

- config – pointer to master configuration structure

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

De-initializes the SPDIF peripheral.

This API gates the SPDIF clock. The SPDIF module can't operate unless SPDIF\_Init is called to enable the clock.

### Parameters

- base – SPDIF base pointer

uint32\_t SPDIF\_GetInstance(SPDIF\_Type \*base)

Get the instance number for SPDIF.

### Parameters

- base – SPDIF base pointer.

static inline void SPDIF\_TxFIFOReset(SPDIF\_Type \*base)

Resets the SPDIF Tx.

This function makes Tx FIFO in reset mode.

### Parameters

- base – SPDIF base pointer

static inline void SPDIF\_RxFIFOReset(SPDIF\_Type \*base)

Resets the SPDIF Rx.

This function enables the software reset and FIFO reset of SPDIF Rx. After reset, clear the reset bit.

### Parameters

- base – SPDIF base pointer

void SPDIF\_TxEnable(SPDIF\_Type \*base, bool enable)

Enables/disables the SPDIF Tx.

### Parameters

- base – SPDIF base pointer

- enable – True means enable SPDIF Tx, false means disable.

static inline void SPDIF\_RxEnable(SPDIF\_Type \*base, bool enable)

Enables/disables the SPDIF Rx.

#### Parameters

- base – SPDIF base pointer
- enable – True means enable SPDIF Rx, false means disable.

static inline uint32\_t SPDIF\_GetStatusFlag(SPDIF\_Type \*base)

Gets the SPDIF status flag state.

#### Parameters

- base – SPDIF base pointer

#### Returns

SPDIF status flag value. Use the `_spdif_interrupt_enable_t` to get the status value needed.

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

Clears the SPDIF status flag state.

#### Parameters

- base – SPDIF base pointer
- mask – State mask. It can be a combination of the `_spdif_interrupt_enable_t` member. Notice these members cannot be included, as these flags cannot be cleared by writing 1 to these bits:
  - kSPDIF\_UChannelReceiveRegisterFull
  - kSPDIF\_QChannelReceiveRegisterFull
  - kSPDIF\_TxFIFOEmpty
  - kSPDIF\_RxFIFOFull

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

Enables the SPDIF Tx interrupt requests.

#### Parameters

- base – SPDIF base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSPDIF\_WordStartInterruptEnable
  - kSPDIF\_SyncErrorInterruptEnable
  - kSPDIF\_FIFOWarningInterruptEnable
  - kSPDIF\_FIFORequestInterruptEnable
  - kSPDIF\_FIFOErrorInterruptEnable

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

Disables the SPDIF Tx interrupt requests.

#### Parameters

- base – SPDIF base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSPDIF\_WordStartInterruptEnable

- kSPDIF\_SyncErrorInterruptEnable
- kSPDIF\_FIFOWarningInterruptEnable
- kSPDIF\_FIFORequestInterruptEnable
- kSPDIF\_FIFOErrorInterruptEnable

static inline void SPDIF\_\_EnableDMA(SPDIF\_Type \*base, uint32\_t mask, bool enable)

Enables/disables the SPDIF DMA requests.

#### Parameters

- base – SPDIF base pointer
- mask – SPDIF DMA enable mask, The parameter can be a combination of the following sources if defined
  - kSPDIF\_RxDMAEnable
  - kSPDIF\_TxDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline uint32\_t SPDIF\_\_TxGetLeftDataRegisterAddress(SPDIF\_Type \*base)

Gets the SPDIF Tx left data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

#### Parameters

- base – SPDIF base pointer.

#### Returns

data register address.

static inline uint32\_t SPDIF\_\_TxGetRightDataRegisterAddress(SPDIF\_Type \*base)

Gets the SPDIF Tx right data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

#### Parameters

- base – SPDIF base pointer.

#### Returns

data register address.

static inline uint32\_t SPDIF\_\_RxGetLeftDataRegisterAddress(SPDIF\_Type \*base)

Gets the SPDIF Rx left data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

#### Parameters

- base – SPDIF base pointer.

#### Returns

data register address.

static inline uint32\_t SPDIF\_\_RxGetRightDataRegisterAddress(SPDIF\_Type \*base)

Gets the SPDIF Rx right data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

#### Parameters

- base – SPDIF base pointer.

#### Returns

data register address.



```
void SPDIF_TxSetSampleRate(SPDIF_Type *base, uint32_t sampleRate_Hz, uint32_t  
                           sourceClockFreq_Hz)
```

Configures the SPDIF Tx sample rate.

The audio format can be changed at run-time. This function configures the sample rate.

**Parameters**

- base – SPDIF base pointer.
- sampleRate\_Hz – SPDIF sample rate frequency in Hz.
- sourceClockFreq\_Hz – SPDIF tx clock source frequency in Hz.

```
uint32_t SPDIF_GetRxSampleRate(SPDIF_Type *base, uint32_t clockSourceFreq_Hz)
```

Configures the SPDIF Rx audio format.

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

**Parameters**

- base – SPDIF base pointer.
- clockSourceFreq\_Hz – SPDIF system clock frequency in hz.

```
void SPDIF_WriteBlocking(SPDIF_Type *base, uint8_t *buffer, uint32_t size)
```

Sends data using a blocking method.

---

**Note:** This function blocks by polling until data is ready to be sent.

---

**Parameters**

- base – SPDIF base pointer.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

```
static inline void SPDIF_WriteLeftData(SPDIF_Type *base, uint32_t data)
```

Writes data into SPDIF FIFO.

**Parameters**

- base – SPDIF base pointer.
- data – Data needs to be written.

```
static inline void SPDIF_WriteRightData(SPDIF_Type *base, uint32_t data)
```

Writes data into SPDIF FIFO.

**Parameters**

- base – SPDIF base pointer.
- data – Data needs to be written.

```
static inline void SPDIF_WriteChannelStatusHigh(SPDIF_Type *base, uint32_t data)
```

Writes data into SPDIF FIFO.

**Parameters**

- base – SPDIF base pointer.
- data – Data needs to be written.

static inline void SPDIF\_WriteChannelStatusLow(SPDIF\_Type \*base, uint32\_t data)

Writes data into SPDIF FIFO.

**Parameters**

- base – SPDIF base pointer.
- data – Data needs to be written.

void SPDIF\_ReadBlocking(SPDIF\_Type \*base, uint8\_t \*buffer, uint32\_t size)

Receives data using a blocking method.

---

**Note:** This function blocks by polling until data is ready to be sent.

---

**Parameters**

- base – SPDIF base pointer.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

static inline uint32\_t SPDIF\_ReadLeftData(SPDIF\_Type \*base)

Reads data from the SPDIF FIFO.

**Parameters**

- base – SPDIF base pointer.

**Returns**

Data in SPDIF FIFO.

static inline uint32\_t SPDIF\_ReadRightData(SPDIF\_Type \*base)

Reads data from the SPDIF FIFO.

**Parameters**

- base – SPDIF base pointer.

**Returns**

Data in SPDIF FIFO.

static inline uint32\_t SPDIF\_ReadChannelStatusHigh(SPDIF\_Type \*base)

Reads data from the SPDIF FIFO.

**Parameters**

- base – SPDIF base pointer.

**Returns**

Data in SPDIF FIFO.

static inline uint32\_t SPDIF\_ReadChannelStatusLow(SPDIF\_Type \*base)

Reads data from the SPDIF FIFO.

**Parameters**

- base – SPDIF base pointer.

**Returns**

Data in SPDIF FIFO.

static inline uint32\_t SPDIF\_ReadQChannel(SPDIF\_Type \*base)

Reads data from the SPDIF FIFO.

**Parameters**

- base – SPDIF base pointer.

**Returns**

Data in SPDIF FIFO.

```
static inline uint32_t SPDIF_ReadUChannel(SPDIF_Type *base)
```

Reads data from the SPDIF FIFO.

**Parameters**

- base – SPDIF base pointer.

**Returns**

Data in SPDIF FIFO.

```
void SPDIF_TransferTxCreateHandle(SPDIF_Type *base, spdif_handle_t *handle,  
                                spdif_transfer_callback_t callback, void *userData)
```

Initializes the SPDIF Tx handle.

This function initializes the Tx handle for the SPDIF Tx transactional APIs. Call this function once to get the handle initialized.

**Parameters**

- base – SPDIF base pointer
- handle – SPDIF handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function

```
void SPDIF_TransferRxCreateHandle(SPDIF_Type *base, spdif_handle_t *handle,  
                                spdif_transfer_callback_t callback, void *userData)
```

Initializes the SPDIF Rx handle.

This function initializes the Rx handle for the SPDIF Rx transactional APIs. Call this function once to get the handle initialized.

**Parameters**

- base – SPDIF base pointer.
- handle – SPDIF handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function.

```
status_t SPDIF_TransferSendNonBlocking(SPDIF_Type *base, spdif_handle_t *handle,  
                                       spdif_transfer_t *xfer)
```

Performs an interrupt non-blocking send transfer on SPDIF.

---

**Note:** This API returns immediately after the transfer initiates. Call the SPDIF\_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus\_SPDIF\_Busy, the transfer is finished.

---

**Parameters**

- base – SPDIF base pointer.
- handle – Pointer to the spdif\_handle\_t structure which stores the transfer state.
- xfer – Pointer to the spdif\_transfer\_t structure.

**Return values**

- kStatus\_Success – Successfully started the data receive.

- `kStatus_SPDIF_TxBusy` – Previous receive still not finished.
- `kStatus_InvalidArgument` – The input parameter is invalid.

*status\_t* SPDIF\_TransferReceiveNonBlocking(SPDIF\_Type \*base, *spdif\_handle\_t* \*handle, *spdif\_transfer\_t* \*xfer)

Performs an interrupt non-blocking receive transfer on SPDIF.

---

**Note:** This API returns immediately after the transfer initiates. Call the `SPDIF_RxGetTransferStatusIRQ` to poll the transfer status and check whether the transfer is finished. If the return status is not `kStatus_SPDIF_Busy`, the transfer is finished.

---

#### Parameters

- `base` – SPDIF base pointer
- `handle` – Pointer to the `spdif_handle_t` structure which stores the transfer state.
- `xfer` – Pointer to the `spdif_transfer_t` structure.

#### Return values

- `kStatus_Success` – Successfully started the data receive.
- `kStatus_SPDIF_RxBusy` – Previous receive still not finished.
- `kStatus_InvalidArgument` – The input parameter is invalid.

*status\_t* SPDIF\_TransferGetSendCount(SPDIF\_Type \*base, *spdif\_handle\_t* \*handle, *size\_t* \*count)

Gets a set byte count.

#### Parameters

- `base` – SPDIF base pointer.
- `handle` – Pointer to the `spdif_handle_t` structure which stores the transfer state.
- `count` – Bytes count sent.

#### Return values

- `kStatus_Success` – Succeed get the transfer count.
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

*status\_t* SPDIF\_TransferGetReceiveCount(SPDIF\_Type \*base, *spdif\_handle\_t* \*handle, *size\_t* \*count)

Gets a received byte count.

#### Parameters

- `base` – SPDIF base pointer.
- `handle` – Pointer to the `spdif_handle_t` structure which stores the transfer state.
- `count` – Bytes count received.

#### Return values

- `kStatus_Success` – Succeed get the transfer count.
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

void SPDIF\_\_TransferAbortSend(SPDIF\_Type \*base, *spdif\_handle\_t* \*handle)

Aborts the current send.

---

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

---

**Parameters**

- base – SPDIF base pointer.
- handle – Pointer to the *spdif\_handle\_t* structure which stores the transfer state.

void SPDIF\_\_TransferAbortReceive(SPDIF\_Type \*base, *spdif\_handle\_t* \*handle)

Aborts the current IRQ receive.

---

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

---

**Parameters**

- base – SPDIF base pointer
- handle – Pointer to the *spdif\_handle\_t* structure which stores the transfer state.

void SPDIF\_\_TransferTxHandleIRQ(SPDIF\_Type \*base, *spdif\_handle\_t* \*handle)

Tx interrupt handler.

**Parameters**

- base – SPDIF base pointer.
- handle – Pointer to the *spdif\_handle\_t* structure.

void SPDIF\_\_TransferRxHandleIRQ(SPDIF\_Type \*base, *spdif\_handle\_t* \*handle)

Tx interrupt handler.

**Parameters**

- base – SPDIF base pointer.
- handle – Pointer to the *spdif\_handle\_t* structure.

FSL\_SPDIF\_DRIVER\_VERSION

Version 2.0.7

SPDIF return status.

*Values:*

enumerator kStatus\_SPDIF\_RxDPLLLocked

SPDIF Rx PLL locked.

enumerator kStatus\_SPDIF\_TxFIFOError

SPDIF Tx FIFO error.

enumerator kStatus\_SPDIF\_TxFIFOResync

SPDIF Tx left and right FIFO resync.

enumerator kStatus\_SPDIF\_RxCnew  
SPDIF Rx status channel value updated.

enumerator kStatus\_SPDIF\_ValidatyNoGood  
SPDIF validaty flag not good.

enumerator kStatus\_SPDIF\_RxIllegalSymbol  
SPDIF Rx receive illegal symbol.

enumerator kStatus\_SPDIF\_RxParityBitError  
SPDIF Rx parity bit error.

enumerator kStatus\_SPDIF\_UChannelOverrun  
SPDIF receive U channel overrun.

enumerator kStatus\_SPDIF\_QChannelOverrun  
SPDIF receive Q channel overrun.

enumerator kStatus\_SPDIF\_UQChannelSync  
SPDIF U/Q channel sync found.

enumerator kStatus\_SPDIF\_UQChannelFrameError  
SPDIF U/Q channel frame error.

enumerator kStatus\_SPDIF\_RxFIFOError  
SPDIF Rx FIFO error.

enumerator kStatus\_SPDIF\_RxFIFOResync  
SPDIF Rx left and right FIFO resync.

enumerator kStatus\_SPDIF\_LockLoss  
SPDIF Rx PLL clock lock loss.

enumerator kStatus\_SPDIF\_TxIdle  
SPDIF Tx is idle

enumerator kStatus\_SPDIF\_RxIdle  
SPDIF Rx is idle

enumerator kStatus\_SPDIF\_QueueFull  
SPDIF queue full

enum \_spdif\_rxfull\_select

SPDIF Rx FIFO full falg select, it decides when assert the rx full flag.

*Values:*

enumerator kSPDIF\_RxFull1Sample  
Rx full at least 1 sample in left and right FIFO

enumerator kSPDIF\_RxFull4Samples  
Rx full at least 4 sample in left and right FIFO

enumerator kSPDIF\_RxFull8Samples  
Rx full at least 8 sample in left and right FIFO

enumerator kSPDIF\_RxFull16Samples  
Rx full at least 16 sample in left and right FIFO

enum \_spdif\_txempty\_select

SPDIF tx FIFO EMPTY falg select, it decides when assert the tx empty flag.

*Values:*

enumerator kSPDIF\_\_TxEmpty0Sample

Tx empty at most 0 sample in left and right FIFO

enumerator kSPDIF\_\_TxEmpty4Samples

Tx empty at most 4 sample in left and right FIFO

enumerator kSPDIF\_\_TxEmpty8Samples

Tx empty at most 8 sample in left and right FIFO

enumerator kSPDIF\_\_TxEmpty12Samples

Tx empty at most 12 sample in left and right FIFO

enum \_\_spdif\_\_uchannel\_\_source

SPDIF U channel source.

*Values:*

enumerator kSPDIF\_\_NoUChannel

No embedded U channel

enumerator kSPDIF\_\_UChannelFromRx

U channel from receiver, it is CD mode

enumerator kSPDIF\_\_UChannelFromTx

U channel from on chip tx

enum \_\_spdif\_\_gain\_\_select

SPDIF clock gain.

*Values:*

enumerator kSPDIF\_\_GAIN\_\_24

Gain select is 24

enumerator kSPDIF\_\_GAIN\_\_16

Gain select is 16

enumerator kSPDIF\_\_GAIN\_\_12

Gain select is 12

enumerator kSPDIF\_\_GAIN\_\_8

Gain select is 8

enumerator kSPDIF\_\_GAIN\_\_6

Gain select is 6

enumerator kSPDIF\_\_GAIN\_\_4

Gain select is 4

enumerator kSPDIF\_\_GAIN\_\_3

Gain select is 3

enum \_\_spdif\_\_tx\_\_source

SPDIF tx data source.

*Values:*

enumerator kSPDIF\_\_txFromReceiver

Tx data directly through SPDIF receiver

enumerator kSPDIF\_\_txNormal

Normal operation, data from processor

enum \_spdif\_validity\_config

SPDIF tx data source.

*Values:*

enumerator kSPDIF\_validityFlagAlwaysSet

Outgoing validity flags always set

enumerator kSPDIF\_validityFlagAlwaysClear

Outgoing validity flags always clear

The SPDIF interrupt enable flag.

*Values:*

enumerator kSPDIF\_RxDPLLLocked

SPDIF DPLL locked

enumerator kSPDIF\_TxFIFOError

Tx FIFO underrun or overrun

enumerator kSPDIF\_TxFIFOResync

Tx FIFO left and right channel resync

enumerator kSPDIF\_RxControlChannelChange

SPDIF Rx control channel value changed

enumerator kSPDIF\_VValidityFlagNoGood

SPDIF validity flag no good

enumerator kSPDIF\_RxIllegalSymbol

SPDIF receiver found illegal symbol

enumerator kSPDIF\_RxParityBitError

SPDIF receiver found parity bit error

enumerator kSPDIF\_UChannelReceiveRegisterFull

SPDIF U channel receive register full

enumerator kSPDIF\_UChannelReceiveRegisterOverrun

SPDIF U channel receive register overrun

enumerator kSPDIF\_QChannelReceiveRegisterFull

SPDIF Q channel receive register full

enumerator kSPDIF\_QChannelReceiveRegisterOverrun

SPDIF Q channel receive register overrun

enumerator kSPDIF\_UQChannelSync

SPDIF U/Q channel sync found

enumerator kSPDIF\_UQChannelFrameError

SPDIF U/Q channel frame error

enumerator kSPDIF\_RxFIFOError

SPDIF Rx FIFO underrun/overrun

enumerator kSPDIF\_RxFIFOResync

SPDIF Rx left and right FIFO resync

enumerator kSPDIF\_LockLoss

SPDIF receiver loss of lock



```

enumerator kSPDIF__TxFIFOEmpty
    SPDIF Tx FIFO empty
enumerator kSPDIF__RxFIFOFull
    SPDIF Rx FIFO full
enumerator kSPDIF__AllInterrupt
    all interrupt

```

The DMA request sources.

*Values:*

```

enumerator kSPDIF__RxDMAEnable
    Rx FIFO full
enumerator kSPDIF__TxDMAEnable
    Tx FIFO empty
typedef enum _spdif_rxfull_select spdif_rxfull_select_t
    SPDIF Rx FIFO full falg select, it decides when assert the rx full flag.
typedef enum _spdif_txempty_select spdif_txempty_select_t
    SPDIF tx FIFO EMPTY falg select, it decides when assert the tx empty flag.
typedef enum _spdif_uchannel_source spdif_uchannel_source_t
    SPDIF U channel source.
typedef enum _spdif_gain_select spdif_gain_select_t
    SPDIF clock gain.
typedef enum _spdif_tx_source spdif_tx_source_t
    SPDIF tx data source.
typedef enum _spdif_validity_config spdif_validity_config_t
    SPDIF tx data source.
typedef struct _spdif_config spdif_config_t
    SPDIF user configuration structure.
typedef struct _spdif_transfer spdif_transfer_t
    SPDIF transfer structure.
typedef struct _spdif_handle spdif_handle_t
typedef void (*spdif_transfer_callback_t)(SPDIF_Type *base, spdif_handle_t *handle, status_t
status, void *userData)
    SPDIF transfer callback prototype.
SPDIF_XFER_QUEUE_SIZE
    SPDIF transfer queue size, user can refine it according to use case.
struct _spdif_config
    #include <fsl_spdif.h> SPDIF user configuration structure.

```

### Public Members

```

bool isTxAutoSync
    If auto sync mechanism open

```

bool isRxAutoSync

If auto sync mechanism open

uint8\_t DPLLClkSource

SPDIF DPLL clock source, range from 0~15, meaning is chip-specific

uint8\_t txClkSource

SPDIF tx clock source, range from 0~7, meaning is chip-specific

*spdif\_rxfull\_select\_t* rxFullSelect

SPDIF rx buffer full select

*spdif\_txempty\_select\_t* txFullSelect

SPDIF tx buffer empty select

*spdif\_uchannel\_source\_t* uChannelSrc

U channel source

*spdif\_tx\_source\_t* txSource

SPDIF tx data source

*spdif\_validity\_config\_t* validityConfig

Validity flag config

*spdif\_gain\_select\_t* gain

Rx receive clock measure gain parameter.

struct *\_spdif\_transfer*

*#include <fsl\_spdif.h>* SPDIF transfer structure.

### Public Members

uint8\_t \*data

Data start address to transfer.

uint8\_t \*qdata

Data buffer for Q channel

uint8\_t \*udata

Data buffer for C channel

size\_t dataSize

Transfer size.

struct *\_spdif\_handle*

*#include <fsl\_spdif.h>* SPDIF handle structure.

### Public Members

uint32\_t state

Transfer status

*spdif\_transfer\_callback\_t* callback

Callback function called at transfer event

void \*userData

Callback parameter passed to callback function

*spdif\_transfer\_t* spdifQueue[(4U)]

Transfer queue storing queued transfer

`size_t transferSize[(4U)]`  
 Data bytes need to transfer

`volatile uint8_t queueUser`  
 Index for user to queue transfer

`volatile uint8_t queueDriver`  
 Index for driver to get the transfer data and size

`uint8_t watermark`  
 Watermark value

## 2.35 SRC: System Reset Controller Driver

`FSL_SRC_DRIVER_VERSION`  
 SRC driver version 2.0.1.

`enum __src_reset_status_flags`  
 SRC reset status flags.

*Values:*

`enumerator kSRC_WarmBootIndicationFlag`  
 WARM boot indication shows that WARM boot was initiated by software.

`enumerator kSRC_TemperatureSensorResetFlag`  
 Indicates whether the reset was the result of software reset from on-chip Temperature Sensor. Temperature Sensor Interrupt needs to be served before this bit can be cleaned.

`enumerator kSRC_JTAGSoftwareResetFlag`  
 Indicates whether the reset was the result of setting SJC\_GPCCR bit 31.

`enumerator kSRC_JTAGGeneratedResetFlag`  
 Indicates a reset has been caused by JTAG selection of certain IR codes: EXTEST or HIGHZ.

`enumerator kSRC_WatchdogResetFlag`  
 Indicates a reset has been caused by the watchdog timer timing out. This reset source can be blocked by disabling the watchdog.

`enum __src_warm_reset_bypass_count`  
 Selection of WARM reset bypass count.

This type defines the 32KHz clock cycles to count before bypassing the MMDC acknowledge for WARM reset. If the MMDC acknowledge is not asserted before this counter is elapsed, a COLD reset will be initiated.

*Values:*

`enumerator kSRC_WarmResetWaitAlways`  
 System will wait until MMDC acknowledge is asserted.

`enumerator kSRC_WarmResetWaitClk16`  
 Wait 16 32KHz clock cycles before switching the reset.

`enumerator kSRC_WarmResetWaitClk32`  
 Wait 32 32KHz clock cycles before switching the reset.

`enumerator kSRC_WarmResetWaitClk64`  
 Wait 64 32KHz clock cycles before switching the reset.

```
typedef enum _src_warm_reset_bypass_count src_warm_reset_bypass_count_t
```

Selection of WARM reset bypass count.

This type defines the 32KHz clock cycles to count before bypassing the MMDC acknowledge for WARM reset. If the MMDC acknowledge is not asserted before this counter is elapsed, a COLD reset will be initiated.

```
static inline void SRC_EnableWDOGReset(SRC_Type *base, bool enable)
```

Enable the WDOG Reset in SRC.

WDOG Reset is enabled in SRC by default. If the WDOG event to SRC is masked, it would not create a reset to the chip. During the time the WDOG event is masked, when the WDOG event flag is asserted, it would remain asserted regardless of servicing the WDOG module. The only way to clear that bit is the hardware reset.

#### Parameters

- base – SRC peripheral base address.
- enable – Enable the reset or not.

```
static inline void SRC_SetWarmResetBypassCount(SRC_Type *base,  
                                              src_warm_reset_bypass_count_t option)
```

Set the delay count of waiting MMDC's acknowledge.

This function would define the 32KHz clock cycles to count before bypassing the MMDC acknowledge for WARM reset. If the MMDC acknowledge is not asserted before this counter is elapsed, a COLD reset will be initiated.

#### Parameters

- base – SRC peripheral base address.
- option – The option of setting mode, see to *src\_warm\_reset\_bypass\_count\_t*.

```
static inline void SRC_EnableWarmReset(SRC_Type *base, bool enable)
```

Enable the WARM reset.

Only when the WARM reset is enabled, the WARM reset requests would be served by WARM reset. Otherwise, all the WARM reset sources would generate COLD reset.

#### Parameters

- base – SRC peripheral base address.
- enable – Enable the WARM reset or not.

```
static inline uint32_t SRC_GetBootModeWord1(SRC_Type *base)
```

Get the boot mode register 1 value.

The Boot Mode register contains bits that reflect the status of BOOT\_CFGx pins of the chip. See to chip-specific document for detail information about value.

#### Parameters

- base – SRC peripheral base address.

#### Returns

status of BOOT\_CFGx pins of the chip.

```
static inline uint32_t SRC_GetBootModeWord2(SRC_Type *base)
```

Get the boot mode register 2 value.

The Boot Mode register contains bits that reflect the status of BOOT\_MODEx Pins and fuse values that controls boot of the chip. See to chip-specific document for detail information about value.

#### Parameters

- base – SRC peripheral base address.

**Returns**

status of BOOT\_MODEx Pins and fuse values that controls boot of the chip.

static inline void SRC\_SetWarmBootIndication(SRC\_Type \*base, bool enable)

Set the warm boot indication flag.

WARM boot indication shows that WARM boot was initiated by software. This indicates to the software that it saved the needed information in the memory before initiating the WARM reset. In this case, software will set this bit to '1', before initiating the WARM reset. The warm\_boot bit should be used as indication only after a warm\_reset sequence. Software should clear this bit after warm\_reset to indicate that the next warm\_reset is not performed with warm\_boot.

**Parameters**

- base – SRC peripheral base address.
- enable – Assert the flag or not.

static inline uint32\_t SRC\_GetResetStatusFlags(SRC\_Type \*base)

Get the status flags of SRC.

**Parameters**

- base – SRC peripheral base address.

**Returns**

Mask value of status flags, see to \_src\_reset\_status\_flags.

void SRC\_ClearResetStatusFlags(SRC\_Type \*base, uint32\_t flags)

Clear the status flags of SRC.

**Parameters**

- base – SRC peripheral base address.
- flags – value of status flags to be cleared, see to \_src\_reset\_status\_flags.

static inline void SRC\_SetGeneralPurposeRegister(SRC\_Type \*base, uint32\_t index, uint32\_t value)

Set value to general purpose registers.

General purpose registers (GPRx) would hold the value during reset process. Wakeup function could be kept in these register. For example, the GPR1 holds the entry function for waking-up from Partial SLEEP mode while the GPR2 holds the argument. Other GPRx register would store the arbitray values.

**Parameters**

- base – SRC peripheral base address.
- index – The index of GPRx register array. Note index 0 reponses the GPR1 register.
- value – Setting value for GPRx register.

static inline uint32\_t SRC\_GetGeneralPurposeRegister(SRC\_Type \*base, uint32\_t index)

Get the value from general purpose registers.

**Parameters**

- base – SRC peripheral base address.
- index – The index of GPRx register array. Note index 0 reponses the GPR1 register.

**Returns**

The setting value for GPRx register.

## 2.36 TMU: Thermal Management Unit Driver

enum \_tmu\_monitor\_site

*Values:*

enumerator kTMU\_MonitorSite0

enumerator kTMU\_MonitorSite1

enumerator kTMU\_MonitorSite2

enumerator kTMU\_MonitorSite3

enumerator kTMU\_MonitorSite4

enumerator kTMU\_MonitorSite5

enumerator kTMU\_MonitorSite6

enumerator kTMU\_MonitorSite7

enumerator kTMU\_MonitorSite8

enumerator kTMU\_MonitorSite9

enumerator kTMU\_MonitorSite10

enumerator kTMU\_MonitorSite11

enumerator kTMU\_MonitorSite12

enumerator kTMU\_MonitorSite13

enumerator kTMU\_MonitorSite14

enumerator kTMU\_MonitorSite15

enum \_tmu\_interrupt\_enable

TMU interrupt enable.

*Values:*

enumerator kTMU\_ImmediateTemperatureInterruptEnable

Immediate temperature threshold exceeded interrupt enable.

enumerator kTMU\_AverageTemperatureInterruptEnable

Average temperature threshold exceeded interrupt enable.

enumerator kTMU\_AverageTemperatureCriticalInterruptEnable

Average temperature critical threshold exceeded interrupt enable. >

enum \_tmu\_interrupt\_status\_flags

TMU interrupt status flags.

*Values:*

enumerator kTMU\_ImmediateTemperatureStatusFlags

Immediate temperature threshold exceeded(ITTE).

enumerator kTMU\_AverageTemperatureStatusFlags

Average temperature threshold exceeded(ATTE).

enumerator kTMU\_AverageTemperatureCriticalStatusFlags

Average temperature critical threshold exceeded.(ATCTE)

enum \_tmu\_status\_flags

TMU status flags.

*Values:*

enumerator kTMU\_IntervalExceededStatusFlags

Monitoring interval exceeded. The time required to perform measurement of all monitored sites has exceeded the monitoring interval as defined by TMTMIR.

enumerator kTMU\_OutOfLowRangeStatusFlags

Out-of-range low temperature measurement detected. A temperature sensor detected a temperature reading below the lowest measurable temperature of 0 °C.

enumerator kTMU\_OutOfHighRangeStatusFlags

Out-of-range high temperature measurement detected. A temperature sensor detected a temperature reading above the highest measurable temperature of 125 °C.

enum \_tmu\_average\_low\_pass\_filter

Average low pass filter setting.

*Values:*

enumerator kTMU\_AverageLowPassFilter1\_0

Average low pass filter = 1.

enumerator kTMU\_AverageLowPassFilter0\_5

Average low pass filter = 0.5.

enumerator kTMU\_AverageLowPassFilter0\_25

Average low pass filter = 0.25.

enumerator kTMU\_AverageLowPassFilter0\_125

Average low pass filter = 0.125.

typedef struct *\_tmu\_threshold\_config* tmu\_threshold\_config\_t

configuration for TMU threshold.

typedef struct *\_tmu\_interrupt\_status* tmu\_interrupt\_status\_t

TMU interrupt status.

typedef enum *\_tmu\_average\_low\_pass\_filter* tmu\_average\_low\_pass\_filter\_t

Average low pass filter setting.

typedef struct *\_tmu\_config* tmu\_config\_t

Configuration for TMU module.

void TMU\_Init(TMU\_Type \*base, const *tmu\_config\_t* \*config)

Enable the access to TMU registers and Initialize TMU module.

#### Parameters

- base – TMU peripheral base address.
- config – Pointer to configuration structure. Refer to “tmu\_config\_t” structure.

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

De-initialize TMU module and Disable the access to DCDC registers.

**Parameters**

- base – TMU peripheral base address.

void TMU\_GetDefaultConfig(*tmu\_config\_t* \*config)

Gets the default configuration for TMU.

This function initializes the user configuration structure to default value. The default value are:

Example:

```
config->monitorInterval = 0U;  
config->monitorSiteSelection = 0U;  
config->averageLPF = kTMU_AverageLowPassFilter1_0;
```

**Parameters**

- config – Pointer to TMU configuration structure.

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

Enable/Disable the TMU module.

**Parameters**

- base – TMU peripheral base address.
- enable – Switcher to enable/disable TMU.

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

Enable the TMU interrupts.

**Parameters**

- base – TMU peripheral base address.
- mask – The interrupt mask. Refer to “\_tmu\_interrupt\_enable” enumeration.

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

Disable the TMU interrupts.

**Parameters**

- base – TMU peripheral base address.
- mask – The interrupt mask. Refer to “\_tmu\_interrupt\_enable” enumeration.

void TMU\_GetInterruptStatusFlags(TMU\_Type \*base, *tmu\_interrupt\_status\_t* \*status)

Get interrupt status flags.

**Parameters**

- base – TMU peripheral base address.
- status – The pointer to interrupt status structure. Record the current interrupt status. Please refer to “tmu\_interrupt\_status\_t” structure.

void TMU\_ClearInterruptStatusFlags(TMU\_Type \*base, uint32\_t mask)

Clear interrupt status flags and corresponding interrupt critical site capture register.

**Parameters**

- base – TMU peripheral base address.



- `mask` – The mask of interrupt status flags. Refer to “`_tmu_interrupt_status_flags`” enumeration.

`static inline uint32_t TMU_GetStatusFlags(TMU_Type *base)`

Get TMU status flags.

#### Parameters

- `base` – TMU peripheral base address.

#### Returns

The mask of status flags. Refer to “`_tmu_status_flags`” enumeration.

`status_t TMU_GetHighestTemperature(TMU_Type *base, uint32_t *temperature)`

Get the highest temperature reached for any enabled monitored site within the temperature sensor range.

#### Parameters

- `base` – TMU peripheral base address.
- `temperature` – Highest temperature recorded in degrees Celsius by any enabled monitored site.

#### Return values

- `kStatus_Success` – Temperature reading is valid.
- `kStatus_Fail` – Temperature reading is not valid due to no measured temperature within the sensor range of 0-125 °C for an enabled monitored site.

#### Returns

Execution status.

`status_t TMU_GetLowestTemperature(TMU_Type *base, uint32_t *temperature)`

Get the lowest temperature reached for any enabled monitored site within the temperature sensor range.

#### Parameters

- `base` – TMU peripheral base address.
- `temperature` – Lowest temperature recorded in degrees Celsius by any enabled monitored site.

#### Return values

- `kStatus_Success` – Temperature reading is valid.
- `kStatus_Fail` – Temperature reading is not valid due to no measured temperature within the sensor range of 0-125 °C for an enabled monitored site.

#### Returns

Execution status.

`status_t TMU_GetImmediateTemperature(TMU_Type *base, uint32_t siteIndex, uint32_t *temperature)`

Get the last immediate temperature at site `n`. The site must be part of the list of enabled monitored sites as defined by `monitorSiteSelection` in “`tmu_config_t`” structure.

#### Parameters

- `base` – TMU peripheral base address.
- `siteIndex` – The index of the site user want to read. 0U: site0 ~ 15U: site15.
- `temperature` – Last immediate temperature reading at site `n`.

**Return values**

- `kStatus_Success` – Temperature reading is valid.
- `kStatus_Fail` – Temperature reading is not valid because temperature out of sensor range or first measurement still pending.

**Returns**

Execution status.

`status_t` `TMU_GetAverageTemperature(TMU_Type *base, uint32_t siteIndex, uint32_t *temperature)`

Get the last average temperature at site n. The site must be part of the list of enabled monitored sites as defined by `monitorSiteSelection` in “`tmu_config_t`” structure.

**Parameters**

- `base` – TMU peripheral base address.
- `siteIndex` – The index of the site user want to read. 0U: site0 ~ 15U: site15.
- `temperature` – Last average temperature reading at site n .

**Return values**

- `kStatus_Success` – Temperature reading is valid.
- `kStatus_Fail` – Temperature reading is not valid because temperature out of sensor range or first measurement still pending.

**Returns**

Execution status.

`void` `TMU_SetHighTemperatureThresold(TMU_Type *base, const tmu_thresold_config_t *config)`

Configure the high temperature thresold value and enable/disable relevant threshold.

**Parameters**

- `base` – TMU peripheral base address.
- `config` – Pointer to configuration structure. Refer to “`tmu_thresold_config_t`” structure.

`FSL_TMU_DRIVER_VERSION`

TMU driver version.

Version 2.0.3.

`struct` `_tmu_thresold_config`

`#include <fsl_tmu.h>` configuration for TMU thresold.

**Public Members**

`bool` `immediateThresoldEnable`

Enable high temperature immediate threshold.

`bool` `AverageThresoldEnable`

Enable high temperature average threshold.

`bool` `AverageCriticalThresoldEnable`

Enable high temperature average critical threshold.

`uint8_t` `immediateThresoldValue`

Range:0U-125U. Valid when corresponding thresold is enabled. High temperature immediate threshold value. Determines the current upper temperature threshold, for anyenabled monitored site.

uint8\_t averageThresoldValue

Range:0U-125U. Valid when corresponding threshold is enabled. High temperature average threshold value. Determines the average upper temperature threshold, for any enabled monitored site.

uint8\_t averageCriticalThresoldValue

Range:0U-125U. Valid when corresponding thresold is enabled. High temperature average critical threshold value. Determines the average upper critical temperature threshold, for any enabled monitored site.

struct \_\_tmu\_interrupt\_status

*#include <fsl\_tmu.h>* TMU interrupt status.

### Public Members

uint32\_t interruptDetectMask

The mask of interrupt status flags. Refer to “\_tmu\_interrupt\_status\_flags” enumeration.

uint16\_t immediateInterruptsSiteMask

The mask of the temperature sensor site associated with a detected ITTE event. Please refer to “\_tmu\_monitor\_site” enumeration.

uint16\_t AverageInterruptsSiteMask

The mask of the temperature sensor site associated with a detected ATTE event. Please refer to “\_tmu\_monitor\_site” enumeration.

uint16\_t AverageCriticalInterruptsSiteMask

The mask of the temperature sensor site associated with a detected ATCTE event. Please refer to “\_tmu\_monitor\_site” enumeration.

struct \_\_tmu\_config

*#include <fsl\_tmu.h>* Configuration for TMU module.

### Public Members

uint8\_t monitorInterval

Temperature monitoring interval in seconds. Please refer to specific table in RM.

uint16\_t monitorSiteSelection

By setting the select bit for a temperature sensor site, it is enabled and included in all monitoring functions. If no site is selected, site 0 is monitored by default. Refer to “\_tmu\_monitor\_site” enumeration. Please look up relevant table in reference manual.

tmu\_average\_low\_pass\_filter\_t averageLPF

The average temperature is calculated as:  $ALPF \times \text{Current\_Temp} + (1 - ALPF) \times \text{Average\_Temp}$ . For proper operation, this field should only change when monitoring is disabled.

## 2.37 UART: Universal Asynchronous Receiver/Transmitter Driver

### 2.38 UART Driver

static inline void UART\_SoftwareReset(UART\_Type \*base)

Resets the UART using software.

This function resets the transmit and receive state machines, all FIFOs and register USR1, USR2, UBIR, UBMR, UBRC, URXD, UTXD and UTS[6-3]

#### Parameters

- base – UART peripheral base address.

status\_t UART\_Init(UART\_Type \*base, const uart\_config\_t \*config, uint32\_t srcClock\_Hz)

Initializes an UART instance with the user configuration structure and the peripheral clock.

This function configures the UART module with user-defined settings. Call the UART\_GetDefaultConfig() function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the UART.

```
uart_config_t uartConfig;
uartConfig.baudRate_Bps = 115200U;
uartConfig.parityMode = kUART_ParityDisabled;
uartConfig.dataBitsCount = kUART_EightDataBits;
uartConfig.stopBitCount = kUART_OneStopBit;
uartConfig.txFifoWatermark = 2;
uartConfig.rxFifoWatermark = 1;
uartConfig.enableAutoBaudrate = false;
uartConfig.enableTx = true;
uartConfig.enableRx = true;
UART_Init(UART1, &uartConfig, 24000000U);
```

#### Parameters

- base – UART peripheral base address.
- config – Pointer to a user-defined configuration structure.
- srcClock\_Hz – UART clock source frequency in HZ.

#### Return values

kStatus\_Success – UART initialize succeed

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

Deinitializes a UART instance.

This function waits for transmit to complete, disables TX and RX, and disables the UART clock.

#### Parameters

- base – UART peripheral base address.

void UART\_GetDefaultConfig(uart\_config\_t \*config)

Gets the default configuration structure.

1

This function initializes the UART configuration structure to a default value. The default values are: uartConfig->baudRate\_Bps = 115200U; uartConfig->parityMode = kUART\_ParityDisabled; uartConfig->dataBitsCount = kUART\_EightDataBits; uartConfig->stopBitCount = kUART\_OneStopBit; uartConfig->txFifoWatermark = 2; uartConfig->rxFifoWatermark = 1; uartConfig->enableAutoBaudrate = false; uartConfig->enableTx = false; uartConfig->enableRx = false;

#### Parameters

- config – Pointer to a configuration structure.

```
status_t UART_SetBaudRate(UART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
```

Sets the UART instance baud rate.

This function configures the UART module baud rate. This function is used to update the UART module baud rate after the UART module is initialized by the UART\_Init.

```
UART_SetBaudRate(UART1, 115200U, 20000000U);
```

#### Parameters

- base – UART peripheral base address.
- baudRate\_Bps – UART baudrate to be set.
- srcClock\_Hz – UART clock source frequency in Hz.

#### Return values

- kStatus\_UART\_BaudrateNotSupport – Baudrate is not support in the current clock source.
- kStatus\_Success – Set baudrate succeeded.

```
static inline void UART_Enable(UART_Type *base)
```

This function is used to Enable the UART Module.

#### Parameters

- base – UART base pointer.

```
static inline void UART_SetIdleCondition(UART_Type *base, uart_idle_condition_t condition)
```

This function is used to configure the IDLE line condition.

#### Parameters

- base – UART base pointer.
- condition – IDLE line detect condition of the enumerators in `uart_idle_condition_t`.

```
static inline void UART_Disable(UART_Type *base)
```

This function is used to Disable the UART Module.

#### Parameters

- base – UART base pointer.

```
bool UART_GetStatusFlag(UART_Type *base, uint32_t flag)
```

This function is used to get the current status of specific UART status flag(including interrupt flag). The available status flag can be select from `uart_status_flag_t` enumeration.

#### Parameters

- base – UART base pointer.
- flag – Status flag to check.

#### Return values

current – state of corresponding status flag.

```
void UART_ClearStatusFlag(UART_Type *base, uint32_t flag)
```

This function is used to clear the current status of specific UART status flag. The available status flag can be select from `uart_status_flag_t` enumeration.

#### Parameters

- base – UART base pointer.
- flag – Status flag to clear.

`void UART_EnableInterrupts(UART_Type *base, uint32_t mask)`

Enables UART interrupts according to the provided mask.

This function enables the UART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See `_uart_interrupt_enable`. For example, to enable TX empty interrupt and RX data ready interrupt, do the following.

```
UART_EnableInterrupts(UART1, kUART_TxEmptyEnable | kUART_RxDataReadyEnable);
```

#### Parameters

- `base` – UART peripheral base address.
- `mask` – The interrupts to enable. Logical OR of `_uart_interrupt_enable`.

`void UART_DisableInterrupts(UART_Type *base, uint32_t mask)`

Disables the UART interrupts according to the provided mask.

This function disables the UART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See `_uart_interrupt_enable`. For example, to disable TX empty interrupt and RX data ready interrupt do the following.

```
UART_EnableInterrupts(UART1, kUART_TxEmptyEnable | kUART_RxDataReadyEnable);
```

#### Parameters

- `base` – UART peripheral base address.
- `mask` – The interrupts to disable. Logical OR of `_uart_interrupt_enable`.

`uint32_t UART_GetEnabledInterrupts(UART_Type *base)`

Gets enabled UART interrupts.

This function gets the enabled UART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators `_uart_interrupt_enable`. To check a specific interrupt enable status, compare the return value with enumerators in `_uart_interrupt_enable`. For example, to check whether the TX empty interrupt is enabled:

```
uint32_t enabledInterrupts = UART_GetEnabledInterrupts(UART1);  
  
if (kUART_TxEmptyEnable & enabledInterrupts)  
{  
    ...  
}
```

#### Parameters

- `base` – UART peripheral base address.

#### Returns

UART interrupt flags which are logical OR of the enumerators in `_uart_interrupt_enable`.

`static inline void UART_EnableTx(UART_Type *base, bool enable)`

Enables or disables the UART transmitter.

This function enables or disables the UART transmitter.

#### Parameters

- `base` – UART peripheral base address.
- `enable` – True to enable, false to disable.

```
static inline void UART_EnableRx(UART_Type *base, bool enable)
```

Enables or disables the UART receiver.

This function enables or disables the UART receiver.

#### Parameters

- base – UART peripheral base address.
- enable – True to enable, false to disable.

```
static inline void UART_WriteByte(UART_Type *base, uint8_t data)
```

Writes to the transmitter register.

This function is used to write data to transmitter register. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

#### Parameters

- base – UART peripheral base address.
- data – Data write to the TX register.

```
static inline uint8_t UART_ReadByte(UART_Type *base)
```

Reads the receiver register.

This function is used to read data from receiver register. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

#### Parameters

- base – UART peripheral base address.

#### Returns

Data read from data register.

```
status_t UART_WriteBlocking(UART_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 – UART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

#### Return values

- kStatus\_UART\_Timeout – Transmission timed out and was aborted.
- kStatus\_Success – Successfully wrote all data.

```
status_t UART_ReadBlocking(UART_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 reads data from the TX register.

#### Parameters

- base – UART peripheral base address.
- data – Start address of the buffer to store the received data.
- length – Size of the buffer.

#### Return values

- `kStatus_UART_RxHardwareOverrun` – Receiver overrun occurred while receiving data.
- `kStatus_UART_NoiseError` – A noise error occurred while receiving data.
- `kStatus_UART_FramingError` – A framing error occurred while receiving data.
- `kStatus_UART_ParityError` – A parity error occurred while receiving data.
- `kStatus_UART_Timeout` – Transmission timed out and was aborted.
- `kStatus_Success` – Successfully received all data.

```
void UART_TransferCreateHandle(UART_Type *base, uart_handle_t *handle,  
                             uart_transfer_callback_t callback, void *userData)
```

Initializes the UART handle.

This function initializes the UART handle which can be used for other UART transactional APIs. Usually, for a specified UART instance, call this API once to get the initialized handle.

#### Parameters

- `base` – UART peripheral base address.
- `handle` – UART handle pointer.
- `callback` – The callback function.
- `userData` – The parameter of the callback function.

```
void UART_TransferStartRingBuffer(UART_Type *base, uart_handle_t *handle, uint8_t  
                                *ringBuffer, size_t ringBufferSize)
```

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received are stored into the ring buffer even when the user doesn't call the `UART_TransferReceiveNonBlocking()` API. If data is already 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, only 31 bytes are used for saving data.

---

#### Parameters

- `base` – UART peripheral base address.
- `handle` – UART 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 UART_TransferStopRingBuffer(UART_Type *base, uart_handle_t *handle)
```

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

#### Parameters

- `base` – UART peripheral base address.
- `handle` – UART handle pointer.



`size_t UART_TransferGetRxRingBufferLength(uart_handle_t *handle)`

Get the length of received data in RX ring buffer.

#### Parameters

- *handle* – UART handle pointer.

#### Returns

Length of received data in RX ring buffer.

`status_t UART_TransferSendNonBlocking(UART_Type *base, uart_handle_t *handle,  
                                          uart_transfer_t *xfer)`

Transmits a buffer of data using the interrupt method.

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in the ISR, the UART driver calls the callback function and passes the `kStatus_UART_TxIdle` as status parameter.

---

**Note:** The `kStatus_UART_TxIdle` is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out. Before disabling the TX, check the `kUART_TransmissionCompleteFlag` to ensure that the TX is finished.

---

#### Parameters

- *base* – UART peripheral base address.
- *handle* – UART handle pointer.
- *xfer* – UART transfer structure. See `uart_transfer_t`.

#### Return values

- `kStatus_Success` – Successfully start the data transmission.
- `kStatus_UART_TxBusy` – Previous transmission still not finished; data not all written to TX register yet.
- `kStatus_InvalidArgument` – Invalid argument.

`void UART_TransferAbortSend(UART_Type *base, uart_handle_t *handle)`

Aborts the interrupt-driven data transmit.

This function aborts the interrupt-driven data sending. The user can get the `remainBytes` to find out how many bytes are not sent out.

#### Parameters

- *base* – UART peripheral base address.
- *handle* – UART handle pointer.

`status_t UART_TransferGetSendCount(UART_Type *base, uart_handle_t *handle, uint32_t  
                                          *count)`

Gets the number of bytes written to the UART TX register.

This function gets the number of bytes written to the UART TX register by using the interrupt method.

#### Parameters

- *base* – UART peripheral base address.
- *handle* – UART handle pointer.
- *count* – Send bytes count.

**Return values**

- `kStatus_NoTransferInProgress` – No send in progress.
- `kStatus_InvalidArgument` – The parameter is invalid.
- `kStatus_Success` – Get successfully through the parameter count;

*status\_t* UART\_TransferReceiveNonBlocking(UART\_Type \*base, *uart\_handle\_t* \*handle, *uart\_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 UART driver. When the new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter `kStatus_UART_RxIdle`. For example, 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 UART 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` – UART peripheral base address.
- `handle` – UART handle pointer.
- `xfer` – UART transfer structure, see `uart_transfer_t`.
- `receivedBytes` – Bytes received from the ring buffer directly.

**Return values**

- `kStatus_Success` – Successfully queue the transfer into transmit queue.
- `kStatus_UART_RxBusy` – Previous receive request is not finished.
- `kStatus_InvalidArgument` – Invalid argument.

*void* UART\_TransferAbortReceive(UART\_Type \*base, *uart\_handle\_t* \*handle)

Aborts the interrupt-driven data receiving.

This function aborts the interrupt-driven data receiving. The user can get the `remainBytes` to know how many bytes are not received yet.

**Parameters**

- `base` – UART peripheral base address.
- `handle` – UART handle pointer.

*status\_t* UART\_TransferGetReceiveCount(UART\_Type \*base, *uart\_handle\_t* \*handle, *uint32\_t* \*count)

Gets the number of bytes that have been received.

This function gets the number of bytes that have been received.

**Parameters**

- `base` – UART peripheral base address.
- `handle` – UART 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 UART_TransferHandleIRQ(UART_Type *base, void *irqHandle)`

UART IRQ handle function.

This function handles the UART transmit and receive IRQ request.

**Parameters**

- `base` – UART peripheral base address.
- `irqHandle` – UART handle pointer.

`static inline void UART_EnableTxDMA(UART_Type *base, bool enable)`

Enables or disables the UART transmitter DMA request.

This function enables or disables the transmit request when the transmitter has one or more slots available in the TxFIFO. The fill level in the TxFIFO that generates the DMA request is controlled by the TXTL bits.

**Parameters**

- `base` – UART peripheral base address.
- `enable` – True to enable, false to disable.

`static inline void UART_EnableRxDMA(UART_Type *base, bool enable)`

Enables or disables the UART receiver DMA request.

This function enables or disables the receive request when the receiver has data in the RxFIFO. The fill level in the RxFIFO at which a DMA request is generated is controlled by the RXTL bits.

**Parameters**

- `base` – UART peripheral base address.
- `enable` – True to enable, false to disable.

`static inline void UART_SetTxFifoWatermark(UART_Type *base, uint8_t watermark)`

This function is used to set the watermark of UART Tx FIFO. A maskable interrupt is generated whenever the data level in the TxFIFO falls below the Tx FIFO watermark.

**Parameters**

- `base` – UART base pointer.
- `watermark` – The Tx FIFO watermark.

`static inline void UART_SetRxRTSWatermark(UART_Type *base, uint8_t watermark)`

This function is used to set the watermark of UART RTS deassertion.

The RTS signal deasserts whenever the data count in RxFIFO reaches the Rx RTS watermark.

**Parameters**

- `base` – UART base pointer.
- `watermark` – The Rx RTS watermark.

`static inline void UART_SetRxFifoWatermark(UART_Type *base, uint8_t watermark)`

This function is used to set the watermark of UART Rx FIFO. A maskable interrupt is generated whenever the data level in the RxFIFO reaches the Rx FIFO watermark.

**Parameters**

- base – UART base pointer.
- watermark – The Rx FIFO watermark.

static inline void UART\_\_EnableAutoBaudRate(UART\_Type \*base, bool enable)

This function is used to set the enable condition of Automatic Baud Rate Detection feature.

#### Parameters

- base – UART base pointer.
- enable – Enable/Disable Automatic Baud Rate Detection feature.
  - true: Enable Automatic Baud Rate Detection feature.
  - false: Disable Automatic Baud Rate Detection feature.

static inline bool UART\_\_IsAutoBaudRateComplete(UART\_Type \*base)

This function is used to read if the automatic baud rate detection has finished.

#### Parameters

- base – UART base pointer.

#### Returns

- true: Automatic baud rate detection has finished.
- false: Automatic baud rate detection has not finished.

FSL\_UART\_DRIVER\_VERSION

UART driver version.

Error codes for the UART driver.

*Values:*

enumerator kStatus\_UART\_TxBusy  
Transmitter is busy.

enumerator kStatus\_UART\_RxBusy  
Receiver is busy.

enumerator kStatus\_UART\_TxIdle  
UART transmitter is idle.

enumerator kStatus\_UART\_RxIdle  
UART receiver is idle.

enumerator kStatus\_UART\_TxWatermarkTooLarge  
TX FIFO watermark too large

enumerator kStatus\_UART\_RxWatermarkTooLarge  
RX FIFO watermark too large

enumerator kStatus\_UART\_FlagCannotClearManually  
UART flag can't be manually cleared.

enumerator kStatus\_UART\_Error  
Error happens on UART.

enumerator kStatus\_UART\_RxRingBufferOverrun  
UART RX software ring buffer overrun.

enumerator kStatus\_UART\_RxHardwareOverrun  
UART RX receiver overrun.

enumerator kStatus\_UART\_NoiseError

UART noise error.

enumerator kStatus\_UART\_FramingError

UART framing error.

enumerator kStatus\_UART\_ParityError

UART parity error.

enumerator kStatus\_UART\_BaudrateNotSupport

Baudrate is not support in current clock source

enumerator kStatus\_UART\_BreakDetect

Receiver detect BREAK signal

enumerator kStatus\_UART\_Timeout

UART times out.

enum \_uart\_data\_bits

UART data bits count.

*Values:*

enumerator kUART\_SevenDataBits

Seven data bit

enumerator kUART\_EightDataBits

Eight data bit

enum \_uart\_parity\_mode

UART parity mode.

*Values:*

enumerator kUART\_ParityDisabled

Parity disabled

enumerator kUART\_ParityEven

Even error check is selected

enumerator kUART\_ParityOdd

Odd error check is selected

enum \_uart\_stop\_bit\_count

UART stop bit count.

*Values:*

enumerator kUART\_OneStopBit

One stop bit

enumerator kUART\_TwoStopBit

Two stop bits

enum \_uart\_idle\_condition

UART idle condition detect.

*Values:*

enumerator kUART\_IdleFor4Frames

Idle for more than 4 frames

enumerator kUART\_IdleFor8Frames

Idle for more than 8 frames

enumerator kUART\_IdleFor16Frames

Idle for more than 16 frames

enumerator kUART\_IdleFor32Frames

Idle for more than 32 frames

enum \_uart\_interrupt\_enable

This structure contains the settings for all of the UART interrupt configurations.

*Values:*

enumerator kUART\_AutoBaudEnable

enumerator kUART\_TxReadyEnable

enumerator kUART\_IdleEnable

enumerator kUART\_RxReadyEnable

enumerator kUART\_TxEmptyEnable

enumerator kUART\_RtsDeltaEnable

enumerator kUART\_EscapeEnable

enumerator kUART\_RtsEnable

enumerator kUART\_AgingTimerEnable

enumerator kUART\_DtrEnable

enumerator kUART\_ParityErrorEnable

enumerator kUART\_FrameErrorEnable

enumerator kUART\_DcdEnable

enumerator kUART\_RiEnable

enumerator kUART\_RxDsEnable

enumerator kUART\_tAirWakeEnable

enumerator kUART\_AwakeEnable

enumerator kUART\_DtrDeltaEnable

enumerator kUART\_AutoBaudCntEnable

enumerator kUART\_IrEnable

enumerator kUART\_WakeEnable

enumerator kUART\_TxCompleteEnable

enumerator kUART\_BreakDetectEnable

enumerator kUART\_RxOverrunEnable

enumerator kUART\_RxDataReadyEnable

enumerator kUART\_RxDmaIdleEnable

enumerator kUART\_AllInterruptsEnable

UART status flags.

This provides constants for the UART status flags for use in the UART functions.

*Values:*

enumerator kUART\_RxCharReadyFlag  
Rx Character Ready Flag.

enumerator kUART\_RxErrorFlag  
Rx Error Detect Flag.

enumerator kUART\_RxOverrunErrorFlag  
Rx Overrun Flag.

enumerator kUART\_RxFrameErrorFlag  
Rx Frame Error Flag.

enumerator kUART\_RxBreakDetectFlag  
Rx Break Detect Flag.

enumerator kUART\_RxParityErrorFlag  
Rx Parity Error Flag.

enumerator kUART\_ParityErrorFlag  
Parity Error Interrupt Flag.

enumerator kUART\_RtsStatusFlag  
RTS\_B Pin Status Flag.

enumerator kUART\_TxReadyFlag  
Transmitter Ready Interrupt/DMA Flag.

enumerator kUART\_RtsDeltaFlag  
RTS Delta Flag.

enumerator kUART\_EscapeFlag  
Escape Sequence Interrupt Flag.

enumerator kUART\_FrameErrorFlag  
Frame Error Interrupt Flag.

enumerator kUART\_RxReadyFlag  
Receiver Ready Interrupt/DMA Flag.

enumerator kUART\_AgingTimerFlag  
Aging Timer Interrupt Flag.

enumerator kUART\_DtrDeltaFlag  
DTR Delta Flag.

enumerator kUART\_RxDsFlag  
Receiver IDLE Interrupt Flag.

enumerator kUART\_tAirWakeFlag  
Asynchronous IR WAKE Interrupt Flag.

enumerator kUART\_AwakeFlag  
Asynchronous WAKE Interrupt Flag.

enumerator kUART\_Rs485SlaveAddrMatchFlag  
RS-485 Slave Address Detected Interrupt Flag.

enumerator `kUART_AutoBaudFlag`  
Automatic Baud Rate Detect Complete Flag.

enumerator `kUART_TxEmptyFlag`  
Transmit Buffer FIFO Empty.

enumerator `kUART_DtrFlag`  
DTR edge triggered interrupt flag.

enumerator `kUART_IdleFlag`  
Idle Condition Flag.

enumerator `kUART_AutoBaudCntStopFlag`  
Auto-baud Counter Stopped Flag.

enumerator `kUART_RiDeltaFlag`  
Ring Indicator Delta Flag.

enumerator `kUART_RiFlag`  
Ring Indicator Input Flag.

enumerator `kUART_IrFlag`  
Serial Infrared Interrupt Flag.

enumerator `kUART_WakeFlag`  
Wake Flag.

enumerator `kUART_DcdDeltaFlag`  
Data Carrier Detect Delta Flag.

enumerator `kUART_DcdFlag`  
Data Carrier Detect Input Flag.

enumerator `kUART_RtsFlag`  
RTS Edge Triggered Interrupt Flag.

enumerator `kUART_TxCompleteFlag`  
Transmitter Complete Flag.

enumerator `kUART_BreakDetectFlag`  
BREAK Condition Detected Flag.

enumerator `kUART_RxOverrunFlag`  
Overrun Error Flag.

enumerator `kUART_RxDataReadyFlag`  
Receive Data Ready Flag.

typedef enum *uart\_data\_bits* `uart_data_bits_t`  
UART data bits count.

typedef enum *uart\_parity\_mode* `uart_parity_mode_t`  
UART parity mode.

typedef enum *uart\_stop\_bit\_count* `uart_stop_bit_count_t`  
UART stop bit count.

typedef enum *uart\_idle\_condition* `uart_idle_condition_t`  
UART idle condition detect.

typedef struct *uart\_config* `uart_config_t`  
UART configuration structure.



```
typedef struct _uart_transfer uart_transfer_t
```

UART transfer structure.

```
typedef struct _uart_handle uart_handle_t
```

Forward declaration of the handle typedef.

```
typedef void (*uart_transfer_callback_t)(UART_Type *base, uart_handle_t *handle, status_t status, void *userData)
```

UART transfer callback function.

```
typedef void (*uart_isr_t)(UART_Type *base, void *handle)
```

```
const IRQn_Type s_uartIRQ[]
```

```
uart_isr_t s_uartIsr
```

```
void *s_uartHandle[]
```

Pointers to uart handles for each instance.

```
uint32_t UART_GetInstance(UART_Type *base)
```

Get the UART instance from peripheral base address.

#### Parameters

- base – UART peripheral base address.

#### Returns

UART instance.

```
UART_RETRY_TIMES
```

Retry times for waiting flag.

```
struct _uart_config
```

*#include <fsl\_uart.h>* UART configuration structure.

#### Public Members

```
uint32_t baudRate_Bps
```

UART baud rate.

```
uart_parity_mode_t parityMode
```

Parity error check mode of this module.

```
uart_data_bits_t dataBitsCount
```

Data bits count, eight (default), seven

```
uart_stop_bit_count_t stopBitCount
```

Number of stop bits in one frame.

```
uint8_t txFifoWatermark
```

TX FIFO watermark

```
uint8_t rxFifoWatermark
```

RX FIFO watermark

```
uint8_t rxRTSWatermark
```

RX RTS watermark, RX FIFO data count being larger than this triggers RTS deassertion

```
bool enableAutoBaudRate
```

Enable automatic baud rate detection

```
bool enableTx
```

Enable TX

bool enableRx

Enable RX

bool enableRxRTS

RX RTS enable

bool enableTxCTS

TX CTS enable

struct \_\_uart\_\_transfer

*#include <fsl\_uart.h>* UART transfer structure.

### Public Members

size\_t dataSize

The byte count to be transfer.

struct \_\_uart\_\_handle

*#include <fsl\_uart.h>* UART handle structure.

### Public Members

const uint8\_t \*volatile txData

Address of remaining data to send.

volatile size\_t txDataSize

Size of the remaining data to send.

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.

*uart\_transfer\_callback\_t* callback

Callback function.

void \*userData

UART callback function parameter.

volatile uint8\_t txState

TX transfer state.

```
volatile uint8_t rxState
    RX transfer state
union __unnamed9__
```

### 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.
```

## 2.39 UART FreeRTOS Driver

### 2.40 UART SDMA Driver

```
void UART_TransferCreateHandleSDMA(UART_Type *base, uart_sdma_handle_t *handle,
                                   uart_sdma_transfer_callback_t callback, void
                                   *userData, sdma_handle_t *txSdmaHandle,
                                   sdma_handle_t *rxSdmaHandle, uint32_t
                                   eventSourceTx, uint32_t eventSourceRx)
```

Initializes the UART handle which is used in transactional functions.

#### Parameters

- base – UART peripheral base address.
- handle – Pointer to the `uart_sdma_handle_t` structure.
- callback – UART callback, NULL means no callback.
- userData – User callback function data.
- rxSdmaHandle – User-requested DMA handle for RX DMA transfer.
- txSdmaHandle – User-requested DMA handle for TX DMA transfer.
- eventSourceTx – Eventsource for TX DMA transfer.
- eventSourceRx – Eventsource for RX DMA transfer.

```
status_t UART_SendSDMA(UART_Type *base, uart_sdma_handle_t *handle, uart_transfer_t
                       *xfer)
```

Sends data using sDMA.

This function sends data using sDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

#### Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.
- xfer – UART sDMA transfer structure. See `uart_transfer_t`.

#### Return values

- `kStatus_Success` – if succeeded; otherwise failed.

- kStatus\_UART\_TxBusy – Previous transfer ongoing.
- kStatus\_InvalidArgument – Invalid argument.

*status\_t* UART\_ReceiveSDMA(UART\_Type \*base, *uart\_sdma\_handle\_t* \*handle, *uart\_transfer\_t* \*xfer)

Receives data using sDMA.

This function receives data using sDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

#### Parameters

- base – UART peripheral base address.
- handle – Pointer to the *uart\_sdma\_handle\_t* structure.
- xfer – UART sDMA transfer structure. See *uart\_transfer\_t*.

#### Return values

- kStatus\_Success – if succeeded; otherwise failed.
- kStatus\_UART\_RxBusy – Previous transfer ongoing.
- kStatus\_InvalidArgument – Invalid argument.

void UART\_TransferAbortSendSDMA(UART\_Type \*base, *uart\_sdma\_handle\_t* \*handle)

Aborts the sent data using sDMA.

This function aborts sent data using sDMA.

#### Parameters

- base – UART peripheral base address.
- handle – Pointer to the *uart\_sdma\_handle\_t* structure.

void UART\_TransferAbortReceiveSDMA(UART\_Type \*base, *uart\_sdma\_handle\_t* \*handle)

Aborts the receive data using sDMA.

This function aborts receive data using sDMA.

#### Parameters

- base – UART peripheral base address.
- handle – Pointer to the *uart\_sdma\_handle\_t* structure.

void UART\_TransferSdmaHandleIRQ(UART\_Type \*base, void \*uartSdmaHandle)

UART IRQ handle function.

This function handles the UART transmit complete IRQ request and invoke user callback.

#### Parameters

- base – UART peripheral base address.
- uartSdmaHandle – UART handle pointer.

FSL\_UART\_SDMA\_DRIVER\_VERSION

UART SDMA driver version.

typedef struct *uart\_sdma\_handle* *uart\_sdma\_handle\_t*

typedef void (\**uart\_sdma\_transfer\_callback\_t*)(UART\_Type \*base, *uart\_sdma\_handle\_t* \*handle, *status\_t* status, void \*userData)

UART transfer callback function.

struct *uart\_sdma\_handle*

#include <*fsl\_uart\_sdma.h*> UART sDMA handle.

**Public Members**

*uart\_sdma\_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.

*sdma\_handle\_t \*txSdmaHandle*

The sDMA TX channel used.

*sdma\_handle\_t \*rxSdmaHandle*

The sDMA RX channel used.

*volatile uint8\_t txState*

TX transfer state.

*volatile uint8\_t rxState*

RX transfer state

## 2.41 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 void USDHC\_SelectVoltage(USDHC\_Type \*base, bool en18v)  
Selects the USDHC output voltage.

#### Parameters

- base – USDHC peripheral base address.
- en18v – True means 1.8V, false means 3.0V.

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

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\_SupportV180Flag  
Support voltage 1.8V.

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.

uint8\_t readBurstLen

Read burst len.

uint8\_t writeBurstLen

Write burst len.

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.

*usdhc\_burst\_len\_t* burstLen

Burst len config.

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.

## 2.42 WDOG: Watchdog Timer Driver

void WDOG\_GetDefaultConfig(*wdog\_config\_t* \*config)

Initializes the WDOG configuration structure.

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

```
wdogConfig->enableWdog = true;
wdogConfig->workMode.enableWait = true;
wdogConfig->workMode.enableStop = true;
wdogConfig->workMode.enableDebug = true;
wdogConfig->enableInterrupt = false;
wdogConfig->enablePowerdown = false;
wdogConfig->resetExtension = false;
wdogConfig->timeoutValue = 0xFFU;
wdogConfig->interruptTimeValue = 0x04u;
```

**See also:**

*wdog\_config\_t*



**Parameters**

- config – Pointer to the WDOG configuration structure.

void WDOG\_Init(WDOG\_Type \*base, const *wdog\_config\_t* \*config)

Initializes the WDOG.

This function initializes the WDOG. When called, the WDOG runs according to the configuration.

This is an example.

```
wdog_config_t config;
WDOG_GetDefaultConfig(&config);
config.timeoutValue = 0xffU;
config->interruptTimeValue = 0x04u;
WDOG_Init(wdog_base,&config);
```

**Parameters**

- base – WDOG peripheral base address
- config – The configuration of WDOG

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

Shuts down the WDOG.

This function shuts down the WDOG. Watchdog Enable bit is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. This bit(WDE) can be set/reset only in debug mode(exception).

static inline void WDOG\_Enable(WDOG\_Type \*base)

Enables the WDOG module.

This function writes a value into the WDOG\_WCR register to enable the WDOG. This is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. only debug mode exception.

**Parameters**

- base – WDOG peripheral base address

static inline void WDOG\_Disable(WDOG\_Type \*base)

Disables the WDOG module.

This function writes a value into the WDOG\_WCR register to disable the WDOG. This is a write one once only bit. It is not possible to clear this bit by a software write,once the bit is set. only debug mode exception

**Parameters**

- base – WDOG peripheral base address

static inline void WDOG\_TriggerSystemSoftwareReset(WDOG\_Type \*base)

Trigger the system software reset.

This function will write to the WCR[SRS] bit to trigger a software system reset. This bit will automatically resets to “1” after it has been asserted to “0”. Note: Calling this API will reset the system right now, please using it with more attention.

**Parameters**

- base – WDOG peripheral base address

```
static inline void WDOG_TriggerSoftwareSignal(WDOG_Type *base)
```

Trigger an output assertion.

This function will write to the WCR[WDA] bit to trigger WDOG\_B signal assertion. The WDOG\_B signal can be routed to external pin of the chip, the output pin will turn to assertion along with WDOG\_B signal. Note: The WDOG\_B signal will remain assert until a power on reset occurred, so, please take more attention while calling it.

#### Parameters

- base – WDOG peripheral base address

```
static inline void WDOG_EnableInterrupts(WDOG_Type *base, uint16_t mask)
```

Enables the WDOG interrupt.

This bit is a write once only bit. Once the software does a write access to this bit, it will get locked and cannot be reprogrammed until the next system reset assertion

#### Parameters

- base – WDOG peripheral base address
- mask – The interrupts to enable The parameter can be combination of the following source if defined.
  - kWDOG\_InterruptEnable

```
uint16_t WDOG_GetStatusFlags(WDOG_Type *base)
```

Gets the WDOG all reset status flags.

This function gets all reset status flags.

```
uint16_t status;  
status = WDOG_GetStatusFlags (wdog_base);
```

#### See also:

`_wdog_status_flags`

- true: a related status flag has been set.
- false: a related status flag is not set.

#### Parameters

- base – WDOG peripheral base address

#### Returns

State of the status flag: asserted (true) or not-asserted (false).

```
void WDOG_ClearInterruptStatus(WDOG_Type *base, uint16_t mask)
```

Clears the WDOG flag.

This function clears the WDOG status flag.

This is an example for clearing the interrupt flag.

```
WDOG_ClearStatusFlags(wdog_base,KWDOG_InterruptFlag);
```

#### Parameters

- base – WDOG peripheral base address
- mask – The status flags to clear. The parameter could be any combination of the following values. kWDOG\_TimeoutFlag

```
static inline void WDOG_SetTimeoutValue(WDOG_Type *base, uint16_t timeoutCount)
```

Sets the WDOG timeout value.

This function sets the timeout value. This function writes a value into WCR registers. The time-out value can be written at any point of time but it is loaded to the counter at the time when WDOG is enabled or after the service routine has been performed.

#### Parameters

- base – WDOG peripheral base address
- timeoutCount – WDOG timeout value; count of WDOG clock tick.

```
static inline void WDOG_SetInterruptTimeoutValue(WDOG_Type *base, uint16_t timeoutCount)
```

Sets the WDOG interrupt count timeout value.

This function sets the interrupt count timeout value. This function writes a value into WIC registers which are write-once. This field is write once only. Once the software does a write access to this field, it will get locked and cannot be reprogrammed until the next system reset assertion.

#### Parameters

- base – WDOG peripheral base address
- timeoutCount – WDOG timeout value; count of WDOG clock tick.

```
static inline void WDOG_DisablePowerDownEnable(WDOG_Type *base)
```

Disable the WDOG power down enable bit.

This function disable the WDOG power down enable(PDE). This function writes a value into WMCR registers which are write-once. This field is write once only. Once software sets this bit it cannot be reset until the next system reset.

#### Parameters

- base – WDOG peripheral base address

```
void WDOG_Refresh(WDOG_Type *base)
```

Refreshes the WDOG timer.

This function feeds the WDOG. This function should be called before the WDOG timer is in timeout. Otherwise, a reset is asserted.

#### Parameters

- base – WDOG peripheral base address

```
FSL_WDOG_DRIVER_VERSION
```

Defines WDOG driver version.

```
WDOG_REFRESH_KEY
```

```
enum _wdog_interrupt_enable
```

WDOG interrupt configuration structure, default settings all disabled.

This structure contains the settings for all of the WDOG interrupt configurations.

*Values:*

```
enumerator kWDOG_InterruptEnable
```

WDOG timeout generates an interrupt before reset

```
enum _wdog_status_flags
```

WDOG status flags.

This structure contains the WDOG status flags for use in the WDOG functions.

*Values:*

enumerator kWDOG\_RunningFlag

Running flag, set when WDOG is enabled

enumerator kWDOG\_PowerOnResetFlag

Power On flag, set when reset is the result of a powerOnReset

enumerator kWDOG\_TimeoutResetFlag

Timeout flag, set when reset is the result of a timeout

enumerator kWDOG\_SoftwareResetFlag

Software flag, set when reset is the result of a software

enumerator kWDOG\_InterruptFlag

interrupt flag, whether interrupt has occurred or not

typedef struct *\_wdog\_work\_mode* wdog\_work\_mode\_t

Defines WDOG work mode.

typedef struct *\_wdog\_config* wdog\_config\_t

Describes WDOG configuration structure.

struct *\_wdog\_work\_mode*

*#include <fsl\_wdog.h>* Defines WDOG work mode.

### Public Members

bool enableWait

If set to true, WDOG continues in wait mode

bool enableStop

If set to true, WDOG continues in stop mode

bool enableDebug

If set to true, WDOG continues in debug mode

struct *\_wdog\_config*

*#include <fsl\_wdog.h>* Describes WDOG configuration structure.

### Public Members

bool enableWdog

Enables or disables WDOG

*wdog\_work\_mode\_t* workMode

Configures WDOG work mode in debug stop and wait mode

bool enableInterrupt

Enables or disables WDOG interrupt

uint16\_t timeoutValue

Timeout value

uint16\_t interruptTimeValue

Interrupt count timeout value

bool softwareResetExtension

software reset extension

bool enablePowerDown

power down enable bit

`bool enableTimeOutAssert`

Enable WDOG\_B timeout assertion.



# Chapter 3

## Middleware

### 3.1 Motor Control

#### 3.1.1 FreeMASTER

*Communication Driver User Guide*

##### Introduction

**What is FreeMASTER?** FreeMASTER is a PC-based application developed by NXP for NXP customers. It is a versatile tool usable as a real-time monitor, visualization tool, and a graphical control panel of embedded applications based on the NXP processing units.

This document describes the embedded-side software driver which implements an interface between the application and the host PC. The interface covers the following communication:

- **Serial** UART communication either over plain RS232 interface or more typically over a USB-to-Serial either external or built in a debugger probe.
- **USB** direct connection to target microcontroller
- **CAN bus**
- **TCP/IP network** wired or WiFi
- **Segger J-Link RTT**
- **JTAG** debug port communication
- ...and all of the above also using a **Zephyr** generic drivers.

The driver also supports so-called “packet-driven BDM” interface which enables a protocol-based communication over a debugging port. The BDM stands for Background Debugging Module and its physical implementation is different on each platform. Some platforms leverage a semi-standard JTAG interface, other platforms provide a custom implementation called BDM. Regardless of the name, this debugging interface enables non-intrusive access to the memory space while the target CPU is running. For basic memory read and write operations, there is no communication driver required on the target when communicating with the host PC. Use this driver to get more advanced FreeMASTER protocol features over the BDM interface. The driver must be configured for the packet-driven BDM mode, in which the host PC uses the debugging interface to write serial command frames directly to the target memory buffer. The same method is then used to read response frames from that memory buffer.

Similar to “packet-driven BDM”, the FreeMASTER also supports a communication over [J-Link RTT](<https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/>) interface defined by SEGGER Microcontroller GmbH for ARM CortexM-based microcontrollers. This method also uses JTAG physical interface and enables high-speed real time communication to run over the same channel as used for application debugging.

**Driver version 3** This document describes version 3 of the FreeMASTER Communication Driver. This version features the implementation of the new Serial Protocol, which significantly extends the features and security of its predecessor. The new protocol internal number is v4 and its specification is available in the documentation accompanying the driver code.

Driver V3 is deployed to modern 32-bit MCU platforms first, so the portfolio of supported platforms is smaller than for the previous V2 versions. It is recommended to keep using the V2 driver for legacy platforms, such as S08, S12, ColdFire, or Power Architecture. Reach out to [FreeMASTER community](#) or to the local NXP representative with requests for more information or to port the V3 driver to legacy MCU devices.

Thanks to a layered approach, the new driver simplifies the porting of the driver to new UART, CAN or networking communication interfaces significantly. Users are encouraged to port the driver to more NXP MCU platforms and contribute the code back to NXP for integration into future releases. Existing code and low-level driver layers may be used as an example when porting to new targets.

**Note:** Using the FreeMASTER tool and FreeMASTER Communication Driver is only allowed in systems based on NXP microcontroller or microprocessor unit. Use with non-NXP MCU platforms is **not permitted** by the license terms.

**Target platforms** The driver implementation uses the following abstraction mechanisms which simplify driver porting and supporting new communication modules:

- **General CPU Platform** (see source code in the `src/platforms` directory). The code in this layer is only specific to native data type sizes and CPU architectures (for example; alignment-aware memory copy routines). This driver version brings two generic implementations of 32-bit platforms supporting both little-endian and big-endian architectures. There are also implementations customized for the 56F800E family of digital signal controllers and S12Z MCUs. **Zephyr** is treated as a specific CPU platform as it brings unified user configuration (Kconfig) and generic hardware device drivers. With Zephyr, the transport layer and low-level communication layers described below are configured automatically using Kconfig and Device Tree technologies.
- **Transport Communication Layer** - The Serial, CAN, Networking, PD-BDM, and other methods of transport logic are implemented as a driver layer called `FMSTR_TRANSPORT` with a uniform API. A support of the Network transport also extends single-client modes of operation which are native for Serial, USB and CAN by a concept of multiple client sessions.
- **Low-level Communication Driver** - Each type of transport further defines a low-level API used to access the physical communication module. For example, the Serial transport defines a character-oriented API implemented by different serial communication modules like UART, LPUART, USART, and also USB-CDC. Similarly, the CAN transport defines a message-oriented API implemented by the FlexCAN or MCAN modules. Moreover, there are multiple different implementations for the same kind of communication peripherals. The difference between the implementation is in the way the low-level hardware registers are accessed. The `mcuxsdk` folder contains implementations which use MCUXpresso SDK drivers. These drivers should be used in applications based on the NXP MCUXpresso SDK. The “ampsdk” drivers target automotive-specific MCUs and their respective SDKs. The “dreg” implementations use a plain C-language access to hardware register addresses which makes it a universal and the most portable solution. In this case, users are encouraged to add more drivers for other communication modules or other respective SDKs and contribute the code back to NXP for integration.



The low-level drivers defined for the Networking transport enable datagram-oriented UDP and stream TCP communication. This implementation is demonstrated using the lwIP software stack but shall be portable to other TCP/IP stacks. It may sound surprisingly, but also the Segger J-Link RTT communication driver is linked to the Networking transport (RTT is stream oriented communication handled similarly to TCP).

**Replacing existing drivers** For all supported platforms, the driver described in this document replaces the V2 implementation and also older driver implementations that were available separately for individual platforms (PC Master SCI drivers).

**Clocks, pins, and peripheral initialization** The FreeMASTER communication driver is only responsible for runtime processing of the communication and must be integrated with an user application code to function properly. The user application code is responsible for general initialization of clock sources, pin multiplexers, and peripheral registers related to the communication speed. Such initialization should be done before calling the `FMSTR_Init` function.

It is recommended to develop the user application using one of the Software Development Kits (SDKs) available from third parties or directly from NXP, such as MCUXpresso SDK, MCUXpresso IDE, and related tools. This approach simplifies the general configuration process significantly.

**MCUXpresso SDK** The MCUXpresso SDK is a software package provided by NXP which contains the device initialization code, linker files, and software drivers with example applications for the NXP family of MCUs. The MCUXpresso Config Tools may be used to generate the clock-setup and pin-multiplexer setup code suitable for the selected processor.

The MCUXpresso SDK also contains this FreeMASTER communication driver as a “middleware” component which may be downloaded along with the example applications from <https://mcuxpresso.nxp.com/en/welcome>.

**MCUXpresso SDK on GitHub** The FreeMASTER communication driver is also released as one of the middleware components of the MCUXpresso SDK on the GitHub. This release enables direct integration of the FreeMASTER source code Git repository into a target applications including Zephyr applications.

Related links:

- [The official FreeMASTER middleware repository.](#)
- [Online version of this document](#)

**FreeMASTER in Zephyr** The FreeMASTER middleware repository can be used with MCUXpresso SDK as well as a Zephyr module. Zephyr-specific samples which include examples of Kconfig and Device Tree configurations for Serial, USB and Network communications are available in separate repository. West manifest in this sample repository fetches the full Zephyr package including the FreeMASTER middleware repository used as a Zephyr module.

## Example applications

**MCUX SDK Example applications** There are several example applications available for each supported MCU platform.

- **fmstr\_uart** demonstrates a plain serial transmission, typically connecting to a computer’s physical or virtual COM port. The typical transmission speed is 115200 bps.

- **fmstr\_can** demonstrates CAN bus communication. This requires a suitable CAN interface connected to the computer and interconnected with the target MCU using a properly terminated CAN bus. The typical transmission speed is 500 kbps. A FreeMASTER-over-CAN communication plug-in must be used.
- **fmstr\_usb\_cdc** uses an on-chip USB controller to implement a CDC communication class. It is connected directly to a computer's USB port and creates a virtual COM port device. The typical transmission speed is above 1 Mbps.
- **fmstr\_net** demonstrates the Network communication over UDP or TCP protocol. Existing examples use lwIP stack to implement the communication, but in general, it shall be possible to use any other TCP/IP stack to achieve the same functionality.
- **fmstr\_wifi** is the fmstr\_net application modified to use a WiFi network interface instead of a wired Ethernet connection.
- **fmstr\_rtt** demonstrates the communication over SEGGER J-Link RTT interface. Both fmstr\_net and fmstr\_rtt examples require the FreeMASTER TCP/UDP communication plug-in to be used on the PC host side.
- **fmstr\_eonce** uses the real-time data unit on the JTAG EOnCE module of the 56F800E family to implement pseudo-serial communication over the JTAG port. The typical transmission speed is around 10 kbps. This communication requires FreeMASTER JTAG/EOnCE communication plug-in.
- **fmstr\_pd\_bdm** uses JTAG or BDM debugging interface to access the target RAM directly while the CPU is running. Note that such approach can be used with any MCU application, even without any special driver code. The computer reads from and writes into the RAM directly without CPU intervention. The Packet-Driven BDM (PD-BDM) communication uses the same memory access to exchange command and response frames. With PD-BDM, the FreeMASTER tool is able to go beyond basic memory read/write operations and accesses also advanced features like Recorder, TSA, or Pipes. The typical transmission speed is around 10 kbps. A PD-BDM communication plug-in must be used in FreeMASTER and configured properly for the selected debugging interface. Note that this communication cannot be used while a debugging interface is used by a debugger session.
- **fmstr\_any** is a special example application which demonstrates how the NXP MCUXpresso Config Tools can be used to configure pins, clocks, peripherals, interrupts, and even the FreeMASTER “middleware” driver features in a graphical and user friendly way. The user can switch between the Serial, CAN, and other ways of communication and generate the required initialization code automatically.

**Zephyr sample applications** Zephyr sample applications demonstrate Kconfig and Device Tree configuration which configure the FreeMASTER middleware module for a selected communication option (Serial, CAN, Network or RTT).

Refer to *readme.md* files in each sample directory for description of configuration options required to implement FreeMASTER connectivity.

## Description

This section shows how to add the FreeMASTER Communication Driver into application and how to configure the connection to the FreeMASTER visualization tool.

**Features** The FreeMASTER driver implements the FreeMASTER protocol V4 and provides the following features which may be accessed using the FreeMASTER visualization tool:

- Read/write access to any memory location on the target.
- Optional password protection of the read, read/write, and read/write/flash access levels.

- Atomic bit manipulation on the target memory (bit-wise write access).
- Optimal size-aligned access to memory which is also suitable to access the peripheral register space.
- Oscilloscope access—real-time access to target variables. The sample rate may be limited by the communication speed.
- Recorder— access to the fast transient recorder running on the board as a part of the FreeMASTER driver. The sample rate is only limited by the MCU CPU speed. The length of the data recorded depends on the amount of available memory.
- Multiple instances of Oscilloscopes and Recorders without the limitation of maximum number of variables.
- Application commands—high-level message delivery from the PC to the application.
- TSA tables—describing the data types, variables, files, or hyperlinks exported by the target application. The TSA newly supports also non-memory mapped resources like external EEPROM or SD Card files.
- Pipes—enabling the buffered stream-oriented data exchange for a general-purpose terminal-like communication, diagnostic data streaming, or other data exchange.

The FreeMASTER driver features:

- Full FreeMASTER protocol V4 implementation with a new V4 style of CRC used.
- Layered approach supporting Serial, CAN, Network, PD-BDM, and other transports.
- Layered low-level Serial transport driver architecture enabling to select UART, LPUART, USART, and other physical implementations of serial interfaces, including USB-CDC.
- Layered low-level CAN transport driver architecture enabling to select FlexCAN, msCAN, MCAN, and other physical implementations of the CAN interface.
- Layered low-level Networking transport enabling to select TCP, UDP or J-Link RTT communication.
- TSA support to write-protect memory regions or individual variables and to deny the access to the unsafe memory.
- The pipe callback handlers are invoked whenever new data is available for reading from the pipe.
- Two Serial Single-Wire modes of operation are enabled. The “external” mode has the RX and TX shorted on-board. The “true” single-wire mode interconnects internally when the MCU or UART modules support it.

The following sections briefly describe all FreeMASTER features implemented by the driver. See the PC-based FreeMASTER User Manual for more details on how to use the features to monitor, tune, or control an embedded application.

**Board Detection** The FreeMASTER protocol V4 defines the standard set of configuration values which the host PC tool reads to identify the target and to access other target resources properly. The configuration includes the following parameters:

- Version of the driver and the version of the protocol implemented.
- MTU as the Maximum size of the Transmission Unit (for example; communication buffer size).
- Application name, description, and version strings.
- Application build date and time as a string.
- Target processor byte ordering (little/big endian).
- Protection level that requires password authentication.

- Number of the Recorder and Oscilloscope instances.
- RAM Base Address for optimized memory access commands.

**Memory Read** This basic feature enables the host PC to read any data memory location by specifying the address and size of the required memory area. The device response frame must be shorter than the MTU to fit into the outgoing communication buffer. To read a device memory of any size, the host uses the information retrieved during the Board Detection and splits the large-block request to multiple partial requests.

The driver uses size-aligned operations to read the target memory (for example; uses proper read-word instruction when an address is aligned to 4 bytes).

**Memory Write** Similarly to the Memory Read operation, the Memory Write feature enables to write to any RAM memory location on the target device. A single write command frame must be shorter than the MTU to fit into the target communication buffer. Larger requests must be split into smaller ones.

The driver uses size-aligned operations to write to the target memory (for example; uses proper write-word instruction when an address is aligned to 4 bytes).

**Masked Memory Write** To implement the write access to a single bit or a group of bits of target variables, the Masked Memory Write feature is available in the FreeMASTER protocol and it is supported by the driver using the Read-Modify-Write approach.

Be careful when writing to bit fields of volatile variables that are also modified in an application interrupt. The interrupt may be serviced in the middle of a read-modify-write operation and it may cause data corruption.

**Oscilloscope** The protocol and driver enables any number of variables to be read at once with a single request from the host. This feature is called Oscilloscope and the FreeMASTER tool uses it to display a real-time graph of variable values.

The driver can be configured to support any number of Oscilloscope instances and enable simultaneously running graphs to be displayed on the host computer screen.

**Recorder** The protocol enables the host to select target variables whose values are then periodically recorded into a dedicated on-board memory buffer. After such data sampling stops (either on a host request or by evaluating a threshold-crossing condition), the data buffer is downloaded to the host and displayed as a graph. The data sampling rate is not limited by the speed of the communication line, so it enables displaying the variable transitions in a very high resolution.

The driver can be configured to support multiple Recorder instances and enable multiple recorder graphs to be displayed on the host screen. Having multiple recorders also enables setting the recording point differently for each instance. For example; one instance may be recording data in a general timer interrupt while another instance may record at a specific control algorithm time in the PWM interrupt.

**TSA** With the TSA feature, data types and variables can be described directly in the application source code. Such information is later provided to the FreeMASTER tool which may use it instead of reading symbol data from the application ELF executable file.

The information is encoded as so-called TSA tables which become direct part of the application code. The TSA tables contain descriptors of variables that shall be visible to the host tool. The descriptors can describe the memory areas by specifying the address and size of the memory

block or more conveniently using the C variable names directly. Different set of TSA descriptors can be used to encode information about the structure types, unions, enumerations, or arrays.

The driver also supports special types of TSA table entries to describe user resources like external EEPROM and SD Card files, memory-mapped files, virtual directories, web URL hyperlinks, and constant enumerations.

**TSA Safety** When the TSA is enabled in the application, the TSA Safety can be enabled and validate the memory accesses directly by the embedded-side driver. When the TSA Safety is turned on, any memory request received from the host is validated and accepted only if it belongs to a TSA-described object. The TSA entries can be declared as Read-Write or Read-Only so that the driver can actively deny the write access to the Read-Only objects.

**Application commands** The Application Commands are high-level messages that can be delivered from the PC Host to the embedded application for further processing. The embedded application can either poll the status, or be called back when a new Application Command arrives to be processed. After the embedded application acknowledges that the command is handled, the host receives the Result Code and reads the other return data from memory. Both the Application Commands and the Result Codes are specific to a given application and it is user's responsibility to define them. The FreeMASTER protocol and the FreeMASTER driver only implement the delivery channel and a set of API calls to enable the Application Command processing in general.

**Pipes** The Pipes enable buffered and stream-oriented data exchange between the PC Host and the target application. Any pipe can be written to and read from at both ends (either on the PC or the MCU). The data transmission is acknowledged using the special FreeMASTER protocol commands. It is guaranteed that the data bytes are delivered from the writer to the reader in a proper order and without losses.

**Serial single-wire operation** The MCU Serial Communication Driver natively supports normal dual-wire operation. Because the protocol is half-duplex only, the driver can also operate in two single-wire modes:

- “External” single-wire operation where the Receiver and Transmitter pins are shorted on the board. This mode is supported by default in the MCU driver because the Receiver and Transmitter units are enabled or disabled whenever needed. It is also easy to extend this operation for the RS485 communication.
- “True” single-wire mode which uses only a single pin and the direction switching is made by the UART module. This mode of operation must be enabled by defining the FMSTR\_SERIAL\_SINGLEWIRE configuration option.

**Multi-session support** With networking interface it is possible for multiple clients to access the target MCU simultaneously. Reading and writing of target memory is processed atomically so there is no risk of data corruption. The state-full resources such as Recorders or Oscilloscopes are locked to a client session upon first use and access is denied to other clients until lock is released..

## Zephyr-specific



**Dedicated communication task** FreeMASTER communication may run isolated in a dedicated task. The task automates the FMSTR\_Init and FMSTR\_Poll calls together with periodic activities enabling the FreeMASTER UI to fetch information about tasks and CPU utilization. The task can be started automatically or manually, and it must be assigned a priority to be able to react on interrupts and other communication events. Refer to Zephyr FreeMASTER sample applications which all use this communication task.

**Zephyr shell and logging over FreeMASTER pipe** FreeMASTER implements a shell backend which may use FreeMASTER pipe as a I/O terminal and logging output. Refer to Zephyr FreeMASTER sample applications which all use this feature.

**Automatic TSA tables** TSA tables can be declared as “automatic” in Zephyr which make them automatically registered in the table list. This may be very useful when there are many TSA tables or when the tables are defined in different (often unrelated) libraries linked together. In this case user does not need to build a list of all tables manually.

**Driver files** The driver source files can be found in a top-level src folder, further divided into the sub-folders:

- **src/platforms** platform-specific folder—one folder exists for each supported processor platform (for example; 32-bit Little Endian platform). Each such folder contains a platform header file with data types and a code which implements the potentially platform-specific operations, such as aligned memory access.
- **src/common** folder—contains the common driver source files shared by the driver for all supported platforms. All the .c files must be added to the project, compiled, and linked together with the application.
  - *freemaster.h* - master driver header file, which declares the common data types, macros, and prototypes of the FreeMASTER driver API functions.
  - *freemaster\_cfg.h.example* - this file can serve as an example of the FreeMASTER driver configuration file. Save this file into a project source code folder and rename it to *freemaster\_cfg.h*. The FreeMASTER driver code includes this file to get the project-specific configuration options and to optimize the compilation of the driver.
  - *freemaster\_defcfg.h* - defines the default values for each FreeMASTER configuration option if the option is not set in the *freemaster\_cfg.h* file.
  - *freemaster\_protocol.h* - defines the FreeMASTER protocol constants used internally by the driver.
  - *freemaster\_protocol.c* - implements the FreeMASTER protocol decoder and handles the basic Get Configuration Value, Memory Read, and Memory Write commands.
  - *freemaster\_rec.c* - handles the Recorder-specific commands and implements the Recorder sampling and triggering routines. When the Recorder is disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
  - *freemaster\_scope.c* - handles the Oscilloscope-specific commands. If the Oscilloscope is disabled by the FreeMASTER driver configuration file, this file compiles as void.
  - *freemaster\_pipes.c* - implements the Pipes functionality when the Pipes feature is enabled.
  - *freemaster\_appcmd.c* - handles the communication commands used to deliver and execute the Application Commands within the context of the embedded application. When the Application Commands are disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.

- *freemaster\_tsa.c* - handles the commands specific to the TSA feature. This feature enables the FreeMASTER host tool to obtain the TSA memory descriptors declared in the embedded application. If the TSA is disabled by the FreeMASTER driver configuration file, this file compiles as void.
- *freemaster\_tsa.h* - contains the declaration of the macros used to define the TSA memory descriptors. This file is indirectly included into the user application code (via *freemaster.h*).
- *freemaster\_sha.c* - implements the SHA-1 hash code used in the password authentication algorithm.
- *freemaster\_private.h* - contains the declarations of functions and data types used internally in the driver. It also contains the C pre-processor statements to perform the compile-time verification of the user configuration provided in the *freemaster\_cfg.h* file.
- *freemaster\_serial.c* - implements the serial protocol logic including the CRC, FIFO queuing, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a character-oriented API exported by the specific low-level driver.
- *freemaster\_serial.h* - defines the low-level character-oriented Serial API.
- *freemaster\_can.c* - implements the CAN protocol logic including the CAN message preparation, signalling using the first data byte in the CAN frame, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a message-oriented API exported by the specific low-level driver.
- *freemaster\_can.h* - defines the low-level message-oriented CAN API.
- *freemaster\_net.c* - implements the Network protocol transport logic including multiple session management code.
- *freemaster\_net.h* - definitions related to the Network transport.
- *freemaster\_pdbdm.c* - implements the packet-driven BDM communication buffer and other communication-related operations.
- *freemaster\_utils.c* - aligned memory copy routines, circular buffer management and other utility functions
- *freemaster\_utils.h* - definitions related to utility code.
- **src/drivers/[sdk]/serial** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster\_serial\_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the UART, LPUART, USART, and other kinds of Serial communication modules.
- **src/drivers/[sdk]/can** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster\_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the FlexCAN, msCAN, MCAN, and other kinds of CAN communication modules.
- **src/drivers/[sdk]/network** - contains low-level code adapting the FreeMASTER Network transport to an underlying TCP/IP or RTT stack.
  - *freemaster\_net\_lwip\_tcp.c* and *\_udp.c* - default networking implementation of TCP and UDP transports using lwIP stack.
  - *freemaster\_net\_segger\_rtt.c* - implementation of network transport using Segger J-Link RTT interface

**Driver configuration** The driver is configured using a single header file (*freemaster\_cfg.h*). Create this file and save it together with other project source files before compiling the driver code. All FreeMASTER driver source files include the *freemaster\_cfg.h* file and use the macros defined here for the conditional and parameterized compilation. The C compiler must locate the configuration file when compiling the driver files. Typically, it can be achieved by putting this file into a folder where the other project-specific included files are stored.

As a starting point to create the configuration file, get the *freemaster\_cfg.h.example* file, rename it to *freemaster\_cfg.h*, and save it into the project area.

**Note:** It is NOT recommended to leave the *freemaster\_cfg.h* file in the FreeMASTER driver source code folder. The configuration file must be placed at a project-specific location, so that it does not affect the other applications that use the same driver.

**Configurable items** This section describes the configuration options which can be defined in *freemaster\_cfg.h*.

### Interrupt modes

```
#define FMSTR_LONG_INTR    [0|1]
#define FMSTR_SHORT_INTR  [0|1]
#define FMSTR_POLL_DRIVEN [0|1]
```

**Value Type** boolean (0 or 1)

**Description** Exactly one of the three macros must be defined to non-zero. The others must be defined to zero or left undefined. The non-zero-defined constant selects the interrupt mode of the driver. See [Driver interrupt modes](#).

- FMSTR\_LONG\_INTR — long interrupt mode
- FMSTR\_SHORT\_INTR — short interrupt mode
- FMSTR\_POLL\_DRIVEN — poll-driven mode

**Note:** Some options may not be supported by all communication interfaces. For example, the FMSTR\_SHORT\_INTR option is not supported by the USB\_CDC interface.

### Protocol transport

```
#define FMSTR_TRANSPORT [identifier]
```

**Value Type** Driver identifiers are structure instance names defined in FreeMASTER source code. Specify one of existing instances to make use of the protocol transport.

**Description** Use one of the pre-defined constants, as implemented by the FreeMASTER code. The current driver supports the following transports:

- FMSTR\_SERIAL - serial communication protocol
- FMSTR\_CAN - using CAN communication
- FMSTR\_PDBDM - using packet-driven BDM communication
- FMSTR\_NET - network communication using TCP or UDP protocol



**Serial transport** This section describes configuration parameters used when serial transport is used:

```
#define FMSTR_TRANSPORT FMSTR_SERIAL
```

**FMSTR\_SERIAL\_DRV** Select what low-level driver interface will be used when implementing the Serial communication.

```
#define FMSTR_SERIAL_DRV [identifier]
```

**Value Type** Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing serial driver instances.

**Description** When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/serial* implementation):

- **FMSTR\_SERIAL\_MCUX\_UART** - UART driver
- **FMSTR\_SERIAL\_MCUX\_LPUART** - LPUART driver
- **FMSTR\_SERIAL\_MCUX\_USART** - USART driver
- **FMSTR\_SERIAL\_MCUX\_MINIUSART** - miniUSART driver
- **FMSTR\_SERIAL\_MCUX\_QSCI** - DSC QSCI driver
- **FMSTR\_SERIAL\_MCUX\_USB** - USB/CDC class driver (also see code in the */support/mcuxsdk\_usb* folder)
- **FMSTR\_SERIAL\_56F800E\_EONCE** - DSC JTAG EOnCE driver

Other SDKs or BSPs may define custom low-level driver interface structure which may be used as FMSTR\_SERIAL\_DRV. For example:

- **FMSTR\_SERIAL\_DREG\_UART** - demonstrates the low-level interface implemented without the MCUXpresso SDK and using direct access to peripheral registers.

## FMSTR\_SERIAL\_BASE

```
#define FMSTR_SERIAL_BASE [address|symbol]
```

**Value Type** Optional address value (numeric or symbolic)

**Description** Specify the base address of the UART, LPUART, USART, or other serial peripheral module to be used for the communication. This value is not defined by default. User application should call FMSTR\_SetSerialBaseAddress() to select the peripheral module.

## FMSTR\_COMM\_BUFFER\_SIZE

```
#define FMSTR_COMM_BUFFER_SIZE [number]
```

**Value Type** 0 or a value in range 32...255

**Description** Specify the size of the communication buffer to be allocated by the driver. Default value, which suits all driver features, is used when this option is defined as 0.

### FMSTR\_COMM\_RQUEUE\_SIZE

```
#define FMSTR_COMM_RQUEUE_SIZE [number]
```

**Value Type** Value in range 0...255

**Description** Specify the size of the FIFO receiver queue used to quickly receive and store characters in the FMSTR\_SHORT\_INTR interrupt mode. The default value is 32 B.

### FMSTR\_SERIAL\_SINGLEWIRE

```
#define FMSTR_SERIAL_SINGLEWIRE [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Set to non-zero to enable the “True” single-wire mode which uses a single MCU pin to communicate. The low-level driver enables the pin direction switching when the MCU peripheral supports it.

**CAN Bus transport** This section describes configuration parameters used when CAN transport is used:

```
#define FMSTR_TRANSPORT FMSTR_CAN
```

**FMSTR\_CAN\_DRV** Select what low-level driver interface will be used when implementing the CAN communication.

```
#define FMSTR_CAN_DRV [identifier]
```

**Value Type** Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing CAN driver instances.

**Description** When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/can implementation*):

- **FMSTR\_CAN\_MCUX\_FLEXCAN** - FlexCAN driver
- **FMSTR\_CAN\_MCUX\_MCAN** - MCAN driver
- **FMSTR\_CAN\_MCUX\_MSCAN** - msCAN driver
- **FMSTR\_CAN\_MCUX\_DSCFLEXCAN** - DSC FlexCAN driver
- **FMSTR\_CAN\_MCUX\_DSCMSCAN** - DSC msCAN driver

Other SDKs or BSPs may define the custom low-level driver interface structure which may be used as FMSTR\_CAN\_DRV.

### FMSTR\_CAN\_BASE

```
#define FMSTR_CAN_BASE [address|symbol]
```

**Value Type** Optional address value (numeric or symbolic)

**Description** Specify the base address of the FlexCAN, msCAN, or other CAN peripheral module to be used for the communication. This value is not defined by default. User application should call `FMSTR_SetCanBaseAddress()` to select the peripheral module.

#### FMSTR\_CAN\_CMDID

```
#define FMSTR_CAN_CMDID [number]
```

**Value Type** CAN identifier (11-bit or 29-bit number)

**Description** CAN message identifier used for FreeMASTER commands (direction from PC Host tool to target application). When declaring 29-bit identifier, combine the numeric value with `FMSTR_CAN_EXTID` bit. Default value is 0x7AA.

#### FMSTR\_CAN\_RSPID

```
#define FMSTR_CAN_RSPID [number]
```

**Value Type** CAN identifier (11-bit or 29-bit number)

**Description** CAN message identifier used for responding messages (direction from target application to PC Host tool). When declaring 29-bit identifier, combine the numeric value with `FMSTR_CAN_EXTID` bit. Note that both *CMDID* and *RSPID* values may be the same. Default value is 0x7AA.

#### FMSTR\_FLEXCAN\_TXMB

```
#define FMSTR_FLEXCAN_TXMB [number]
```

**Value Type** Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

**Description** Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame transmission. Default value is 0.

#### FMSTR\_FLEXCAN\_RXMB

```
#define FMSTR_FLEXCAN_RXMB [number]
```

**Value Type** Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

**Description** Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame reception. Note that the FreeMASTER driver may also operate with a common message buffer used by both TX and RX directions. Default value is 1.

**Network transport** This section describes configuration parameters used when Network transport is used:

```
#define FMSTR_TRANSPORT FMSTR_NET
```

**FMSTR\_NET\_DRV** Select network interface implementation.

```
#define FMSTR_NET_DRV [identifier]
```

**Value Type** Identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing NET driver instances.

**Description** When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/network implementation*):

- **FMSTR\_NET\_LWIP\_TCP** - TCP communication using lwIP stack
- **FMSTR\_NET\_LWIP\_UDP** - UDP communication using lwIP stack
- **FMSTR\_NET\_SEGGER\_RTT** - Communication using SEGGER J-Link RTT interface

Other SDKs or BSPs may define the custom networking interface which may be used as FMSTR\_CAN\_DRV.

Add another row below:

#### FMSTR\_NET\_PORT

```
#define FMSTR_NET_PORT [number]
```

**Value Type** TCP or UDP port number (short integer)

**Description** Specifies the server port number used by TCP or UDP protocols.

#### FMSTR\_NET\_BLOCKING\_TIMEOUT

```
#define FMSTR_NET_BLOCKING_TIMEOUT [number]
```

**Value Type** Timeout as number of milliseconds

**Description** This value specifies a timeout in milliseconds for which the network socket operations may block the execution inside *FMSTR\_Poll*. This may be set high (e.g. 250) when a dedicated RTOS task is used to handle FreeMASTER protocol polling. Set to a lower value when the polling task is also responsible for other operations. Set to 0 to attempt to use non-blocking socket operations.

**FMSTR\_NET\_AUTODISCOVERY**

```
#define FMSTR_NET_AUTODISCOVERY [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** This option enables the FreeMASTER driver to use a separate UDP socket to broadcast auto-discovery messages to network. This helps the FreeMASTER tool to discover the target device address, port and protocol options.

**Debugging options****FMSTR\_DISABLE**

```
#define FMSTR_DISABLE [0|1]
```

**Value Type** boolean (0 or 1)

**Description** Define as non-zero to disable all FreeMASTER features, exclude the driver code from build, and compile all its API functions empty. This may be useful to remove FreeMASTER without modifying any application source code. Default value is 0 (false).

**FMSTR\_DEBUG\_TX**

```
#define FMSTR_DEBUG_TX [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to enable the driver to periodically transmit test frames out on the selected communication interface (SCI or CAN). With the debug transmission enabled, it is simpler to detect problems in the baudrate or other communication configuration settings.

The test frames are transmitted until the first valid command frame is received from the PC Host tool. The test frame is a valid error status frame, as defined by the protocol format. On the serial line, the test frame consists of three printable characters (+@W) which are easy to capture using the serial terminal tools.

This feature requires the FMSTR\_Poll() function to be called periodically. Default value is 0 (false).

**FMSTR\_APPLICATION\_STR**

```
#define FMSTR_APPLICATION_STR
```

**Value Type** String.

**Description** Name of the application visible in FreeMASTER host application.

**Memory access**

**FMSTR\_USE\_READMEM**

```
#define FMSTR_USE_READMEM [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the Memory Read command and enable FreeMASTER to have read access to memory and variables. The access can be further restricted by using a TSA feature.  
Default value is 1 (true).

**FMSTR\_USE\_WRITEMEM**

```
#define FMSTR_USE_WRITEMEM [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the Memory Write command.  
The default value is 1 (true).

**Oscilloscope options****FMSTR\_USE\_SCOPE**

```
#define FMSTR_USE_SCOPE [number]
```

**Value Type** Integer number.

**Description** Number of Oscilloscope instances to be supported. Set to 0 to disable the Oscilloscope feature.  
Default value is 0.

**FMSTR\_MAX\_SCOPE\_VARS**

```
#define FMSTR_MAX_SCOPE_VARS [number]
```

**Value Type** Integer number larger than 2.

**Description** Number of variables to be supported by each Oscilloscope instance.  
Default value is 8.

**Recorder options****FMSTR\_USE\_RECORDER**

```
#define FMSTR_USE_RECORDER [number]
```

**Value Type** Integer number.

**Description** Number of Recorder instances to be supported. Set to 0 to disable the Recorder feature.  
Default value is 0.

#### FMSTR\_REC\_BUFF\_SIZE

```
#define FMSTR_REC_BUFF_SIZE [number]
```

**Value Type** Integer number larger than 2.

**Description** Defines the size of the memory buffer used by the Recorder instance #0.  
Default: not defined, user shall call 'FMSTR\_RecorderCreate()' API function to specify this parameter in run time.

#### FMSTR\_REC\_TIMEBASE

```
#define FMSTR_REC_TIMEBASE [time specification]
```

**Value Type** Number (nanoseconds time).

**Description** Defines the base sampling rate in nanoseconds (sampling speed) Recorder instance #0.

Use one of the following macros:

- FMSTR\_REC\_BASE\_SECONDS(x)
- FMSTR\_REC\_BASE\_MILLISEC(x)
- FMSTR\_REC\_BASE\_MICROSEC(x)
- FMSTR\_REC\_BASE\_NANOSEC(x)

Default: not defined, user shall call 'FMSTR\_RecorderCreate()' API function to specify this parameter in run time.

#### FMSTR\_REC\_FLOAT\_TRIG

```
#define FMSTR_REC_FLOAT_TRIG [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the floating-point triggering. Be aware that floating-point triggering may grow the code size by linking the floating-point standard library.  
Default value is 0 (false).

### Application Commands options

**FMSTR\_USE\_APPCMD**

```
#define FMSTR_USE_APPCMD [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the Application Commands feature. Default value is 0 (false).

**FMSTR\_APPCMD\_BUFF\_SIZE**

```
#define FMSTR_APPCMD_BUFF_SIZE [size]
```

**Value Type** Numeric buffer size in range 1..255

**Description** The size of the Application Command data buffer allocated by the driver. The buffer stores the (optional) parameters of the Application Command which waits to be processed.

**FMSTR\_MAX\_APPCMD\_CALLS**

```
#define FMSTR_MAX_APPCMD_CALLS [number]
```

**Value Type** Number in range 0..255

**Description** The number of different Application Commands that can be assigned a callback handler function using FMSTR\_RegisterAppCmdCall(). Default value is 0.

**TSA options****FMSTR\_USE\_TSA**

```
#define FMSTR_USE_TSA [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable the FreeMASTER TSA feature to be used. With this option enabled, the TSA tables defined in the applications are made available to the FreeMASTER host tool. Default value is 0 (false).

**FMSTR\_USE\_TSA\_SAFETY**

```
#define FMSTR_USE_TSA_SAFETY [0|1]
```

**Value Type** Boolean 0 or 1.



**Description** Enable the memory access validation in the FreeMASTER driver. With this option, the host tool is not able to access the memory which is not described by at least one TSA descriptor. Also a write access is denied for objects defined as read-only in TSA tables. Default value is 0 (false).

#### FMSTR\_USE\_TSA\_INROM

```
#define FMSTR_USE_TSA_INROM [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Declare all TSA descriptors as *const*, which enables the linker to put the data into the flash memory. The actual result depends on linker settings or the linker commands used in the project. Default value is 0 (false).

#### FMSTR\_USE\_TSA\_DYNAMIC

```
#define FMSTR_USE_TSA_DYNAMIC [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable runtime-defined TSA entries to be added to the TSA table by the FMSTR\_SetUpTsaBuff() and FMSTR\_TsaAddVar() functions. Default value is 0 (false).

### Pipes options

#### FMSTR\_USE\_PIPES

```
#define FMSTR_USE_PIPES [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable the FreeMASTER Pipes feature to be used. Default value is 0 (false).

#### FMSTR\_MAX\_PIPES\_COUNT

```
#define FMSTR_MAX_PIPES_COUNT [number]
```

**Value Type** Number in range 1..63.

**Description** The number of simultaneous pipe connections to support. The default value is 1.

**Driver interrupt modes** To implement the communication, the FreeMASTER driver handles the Serial or CAN module's receive and transmit requests. Use the *freemaster\_cfg.h* configuration file to select whether the driver processes the communication automatically in the interrupt service routine handler or if it only polls the status of the module (typically during the application idle time).

This section describes each of the interrupt mode in more details.

### Completely Interrupt-Driven operation Activated using:

```
#define FMSTR_LONG_INTR 1
```

In this mode, both the communication and the FreeMASTER protocol decoding is done in the *FMSTR\_SerialIsr*, *FMSTR\_CanIsr*, or other interrupt service routine. Because the protocol execution may be a lengthy task (especially with the TSA-Safety enabled) it is recommended to use this mode only if the interrupt prioritization scheme is possible in the application and the FreeMASTER interrupt is assigned to a lower (the lowest) priority.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR\_SerialIsr* or *FMSTR\_CanIsr* functions from that handler.

### Mixed Interrupt and Polling Modes Activated using:

```
#define FMSTR_SHORT_INTR 1
```

In this mode, the communication processing time is split between the interrupt routine and the main application loop or task. The raw communication is handled by the *FMSTR\_SerialIsr*, *FMSTR\_CanIsr*, or other interrupt service routine, while the protocol decoding and execution is handled by the *FMSTR\_Poll* routine. Call *FMSTR\_Poll* during the idle time in the application main loop.

The interrupt processing in this mode is relatively fast and deterministic. Upon a serial-receive event, the received character is only placed into a FIFO-like queue and it is not further processed. Upon a CAN receive event, the received frame is stored into a receive buffer. When transmitting, the characters are fetched from the prepared transmit buffer.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR\_SerialIsr* or *FMSTR\_CanIsr* functions from that handler.

When the serial interface is used as the serial communication interface, ensure that the *FMSTR\_Poll* function is called at least once per *N* character time periods. *N* is the length of the FreeMASTER FIFO queue (*FMSTR\_COMM\_QUEUE\_SIZE*) and the character time is the time needed to transmit or receive a single byte over the SCI line.

### Completely Poll-driven

```
#define FMSTR_POLL_DRIVEN 1
```

In this mode, both the communication and the FreeMASTER protocol decoding are done in the *FMSTR\_Poll* routine. No interrupts are needed and the *FMSTR\_SerialIsr*, *FMSTR\_CanIsr*, and similar handlers compile to an empty code.

When using this mode, ensure that the *FMSTR\_Poll* function is called by the application at least once per the serial "character time" which is the time needed to transmit or receive a single character.

In the latter two modes (*FMSTR\_SHORT\_INTR* and *FMSTR\_POLL\_DRIVEN*), the protocol handling takes place in the *FMSTR\_Poll* routine. An application interrupt can occur in the middle of the

Read Memory or Write Memory commands' execution and corrupt the variable being accessed by the FreeMASTER driver. In these two modes, some issues or glitches may occur when using FreeMASTER to visualize or monitor volatile variables modified in interrupt servicing code.

The same issue may appear even in the full interrupt mode (FMSTR\_LONG\_INTR), if volatile variables are modified in the interrupt code with a priority higher than the priority of the communication interrupt.

**Data types** Simple portability was one of the main requirements when writing the FreeMASTER driver. This is why the driver code uses the privately-declared data types and the vast majority of the platform-dependent code is separated in the platform-dependent source files. The data types used in the driver API are all defined in the platform-specific header file.

To prevent name conflicts with the symbols used in the application, all data types, macros, and functions have the FMSTR\_ prefix. The only global variables used in the driver are the transport and low-level API structures exported from the driver-implementation layer to upper layers. Other than that, all private variables are declared as static and named using the fmstr\_ prefix.

**Communication interface initialization** The FreeMASTER driver does not perform neither the initialization nor the configuration of the peripheral module that it uses to communicate. It is the application startup code responsibility to configure the communication module before the FreeMASTER driver is initialized by the FMSTR\_Init call.

When the Serial communication module is used as the FreeMASTER communication interface, configure the UART receive and transmit pins, the serial communication baud rate, parity (no-parity), the character length (eight bits), and the number of stop bits (one) before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see [Driver interrupt modes](#)), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected serial peripheral module. Call the FMSTR\_SerialIsr function from the application handler.

When a CAN module is used as the FreeMASTER communication interface, configure the CAN receive and transmit pins and the CAN module bit rate before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see [Driver interrupt modes](#)), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected CAN peripheral module. Call the FMSTR\_CanIsr function from the application handler.

**Note:** It is not necessary to enable or unmask the serial nor the CAN interrupts before initializing the FreeMASTER driver. The driver enables or disables the interrupts and communication lines, as required during runtime.

**FreeMASTER Recorder calls** When using the FreeMASTER Recorder in the application (FMSTR\_USE\_RECORDER > 0), call the FMSTR\_RecorderCreate function early after FMSTR\_Init to set up each recorder instance to be used in the application. Then call the FMSTR\_Recorder function periodically in the code where the data recording should occur. A typical place to call the Recorder routine is at the timer or PWM interrupts, but it can be anywhere else. The example applications provided together with the driver code call the FMSTR\_Recorder in the main application loop.

In applications where FMSTR\_Recorder is called periodically with a constant period, specify the period in the Recorder configuration structure before calling FMSTR\_RecorderCreate. This setting enables the PC Host FreeMASTER tool to display the X-axis of the Recorder graph properly scaled for the time domain.

**Driver usage** Start using or evaluating FreeMASTER by opening some of the example applications available in the driver setup package.

Follow these steps to enable the basic FreeMASTER connectivity in the application:

- Make sure that all \*.c files of the FreeMASTER driver from the `src/common/platforms/[your_platform]` folder are a part of the project. See [Driver files](#) for more details.
- Configure the FreeMASTER driver by creating or editing the `freemaster_cfg.h` file and by saving it into the application project directory. See [Driver configuration](#) for more details.
- Include the `freemaster.h` file into any application source file that makes the FreeMASTER API calls.
- Initialize the Serial or CAN modules. Set the baud rate, parity, and other parameters of the communication. Do not enable the communication interrupts in the interrupt mask registers.
- For the FMSTR\_LONG\_INTR and FMSTR\_SHORT\_INTR modes, install the application-specific interrupt routine and call the FMSTR\_SerialIsr or FMSTR\_CanIsr functions from this handler.
- Call the FMSTR\_Init function early on in the application initialization code.
- Call the FMSTR\_RecorderCreate functions for each Recorder instance to enable the Recorder feature.
- In the main application loop, call the FMSTR\_Poll API function periodically when the application is idle.
- For the FMSTR\_SHORT\_INTR and FMSTR\_LONG\_INTR modes, enable the interrupts globally so that the interrupts can be handled by the CPU.

**Communication troubleshooting** The most common problem that causes communication issues is a wrong baud rate setting or a wrong pin multiplexer setting of the target MCU. When a communication between the PC Host running FreeMASTER and the target MCU cannot be established, try enabling the FMSTR\_DEBUG\_TX option in the `freemaster_cfg.h` file and call the FMSTR\_Poll function periodically in the main application task loop.

With this feature enabled, the FreeMASTER driver periodically transmits a test frame through the Serial or CAN lines. Use a logic analyzer or an oscilloscope to monitor the signals at the communication pins of the CPU device to examine whether the bit rate and signal polarity are configured properly.

## Driver API

This section describes the driver Application Programmers' Interface (API) needed to initialize and use the FreeMASTER serial communication driver.

**Control API** There are three key functions to initialize and use the driver.

### FMSTR\_Init

#### Prototype

```
FMSTR_BOOL FMSTR_Init(void);
```

- Declaration: `freemaster.h`
- Implementation: `freemaster_protocol.c`

**Description** This function initializes the internal variables of the FreeMASTER driver and enables the communication interface. This function does not change the configuration of the selected communication module. The hardware module must be initialized before the *FMSTR\_Init* function is called.

A call to this function must occur before calling any other FreeMASTER driver API functions.

## FMSTR\_Poll

### Prototype

```
void FMSTR_Poll(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_protocol.c*

**Description** In the poll-driven or short interrupt modes, this function handles the protocol decoding and execution (see *Driver interrupt modes*). In the poll-driven mode, this function also handles the communication interface with the PC. Typically, the *FMSTR\_Poll* function is called during the “idle” time in the main application task loop.

To prevent the receive data overflow (loss) on a serial interface, make sure that the *FMSTR\_Poll* function is called at least once per the time calculated as:

$$N * Tchar$$

where:

- *N* is equal to the length of the receive FIFO queue (configured by the *FMSTR\_COMM\_RQUEUE\_SIZE* macro). *N* is 1 for the poll-driven mode.
- *Tchar* is the character time, which is the time needed to transmit or receive a single byte over the SCI line.

**Note:** In the long interrupt mode, this function typically compiles as an empty function and can still be called. It is worthwhile to call this function regardless of the interrupt mode used in the application. This approach enables a convenient switching between the different interrupt modes only by changing the configuration macros in the *freemaster\_cfg.h* file.

## FMSTR\_SerialIsr / FMSTR\_CanIsr

### Prototype

```
void FMSTR_SerialIsr(void);
void FMSTR_CanIsr(void);
```

- Declaration: *freemaster.h*
- Implementation: *hw-specific low-level driver C file*

**Description** This function contains the interrupt-processing code of the FreeMASTER driver. In long or short interrupt modes (see *Driver interrupt modes*), this function must be called from the application interrupt service routine registered for the communication interrupt vector. On platforms where the communication module uses multiple interrupt vectors, the application should register a handler for all vectors and call this function at each interrupt.

**Note:** In a poll-driven mode, this function is compiled as an empty function and does not have to be used.

## Recorder API

### FMSTR\_RecorderCreate

#### Prototype

```
FMSTR_BOOL FMSTR_RecorderCreate(FMSTR_INDEX recIndex, FMSTR_REC_BUFF* buffCfg);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_rec.c*

**Description** This function registers a recorder instance and enables it to be used by the PC Host tool. Call this function for all recorder instances from 0 to the maximum number defined by the FMSTR\_USE\_RECORDER configuration option (minus one). An exception to this requirement is the recorder of instance 0 which may be automatically configured by FMSTR\_Init when the *freemaster\_cfg.h* configuration file defines the *FMSTR\_REC\_BUFF\_SIZE* and *FMSTR\_REC\_TIMEBASE* options.

For more information, see [Configurable items](#).

### FMSTR\_Recorder

#### Prototype

```
void FMSTR_Recorder(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_rec.c*

**Description** This function takes a sample of the variables being recorded using the FreeMASTER Recorder instance *recIndex*. If the selected Recorder is not active when the *FMSTR\_Recorder* function is being called, the function returns immediately. When the Recorder is active, the values of the variables being recorded are copied into the recorder buffer and the trigger conditions are evaluated.

If a trigger condition is satisfied, the Recorder enters the post-trigger mode, where it counts down the follow-up samples (number of *FMSTR\_Recorder* function calls) and de-activates the Recorder when the required post-trigger samples are finished.

The *FMSTR\_Recorder* function is typically called in the timer or PWM interrupt service routines. This function can also be called in the application main loop (for testing purposes).

### FMSTR\_RecorderTrigger

#### Prototype

```
void FMSTR_RecorderTrigger(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_rec.c*

**Description** This function forces the Recorder trigger condition to happen, which causes the Recorder to be automatically deactivated after the post-trigger samples are sampled. Use this function in the application code for programmatic control over the Recorder triggering. This can be useful when a more complex triggering conditions need to be used.

**Fast Recorder API** The Fast Recorder feature is not available in the FreeMASTER driver version 3. This feature was heavily dependent on the target platform and it was only available for the 56F8xxxx DSCs.

**TSA Tables** When the TSA is enabled in the FreeMASTER driver configuration file (by setting the FMSTR\_USE\_TSA macro to a non-zero value), it defines the so-called TSA tables in the application. This section describes the macros that must to be used to define the TSA tables.

There can be any number of TSA tables spread across the application source files. There must be always exactly one TSA Table List defined, which informs the FreeMASTER driver about the active TSA tables.

When there is at least one TSA table and one TSA Table List defined in the application, the TSA information automatically appears in the FreeMASTER symbols list. The symbols can then be used to create FreeMASTER variables for visualization or control.

**TSA table definition** The TSA table describes the static or global variables together with their address, size, type, and access-protection information. If the TSA-described variables are of a structure type, the TSA table may also describe this type and provide an access to the individual structure members of the variable.

The TSA table definition begins with the FMSTR\_TSA\_TABLE\_BEGIN macro with a *table\_id* identifying the table. The *table\_id* shall be a valid C-language symbol.

```
FMSTR_TSA_TABLE_BEGIN(table_id)
```

After this opening macro, the TSA descriptors are placed using these macros:

```
/* Adding variable descriptors */
FMSTR_TSA_RW_VAR(name, type) /* read/write variable entry */
FMSTR_TSA_RO_VAR(name, type) /* read-only variable entry */

/* Description of complex data types */
FMSTR_TSA_STRUCT(struct_name) /* structure or union type entry */
FMSTR_TSA_MEMBER(struct_name, member_name, type) /* structure member entry */

/* Memory blocks */
FMSTR_TSA_RW_MEM(name, type, address, size) /* read/write memory block */
FMSTR_TSA_RO_MEM(name, type, address, size) /* read-only memory block */
```

The table is closed using the FMSTR\_TSA\_TABLE\_END macro:

```
FMSTR_TSA_TABLE_END()
```

**TSA descriptor parameters** The TSA descriptor macros accept these parameters:

- *name* — variable name. The variable must be defined before the TSA descriptor references it.
- *type* — variable or member type. Only one of the pre-defined type constants may be used (see below).
- *struct\_name* — structure type name. The type must be defined (typedef) before the TSA descriptor references it.



- *member\_name* — structure member name.

**Note:** The structure member descriptors (FMSTR\_TSA\_MEMBER) must immediately follow the parent structure descriptor (FMSTR\_TSA\_STRUCT) in the table.

**Note:** To write-protect the variables in the FreeMASTER driver (FMSTR\_TSA\_RO\_VAR), enable the TSA-Safety feature in the configuration file.

**TSA variable types** The table lists *type* identifiers which can be used in TSA descriptors:

| Constant                           | Description                                                    |
|------------------------------------|----------------------------------------------------------------|
| FMSTR_TSA_UINTn                    | Unsigned integer type of size <i>n</i> bits (n=8,16,32,64)     |
| FMSTR_TSA_SINTn                    | Signed integer type of size <i>n</i> bits (n=8,16,32,64)       |
| FMSTR_TSA_FRACn                    | Fractional number of size <i>n</i> bits (n=16,32,64).          |
| FMSTR_TSA_FRAC_Q( <i>m,n</i> )     | Signed fractional number in general Q form (m+n+1 total bits)  |
| FMSTR_TSA_FRAC_UQ( <i>m,n</i> )    | Unsigned fractional number in general UQ form (m+n total bits) |
| FMSTR_TSA_FLOAT                    | 4-byte standard IEEE floating-point type                       |
| FMSTR_TSA_DOUBLE                   | 8-byte standard IEEE floating-point type                       |
| FMSTR_TSA_POINTER                  | Generic pointer type defined (platform-specific 16 or 32 bit)  |
| FM-STR_TSA_USERTYPE( <i>name</i> ) | Structure or union type declared with FMSTR_TSA_STRUCT record  |

**TSA table list** There shall be exactly one TSA Table List in the application. The list contains one entry for each TSA table defined anywhere in the application.

The TSA Table List begins with the FMSTR\_TSA\_TABLE\_LIST\_BEGIN macro and continues with the TSA table entries for each table.

```
FMSTR_TSA_TABLE_LIST_BEGIN()
```

```
FMSTR_TSA_TABLE(table_id)
FMSTR_TSA_TABLE(table_id2)
FMSTR_TSA_TABLE(table_id3)
...
```

The list is closed with the FMSTR\_TSA\_TABLE\_LIST\_END macro:

```
FMSTR_TSA_TABLE_LIST_END()
```

**TSA Active Content entries** FreeMASTER v2.0 and higher supports TSA Active Content, enabling the TSA tables to describe the memory-mapped files, virtual directories, and URL hyperlinks. FreeMASTER can access such objects similarly to accessing the files and folders on the local hard drive.

With this set of TSA entries, the FreeMASTER pages can be embedded directly into the target MCU flash and accessed by FreeMASTER directly over the communication line. The HTML-coded pages rendered inside the FreeMASTER window can access the TSA Active Content resources using a special URL referencing the *fmrstr:* protocol.

This example provides an overview of the supported TSA Active Content entries:

```
FMSTR_TSA_TABLE_BEGIN(files_and_links)
```

```
/* Directory entry applies to all subsequent MEMFILE entries */
```

```
FMSTR_TSA_DIRECTORY("/text_files") /* entering a new virtual directory */
```

(continues on next page)



(continued from previous page)

```

/* The readme.txt file will be accessible at the fmstr://text_files/readme.txt URL */
FMSTR_TSA_MEMFILE("readme.txt", readme_txt, sizeof(readme_txt)) /* memory-mapped file */

/* Files can also be specified with a full path so the DIRECTORY entry does not apply */
FMSTR_TSA_MEMFILE("/index.htm", index, sizeof(index)) /* memory-mapped file */
FMSTR_TSA_MEMFILE("/prj/demo.pmp", demo_pmp, sizeof(demo_pmp)) /* memory-mapped file */

/* Hyperlinks can point to a local MEMFILE object or to the Internet */
FMSTR_TSA_HREF("Board's Built-in Welcome Page", "/index.htm")
FMSTR_TSA_HREF("FreeMASTER Home Page", "http://www.nxp.com/freemaster")

/* Project file links simplify opening the projects from any URLs */
FMSTR_TSA_PROJECT("Demonstration Project (embedded)", "/prj/demo.pmp")
FMSTR_TSA_PROJECT("Full Project (online)", "http://mycompany.com/prj/demo.pmp")

FMSTR_TSA_TABLE_END()

```

## TSA API

### FMSTR\_SetUpTsaBuff

#### Prototype

```
FMSTR_BOOL FMSTR_SetUpTsaBuff(FMSTR_ADDR buffAddr, FMSTR_SIZE buffSize);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_tsa.c*

#### Arguments

- *buffAddr* [in] - address of the memory buffer for the dynamic TSA table
- *buffSize* [in] - size of the memory buffer which determines the maximum number of TSA entries to be added in the runtime

**Description** This function must be used to assign the RAM memory buffer to the TSA subsystem when FMSTR\_USE\_TSA\_DYNAMIC is enabled. The memory buffer is then used to store the TSA entries added dynamically to the runtime TSA table using the FMSTR\_TsaAddVar function call. The runtime TSA table is processed by the FreeMASTER PC Host tool along with all static tables as soon as the communication port is open.

The size of the memory buffer determines the number of TSA entries that can be added dynamically. Depending on the MCU platform, one TSA entry takes either 8 or 16 bytes.

### FMSTR\_TsaAddVar

#### Prototype

```

FMSTR_BOOL FMSTR_TsaAddVar(FMSTR_TSATBL_STRPTR tsaName, FMSTR_TSATBL_STRPTR
↪ tsaType,
    FMSTR_TSATBL_VOIDPTR varAddr, FMSTR_SIZE32 varSize,
    FMSTR_SIZE flags);

```

- Declaration: *freemaster.h*

- Implementation: *freemaster\_tsa.c*

### Arguments

- *tsaName* [in] - name of the object
- *tsaType* [in] - name of the object type
- *varAddr* [in] - address of the object
- *varSize* [in] - size of the object
- *flags* [in] - access flags; a combination of these values:
  - *FMSTR\_TSA\_INFO\_RO\_VAR* — read-only memory-mapped object (typically a variable)
  - *FMSTR\_TSA\_INFO\_RW\_VAR* — read/write memory-mapped object
  - *FMSTR\_TSA\_INFO\_NON\_VAR* — other entry, describing structure types, structure members, enumerations, and other types

**Description** This function can be called only when the dynamic TSA table is enabled by the *FMSTR\_USE\_TSA\_DYNAMIC* configuration option and when the *FMSTR\_SetUpTsaBuff* function call is made to assign the dynamic TSA table memory. This function adds an entry into the dynamic TSA table. It can be used to register a read-only or read/write memory object or describe an item of the user-defined type.

See [TSA table definition](#) for more details about the TSA table entries.

## Application Commands API

### FMSTR\_GetAppCmd

#### Prototype

```
FMSTR_APPCMD_CODE FMSTR_GetAppCmd(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*

**Description** This function can be used to detect if there is an Application Command waiting to be processed by the application. If no command is pending, this function returns the *FMSTR\_APPCMDRESULT\_NOCMD* constant. Otherwise, this function returns the code of the Application Command that must be processed. Use the *FMSTR\_AppCmdAck* call to acknowledge the Application Command after it is processed and to return the appropriate result code to the host.

The *FMSTR\_GetAppCmd* function does not report the commands for which a callback handler function exists. If the *FMSTR\_GetAppCmd* function is called when a callback-registered command is pending (and before it is actually processed by the callback function), this function returns *FMSTR\_APPCMDRESULT\_NOCMD*.

### FMSTR\_GetAppCmdData

### Prototype

```
FMSTR_APPCMD_PDATA FMSTR_GetAppCmdData(FMSTR_SIZE* dataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*

### Arguments

- *dataLen* [out] - pointer to the variable that receives the length of the data available in the buffer. It can be NULL when this information is not needed.

**Description** This function can be used to retrieve the Application Command data when the application determines that an Application Command is pending (see [FMSTR\\_GetAppCmd](#)).

There is just a single buffer to hold the Application Command data (the buffer length is FMSTR\_APPCMD\_BUFF\_SIZE bytes). If the data are to be used in the application after the command is processed by the FMSTR\_AppCmdAck call, copy the data out to a private buffer.

### FMSTR\_AppCmdAck

#### Prototype

```
void FMSTR_AppCmdAck(FMSTR_APPCMD_RESULT resultCode);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*

### Arguments

- *resultCode* [in] - the result code which is to be returned to FreeMASTER

**Description** This function is used when the Application Command processing finishes in the application. The resultCode passed to this function is returned back to the host and the driver is re-initialized to expect the next Application Command.

After this function is called and before the next Application Command arrives, the return value of the FMSTR\_GetAppCmd function is FMSTR\_APPCMDRESULT\_NOCMD.

### FMSTR\_AppCmdSetResponseData

#### Prototype

```
void FMSTR_AppCmdSetResponseData(FMSTR_ADDR resultDataAddr, FMSTR_SIZE resultDataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*

### Arguments

- *resultDataAddr* [in] - pointer to the data buffer that is to be copied to the Application Command data buffer
- *resultDataLen* [in] - length of the data to be copied. It must not exceed the FMSTR\_APPCMD\_BUFF\_SIZE value.

**Description** This function can be used before the Application Command processing finishes, when there are data to be returned back to the PC.

The response data buffer is copied into the Application Command data buffer, from where it is accessed when the host requires it. Do not use FMSTR\_GetAppCmdData and the data buffer after FMSTR\_AppCmdSetResponseData is called.

**Note:** The current version of FreeMASTER does not support the Application Command response data.

### FMSTR\_RegisterAppCmdCall

#### Prototype

```
FMSTR_BOOL FMSTR_RegisterAppCmdCall(FMSTR_APPCMD_CODE appCmdCode, FMSTR_
↪PAPPCMDFUNC callbackFunc);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*

### Arguments

- *appCmdCode* [in] - the Application Command code for which the callback is to be registered
- *callbackFunc* [in] - pointer to the callback function that is to be registered. Use NULL to unregister a callback registered previously with this Application Command.

**Return value** This function returns a non-zero value when the callback function was successfully registered or unregistered. It can return zero when trying to register a callback function for more than FMSTR\_MAX\_APPCMD\_CALLS different Application Commands.

**Description** This function can be used to register the given function as a callback handler for the Application Command. The Application Command is identified using single-byte code. The callback function is invoked automatically by the FreeMASTER driver when the protocol decoder obtains a request to get the application command result code.

The prototype of the callback function is

```
FMSTR_APPCMD_RESULT HandlerFunction(FMSTR_APPCMD_CODE nAppcmd,
FMSTR_APPCMD_PDATA pData, FMSTR_SIZE nDataLen);
```

Where:

- *nAppcmd* -Application Command code
- *pData* —points to the Application Command data received (if any)
- *nDataLen* —information about the Application Command data length

The return value of the callback function is used as the Application Command Result Code and returned to FreeMASTER.

**Note:** The FMSTR\_MAX\_APPCMD\_CALLS configuration macro defines how many different Application Commands may be handled by a callback function. When FMSTR\_MAX\_APPCMD\_CALLS is undefined or defined as zero, the FMSTR\_RegisterAppCmdCall function always fails.

## Pipes API

### FMSTR\_PipeOpen

#### Prototype

```
FMSTR_HPIPE FMSTR_PipeOpen(FMSTR_PIPE_PORT pipePort, FMSTR_PPIPEFUNC pipeCallback,
    FMSTR_ADDR pipeRxBuff, FMSTR_PIPE_SIZE pipeRxSize,
    FMSTR_ADDR pipeTxBuff, FMSTR_PIPE_SIZE pipeTxSize,
    FMSTR_U8 type, const FMSTR_CHAR *name);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_pipes.c*

#### Arguments

- *pipePort* [in] - port number that identifies the pipe for the client
- *pipeCallback* [in] - pointer to the callback function that is called whenever a pipe data status changes
- *pipeRxBuff* [in] - address of the receive memory buffer
- *pipeRxSize* [in] - size of the receive memory buffer
- *pipeTxBuff* [in] - address of the transmit memory buffer
- *pipeTxSize* [in] - size of the transmit memory buffer
- *type* [in] - a combination of FMSTR\_PIPE\_MODE\_XXX and FMSTR\_PIPE\_SIZE\_XXX constants describing primary pipe data format and usage. This type helps FreeMASTER decide how to access the pipe by default. Optional, use 0 when undetermined.
- *name* [in] - user name of the pipe port. This name is visible to the FreeMASTER user when creating the graphical pipe interface.

**Description** This function initializes a new pipe and makes it ready to accept or send the data to the PC Host client. The receive memory buffer is used to store the received data before they are read out by the FMSTR\_PipeRead call. When this buffer gets full, the PC Host client denies the data transmission into this pipe until there is enough free space again. The transmit memory buffer is used to store the data transmitted by the application to the PC Host client using the FMSTR\_PipeWrite call. The transmit buffer can get full when the PC Host is disconnected or when it is slow in receiving and reading out the pipe data.

The function returns the pipe handle which must be stored and used in the subsequent calls to manage the pipe object.

The callback function (if specified) is called whenever new data are received through the pipe and available for reading. This callback is also called when the data waiting in the transmit buffer are successfully pushed to the PC Host and the transmit buffer free space increases. The prototype of the callback function provided by the user application must be as follows. The *PipeHandler* name is only a placeholder and must be defined by the application.

```
void PipeHandler(FMSTR_HPIPE pipeHandle);
```

## FMSTR\_PipeClose

### Prototype

```
void FMSTR_PipeClose(FMSTR_HPIPE pipeHandle);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_pipes.c*

### Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR\_PipeOpen function call

**Description** This function de-initializes the pipe object. No data can be received or sent on the pipe after this call.

## FMSTR\_PipeWrite

### Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeWrite(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,  
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE writeGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_pipes.c*

### Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR\_PipeOpen function call
- *pipeData* [in] - address of the data to be written
- *pipeDataLen* [in] - length of the data to be written
- *writeGranularity* [in] - size of the minimum unit of data which is to be written

**Description** This function puts the user-specified data into the pipe's transmit memory buffer and schedules it for transmission. This function returns the number of bytes that were successfully written into the buffer. This number may be smaller than the number of the requested bytes if there is not enough free space in the transmit buffer.

The *writeGranularity* argument can be used to split the data into smaller chunks, each of the size given by the *writeGranularity* value. The FMSTR\_PipeWrite function writes as many data chunks as possible into the transmit buffer and does not attempt to write an incomplete chunk. This feature can prove to be useful to avoid the intermediate caching when writing an array of integer values or other multi-byte data items. When making the *nGranularity* value equal to the *nLength* value, all data are considered as one chunk which is either written successfully as a whole or not at all. The *nGranularity* value of 0 or 1 disables the data-chunk approach.

## FMSTR\_PipeRead

## Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeRead(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE readGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_pipes.c*

## Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR\_PipeOpen function call
- *pipeData* [in] - address of the data buffer to be filled with the received data
- *pipeDataLen* [in] - length of the data to be read
- *readGranularity* [in] - size of the minimum unit of data which is to be read

**Description** This function copies the data received from the pipe from its receive buffer to the user buffer for further processing. The function returns the number of bytes that were successfully copied to the buffer. This number may be smaller than the number of the requested bytes if there is not enough data bytes available in the receive buffer.

The readGranularity argument can be used to copy the data in larger chunks in the same way as described in the FMSTR\_PipeWrite function.

**API data types** This section describes the data types used in the FreeMASTER driver. The information provided here can be useful when modifying or porting the FreeMASTER Communication Driver to new NXP platforms.

**Note:** The licensing conditions prohibit use of FreeMASTER and the FreeMASTER Communication Driver with non-NXP MPU or MCU products.

**Public common types** The table below describes the public data types used in the FreeMASTER driver API calls. The data types are declared in the *freemaster.h* header file.

| Type name                                                                                                                                                                                                                 | Description                                                                                                                         |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>FM-STR_ADDR</i><br>For example, this type is defined as long integer on the 56F8xxx platform where the 24-bit addresses must be supported, but the C-pointer may be only 16 bits wide in some compiler configurations. | Data type used to hold the memory address. On most platforms, this is normally a C-pointer, but it may also be a pure integer type. |
| <i>FM-STR_SIZE</i><br>It is required that this type is unsigned and at least 16 bits wide integer.                                                                                                                        | Data type used to hold the memory block size.                                                                                       |
| <i>FM-STR_BOOL</i><br>This type is used only in zero/non-zero conditions in the driver code.                                                                                                                              | Data type used as a general boolean type.                                                                                           |
| <i>FM-STR_APPCM</i><br>Generally, this is an unsigned 8-bit value.                                                                                                                                                        | Data type used to hold the Application Command code.                                                                                |
| <i>FM-STR_APPCM</i><br>Generally, this is an unsigned 8-bit value.                                                                                                                                                        | Data type used to create the Application Command data buffer.                                                                       |
| <i>FM-STR_APPCM</i><br>Generally, this is an unsigned 8-bit value.                                                                                                                                                        | Data type used to hold the Application Command result code.                                                                         |



**Public TSA types** The table describes the TSA-specific public data types. These types are declared in the *freemaster\_tsa.h* header file, which is included in the user application indirectly by the *freemaster.h* file.

|                       |                                                                                                                                                               |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>FM-STR_TSA_TII</i> | Data type used to hold a descriptor index in the TSA table or a table index in the list of TSA tables.<br>By default, this is defined as <i>FM-STR_SIZE</i> . |
| <i>FM-STR_TSA_TS</i>  | Data type used to hold a memory block size, as used in the TSA descriptors.<br>By default, this is defined as <i>FM-STR_SIZE</i> .                            |

**Public Pipes types** The table describes the data types used by the FreeMASTER Pipes API:

|                       |                                                                                                                |
|-----------------------|----------------------------------------------------------------------------------------------------------------|
| <i>FM-STR_HPIPE</i>   | Pipe handle that identifies the open-pipe object.<br>Generally, this is a pointer to a void type.              |
| <i>FM-STR_PIPE_PC</i> | Integer type required to hold at least 7 bits of data.<br>Generally, this is an unsigned 8-bit or 16-bit type. |
| <i>FM-STR_PIPE_SI</i> | Integer type required to hold at least 16 bits of data.<br>This is used to store the data buffer sizes.        |
| <i>FM-STR_PPIPEF</i>  | Pointer to the pipe handler function.<br>See <a href="#">FM-STR_PipeOpen</a> for more details.                 |

**Internal types** The table describes the data types used internally by the FreeMASTER driver. The data types are declared in the platform-specific header file and they are not available in the application code.

|                                                                                                          |                                                                         |
|----------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|
| <i>FMSTR_U8</i>                                                                                          | The smallest memory entity.                                             |
| On the vast majority of platforms, this is an unsigned 8-bit integer.                                    |                                                                         |
| On the 56F8xx DSP platform, this is defined as an unsigned 16-bit integer.                               |                                                                         |
| <i>FMSTR_U16</i>                                                                                         | Unsigned 16-bit integer.                                                |
| <i>FMSTR_U32</i>                                                                                         | Unsigned 32-bit integer.                                                |
| <i>FMSTR_S8</i>                                                                                          | Signed 8-bit integer.                                                   |
| <i>FMSTR_S16</i>                                                                                         | Signed 16-bit integer.                                                  |
| <i>FMSTR_S32</i>                                                                                         | Signed 32-bit integer.                                                  |
| <i>FMSTR_FLOAT</i>                                                                                       | 4-byte standard IEEE floating-point type.                               |
| <i>FMSTR_FLAGS</i>                                                                                       | Data type forming a union with a structure of flag bit-fields.          |
| <i>FMSTR_SIZE8</i>                                                                                       | Data type holding a general size value, at least 8 bits wide.           |
| <i>FMSTR_INDEX</i>                                                                                       | General for-loop index. Must be signed, at least 16 bits wide.          |
| <i>FMSTR_BCHR</i>                                                                                        | A single character in the communication buffer.                         |
| Typically, this is an 8-bit unsigned integer, except for the DSP platforms where it is a 16-bit integer. |                                                                         |
| <i>FMSTR_BPTR</i>                                                                                        | A pointer to the communication buffer (an array of <i>FMSTR_BCHR</i> ). |

## Document references

### Links

- This document online: <https://mcuxpresso.nxp.com/mcuxsdk/latest/html/middleware/freemaster/doc/index.html>

- FreeMASTER tool home: [www.nxp.com/freemaster](http://www.nxp.com/freemaster)
- FreeMASTER community area: [community.nxp.com/community/freemaster](http://community.nxp.com/community/freemaster)
- FreeMASTER GitHub code repo: <https://github.com/nxp-mcuxpresso/mcux-freemaster>
- MCUXpresso SDK home: [www.nxp.com/mcuxpresso](http://www.nxp.com/mcuxpresso)
- MCUXpresso SDK builder: [mcuxpresso.nxp.com/en](http://mcuxpresso.nxp.com/en)

## Documents

- *FreeMASTER Usage Serial Driver Implementation* (document [AN4752](#))
- *Integrating FreeMASTER Time Debugging Tool With CodeWarrior For Microcontrollers v10.X Project* (document [AN4771](#))
- *Flash Driver Library For MC56F847xx And MC56F827xx DSC Family* (document [AN4860](#))

**Revision history** This Table summarizes the changes done to this document since the initial release.

| Revision | Date    | Description                                                                                                                                                                                                                                                                                                          |
|----------|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.0      | 03/2006 | Limited initial release                                                                                                                                                                                                                                                                                              |
| 2.0      | 09/2007 | Updated for FreeMASTER version. New Freescale document template used.                                                                                                                                                                                                                                                |
| 2.1      | 12/2007 | Added description of the new Fast Recorder feature and its API.                                                                                                                                                                                                                                                      |
| 2.2      | 04/2010 | Added support for MPC56xx platform, Added new API for use CAN interface.                                                                                                                                                                                                                                             |
| 2.3      | 04/2011 | Added support for Kxx Kinetis platform and MQX operating system.                                                                                                                                                                                                                                                     |
| 2.4      | 06/2011 | Serial driver update, adds support for USB CDC interface.                                                                                                                                                                                                                                                            |
| 2.5      | 08/2011 | Added Packet Driven BDM interface.                                                                                                                                                                                                                                                                                   |
| 2.7      | 12/2013 | Added FLEXCAN32 interface, byte access and isr callback configuration option.                                                                                                                                                                                                                                        |
| 2.8      | 06/2014 | Removed obsolete license text, see the software package content for up-to-date license.                                                                                                                                                                                                                              |
| 2.9      | 03/2015 | Update for driver version 1.8.2 and 1.9: FreeMASTER Pipes, TSA Active Content, LIN Transport Layer support, DEBUG-TX communication troubleshooting, Kinetis SDK support.                                                                                                                                             |
| 3.0      | 08/2016 | Update for driver version 2.0: Added support for MPC56xx, MPC57xx, KEAxx and S32Kxx platforms. New NXP document template as well as new license agreement used. added MCAN interface. Folders structure at the installation destination was rearranged.                                                              |
| 4.0      | 04/2019 | Update for driver released as part of FreeMASTER v3.0 and MCUXpresso SDK 2.6. Updated to match new V4 serial communication protocol and new configuration options. This version of the document removes substantial portion of outdated information related to S08, S12, ColdFire, Power and other legacy platforms. |
| 4.1      | 04/2020 | Minor update for FreeMASTER driver included in MCUXpresso SDK 2.8.                                                                                                                                                                                                                                                   |
| 4.2      | 09/2020 | Added example applications description and information about the MCUXpresso Config Tools. Fixed the pipe-related API description.                                                                                                                                                                                    |
| 4.3      | 10/2024 | Added description of Network and Segger J-Link RTT interface configuration. Accompanying the MCUXpresso SDK version 24.12.00.                                                                                                                                                                                        |
| 4.4      | 04/2025 | Added Zephyr-specific information. Accompanying the MCUXpresso SDK version 25.06.00.                                                                                                                                                                                                                                 |

# Chapter 4

## RTOS

### 4.1 FreeRTOS

#### 4.1.1 FreeRTOS kernel

Open source RTOS kernel for small devices.

[FreeRTOS kernel for MCUXpresso SDK Readme](#)

[FreeRTOS kernel for MCUXpresso SDK ChangeLog](#)

[FreeRTOS kernel Readme](#)

#### 4.1.2 FreeRTOS drivers

This is set of NXP provided FreeRTOS reentrant bus drivers.

#### 4.1.3 backoffalgorithm

Algorithm for calculating exponential backoff with jitter for network retry attempts.

[Readme](#)

#### 4.1.4 corehttp

C language HTTP client library designed for embedded platforms.

#### 4.1.5 corejson

JSON parser.

## **Readme**

### **4.1.6 coremqtt**

MQTT publish/subscribe messaging library.

### **4.1.7 corepkcs11**

PKCS #11 key management library.

## **Readme**

### **4.1.8 freertos-plus-tcp**

Open source RTOS FreeRTOS Plus TCP.

## **Readme**