



# MCUXpresso SDK Documentation

Release 25.12.00



NXP  
Dec 18, 2025



# Table of contents

|          |                                                              |            |
|----------|--------------------------------------------------------------|------------|
| <b>1</b> | <b>MCIMX93-EVK</b>                                           | <b>3</b>   |
| 1.1      | Overview                                                     | 3          |
| 1.2      | Getting Started with MCUXpresso SDK Package                  | 3          |
| 1.2.1    | MCUXSDKMCIMX93EVKGSUG                                        | 3          |
| 1.3      | Getting Started with MCUXpresso SDK GitHub                   | 17         |
| 1.3.1    | Getting Started with MCUXpresso SDK Repository               | 17         |
| 1.4      | Release Notes                                                | 41         |
| 1.4.1    | MCUXpresso SDK Release Notes                                 | 41         |
| 1.5      | ChangeLog                                                    | 46         |
| 1.5.1    | MCUXpresso SDK Changelog                                     | 46         |
| 1.6      | Driver API Reference Manual                                  | 146        |
| 1.7      | Middleware Documentation                                     | 146        |
| 1.7.1    | Multicore                                                    | 146        |
| 1.7.2    | FreeMASTER                                                   | 146        |
| 1.7.3    | FreeRTOS                                                     | 146        |
| 1.7.4    | lwIP                                                         | 147        |
| <b>2</b> | <b>MIMX9352</b>                                              | <b>149</b> |
| 2.1      | eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver | 149        |
| 2.2      | eDMA core Driver                                             | 181        |
| 2.3      | eDMA soc Driver                                              | 188        |
| 2.4      | ENET: Ethernet MAC Driver                                    | 188        |
| 2.5      | EQOS-TSN: Ethernet QoS with TSN Driver                       | 219        |
| 2.6      | Enet_qos_qos                                                 | 219        |
| 2.7      | FGPIO Driver                                                 | 260        |
| 2.8      | FlexCAN: Flex Controller Area Network Driver                 | 260        |
| 2.9      | FlexCAN Driver                                               | 260        |
| 2.10     | FlexCAN eDMA Driver                                          | 308        |
| 2.11     | FlexIO: FlexIO Driver                                        | 310        |
| 2.12     | FlexIO Driver                                                | 310        |
| 2.13     | FlexIO eDMA I2S Driver                                       | 328        |
| 2.14     | FlexIO eDMA SPI Driver                                       | 331        |
| 2.15     | FlexIO eDMA UART Driver                                      | 334        |
| 2.16     | FlexIO I2C Master Driver                                     | 337        |
| 2.17     | FlexIO I2S Driver                                            | 346        |
| 2.18     | FlexIO SPI Driver                                            | 356        |
| 2.19     | FlexIO UART Driver                                           | 369        |
| 2.20     | FLEXSPI: Flexible Serial Peripheral Interface Driver         | 380        |
| 2.21     | FLEXSPI eDMA Driver                                          | 396        |
| 2.22     | I3C: I3C Driver                                              | 399        |
| 2.23     | I3C Common Driver                                            | 401        |
| 2.24     | I3C Master Driver                                            | 403        |
| 2.25     | I3C Master DMA Driver                                        | 429        |
| 2.26     | I3C Slave Driver                                             | 432        |
| 2.27     | I3C Slave DMA Driver                                         | 445        |
| 2.28     | Iomuxc_driver                                                | 447        |

|          |                                                                      |            |
|----------|----------------------------------------------------------------------|------------|
| 2.29     | Common Driver                                                        | 467        |
| 2.30     | LPI2C: Low Power Inter-Integrated Circuit Driver                     | 480        |
| 2.31     | LPI2C Master Driver                                                  | 481        |
| 2.32     | LPI2C Master DMA Driver                                              | 496        |
| 2.33     | LPI2C Slave Driver                                                   | 498        |
| 2.34     | LPIT: Low-Power Interrupt Timer                                      | 508        |
| 2.35     | LPSPi: Low Power Serial Peripheral Interface                         | 515        |
| 2.36     | LPSPi Peripheral driver                                              | 515        |
| 2.37     | LPSPi eDMA Driver                                                    | 537        |
| 2.38     | LPTMR: Low-Power Timer                                               | 544        |
| 2.39     | LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver | 549        |
| 2.40     | LPUART Driver                                                        | 549        |
| 2.41     | LPUART eDMA Driver                                                   | 568        |
| 2.42     | MCM: Miscellaneous Control Module                                    | 571        |
| 2.43     | Mipi_dsi                                                             | 576        |
| 2.44     | MIPI_DSI: MIPI DSI Host Controller                                   | 592        |
| 2.45     | MU: Messaging Unit                                                   | 592        |
| 2.46     | OTFAD: On The Fly AES-128 Decryption Driver                          | 605        |
| 2.47     | PDM: Microphone Interface                                            | 608        |
| 2.48     | PDM Driver                                                           | 608        |
| 2.49     | PDM EDMA Driver                                                      | 620        |
| 2.50     | RGPIO: Rapid General-Purpose Input/Output Driver                     | 624        |
| 2.51     | RGPIO Driver                                                         | 626        |
| 2.52     | SAI: Serial Audio Interface                                          | 630        |
| 2.53     | SAI Driver                                                           | 630        |
| 2.54     | SAI EDMA Driver                                                      | 656        |
| 2.55     | SAR_ADC: SAR_ADC Module                                              | 663        |
| 2.56     | SEMA42: Hardware Semaphores Driver                                   | 691        |
| 2.57     | TMU: Thermal Management Unit Driver                                  | 694        |
| 2.58     | TPM: Timer PWM Module                                                | 700        |
| 2.59     | TRDC: Trusted Resource Domain Controller                             | 716        |
| 2.60     | Trdc_core                                                            | 738        |
| 2.61     | Trdc_soc                                                             | 752        |
| 2.62     | TSTMR: Timestamp Timer Driver                                        | 755        |
| 2.63     | WDOG32: 32-bit Watchdog Timer                                        | 756        |
| 2.64     | CACHE: CACHE Memory Controller                                       | 762        |
| <b>3</b> | <b>Middleware</b>                                                    | <b>767</b> |
| 3.1      | Connectivity                                                         | 767        |
| 3.1.1    | lwIP                                                                 | 767        |
| 3.2      | Motor Control                                                        | 768        |
| 3.2.1    | FreeMASTER                                                           | 768        |
| 3.3      | MultiCore                                                            | 805        |
| 3.3.1    | Multicore SDK                                                        | 805        |
| <b>4</b> | <b>RTOS</b>                                                          | <b>905</b> |
| 4.1      | FreeRTOS                                                             | 905        |
| 4.1.1    | FreeRTOS kernel                                                      | 905        |
| 4.1.2    | FreeRTOS drivers                                                     | 905        |
| 4.1.3    | backoffalgorithm                                                     | 905        |
| 4.1.4    | corehttp                                                             | 905        |
| 4.1.5    | corejson                                                             | 905        |
| 4.1.6    | coremqtt                                                             | 906        |
| 4.1.7    | corepkcs11                                                           | 906        |
| 4.1.8    | freertos-plus-tcp                                                    | 906        |

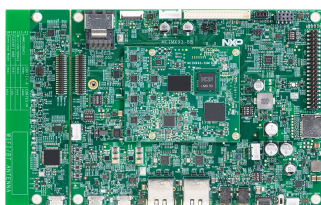
This documentation contains information specific to the mcimx93evk board.



# Chapter 1

## MCIMX93-EVK

### 1.1 Overview



MCU device and part on board is shown below:

- Device: MIMX9352
- PartNumber: MIMX9352DVVXM

### 1.2 Getting Started with MCUXpresso SDK Package

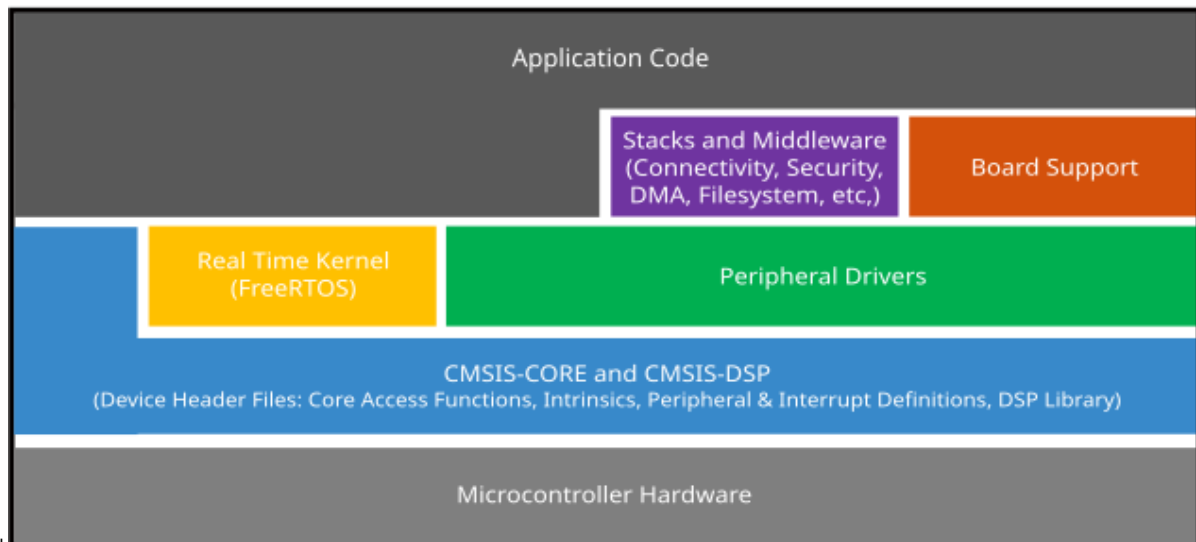
#### 1.2.1 MCUXSDKMCIMX93EVKGSUG

##### 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 for MCIMX93-EVK (document MCUXSDKMCIMX93EVKRN)*.

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):

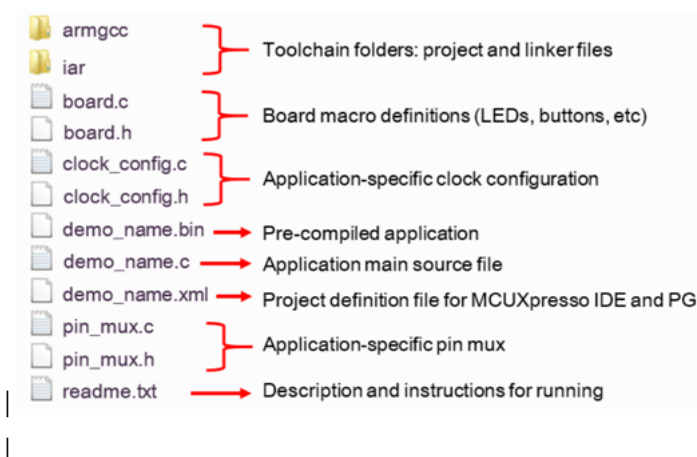
- `cmsis_driver_examples`: Simple applications intended to concisely illustrate how to use CMSIS drivers.
- `demo_apps`: Full-featured applications intended to highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks 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 93 includes the build system to be used with some toolchains. In this chapter, the toolchain support is presented and detailed.

**Compiler/Debugger** The MCUXpresso SDK i.MX 93 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 93, the patch for IAR and Segger J-Link should be installed. The patch named `iar_segger_support_patch_imx93.zip` can be used with MCUXpresso SDK. Contact the NXP representative for the zip package. For additional information about patch installation, see the *readme.txt* in the patch.

**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 i.MX 93 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** Perform 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
```

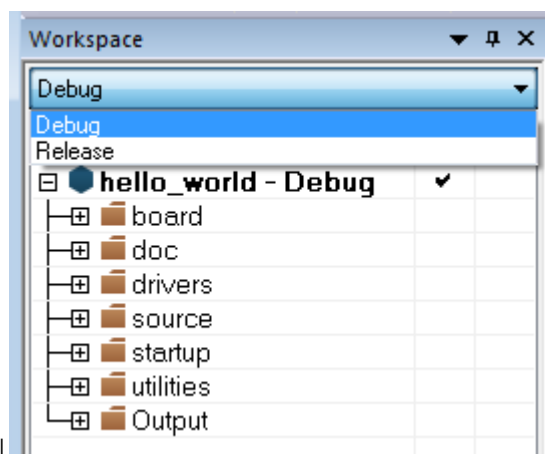
Using the i.MX 93 EVK hardware platform as an example, the `hello_world` workspace is located in:

```
<install_dir>/boards/mcimx93evk/demo_apps/hello_world/iar/hello_world.eww
```

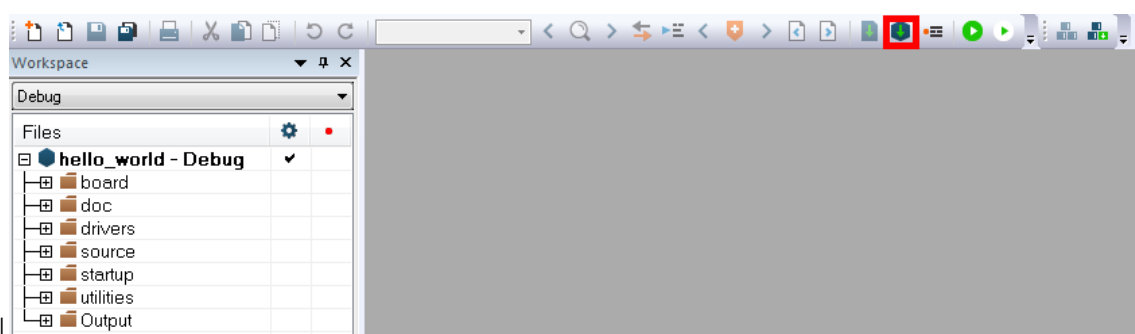
Other example applications may have additional folders in their path.

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.

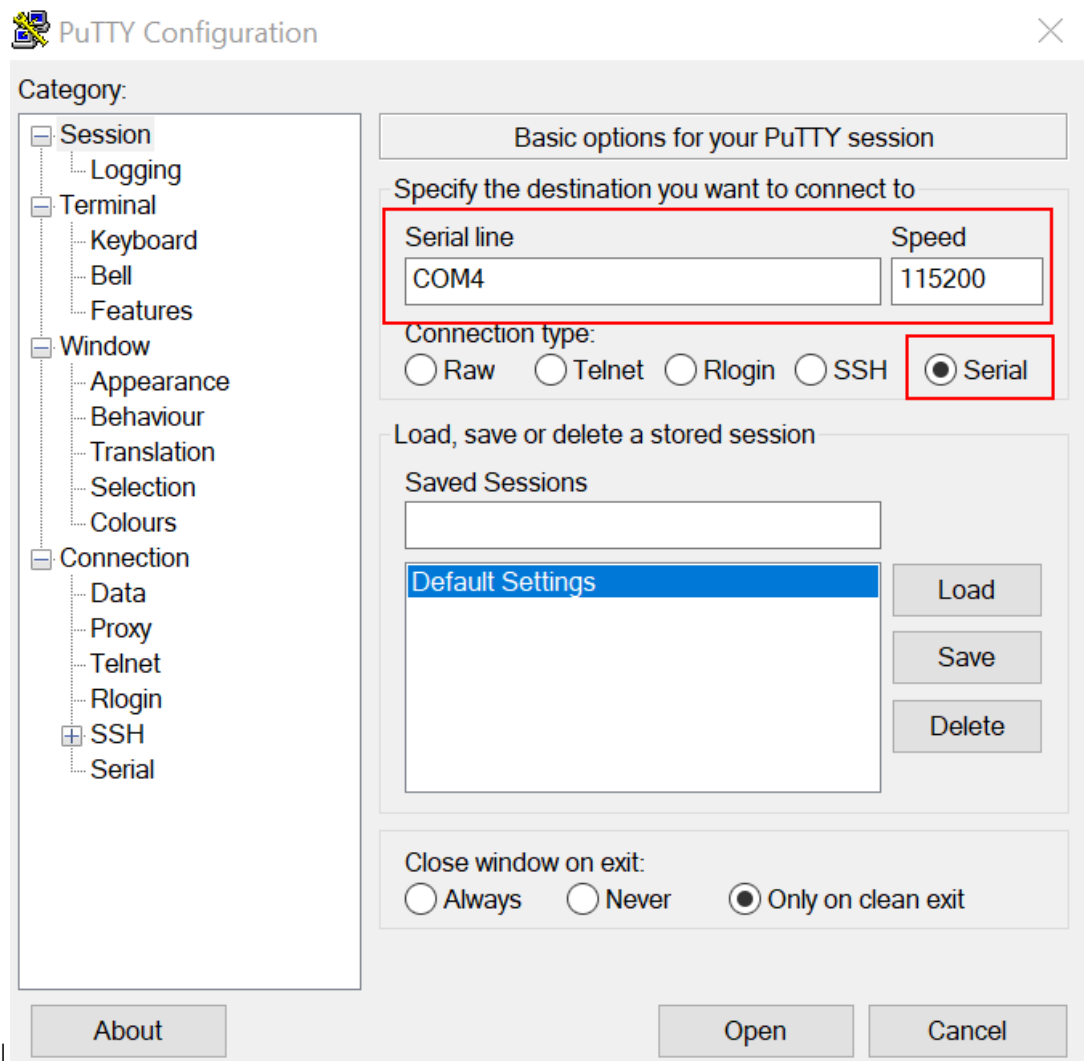


4. The build completes without errors.

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

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

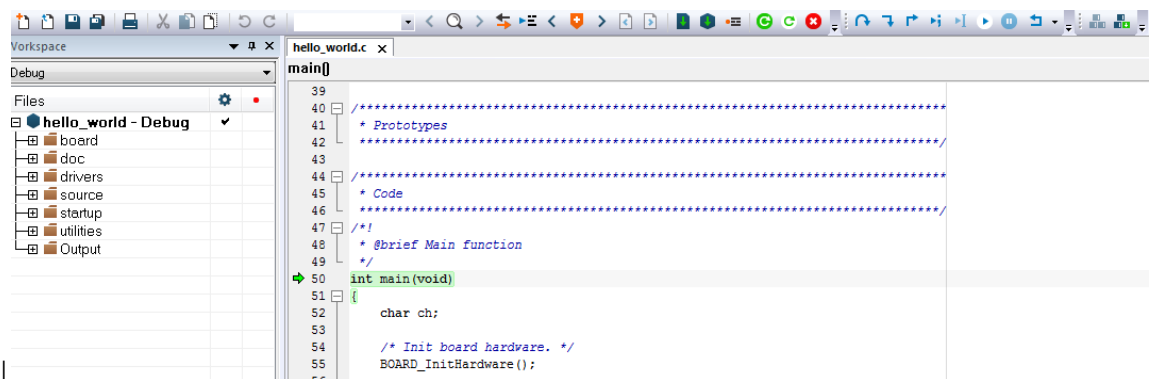
1. This board supports the J-Link PLUS debug probe. Before using it, install SEGGER J-Link software, which can be downloaded from <http://www.segger.com/downloads/jlink/>.
2. Connect the development platform to your PC via USB cable between the DBG USB connector (J1401) and the PC USB connector.
3. Connect 12 V ~ 20 V power supply and J-Link Plus to the device.
4. Switch SW1301[3:0] to the M core boot and ensure that the image is not available on the boot source. For example, 0b1010 for MicroSD boot. Keep the SD slot empty.
5. Open the terminal application on the PC, such as PuTTY or TeraTerm, connect to the debug COM port, see [How to determine COM port](#), and configure the terminal with these settings:
  1. 115,200 baud rate
  2. No parity
  3. 8 data bits
  4. 1 stop bit



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



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



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



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



**\*\*Note:\*\*** If the software is already running on the M core, the debugger loading image into TCM may get  
 ↳ HardFault or a data verification issue. NXP recommends you to follow the steps above to use the debugger.  
 ↳ Repowering the board is required to restart the debugger.

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

## Run a demo using ARMGCC / VSCODE

This section describes the steps to run an example application from the SDK archive using the ARMGCC / VSCODE toolchain.

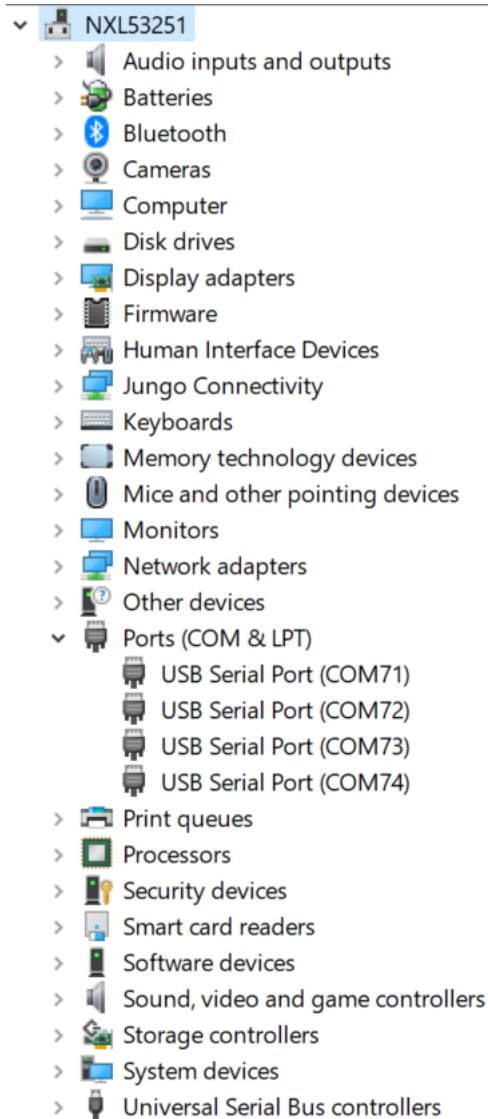
Refer to the [running a demo using MCUXpresso VSC](#) section for detailed instructions on setting up and configuring your project in Visual Studio Code.

Refer to the [CLI](#) section for detailed instructions on building and running your project from the command line.

## Running an application by U-Boot

This section describes the steps to write a bootable SDK bin file to TCM 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 & LPT)**. Assume that the ports are COM71 - COM74. COM73 is for the debug message from the Cortex-A55 and COM74 is for the Cortex-M33. 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 A55 terminal.
7. Enter to U-Boot command line mode. You can then write the image and run it from TCM with the following commands:
  - `fatload mmc 1:1 80000000 hello_world.bin; cp.b 0x80000000 0x201e0000 0x10000;`
  - `bootaux 0x201e0000 0`

```

U-Boot 2022.04-imx_v2022.04+g3d51d4f18b (Jun 30 2022 - 06:55:33 +0000)

CPU:   i.MX93 rev1.0
Model: NXP i.MX93 11X11 EVK board
DRAM:  2 GiB
TCPC:   Vendor ID [0x1fc9], Product ID [0x5110], Addr [I2C0 0x52]
SNK.Power3.0 on CC1
PDO 0: type 0, 5000 mV, 3000 mA [E]
PDO 1: type 0, 9000 mV, 3000 mA []
PDO 2: type 0, 15000 mV, 3000 mA []
PDO 3: type 0, 20000 mV, 2250 mA []
Requesting PDO 3: 20000 mV, 750 mA
Source accept request
PD source ready!
tcpc_pd_receive_message: Polling ALERT register, TCPC_ALERT_RX_STATUS bit failed
, ret = -62
TCPC:   Vendor ID [0x1fc9], Product ID [0x5110], Addr [I2C0 0x51]
TCPC:   Vendor ID [0x1fc9], Product ID [0x5110], Addr [I2C0 0x50]
Core:   204 devices, 26 uclasses, devicetree: separate
MMC:    FSL_SDHC: 0, FSL_SDHC: 1
Loading Environment from MMC... *** Warning - bad CRC, using default environment

[*-Video Link 0adv7535_mipi2hdmi hdmi@3d: Can't find cec device id=0x3c
fail to probe panel device hdmi@3d
fail to get display timings
probe video device failed, ret -19

        [0] lcd-controller@4ae30000, video
        [1] dsi@4ae10000, video_bridge
        [2] hdmi@3d, panel
adv7535_mipi2hdmi hdmi@3d: Can't find cec device id=0x3c
fail to probe panel device hdmi@3d
fail to get display timings
probe video device failed, ret -19
In:     serial
Out:    serial
Err:    serial
switch to partitions #0, OK
mmio1 is current device
UID: 0x9d96f518 0x944d9627 0xe7f8e89 0xc47fcb39
flash target is MMC:1
Net:
Warning: ethernet@428a0000 (eth1) using random MAC address - 82:dc:44:72:38:9d

Warning: ethernet@42890000 (eth0) using random MAC address - ce:ff:2c:ad:e5:e6
eth0: ethernet@42890000 [PRIME], eth1: ethernet@428a0000
Fastboot: Normal
Normal Boot
Hit any key to stop autoboot:  0
=>
=> fatload mmc 0:1 80000000 m33_image.bin
8788 bytes read in 6 ms (1.4 MiB/s)
=> cp.b 0x80000000 0x201e0000 0x10000
=> bootaux 0x201e0000 0
## Starting auxiliary core stack = 0x20020000, pc = 0x0FFE1B09...
=> █

```

8. The `hello_world` application is now running and a banner is displayed on the M33 terminal. If this is not true, check your terminal settings and connections.



## Program flash.bin to SD/eMMC with UUU

This section describes the steps to use the UUU to run the example applications provided in the MCUXpresso SDK. Download the flash.bin to emmc/sd with UUU. The hello\_world demo application targeted for the i.MX 93 hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

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

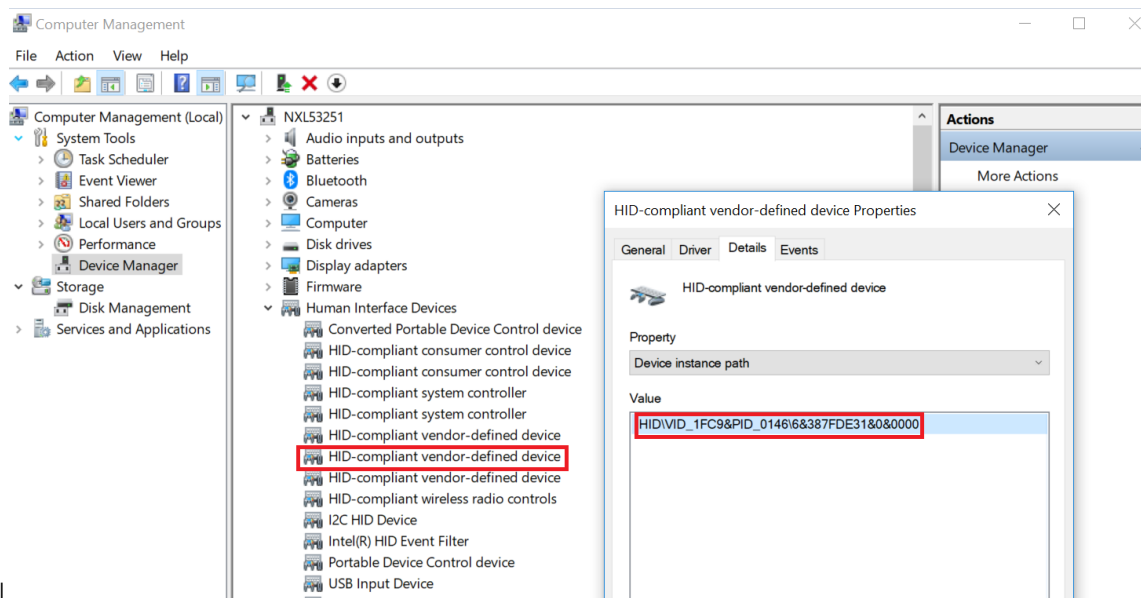
**Download the Universal Upgrade Utility** The Universal Upgrade Utility (UUU) is an upgraded version of MfgTool. It is a command line tool that aims at installing the bootloader to various storage including SD, QSPI, and so on, for i.MX series devices with ease.

The tool can be accessed from corresponding Linux BSP release. Download uuu.exe for Windows OS, or download UUU for Linux. Configure the path so that the executable can later be called anywhere in the command line.

**Parent topic:**Set up environment

**Switch to Download Mode** The board needs to be in Download Mode mode for UUU to download images:

1. Set the board boot mode to Download Mode [SW1301[3:0] = 0011(Single Boot Mode)/1011(Low Power Boot Mode)].
2. Connect the development platform to your PC via USB cable between the DBG USB connector (J1401) and the PC USB connector.
3. Connect J403 (USB1) to PC USB connector for downloading.
4. The PC recognizes the i.MX 93 device as (VID:PID)=(1FC9:0146), as shown in Figure 1.



**Parent topic:**Set up environment

**Parent topic:**[Program flash.bin to SD/eMMC with UUU](#)



**Build an example application** The following steps guide you through opening the `hello_world` example application. These steps may change slightly for other example applications, as some of these applications may have additional layers of folders in their paths.

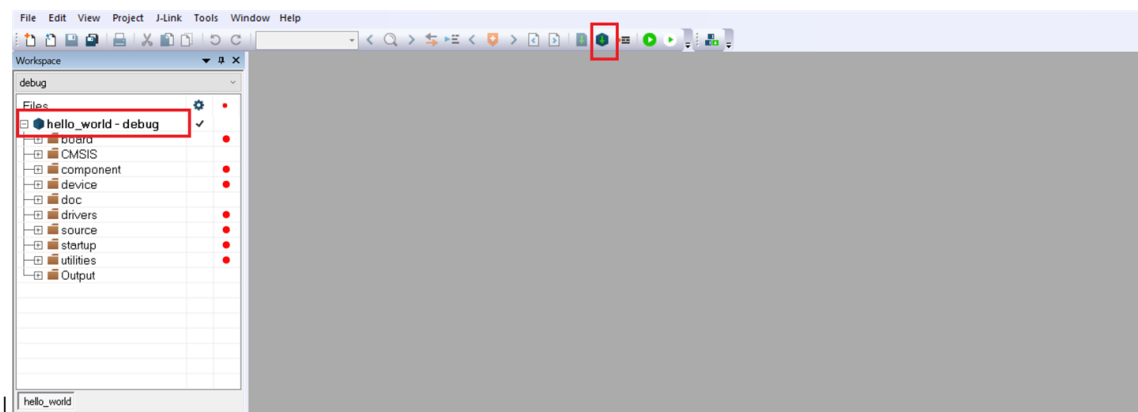
1. If not already done, open the desired demo application workspace. Most example application workspace files can be located using the following path:

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

Using the i.MX 93 EVK board as an example, the workspace is located in:

```
<install_dir>/boards/mcimx93evk/demo_apps/hello_world/iar/hello_world.eww
```

2. Select the desired build target from the drop-down. For this example, select **hello\_world – debug**.
3. To build the demo application, click **Make**.

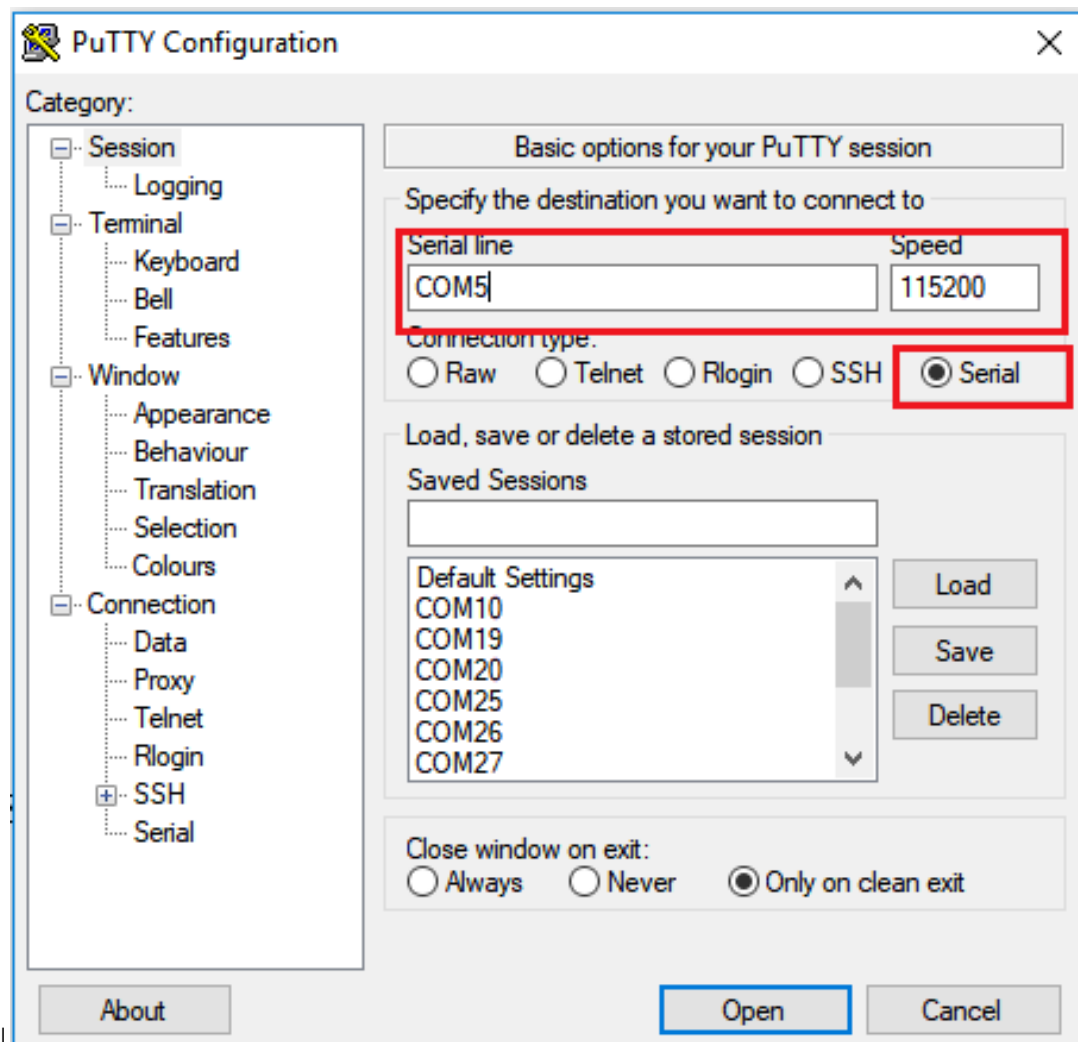


4. The build completes without errors.

**Parent topic:** [Program flash.bin to SD/eMMC with UUU](#)

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

1. Connect the development platform to your PC via USB cable between the DBG USB connector (J1401) and the PC. It provides console output while using UUU.
2. Connect the J403 (USB1) connector and the PC. It provides the data path for UUU.
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. Get the boot images and the imx-mkimage source repository from corresponding Linux BSP release. The boot images required to be put into imx-mkimage/i.MX9 are:

- u-boot-imx93evk.bin-sd (rename to u-boot.bin)
- u-boot-spl.bin-imx93evk-sd (rename to u-boot-spl.bin)
- bl31-imx93.bin (rename to bl31.bin)
- mx93a0-ahab-container.img
- lpddr4\_dmem\_1d\_v202201.bin
- lpddr4\_dmem\_2d\_v202201.bin
- lpddr4\_imem\_1d\_v202201.bin
- lpddr4\_imem\_2d\_v202201.bin

5. Make flash.bin with imx-mkimage.

make SOC=iMX9 flash\_singleboot\_m33 (for single boot mode)

or

make SOC=iMX9 flash\_lpboot (for low power boot mode)

6. Type the UUU command to the flash image.

uuu -b emmc flash.bin (for single boot on eMMC)

`uuu -b sd flash.bin` (for single boot on SD)

For low power boot, a single boot flash.bin is needed besides the target flash.bin.

`uuu -b emmc <singleboot flash.bin> flash.bin` (for lowpower boot on eMMC)

`uuu -b sd <singleboot flash.bin> flash.bin` (for lowpower boot on SD)

The UUU puts the platform into fast boot mode and automatically flashes the target boot-loader to emmc/sd. The command line and fast boot console is as shown in Figure 2.

```

Switch to partitions #0, OK
mmc0(part 0) is current device
flash target is MMC:0
Net:
Warning: ethernet@428a0000 MAC addresses don't match:
Address in ROM is      01:02:03:04:05:06
Address in environment is 36:e5:5c:42:fa:e5

Warning: ethernet@42890000 MAC addresses don't match:
Address in ROM is      01:02:03:04:05:06
Address in environment is be:8a:6c:54:c1:5a
eth0: ethernet@42890000 [PRIME], eth1: ethernet@428a0000
Fastboot: Normal
Normal Boot
Hit any key to stop autoboot: 0
=> fastboot 0
Failed to configure default pinctrl
switch to partitions #0, OK
mmc0(part 0) is current device
Starting download of 1014784 bytes
.....
downloading of 1014784 bytes finished
writing to partition 'bootloader'
Initializing 'bootloader'
switch to partitions #1, OK
mmc0(part 1) is current device
Writing 'bootloader'

MMC write: dev # 0, block # 0, count 1982 ... 1982 blocks written: OK
Writing 'bootloader' DONE!

CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyUSB4

sudo uuu -b emmc flash.bin
uuu (universal update utility) for nxp imx cnips -- libuuu_1.4.224-0-g70d1e85

Success 1   Failure 0

1:141    7/ 7 [Done]          ] FB: Done

nxf49783@biwen:/mnt/home/nxf49783/logs/imx/imx93/imx93evk/verify_rpmsg$

```

7. Then, power off the board and change the boot mode to the corresponding one.

- For single-boot mode:
  - when boot device is emmc, then  $\text{SW1301}[3:0] = 0000$ ;
  - when boot device is sd, then  $\text{SW1301}[3:0] = 0010$ .
- For low-power boot mode:
  - when boot device is emmc, then  $\text{SW1301}[3:0] = 1000$ ;
  - when boot device is sd, then  $\text{SW1301}[3:0] = 1010$ .

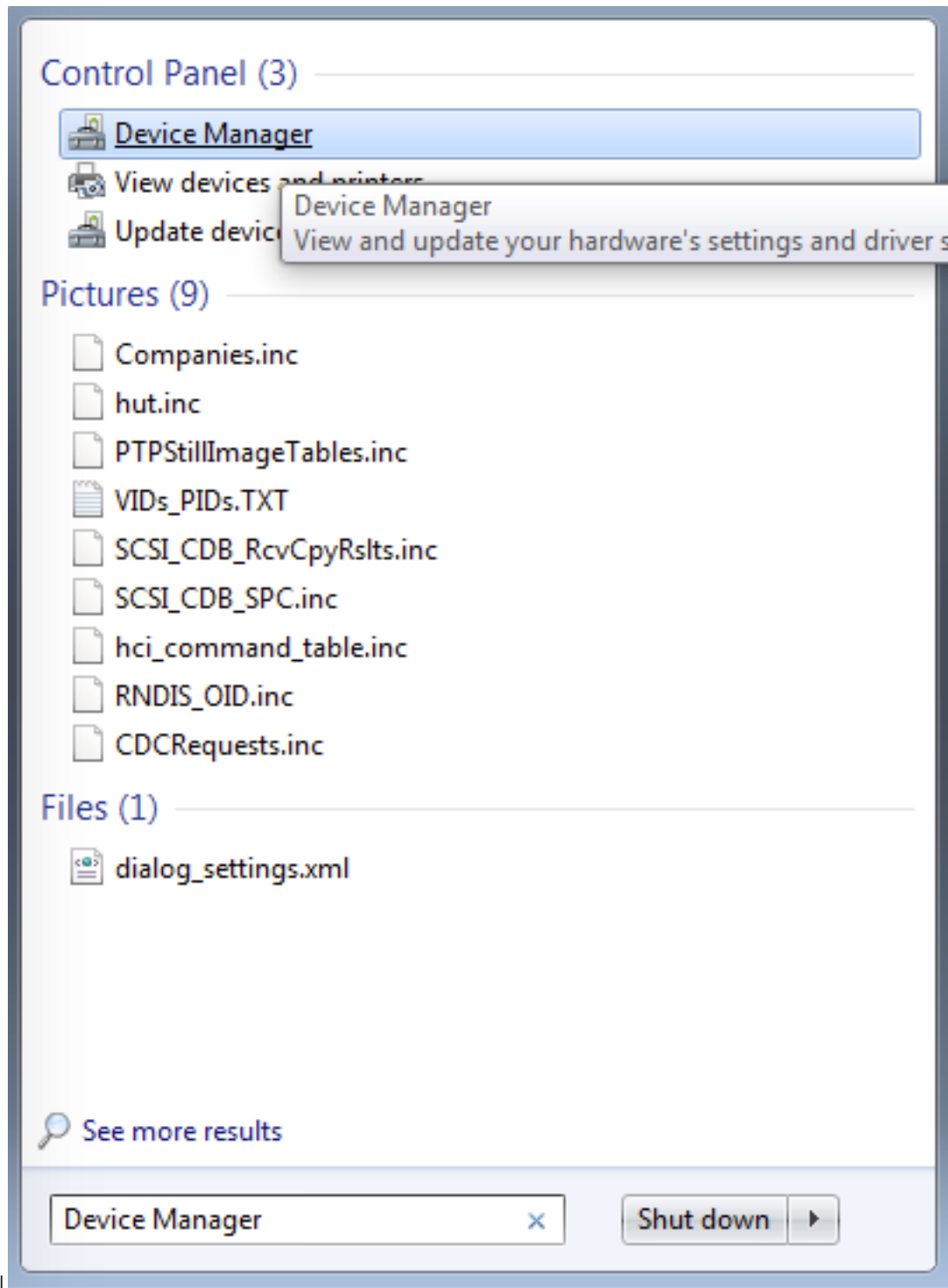
8. Power on the board again.

Parent topic: [Program flash.bin to SD/eMMC with UUU](#)

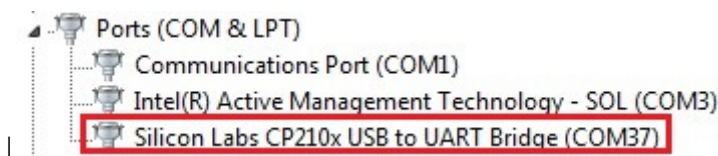
## How to determine COM port

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



## 1.3 Getting Started with MCUXpresso SDK GitHub

### 1.3.1 Getting Started with MCUXpresso SDK Repository

Welcome to the **GitHub Repository SDK Guide**. This documentation provides instructions for setting up and working with the MCUXpresso SDK distributed in a **multi-repository model**. The SDK is distributed across multiple GitHub repositories and managed using the **Zephyr West** tool, enabling modular development and streamlined workflows.

#### Overview

The GitHub Repository SDK approach offers:

- **Modular Structure:** Multiple repositories for flexibility and scalability.
- **Zephyr West Integration:** Simplified repository management and synchronization.
- **Cross-Platform Support:** Designed for MCUXpresso SDK development environments.

#### Benefits of the Multi-Repository Approach

- **Scalability:** Easily add or update components without impacting the entire SDK.
- **Collaboration:** Enables distributed development across teams and repositories.
- **Version Control:** Independent versioning for components ensures better stability.
- **Automation:** Zephyr West simplifies dependency handling and repository synchronization.

#### Setup and Configuration

Follow these steps to prepare your development environment:

**Development Tools Installation** This guide explains how to install the essential tools for development with the MCUXpresso SDK.

**Quick Start: Automated Installation (Recommended)** The **MCUXpresso Installer** is the fastest way to get started. It automatically installs all the basic tools you need.

1. **Download the MCUXpresso Installer** from: [Dependency-Installation](#)
2. **Run the installer** and select “**MCUXpresso SDK Developer**” from the menu
3. **Click Install** and let it handle everything automatically

**Manual Installation** If you prefer to install tools manually or need specific versions, follow these steps:

## Essential Tools

**Git - Version Control** **What it does:** Manages code versions and downloads SDK repositories from GitHub.

### Installation:

- Visit [git-scm.com](https://git-scm.com)
- Download for your operating system
- Run installer with default settings
- **Important:** Make sure “Add Git to PATH” is selected during installation

### Setup:

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

**Python - Scripting Environment** **What it does:** Runs build scripts and SDK tools.

### Installation:

- Install Python **3.10 or newer** from [python.org](https://python.org)
- **Important:** Check “Add Python to PATH” during installation

**West - SDK Management Tool** **What it does:** Manages SDK repositories and provides build commands. The west tool is developed by the Zephyr project for managing multiple repositories.

### Installation:

```
pip install -U west
```

**Minimum version:** 1.2.0 or newer

## Build System Tools

**CMake - Build Configuration** **What it does:** Configures how your projects are built.

**Recommended version:** 3.30.0 or newer

### Installation:

- **Windows:** Download .msi installer from [cmake.org/download](https://cmake.org/download)
- **Linux:** Use package manager or download from [cmake.org](https://cmake.org)
- **macOS:** Use Homebrew (brew install cmake) or download from [cmake.org](https://cmake.org)

**Ninja - Fast Build System** **What it does:** Compiles your code quickly.

**Minimum version:** 1.12.1 or newer

**Installation:**

- **Windows:** Usually included, or download from [ninja-build.org](https://ninja-build.org)
- **Linux:** `sudo apt install ninja-build` or download binary
- **macOS:** `brew install ninja` or download binary

**Ruby - IDE Project Generation (Optional)** **What it does:** Generates project files for IDEs like IAR and Keil.

**When needed:** Only if you want to use traditional IDEs instead of VS Code.

**Installation:** Follow the Ruby environment setup guide

**Compiler Toolchains** Choose and install the compiler toolchain you want to use:

| Toolchain                    | Best For                 | Download Link                                     | Environment Variable |
|------------------------------|--------------------------|---------------------------------------------------|----------------------|
| <b>ARM GCC</b> (Recommended) | Most users, free         | <a href="#">ARM Toolchain</a> <a href="#">GNU</a> | ARMGCC_DIR           |
| <b>IAR EWARM</b>             | Professional development | <a href="#">IAR Systems</a>                       | IAR_DIR              |
| <b>Keil MDK</b>              | ARM ecosystem            | <a href="#">ARM Developer</a>                     | MDK_DIR              |
| <b>ARM Compiler</b>          | Advanced optimization    | <a href="#">ARM Developer</a>                     | ARMCLANG_DIR         |

**Setting Up Environment Variables** After toolchain installation, set an environment variable so the build system locates it:

**Windows:**

```
# Example for ARM GCC installed in C:\armgcc
setx ARMGCC_DIR "C:\armgcc"
```

**Linux/macOS:**

```
# Add to ~/.bashrc or ~/.zshrc
export ARMGCC_DIR="/usr" # or your installation path
```

**Verify Your Installation** After installation, verify everything works by opening a terminal/command prompt and running these commands:

```
# Check each tool - you should see version numbers
git --version
python --version
west --version
cmake --version
ninja --version
arm-none-eabi-gcc --version # (if using ARM GCC)
```

**Troubleshooting Installation Issues** “Command not found” errors:

- The tool isn’t in your system PATH
- **Solution:** Add the installation directory to your PATH environment variable

**Python/pip issues:**

- Try using python3 and pip3 instead of python and pip
- On Windows, run the Command Prompt as an Administrator

**Slow downloads:**

- Add timeout option: `pip install -U west --default-timeout=1000`
- Use alternative mirror: `pip install -U west -i https://pypi.tuna.tsinghua.edu.cn/simple`

**GitHub Repository Setup** This guide explains how to initialize your MCUXpresso SDK workspace from GitHub repositories using the west tool. The GitHub Repository SDK uses multiple repositories hosted on GitHub to provide modular, flexible development.

**Prerequisites** Verify the requirements:

**System Requirements:**

- Python 3.8 or later
- Git 2.25 or later
- CMake 3.20 or later
- Build tools for your target platform

**Verification Commands:**

```
python --version    # Should show 3.8+
git --version       # Should show 2.25+
cmake --version     # Should show 3.20+
west --version      # Should show west tool installation
```

**Workspace Initialization** The GitHub Repository SDK uses the Zephyr west tool to manage multiple repositories containing different SDK components.

**Step 1: Initialize Workspace** Create and initialize your SDK workspace from GitHub:

**Get the latest SDK from main branch:**

```
west init -m https://github.com/nxp-mcuxpresso/mcuxsdk-manifests.git mcuxpresso-sdk
```

**Get SDK at specific revision:**

```
west init -m https://github.com/nxp-mcuxpresso/mcuxsdk-manifests.git mcuxpresso-sdk --mr {revision}
```

*Note: Replace {revision} with the desired release tag, such as v25.09.00*

**Step 2: Choose Your Repository Update Strategy** Navigate to the SDK workspace:

```
cd mcuxpresso-sdk
```

The west tool manages multiple GitHub repositories containing different SDK components. You have two options for downloading:



**Option A: Download All Repositories (Complete SDK)** Download all SDK repositories for comprehensive development:

```
west update
```

This command downloads all the repositories defined in the manifest from GitHub. Initial download takes several minutes and requires ~7 GB of disk space.

**Best for:**

- Exploring the complete SDK
- Multi-board development projects
- Comprehensive middleware evaluation

**Option B: Targeted Repository Download (Recommended)** Download only repositories needed for your specific board or device to save time and disk space:

```
# For specific board development
west update_board --set board your_board_name

# For specific device family development
west update_board --set device your_device_name

# List available repositories before downloading
west update_board --set board your_board_name --list-repo
```

**Best for:**

- Single board development
- Faster setup and reduced disk usage
- Focused development workflows

**Examples:**

```
# Update only repositories for FRDM-MCXW23 board
west update_board --set board frdm-mcxw23

# Update only repositories for MCXW23 device family
west update_board --set device mcxw23
```

**Step 3: Verify Installation** Confirm successful setup:

```
# Verify workspace structure
ls -la
# Should show: manifests/ and mcuxsdk/ directories

# Test build system
west list_project -p examples/demo_apps/hello_world
# Should display available build configurations
```

**Advanced Repository Management** The west extension command `update_board` provides advanced repository management capabilities for optimized workspace setup with GitHub repositories.

**Board-Specific Setup** Update only repositories required for a specific board:

```
# Update only repositories for specific board, e.g., frdm-mcxw23
west update_board --set board frdm-mcxw23

# List available repositories for the board before updating
west update_board --set board frdm-mcxw23 --list-repo
```

**Device-Specific Setup** Update only repositories required for a specific device family:

```
# Update only repositories for specific device, e.g., MCXW235
west update_board --set device mcxw23

# List available repositories for the device family
west update_board --set device mcxw23 --list-repo
```

**Custom Configuration** For advanced users who want to create custom repository combinations:

```
# Use custom configuration file
west update_board --set custom path/to/custom-config.yml

# Generate custom configuration template
cp manifests/boards/custom.yml.template my-custom-config.yml
```

### Benefits of Targeted Setup Reduced Download Size

- Download only components needed for your target board or device
- Significantly faster initial setup for focused development
- Typical reduction from 7 GB to 2GB

### Optimized Workspace

- Cleaner workspace with relevant components only
- Reduced disk space usage
- Faster repository operations

### Flexible Development

- Switch between different board configurations easily
- Maintain separate workspaces for different projects
- Include optional components as needed

**Repository Information** Before setting up your workspace, you can explore what repositories are available:

```
# Display repository information in console
west update_board --set board frdm-mcxw23 --list-repo

# Export repository information to YAML file for reference
west update_board --set board frdm-mcxw23 --list-repo -o board-repos.yml
```

This command lists all the available repositories with descriptions and outlines the included components in the workspace.

**Package Generation (Optional)** The `update_board` command can also generate ZIP packages for offline distribution:

```
# Generate board-specific SDK package
west update_board --set board frdm-mcxw23 -o frdm-mcxw23-sdk.zip
```

**Note:** Package generation is primarily intended for creating custom SDK distributions. For regular development, use the workspace update commands without the `-o` option.

## Workspace Management

**Updating Your Workspace** Keep your SDK current with latest updates from GitHub:

**For Complete SDK Workspace:**

```
# Update manifest repository
cd manifests
git pull

# Update all component repositories
cd ..
west update
```

**For Targeted Workspace:**

```
# Update manifest repository
cd manifests
git pull

# Update board-specific repositories
cd ..
west update_board --set board your_board_name
```

**Workspace Status** Check workspace synchronization status:

```
# Show status of all repositories
west status

# Show detailed information about repositories
west list
```

**Troubleshooting Network Issues:**

- Use `west update --keep-descendants` for partial failures
- Configure Git credentials for private repositories
- Check firewall settings for Git protocol access

**Permission Issues:**

- Ensure write permissions in workspace directory
- Run commands without `sudo`/administrator privileges
- Verify Git SSH key configuration for authenticated access

**Disk Space:**

- Full SDK workspace requires approximately 7-8 GB
- Targeted workspace typically requires 1-2 GB

- Use board-specific setup to reduce workspace size

#### Repository Management Issues:

- Verify board/device names match available configurations
- Check that custom YAML files follow the correct template format
- Use `--list-repo` to verify available repositories before setup

**Next Steps** With your workspace initialized:

1. Review [Workspace Structure](#) to understand the layout
2. Build your first project with [First Build Guide](#)
3. Explore [Development Workflows MCUXpresso VSCode](#) or [Development Workflows Command Line](#) for the details on project setup and execution

For advanced repository management, see the [west tool documentation](#).

### Explore SDK Structure and Content

Learn about the organization of the SDK and its components:

**SDK Architecture Overview** The MCUXpresso SDK uses a modular architecture where software components are distributed across multiple repositories hosted on GitHub and managed through the west tool. This approach provides flexibility, maintainability, and enables selective component inclusion.

**Repository Organization** Based on the manifest structure, the SDK consists of four main repository categories:

**Manifest Repository** The manifest repo (mcuxsdk-manifests) contains the west.yml manifest file that tracks all other repositories in the SDK.

**Base Repositories** Recorded in submanifests/base.yml and loaded in the root west.yml manifest file. These are the foundational repositories that build the SDK:

- **Devices:** MCU-specific support packages
- **Examples:** Demonstration applications and code samples
- **Boards:** Board support packages

**Middleware Repositories** Recorded in the submanifests/middleware subdirectory, categorized according to functionality:

- **Connectivity:** Networking stacks, USB, and communication protocols
- **Security:** Cryptographic libraries and secure boot components
- **Wireless:** Bluetooth, IEEE 802.15.4, and other wireless protocols
- **Graphics:** Display drivers and UI frameworks
- **Audio:** Audio processing and voice recognition libraries
- **Machine Learning:** AI inference engines and neural network libraries

- **Safety:** IEC60730B safety libraries
- **Motor Control:** Motor control and real-time control libraries

**Internal Repositories** Recorded in `submanifests/internal.yml` and grouped into the “bifrost” group. These are only visible to NXP internal developers and hosted on NXP internal git servers.

**Repository Hosting** Public repositories are hosted on GitHub under these organizations:

- [nxp-mcuxpresso](#)
- [NXP](#)
- [nxp-zephyr](#)

Internal repositories are hosted on NXP’s private Git infrastructure.

**Benefits of This Architecture** **Selective Integration:** Projects include only required components, reducing memory footprint and build complexity.

**Independent Versioning:** Each component maintains its own release cycle and version control.

**Community Collaboration:** Public repositories accept community contributions through standard Git workflows.

**Scalable Maintenance:** Component owners can update their repositories without affecting the entire SDK.

**Workspace Management** The west tool manages repository synchronization, version tracking, and workspace updates. All repositories are checked out under the `mcuxsdk/` directory with their designated paths defined in the manifest files.

**Workspace Structure** After you initialize your SDK workspace, it creates a specific directory structure that organizes all SDK components. This structure is identical for both GitHub Repository SDK and Repository-Layout SDK Package.

### Top-Level Organization

```
your-sdk-workspace/  
  manifests/      # West manifest repository  
  mcuxsdk/        # Main SDK content
```

The `mcuxsdk/` directory serves as your primary working directory and contains all the SDK components.

**SDK Component Layout** Based on the actual SDK structure, the main directories include:

| Directory | Contents                                                                                                                                                                                            | Purpose                                             |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------|
| arch/     | Architecture-specific files                                                                                                                                                                         | ARM CMSIS, build configurations                     |
| cmake/    | Build system modules                                                                                                                                                                                | CMake configuration and build rules                 |
| compo     | Software components                                                                                                                                                                                 | Reusable software libraries and utilities           |
| devices   | Device support packages                                                                                                                                                                             | MCU-specific headers, startup code, linker scripts  |
| drivers   | Peripheral drivers                                                                                                                                                                                  | Hardware abstraction layer for MCU peripherals      |
| examp     | Sample applications                                                                                                                                                                                 | Demonstration code and reference implementations    |
| middle    | Optional software stacks                                                                                                                                                                            | Networking, graphics, security, and other libraries |
| rtos/     | Operating system support                                                                                                                                                                            | FreeRTOS integration                                |
| scripts   | Build and utility scripts                                                                                                                                                                           | West extensions and development tools               |
| 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. |                                                     |

**Example Organization** Examples follow a two-tier structure separating common code from board-specific implementations:

### Common Example Files

```
examples/demo_apps/hello_world/
  CMakeLists.txt      # Build configuration
  example.yml         # Example metadata
  hello_world.c       # Application source code
  Kconfig             # Configuration options
  readme.md           # General documentation
```

### Board-Specific Files

```
examples/_boards/your_board/demo_apps/hello_world/
  app.h               # Board specific application header
  example_board_readme.md # Board specific documentation
  hardware_init.c     # Board specific hardware initialization
  pin_mux.c           # Pin multiplexing configuration
  pin_mux.h           # Pin multiplexing header definitions
  hello_world.bin     # Pre-built binary for quick testing
  hello_world.mex     # MCUXpresso Config Tools project file
  prj.conf            # Board specific Kconfig configuration
  reconfig.cmake      # Board specific cmake configuration overrides
```

**Device Support Structure** Device support is organized hierarchically by MCU family:

```

devices/
  MCX/           # MCU portfolio
  MCXW/          # MCU family
  MCXW235/       # Specific device
  MCXW235.h      # Device register definitions
  drivers/       # Device-specific drivers
  gcc/           # GNU toolchain files
  iar/           # IAR toolchain files
  mcuxpresso/    # MCUXpresso IDE files
  startup_MCXW235.c # Startup and vector table
  system_MCXW235.c # System initialization

```

**Middleware Organization** Middleware components are categorized by functionality and maintained in separate repositories. Based on the manifest files, common middleware categories include:

- **Connectivity:** USB, TCP/IP, industrial protocols
- **Security:** Cryptographic libraries, secure boot
- **Wireless:** Bluetooth, IEEE 802.15.4, Wi-Fi
- **Graphics:** Display drivers, UI frameworks
- **Audio:** Processing libraries, voice recognition
- **Machine Learning:** Inference engines, neural networks
- **Safety:** IEC60730B safety libraries
- **Motor Control:** Motor control and real-time control libraries

**Documentation Structure** SDK documentation is distributed across multiple locations:

- docs/ - Core SDK documentation and build infrastructure
- Component repositories - API documentation and integration guides
- Board directories - Hardware-specific setup instructions

For complete documentation, refer to the [online documentation](#).

**Understanding Example Structure** Each example has **two README files**:

**1. General README:** examples/demo\_apps/hello\_world/readme.md

- What the example does
- General functionality description
- Common usage information

**2. Board-Specific README:** examples/\_boards/{board\_name}/demo\_apps/hello\_world/example\_board\_readme.md

- Board-specific setup instructions
- Hardware connections required
- Board-specific behavior notes

**Tip:** Always check both readme files - start with the general one, then read the board-specific one for detailed setup.

## Development Workflows

Get started with building and running projects:

**Building Your First Project** This guide explains how to build and run your first SDK example project using the west build system. This applies to both GitHub Repository SDK and Repository-Layout SDK Package.

### Prerequisites

- GitHub Repository SDK workspace initialized OR Repository-Layout SDK Package extracted
- Development board connected via USB
- Build tools installed per [Installation Guide](#)

**Understanding Board Support** Use the west extension to discover available examples for your board:

```
west list_project -p examples/demo_apps/hello_world
```

This shows all supported build configurations. You can filter by toolchain:

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

## Basic Build Process

**Simple Build** Build the hello\_world example with default settings:

```
west build -b your_board examples/demo_apps/hello_world
```

The default toolchain is armgcc, and the build system will select the first debug target as default if no config is specified.

### Specifying Configuration

```
# Release build
west build -b your_board examples/demo_apps/hello_world --config release
```

```
# Debug build (default)
west build -b your_board examples/demo_apps/hello_world --config debug
```

### Alternative Toolchains

```
# IAR toolchain
west build -b your_board examples/demo_apps/hello_world --toolchain iar
```

```
# Other toolchains as supported by the example
```



**Multicore Applications** For multicore devices, specify the core ID:

```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world --toolchain iar -Dcore_id=cm7 --config ↵
↵ flexspi_nor_debug
```

For multicore projects using sysbuild:

```
west build -b evkbmimxrt1170 --sysbuild ./examples/multicore_examples/hello_world/primary -Dcore_ ↵
↵ id=cm7 --config flexspi_nor_debug --toolchain=armgcc -p always
```

**Flash an Application** Flash the built application to your board:

```
west flash -r linkserver
```

**Debug** Start a debug session:

```
west debug -r linkserver
```

## Common Build Options

**Clean Build** Force a complete rebuild:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

**Dry Run** See the commands that get executed without running them:

```
west build -b your_board examples/demo_apps/hello_world --dry-run
```

**Device Variants** For boards supporting multiple device variants:

```
west build -b your_board examples/demo_apps/hello_world --device DEVICE_PART_NUMBER --config ↵
↵ release
```

## Project Configuration

**CMake Configuration Only** Run configuration without building:

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

**Interactive Configuration** Launch the configuration GUI:

```
west build -t guiconfig
```

## Troubleshooting

**Build Failures** Use pristine builds to resolve dependency issues:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

**Getting Help** View the help information for west build:

```
west build -h
```

**Check Supported Configurations** To see available configuration options and board targets for an example, refer to the below command:

```
west list_project -p examples/demo_apps/hello_world
```

## Next Steps

- Explore other examples in the SDK
- Learn about [Command Line Development](#) for advanced options
- Try [VS Code Development](#) for integrated development
- Refer [Workspace Structure](#) to understand the SDK layout

**MCUXpresso for VS Code Development** This guide covers using MCUXpresso for VS Code extension to build, debug, and develop SDK applications with an integrated development environment.

## Prerequisites

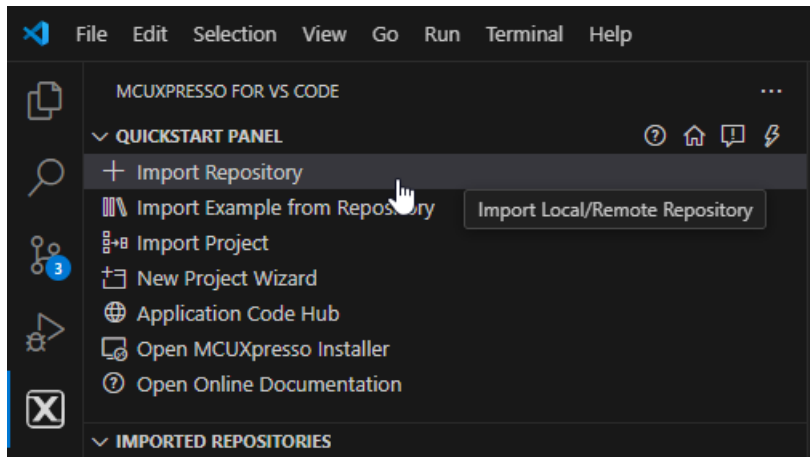
- SDK workspace initialized (GitHub Repository SDK or Repository-Layout SDK Package)
- Development tools installed per [Installation Guide](#)
- Visual Studio Code installed
- MCUXpresso for VS Code extension installed

## Extension Installation

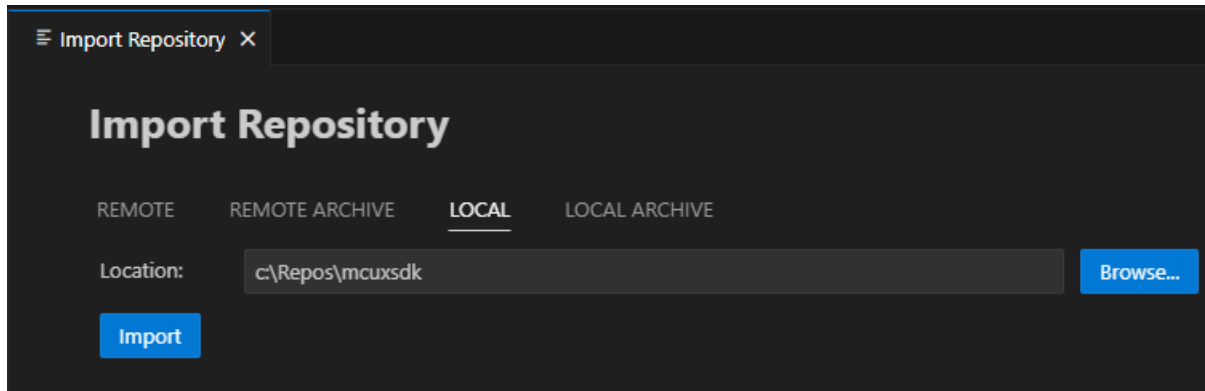
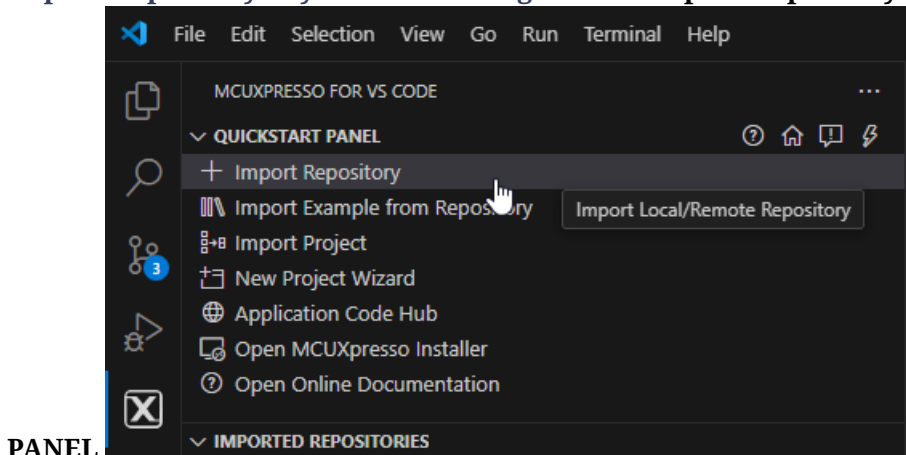
**Install MCUXpresso for VS Code** The MCUXpresso for VS Code extension provides integrated development capabilities for MCUXpresso SDK projects. Refer to the [MCUXpresso for VS Code Wiki](#) for detailed installation and setup instructions.

## SDK Import and Setup

**Import Methods** The SDK can be imported in several ways. The MCUXpresso for VS Code extension supports both GitHub Repository SDK and Repository-Layout SDK Package distributions.

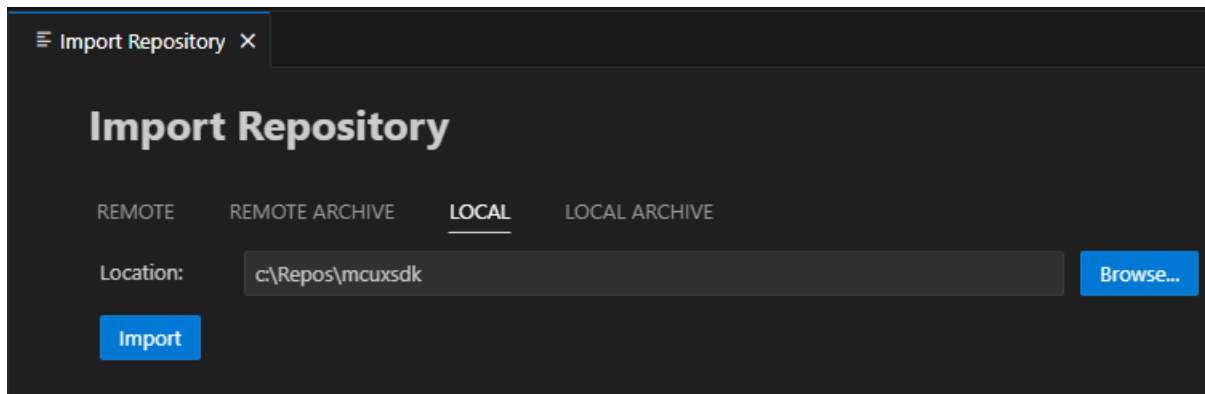
**Import GitHub Repository SDK** Click **Import Repository** from the **QUICKSTART PANEL**

**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 according to [setting up the repo](#). Select your location and click **Import**.

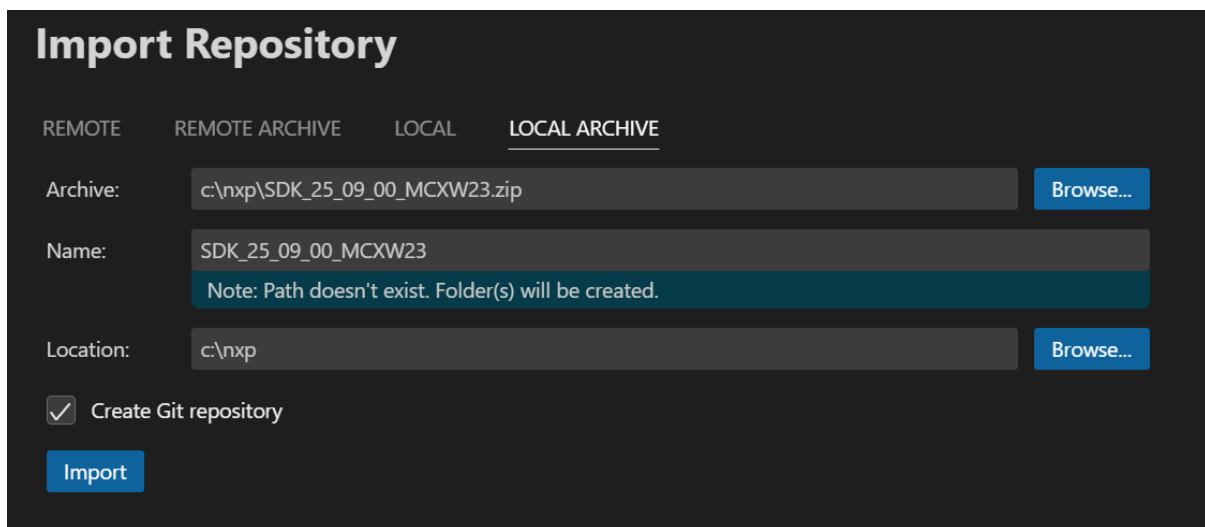
**Import Repository-Layout SDK Package** Click **Import Repository** from the **QUICKSTART**

**PANEL**

Select **Local** if you've already unzipped the Repository-Layout SDK Package. Select your location and click **Import**.



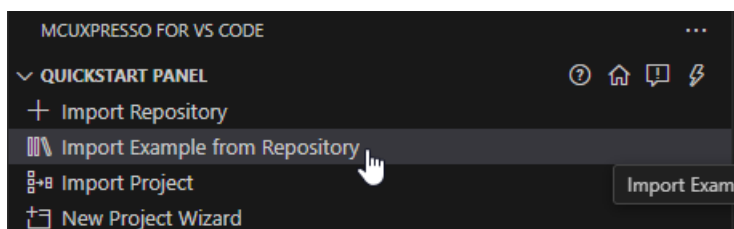
Else if the SDK is ZIP archive, select **Local Archive**, browse to the downloaded SDK ZIP file, fill the link of expect location, then click **Import**.



## Building Example Applications

### Import Example Project

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



2. Configure project settings:
  - **MCUXpresso SDK:** Select your imported SDK
  - **Arm GNU Toolchain:** Choose toolchain
  - **Board:** Select your target development board
  - **Template:** Choose example category
  - **Application:** Select specific example (e.g., hello\_world)
  - **App type:** Choose between Repository applications or Freestanding applications

### 3. Click **Import**

Import Example from Repository

## Import Example from Repository

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

Toolchain:
(Arm GNU Toolchain 13.2.rel1 (Build arm-13.7)) 13.2.1 20231009 (C:\NXP\MCUXpressoIDE\_24.

Board:
FRDM-MCXC444


FRDM-MCXC444

Template:
demo\_apps/hello\_world

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

Name:
frdmmcxc444\_hello\_world

Location:
c:\nxp\_examples
Browse...

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

☐ Open readme file after project is imported

Import

#### Application Types    Repository Applications:

- Located inside the MCUXpresso SDK
- Integrated with SDK workspace

#### Freestanding Applications:

- Imported to user-defined location
- Independent of SDK location

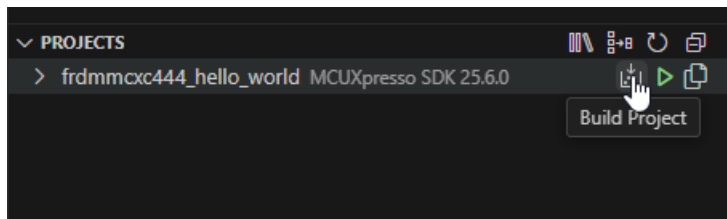
**Trust Confirmation** VS Code will prompt you to confirm if the imported files are trusted. Click **Yes** to proceed.

## Building Projects

### Build Process

1. Navigate to **PROJECTS** view

2. Find your project
3. Click the **Build Project** icon

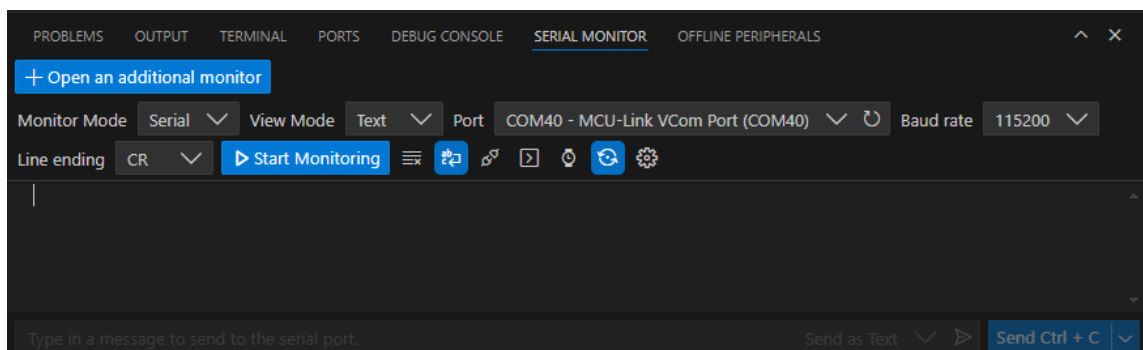


The integrated terminal will display build output at the bottom of the VS Code window.

## Running and Debugging

### Serial Monitor Setup

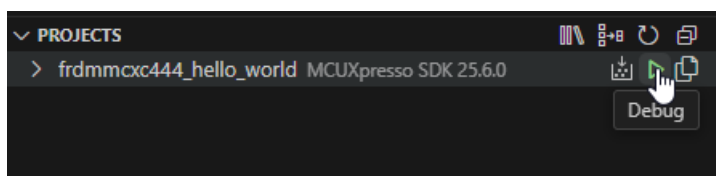
1. Open **Serial Monitor** from VS Code's integrated terminal



2. Configure serial settings:
  - **VCom Port:** Select port for your device
  - **Baud Rate:** Set to 115200

### Debug Session

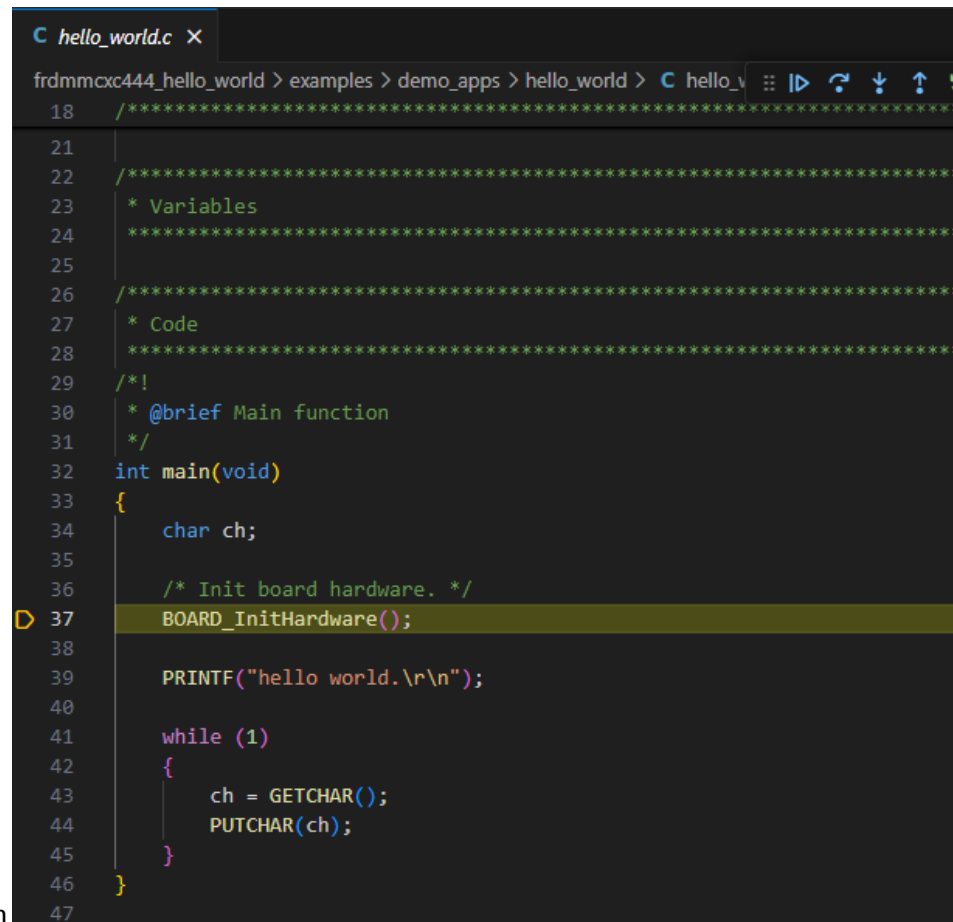
1. Navigate to **PROJECTS** view
2. Click the play button to initiate a debug session



The debug session will begin with debug controls initially at the top of the interface.

**Debug Controls** Use the debug controls to manage execution:

- **Continue:** Resume code execution
- **Step controls:** Navigate through code



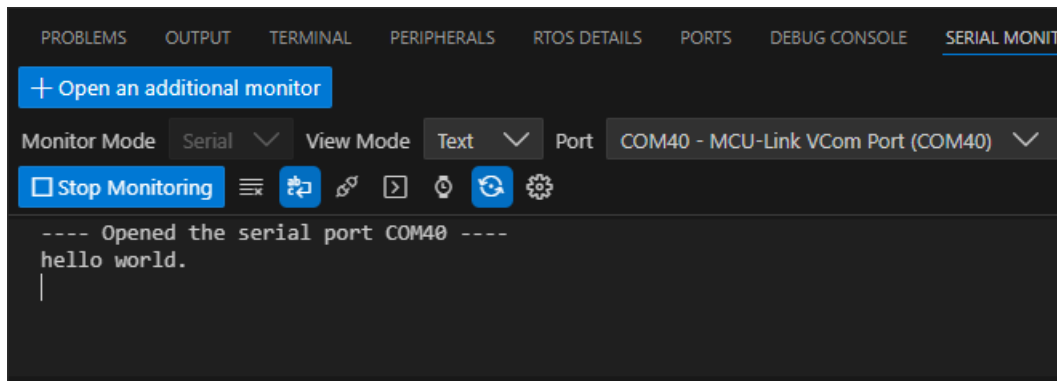
```

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

```

- **Stop:** Terminate debug session

**Monitor Output** Observe application output in the **Serial Monitor** to verify correct operation.



**Debug Probe Support** For comprehensive information on debug probe support and configuration, refer to the [MCUXpresso for VS Code Wiki DebugK](#) section.

## Project Configuration

**Workspace Management** The extension integrates with the MCUXpresso SDK workspace structure, providing access to:

- Example applications
- Board configurations

- Middleware components
- Build system integration

**Multi-Project Support** The PROJECTS view allows management of multiple imported projects within the same workspace.

## Troubleshooting

### Import Issues SDK not detected:

- Verify SDK workspace is properly initialized
- Ensure all required repositories are updated
- Check SDK manifest files are present

### Project import failures:

- Confirm board support exists for selected example
- Verify toolchain installation
- Check example compatibility with selected board

### Build Problems Build failures:

- Check integrated terminal for error messages
- Verify all dependencies are installed
- Ensure toolchain is properly configured

### Debug Issues Debug session fails:

- Verify board connection via USB
- Check debug probe drivers are installed
- Confirm build completed successfully

### Serial monitor problems:

- Verify correct VCom port selection
- Check baud rate configuration (115200)
- Ensure board drivers are installed

**Integration with Command Line** MCUXpresso for VS Code integrates with the underlying west build system, allowing seamless integration with command line workflows described in [Command Line Development](#).

## Advanced Features

**Project Types** The extension supports both repository-based and freestanding project types, providing flexibility in project organization and SDK integration.



**Build System Integration** The extension leverages the MCUXpresso SDK build system, providing access to all build configurations and options available through command line tools.

### Next Steps

- Explore additional examples in the SDK
- Review [Command Line Development](#) for advanced build options
- Refer [MCUXpresso for VS Code Wiki](#) for detailed documentation
- Learn about [SDK Architecture](#) for better understanding of the development environment

**Command Line Development** This guide covers developing with the MCUXpresso SDK using command line tools and the west build system. This workflow applies to both GitHub Repository SDK and Repository-Layout SDK Package distributions.

### Prerequisites

- GitHub Repository SDK workspace initialized OR Repository-Layout SDK Package extracted
- Development tools installed per [Installation Guide](#)
- Target board connected via USB

**Understanding Board Support** Use the west extension to discover available examples for your board:

```
west list _project -p examples/demo_apps/hello_world
```

This shows all supported build configurations. You can filter by toolchain:

```
west list _project -p examples/demo_apps/hello_world -t armgcc
```

### Basic Build Commands

**Standard Build Process** Build with default settings (armgcc toolchain, first debug config):

```
west build -b your_board examples/demo_apps/hello_world
```

### Specifying Build Configuration

```
# Release build
west build -b your_board examples/demo_apps/hello_world --config release

# Debug build with specific toolchain
west build -b your_board examples/demo_apps/hello_world --toolchain iar --config debug
```

**Multicore Applications** For multicore devices, specify the core ID:

```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world --toolchain iar -Dcore_id=cm7 --config ↵
↵ flexspi_nor_debug
```

For multicore projects using sysbuild:

```
west build -b evkbmimxrt1170 --sysbuild ./examples/multicore_examples/hello_world/primary -Dcore_
↪id=cm7 --config flexspi_nor_debug --toolchain=armgcc -p always
```

**Shield Support** For boards with shields:

```
west build -b mimxrt700evk --shield a8974 examples/issdk_examples/sensors/fxls8974cf/fxls8974cf_poll -
↪Dcore_id=cm33_core0
```

## Advanced Build Options

**Clean Builds** Force a complete rebuild:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

**Dry Run** See what commands would be executed:

```
west build -b your_board examples/demo_apps/hello_world --dry-run
```

**Device Variants** For boards supporting multiple device variants:

```
west build -b your_board examples/demo_apps/hello_world --device MK22F12810 --config release
```

## Project Configuration

**CMake Configuration Only** Run configuration without building:

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

**Interactive Configuration** Launch the configuration GUI:

```
west build -t guiconfig
```

## Flashing and Debugging

**Flash Application** Flash the built application to your board:

```
west flash -r linkserver
```

**Debug Session** Start a debugging session:

```
west debug -r linkserver
```

**IDE Project Generation** Generate IDE project files for traditional IDEs:

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

IDE project files are generated in `mcuxsdk/build/<toolchain>` folder.

**Note:** Ruby installation is required for IDE project generation. See [Installation Guide](#) for setup instructions.

## Troubleshooting

**Build Failures** Use pristine builds to resolve dependency issues:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

**Toolchain Issues** Verify environment variables are set correctly:

```
# Check ARM GCC
echo $ARMGCC_DIR
arm-none-eabi-gcc --version

# Check IAR (if using)
echo $IAR_DIR
```

**Getting Help** Display help information:

```
west build -h
west flash -h
west debug -h
```

**Check Supported Configurations** If unsure about supported options for an example:

```
west list_project -p examples/demo_apps/hello_world
```

## Best Practices

### Project Organization

- Keep custom projects outside the SDK tree
- Use version control for your application code
- Document any SDK modifications

### Build Efficiency

- Use `-p always` for clean builds when troubleshooting
- Leverage `--dry-run` to understand build processes
- Use specific configs and toolchains to reduce build time

## Development Workflow

1. Start with existing examples closest to your requirements
2. Copy and modify rather than building from scratch
3. Test with hello\_world before moving to complex examples
4. Use configuration tools for pin muxing and clock setup

## Next Steps

- Explore [VS Code Development](#) for integrated development experience
- Review [Workspace Structure](#) to understand SDK organization
- Refer build system documentation for advanced configurations

**Using MCUXpresso Config Tools** MCUXpresso Config tools provide a user-friendly way to configure hardware initialization of your projects. This guide explains the basic workflow with the MCUXpresso SDK west build system and the Config Tools.

## Prerequisites

- GitHub Repository SDK workspace initialized OR Repository-Layout SDK Package extracted
- MCUXpresso Config Tools standalone installed (version 25.09 or above)
- MCUXpresso SDK Project that can be successfully built

**Board Files** MCUXpresso Config Tools generate source files for the board. These files include pin\_mux.c/h and clock\_config.c/h. The files contain initialization code functions that reflect the hardware configuration in the Config Tools. Within the SDK codebase, these files are specific for the board and either shared by multiple example projects or specific for one example. Open or import the configuration from the SDK project in the Config Tools and customize the settings to match the custom board or specific project use case and regenerate the code. See *User Guide for MCUXpresso Config Tools (Desktop)* (document [GSMCUXCTUG](#)) for details.

**Note:** When opening the configuration for SDK example projects, the board files may be shared across multiple examples. To ensure a separate copy of the board configuration files exists, create a freestanding project with copied board files.

**Visual Studio Code** To open the configuration in Visual Studio Code, use the context menu for the project to access Config Tools. See [MCUXpresso Extension Documentation](#) for details. Otherwise, use the manual workflow described in detail in the following section.

**Manual Workflow** Use the following steps:

1. Before using Config Tools, run the west command to get the project information for Config Tools from the SDK project files, for example:

```
west cfg_project_info -b lpcxpresso55s69 ...mcuxsdk/examples/demo_apps/hello_world/ -Dcore_
↪ id=cm33_core0
```

This results in the creation of the project information json file that is searched by the config tools when the configuration is created. The parameters of the command should match the build parameters that will be used for the project.

2. Launch the MCUXpresso Config Tools and in the **Start development** wizard, select **Create a new configuration based on the existing IDE/Toolchain project**. Select the created “cfg\_tools” subfolder as a project folder (for example: ...mcuxsdk/examples/demo\_apps/hello\_world/cfg\_tools/).

**Updating the SDK West project** **Note:** Updating project is supported with Config Tools V25.12 or newer only.

Changes in the Config tools generated source code modules may require adjustments to the toolchain project to ensure a successful build. These changes may mean, for example, adding the newly generated files, adding include paths, required drivers, or other SDK components. This section describes how to manually resolve the changes needed in the project within the toolchain projects based on the SDK project managed by the West tool.

After the configuration in the Config Tools is finished, write updated files to the disk using the ‘Update Code’ command. The written files include a json file with the required changes for the toolchain project.

To resolve the changes in the project in the terminal, launch the west command that updates the project. For example:

```
west cfg_resolve -b lpcxpresso55s69 ...mcuxsdk/examples/demo_apps/hello_world/ -Dcore_id=cm33_core0
```

This command updates the appropriate cmake and kconfig files to address the changes. After this, the application can be built.

**Note:** The `cfg_resolve` command supports additional arguments. Launch the `west cfg_resolve -h` command to get the list and description.

## 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.

| De-<br>vel-<br>op-<br>ment<br>board | MCU devices    |                |                       |                |
|-------------------------------------|----------------|----------------|-----------------------|----------------|
| <b>MCIM<br/>EVK</b>                 | MIMX9331AVTXM, | MIMX9331CVVXM, | MIMX9331DVVXM,        | MIMX9331XVVXM, |
|                                     | MIMX9332AVTXM, | MIMX9332CVVXM, | MIMX9332DVVXM,        | MIMX9332XVVXM, |
|                                     | MIMX9351AVTXM, | MIMX9351CVVXM, | MIMX9351DVVXM,        | MIMX9351XVVXM, |
|                                     | MIMX9352AVTXM, | MIMX9352CVVXM, | <b>MIMX9352DVVXM,</b> | MIMX9352XVVXM, |
|                                     | MIMX9301CVVXD, | MIMX9301DVVXD, | MIMX9302CVVXD,        | MIMX9302DVVXD  |

## 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.

**IEC60730B Safety Library** NXP IEC60730B Safety Library

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

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

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

**VoiceSpot** VoiceSpot is a highly accurate, small memory and MIPS profile wake word engine supporting custom voice trigger words and phrases. MCUXpresso SDK version of VoiceSpot is trained to “Hey NXP” wake word only and has 25 hour trial timeout.

**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

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

eIQ machine learning SDK containing:

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

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

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

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

**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.



### SEGGER J-Link debugger usage problem

When an M core software is already running, it is possible to get HardFault or data verification issue during loading image into TCM by debugger.

The following steps are recommended to use the J-Link debugger.

1. Configure switch SW1301 to M core boot; low-power boot. Ensure that there is no image on the boot source.
2. Power the board and start the debugger for use.
3. To restart the debugger, stop the debugger, power off the board, and repeat step 2.

### eDMA examples accessing AIPS peripheral bridge memory must run through U-Boot loading method

Non-secure access to Arm IP Bus (AIPS) must be configured in Trusted Resource Domain Control (TRDC) for enhanced direct memory access (eDMA) controller. Due to the limitation that Sentinel ROM can release TRDC only once, such examples must run through U-Boot loading method after Trusted Firmware-A (TF-A) configuring TRDC.

To make low-power boot mode work for only M core in such example, you need to implement the request of the TRDC release and configure TRDC. However, it will break the single boot mode with TF-A/Linux BSP.

The following eDMA examples need access to AIPS.

- cmsis\_lpi2c\_edma\_b2b\_transfer\_master
- cmsis\_lpi2c\_edma\_b2b\_transfer\_slave
- cmsis\_lpuart\_edma\_transfer
- flexcan\_loopback\_edma\_transfer
- lpi2c\_edma\_b2b\_transfer\_master
- lpi2c\_edma\_b2b\_transfer\_slave
- lpuart\_edma\_transfer
- pdm\_edma\_transfer
- sai\_edma\_transfer

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

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

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

**Affected toolchains:** mdk, iar

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

## 1.5 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 XCACHE

###### [2.0.4]

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

###### [2.0.3]

- Bug Fixes
  - Fixed CERT INT30-C violations.

###### [2.0.2]

- Bug Fixes
  - Updated `XCACHE_InvalidateCacheByRange()`, `XCACHE_CleanCacheByRange()`, `XCACHE_CleanInvalidateCacheByRange()` in case of startAddr equal to endAddr.

###### [2.0.1]

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

**[2.0.0]**

- Initial version.
- 

**COMMON****[2.6.3]**

- Bug Fixes
  - Fixed build issue of CMSIS PACK BSP example caused by CMSIS 6.1 issue.

**[2.6.2]**

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule for implicit conversions in boolean contexts

**[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 irqs that mount under `irqsteer` interrupt extender.

**[2.4.2]**

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

**[2.4.1]**

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

#### [2.4.0]

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

#### [2.3.3]

- New Features
  - Added NETC into status group.

#### [2.3.2]

- Improvements
  - Make driver aarch64 compatible

#### [2.3.1]

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

#### [2.3.0]

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

#### [2.2.10]

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

#### [2.2.9]

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

#### [2.2.8]

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

#### [2.2.7]

- Other Change
  - Added MECC status group definition.

**[2.2.6]**

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

**[2.2.5]**

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

**[2.2.4]**

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

**[2.2.3]**

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

**[2.2.2]**

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

**[2.2.1]**

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

**[2.2.0]**

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

**[2.1.4]**

- New Features
  - Added OTFAD into status group.

### [2.1.3]

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

### [2.1.2]

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

### [2.1.1]

- Bug Fixes
  - Deleted and optimized repeated macro.

### [2.1.0]

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

### [2.0.2]

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

### [2.0.1]

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

### [2.0.0]

- Initial version.
- 

## EDMA

### [2.10.9]

- Bug Fixes
  - Add new api EDMA\_TcdInit to avoid destroying code logic by reordering blocks in the toolchain.

**[2.10.8]**

- Bug Fixes
  - Fixed coverity issues with CERT INT30-C, CERT INT31-C compliance.
  - Fixed incorrect enabling of preemption capability issue.

**[2.10.7]**

- Improvements
  - Add condition to fix build warnings(array subscript 4 is above array bounds of 'edma\_handle\_t \*[4][64]')
- Bug Fixes
  - Fixed the EDMA header index retrieval error caused by done bit calculation mistake issue.

**[2.10.6]**

- Improvements
  - Add macro FSL\_FEATURE\_EDMA\_HAS\_EDMA\_TCD\_CLOCK\_ENABLE to enable tcd clocks in EDMA\_Init function.

**[2.10.5]**

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

**[2.10.4]**

- Improvements
  - Add new MP register macros to ensure compatibility with different devices.
  - Add macro DMA\_CHANNEL\_ARRAY\_STEPn to adapt to complex addressing of edma tcd registers.

**[2.10.3]**

- Bug Fixes
  - Clear interrupt status flags in EDMA\_CreateHandle to avoid triggering interrupt by mistake.

**[2.10.2]**

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

**[2.10.1]**

- Bug Fixes
  - Fixed EDMA\_GetRemainingMajorLoopCount may return wrong value issue.
  - Fixed violations of the MISRA C-2012 rules 13.5, 10.4.

### [2.10.0]

- Improvements
  - Modify the structures `edma_core_mp_t`, `edma_core_channel_t`, `edma_core_tcd_t` to adapt to `edma5`.
  - Add TCD register macro to facilitate confirmation of `tcd` type.
  - Modify the mask macro to a fixed value.
  - Add `EDMA_TCD_TYPE` macro to determine `edma tcd` type.
  - Add extension API to the following API to determine `edma tcd` type.
    - \* `EDMA_ConfigChannelSoftwareTCD` -> `EDMA_ConfigChannelSoftwareTCDExt`
    - \* `EDMA_TcdReset` -> `EDMA_TcdResetExt`
    - \* `EDMA_TcdSetTransferConfig` -> `EDMA_TcdSetTransferConfigExt`
    - \* `EDMA_TcdSetMinorOffsetConfig` -> `EDMA_TcdSetMinorOffsetConfigExt`
    - \* `EDMA_TcdSetChannelLink` -> `EDMA_TcdSetChannelLinkExt`
    - \* `EDMA_TcdSetBandWidth` -> `EDMA_TcdSetBandWidthExt`
    - \* `EDMA_TcdSetModulo` -> `EDMA_TcdSetModuloExt`
    - \* `EDMA_TcdEnableAutoStopRequest` -> `EDMA_TcdEnableAutoStopRequestExt`
    - \* `EDMA_TcdEnableInterrupts` -> `EDMA_TcdEnableInterruptsExt`
    - \* `EDMA_TcdDisableInterrupts` -> `EDMA_TcdDisableInterruptsExt`
    - \* `EDMA_TcdSetMajorOffsetConfig` -> `EDMA_TcdSetMajorOffsetConfigExt`

### [2.9.2]

- Improvements
  - Remove `tcd` alignment check in API that is low level and does not necessarily use scatter/gather mode.

### [2.9.1]

- Bug Fixes
  - Deinit channel request source before set channel mux.

### [2.9.0]

- Improvements
  - Release peripheral from reset if necessary in `init` function.
- Bug Fixes
  - Fixed the variable type definition error issue.
  - Fixed doxygen warning.
  - Fixed violations of MISRA C-2012 rule 18.1.

### [2.8.1]

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



**[2.8.0]**

- Improvements
  - Added feature FSL\_FEATURE\_EDMA\_HAS\_NO\_CH\_SBR\_SEC to separate DMA without SEC bitfield.

**[2.7.1]**

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.3, 10.4, 11.6, 11.8, 14.3,.

**[2.7.0]**

- Improvements
  - Use more accurate DMA instance based feature macros.
- New Features
  - Add new APIs EDMA\_PrepareTransferTCD and EDMA\_SubmitTransferTCD, which support EDMA transfer using TCD.

**[2.6.0]**

- Improvements
  - Modify the type of parameter channelRequestSource from dma\_request\_source\_t to int32\_t in the EDMA\_SetChannelMux.

**[2.5.3]**

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.3, 10.4, 11.6, 20.7, 12.2, 20.9, 5.3, 10.8, 8.4, 9.3.

**[2.5.2]**

- Improvements
  - Applied ERRATA 51327.

**[2.5.1]**

- Bug Fixes
  - Fixed the EDMA\_ResetChannel function cannot reset channel DONE/ERROR status.

**[2.5.0]**

- Improvements
  - Added feature FSL\_FEATURE\_EDMA\_HAS\_NO\_SBR\_ATTR\_BIT to separate DMA without ATTR bitfield.
  - Added api EDMA\_GetChannelSystemBusInformation to gets the channel identification and attribute information on the system bus interface.
- Bug Fixes
  - Fixed the ESG bit not set in scatter gather mode issue.

- Fixed the DREQ bit configuration missed in single transfer issue.
- Cleared the interrupt status before invoke callback to avoid miss interrupt issue.
- Removed `disableRequestAfterMajorLoopComplete` from `edma_transfer_config_t` structure as driver will handle it.
- Fixed the channel mux configuration not compatible issue.
- Fixed the out of bound access in function `EDMA_DriverIRQHandler`.

#### [2.4.4]

- Bug Fixes
  - Fixed comments by replacing STCD with TCD
  - Fixed the TCD overwrite issue when submit transfer request in the callback if there is a active TCD in hardware.

#### [2.4.3]

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

#### [2.4.2]

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

#### [2.4.1]

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

#### [2.4.0]

- Improvements
  - Added api `EDMA_EnableContinuousChannelLinkMode` to support continuous link mode.
  - Added apis `EDMA_SetMajorOffsetConfig/EDMA_TcdSetMajorOffsetConfig` to support major loop address offset feature.
  - Added api `EDMA_EnableChannelMinorLoopMapping` for minor loop offset feature.
  - Removed the redundant IRQ Handler in edma driver.

**[2.3.2]**

- Improvements
  - Fixed HIS ccm issue in function `EDMA_PrepareTransferConfig`.
  - Fixed violations of MISRA C-2012 rule 11.6, 10.7, 10.3, 18.1.
- Bug Fixes
  - Added ACTIVE & BITER & CITER bitfields to determine the channel status to fixed the issue of the transfer request cannot submit by function `EDMA_SubmitTransfer` when channel is idle.

**[2.3.1]**

- Improvements
  - Added source/destination address alignment check.
  - Added driver IRQ handler support for multi DMA instance in one SOC.

**[2.3.0]**

- Improvements
  - Added new api `EDMA_PrepareTransferConfig` to allow different configurations of width and offset.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.4, 10.1.
  - Fixed the Coverity issue regarding out-of-bounds write.

**[2.2.0]**

- Improvements
  - Added peripheral-to-peripheral support in EDMA driver.

**[2.1.9]**

- Bug Fixes
  - Fixed MISRA issue: Rule 10.7 and 10.8 in function `EDMA_DisableChannelInterrupts` and `EDMA_SubmitTransfer`.
  - Fixed MISRA issue: Rule 10.7 in function `EDMA_EnableAsyncRequest`.

**[2.1.8]**

- Bug Fixes
  - Fixed incorrect channel preemption base address used in `EDMA_SetChannelPreemptionConfig` API which causes incorrect configuration of the channel preemption register.

#### [2.1.7]

- Bug Fixes
  - Fixed incorrect transfer size setting.
    - \* Added 8 bytes transfer configuration and feature for RT series;
    - \* Added feature to support 16 bytes transfer for Kinetis.
  - Fixed the issue that EDMA\_HandleIRQ would go to incorrect branch when TCD was not used and callback function not registered.

#### [2.1.6]

- Bug Fixes
  - Fixed KW3X MISRA Issue.
    - \* Rule 14.4, 10.8, 10.4, 10.7, 10.1, 10.3, 13.5, and 13.2.
- Improvements
  - Cleared the IRQ handler unavailable for specific platform with macro FSL\_FEATURE\_EDMA\_MODULE\_CHANNEL\_IRQ\_ENTRY\_SHARED\_OFFSET.

#### [2.1.5]

- Improvements
  - Improved EDMA IRQ handler to support half interrupt feature.

#### [2.1.4]

- Bug Fixes
  - Cleared enabled request, status during EDMA\_Init for the case that EDMA is halted before reinitialization.

#### [2.1.3]

- Bug Fixes
  - Added clear DONE bit in IRQ handler to avoid overwrite TCD issue.
  - Optimized above solution for the case that transfer request occurs in callback.

#### [2.1.2]

- Improvements
  - Added interface to get next TCD address.
  - Added interface to get the unused TCD number.

#### [2.1.1]

- Improvements
  - Added documentation for eDMA data flow when scatter/gather is implemented for the EDMA\_HandleIRQ API.
  - Updated and corrected some related comments in the EDMA\_HandleIRQ API and edma\_handle\_t struct.

**[2.1.0]**

- Improvements
  - Changed the EDMA\_GetRemainingBytes API into EDMA\_GetRemainingMajorLoopCount due to eDMA IP limitation (see API comments/note for further details).

**[2.0.5]**

- Improvements
  - Added pubweak DriverIRQHandler for K32H844P (16 channels shared).

**[2.0.4]**

- Improvements
  - Added support for SoCs with multiple eDMA instances.
  - Added pubweak DriverIRQHandler for KL28T DMA1 and MCIMX7U5\_M4.

**[2.0.3]**

- Bug Fixes
  - Fixed the incorrect pubweak IRQHandler name issue, which caused re-definition build errors when client set his/her own IRQHandler, by changing the 32-channel IRQHandler name to DriverIRQHandler.

**[2.0.2]**

- Bug Fixes
  - Fixed incorrect minorLoopBytes type definition in \_edma\_transfer\_config struct, and defined minorLoopBytes as uint32\_t instead of uint16\_t.

**[2.0.1]**

- Bug Fixes
  - Fixed the eDMA callback issue (which did not check valid status) in EDMA\_HandleIRQ API.

**[2.0.0]**

- Initial version.
- 

**ENET****[2.11.0]**

- New Features
  - Added function ENET\_Ptp1588JumpTimer which adjusts the ENET PTP 1588 timer by jumping a relative time difference. Compared to ENET\_Ptp1588SetTimer, this function yields more accurate results when the relative time difference between the PTP clock and the target clock is known.

### [2.10.1]

- Bug Fixes
  - Fixed WAKEUP interrupt not being handled.

### [2.10.0]

- New Features
  - Added function `ENET_Ptp1588GetChannelCaptureValue` to read last captured time from PTP 1588 timer.

### [2.9.3]

- Bug Fixes
  - Fixed `ENET_Ptp1588GetTimer` incorrect timestamps when timer wraps occur after nanosecond capture:
    - \* Now only increments second field when nanosecond value is less than half a second

### [2.9.2]

- Bug Fixes
  - RGMII mode is (temporarily) disabled before selecting between 10/100-Mbit/s and 1000-Mbit/s modes of operation. The bit `RGMII_EN` of RCR register must not be set while changing ECR register's speed bit, otherwise there is a possibility of ENET IP ending in an incorrect state.

### [2.9.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 8.4, 10.4.

### [2.9.0]

- Bug Fixes
  - Enabled collection of transfer statistics, so the function `ENET_GetStatistics` does not always return zeroes.
- New Features
  - Added new function `ENET_EnableStatistics` to enable/disable collection of transfer statistics.
  - Added new function `ENET_ResetStatistics` to reset transfer statistics.
- Improvements
  - Renamed the function `ENET_ResetHareware` to `ENET_ResetHardware`.

### [2.8.0]

- New Features
  - Added the function to reset hardware on certain devices.

**[2.7.1]**

- Bug Fixes
  - Fixed the issue that free wrong buffer address when one frame stores in multiple buffers and memory pool is not enough to allocate these buffers to receive one complete frame.

**[2.7.0]**

- Improvements
  - Deleted deprecated zero copy Tx/Rx functions and set callback function which can be configured in ENET\_Init.
  - Moved the Rx zero copy buffer allocation to Rx BD initialization function to reduce unnecessary looping code.
- Bug Fixes
  - Fixed the issue that predefined Rx buffers which should not be used when enabling Rx zero copy are still be handled by cache operation, it causes hardfault on some platforms.
  - Fixed the issue that zero-copy Rx function doesn't check Rx length of 0 in the BD with EMPTY bit is 0, it may occur in the corner case reported by customer. Not sure how it turns out, consider it as an ENET IP issue and drop this abnormal BD.

**[2.6.3]**

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 11.6.

**[2.6.2]**

- Improvements
  - Changed ENET1\_MAC0\_Rx\_Tx\_Done0\_DriverIRQHandler/ENET1\_MAC0\_Rx\_Tx\_Done1\_DriverIRQHandler to ENET1\_MAC0\_Rx\_Tx\_Done1\_DriverIRQHandler/ENET1\_MAC0\_Rx\_Tx\_Done2\_DriverIRQHandler which represent ring 1 and ring 2.

**[2.6.1]**

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 10.4, 10.7, 11.6, 11.8.

**[2.6.0]**

- Improvements
  - Added MDIO access wrapper APIs for ease of use.
  - Fixed the build warning introduced by 64-bit compatibility patch.

**[2.5.4]**

- Improvements
  - Made the driver compatible with 64-bit platforms.

#### [2.5.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 11.6.

#### [2.5.2]

- Improvements
  - Updated the TXIC/RXIC register handling code according to the new header file.

#### [2.5.1]

- Bug Fixes
  - Fixed document typo.

#### [2.5.0]

- Bug Fixes
  - Fixed the SendFrame/SendFrameZeroCopy functions issue with scattered buffers.
  - Updated the formula of MDC calculation.
  - Used a feature macro to distinguish the old IP design from the new design, because old IP design always reads a value zero from ATCR->CAPTURE bit. For old IP, driver calculates and wait the necessary delay cycles after setting ATCR->CAPTURE then gets the timestamp value.
- New Features
  - Added new zero copy Tx/Rx function.
  - New zero copy Tx function combines scattered and contiguous Tx buffer in one API, it also supports more Tx features which buffer descriptor supports but previous Tx function doesn't support.
  - New zero copy Rx function use dynamic buffer mechanism and simpler interface.
- Improvements
  - Corrected the interrupt handler for PTP timestamp IRQ and PTP1588 event IRQ since platform difference.
  - Added missing IRQ handlers for PTP1588 events on some platforms.
  - Corrected the max Tx frame length verification, it will not depend on a fixed macro. The ENET\_FRAME\_MAX\_FRAMELEN is only an default value for driver, application can configure it. Driver calculates the limitation with the max frame length in register which may takes extended 4 or 8 bytes VLAN tag if VLAN/SVLAN enables.
  - Deleted deprecated Clause 45 read/write legacy APIs.

#### [2.4.3]

- Improvements
  - Aligned the IRQ handler name with header file.



**[2.4.2]**

- Bug Fixes
  - Fixed the MISRA issue of speculative out-of-bounds access.

**[2.4.1]**

- Bug Fixes
  - Fixed the PTP time capture issue.

**[2.4.0]**

- Improvements
  - Exposed API ENET\_ReclaimTxDescriptor for user application to reclaim tx descriptors in their application.
  - Added counter to record multicast hash conflict in struct \_enet\_handle, improved the situation that one multicast group could be left by other conflict multicast address left operation.
  - Improved concurrent usage of reclaim and send frame operation.

**[2.3.4]**

- Bug Fixes
  - Fixed the issue that interrupt handler only checks the interrupt event flag but not checks interrupt mask flag.

**[2.3.3]**

- Bug Fixes
  - Fixed the issue that some compilers may choose the memcpy with 4-bit aligned address limitation due to the type of address pointer is 'unsigned int \*', the data address doesn't have to be 4-bit aligned.

**[2.3.2]**

- New Features
  - Added the feature that ENET driver can be used in the platform which integrates both 10/100M and 1G ENET IP.
  - Deleted duplicated code about ARM errata 838869 in first/second level IRQ handler.

**[2.3.1]**

- Improvements
  - Added function pointer checking in IRQ handler to make sure code can be used even it runs into the interrupt when the second level interrupt handler is NULL.

### [2.3.0]

- Bug Fixes
  - Fixed the issue that clause 45 MDIO read/write API doesn't check the transmission over status between two transmissions.
  - Fixed violations of the MISRA C-2012 rules 2.2,10.3,10.4,10.7,11.6,11.8,13.5,14.4,15.7,17.7.
- New Features
  - Added APIs to support send/receive frame with Zero-Copy.
- Improvements
  - Separated the clock configuration from module configuration when init and deinit.
  - Added functions to set second level interrupt handler.
  - Provided new function to get 1588 timer count without disabling interrupt.
  - Improved timestamp controlling, deleted all old timestamp management APIs and data structures.
  - Merged the single/multiple ring(s) APIs, now these APIs can handle both.
  - Used base and index to control buffer descriptor, aligned with qos and lpc enet driver.

### [2.2.6]

- Bug Fixes
  - Updated MII speed formula referring to the manual.

### [2.2.5]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.4, 10.6, 10.7, 11.6, 11.9, 13.5, 14.4, 16.4, 17.7, 21.15, 3.1, 8.4.
  - Changed to use ARRAY\_SIZE(s\_enetBases) as the array size for s\_ENETHandle, fixed the hardfault issue for using some ENET instance when ARRAY\_SIZE(s\_enetBases) is not same as FSL\_FEATURE\_SOC\_ENET\_COUNT.

### [2.2.4]

- Improvements
  - Added call to Data Synchronization Barrier instruction before activating Tx/Rx buffer descriptor to ensure previous data update is completed.
  - Improved ENET\_TransmitIRQHandler to store timestamps for multiple transmit buffer descriptors.
- Bug Fixes
  - Fixed the issue that ENET\_Ptp1588GetTimer did not handle the timer wrap situation.

### [2.2.3]

- Improvements
  - Improved data buffer cache maintenance in the ENET driver.

### [2.2.2]

- New Features
  - Added APIs for extended multi-ring support.
  - Added the AVB configure API for extended AVB feature support.

### [2.2.1]

- Improvements
  - Changed the input data pointer attribute to const in ENET\_SendFrame().

### [2.1.1]

- New Features
  - Added the extended MDIO IEEE802.3 Clause 45 MDIO format SMI command APIs.
  - Added the extended interrupt coalescing feature.
- Improvements
  - Combined all storage operations in the ENET\_Init to ENET\_SetHandler API.

### [2.0.1]

- Bug Fixes
  - Used direct transmit busy check when doing data transmit.
- Miscellaneous Changes
  - Updated IRQ handler work flow.
  - Changed the TX/RX interrupt macro from kENET\_RxByteInterrupt to kENET\_RxBufferInterrupt, from kENET\_TxByteInterrupt to kENET\_TxBufferInterrupt.
  - Deleted unnecessary parameters in ENET handler.

### [2.0.0]

- Initial version.
- 

## ENET\_QOS

### [2.7.2]

- Bug Fixes
  - Not disabling ENET\_QOS\_ConfigureRxParser function when compiled for EMAC, as the MTL\_RXP\_INDIRECT\_ACC\_DATA register is writeable now.

### [2.7.1]

- Bug Fixes
  - Fixed writing after DMA\_CH array on platforms with smaller number of channels (EMAC).

#### [2.7.0]

- New features
  - The driver can work with EMAC IP now. EMAC is similar to ENET\_QOS but doesn't support 100 % of its features.

#### [2.6.5]

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

#### [2.6.4]

- Improvements
  - ENET\_QOS\_SetMII returns success or failure status now (related to i.MX RT1170 errata ERR050539 - ENET\_QOS doesn't support RMII 10Mbps mode).
- Bug Fixes
  - Fixed the MISRA C-2012 issue rule 14.3.

#### [2.6.3]

- Bug Fixes
  - Fixed the issue that ENET\_QOS\_GetRxFram, ENET\_QOS\_ReadFrame and ENET\_QOS\_DropFrame did not properly restart the receiving once it stopped.

#### [2.6.2]

- Bug Fixes
  - Fixed the issue that free wrong buffer address when one frame stores in multiple buffers and memory pool is not enough to allocate these buffers to receive one complete frame.

#### [2.6.1]

- Bug Fixes
  - Fixed the issue that ENET\_QOS\_ReadFrame doesn't check timestamp available bit before check the context BD bit, it makes software update extra BD. If DMA receives new frame to this BD before software update, software will lose this frame.

#### [2.6.0]

- New features
  - Added hardware checksum acceleration support.

#### [2.5.3]

- Bug Fixes
  - Fixed the MISRA issue rule 14.3, 5.3.

**[2.5.2]**

- Bug Fixes
  - Fixed the issue that ENET\_QOS\_Init reset the MDIO setting of ENET\_QOS\_SetSML.

**[2.5.1]**

- Improvements
  - Supported RMII mode.

**[2.5.0]**

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

**[2.4.1]**

- Improvements
  - Supported cache control.
  - Supported BD address conversion to system address.
  - Make driver aarch64 compatible
- Bug Fixes
  - Fixed the issue that driver internal interface ENET\_QOS\_DropFrame drops all frames in whole BD ring rather than one frame as design. Impact case: 1. Rx drop occurs in zero copy Rx API ENET\_QOS\_GetRxFrame. 2. Call ENET\_QOS\_ReadFrame with data pointer is NULL, driver will drop all Rx frames.

**[2.4.0]**

- New features
  - Added MDIO IEEE802.3 Clause 45 access support.
  - Added get statistics API to get some statistical data in transfer.
  - Added new APIs to support zero copy Rx.
  - Fixed the MISRA issue rule 8.4, 8.6.

**[2.3.0]**

- Improvements
  - Added counter to record multicast hash conflict in struct \_enet\_handle, improved the situation that one multicast group could be left by other conflict multicast address left operation.
- Bug Fixes
  - Updated txDirtyRing maintenance in reclaim and send frame process, allow txDirtyRing to be overwritten.
  - Disabled carrier sensing in full duplex mode configuration in ethernet initialization
  - Fixed 1588 sub-second calculate issue.

### [2.2.2]

- Bug Fixes
  - Fixed the issue that ENET\_QOS\_SetupTxDescriptor didn't handle the DMA access address mapping for SoCs have feature FSL\_FEATURE\_MEMORY\_HAS\_ADDRESS\_OFFSET.
  - Fixed MISRA 2012 violations detected in examples build.

### [2.2.1]

- Bug Fixes
  - Fixed MISRA 2012 violations, fixed doxygen warning.
  - Fixed the issue that cache invalidate to invalid converted memory address in ENET\_QOS\_ReadFrame for SoCs have feature FSL\_FEATURE\_MEMORY\_HAS\_ADDRESS\_OFFSET.

### [2.2.0]

- Removed the ptp time data ring management, below structures and APIs are removed:
  - structure enet\_qos\_ptp\_time\_data\_t
  - structure enet\_qos\_ptp\_time\_data\_ring\_t
  - API ENET\_QOS\_GetRxFrameTime
  - API ENET\_QOS\_GetTxFrameTime
- Added API for GCL list read and AVB configuration
  - ENET\_QOS\_EstReadGcl
  - ENET\_QOS\_AVBConfigure
- Improved driver for PTP system time configuration, timestamp read.
- Added IRQ lock and memory barrier instruction for descriptor operation.
- Fixed MISRA 2012 violations

### [2.1.1]

- Bug Fixes
  - Fixed the bug that data pointer is not converted to local memory address in the call to ENET\_QOS\_Ptp1588ParseFrame.

### [2.1.0]

- New feature
  - Update driver to support feature FSL\_FEATURE\_MEMORY\_HAS\_ADDRESS\_OFFSET which convert buffer address to visible address for DMA.
  - Require user to provide implementation for ENET\_QOS\_SetSYSControl API, which set the PHY interface and enable clock generation for IP.

### [2.0.0]

- Initial version.

## FLEXCAN

### [2.14.5]

- Improvements
  - Make API FLEXCAN\_GetFDMailboxOffset **public**.
  - Add API FLEXCAN\_SetMbID and FLEXCAN\_SetFDMbID to configure Message Buffer ID individually.
- Bug Fixes
  - Fixed violations of the CERT INT30-C INT31-C.
  - Fixed violations of the CERT ARR30-C.

### [2.14.4]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 8.4, 10.1, 10.4, 18.1.

### [2.14.3]

- Improvements
  - Add unhandled interrupt events check for following API:
    - \* FLEXCAN\_MbHandleIRQ
    - \* FLEXCAN\_EhancedRxFifoHandleIRQ
- Bug Fixes
  - Remove FLEXCAN\_MemoryErrorHandleIRQ on some platform without memory error interrupt.
  - Add conditional compile for CTRL2[ISOCANFDEN] because some platform do not have this bit.

### [2.14.2]

- Improvements
  - Add Coverage Justification for uncovered code.
  - Adjust API FLEXCAN\_TransferAbortReceive **order**.
  - Update FLEXCAN\_Enable to enter Freeze Mode first when enter Disable mode on some platform.
  - Added while loop timeout for following API:
    - \* FLEXCAN\_EnterFreezeMode
    - \* FLEXCAN\_ExitFreezeMode
    - \* FLEXCAN\_Enable
    - \* FLEXCAN\_Reset
    - \* FLEXCAN\_TransferSendBlocking
    - \* FLEXCAN\_TransferReceiveBlocking
    - \* FLEXCAN\_TransferFDSendBlocking
    - \* FLEXCAN\_TransferFDReceiveBlocking

- \* FLEXCAN\_TransferReceiveFifoBlocking
- \* FLEXCAN\_TransferReceiveEnhancedFifoBlocking
- Bug Fixes
  - Remove remote frame feature in CANFD mode because there is no remote frame in the CANFD format.
  - Remove legacy Rx FIFO disabled branch in FLEXCAN\_SubHandlerForLegacyRxFIFO and FLEXCAN\_SubHandlerForDataTransferred.

#### [2.14.1]

- Bug Fixes
  - Fixed register IMASK2-4 IFLAG2-4 HR\_TIME\_STAMP<sub>n</sub> access issue on FlexCAN instances with different number of MBs.
  - Fixed bit field MBDSR1-3 access issue on FlexCAN instances with different number of MBs.
- Improvements
  - Unified following API as same parameter and return value type:
    - \* FLEXCAN\_GetMbStatusFlags
    - \* FLEXCAN\_ClearMbStatusFlags
    - \* FLEXCAN\_EnableMbInterrupts
    - \* FLEXCAN\_DisableMbInterrupts
  - Add workaround for ERR050443 and ERR052403.
  - Update message buffer read process in API FLEXCAN\_ReadRxMb and FLEXCAN\_ReadFDRxMb to make critical section as short as possible.
  - Simplify API FLEXCAN\_DriverDataIRQHandler implementation by remove parameter type.

#### [2.14.0]

- Improvements
  - Support external time tick feature.
  - Support high resolution timestamp feature.
  - Enter Freeze Mode first when enter Disable Mode on some platform.
  - Add feature macro for Pretended Networking because some FlexCAN instance do not have this feature.
  - Add feature macro for enhanced Rx FIFO because some FlexCAN instance do not have this feature.
  - Add new FlexCAN IRQ Handler FLEXCAN\_DriverDataIRQHandler and FLEXCAN\_DriverEventIRQHandler. These IRQ Handlers are used on soc which FlexCAN interrupts are grouped by specific function and assigned to different vector.
  - Update macro FLEXCAN\_WAKE\_UP\_FLAG and FLEXCAN\_PNWAKE\_UP\_FLAG to simplify code.
  - Replace macro FSL\_FEATURE\_FLEXCAN\_HAS\_NO\_WAKMSK\_SUPPORT with FSL\_FEATURE\_FLEXCAN\_HAS\_NO\_SLFWAK\_SUPPORT.



- Replace macro `FSL_FEATURE_FLEXCAN_HAS_NO_WAKSRC_SUPPORT` with `FSL_FEATURE_FLEXCAN_HAS_GLITCH_FILTER`.
- Bug Fixes
  - Fixed wrong interrupt and status flag helper macro in enumeration `_flexcan_flags` and API `FLEXCAN_DisableInterrupts`.
  - Fixed interrupt flag helper macro typo issue.
  - Remove flags which will be unassociated with interrupt in macro `FLEXCAN_MEMORY_ERROR_INT_FLAG`.
  - Remove flags which will be unassociated with interrupt in macro `FLEXCAN_ERROR_AND_STATUS_INT_FLAG`.
  - Fixed array out-of-bounds access when read enhanced Rx FIFO.

#### [2.13.1]

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

#### [2.13.0]

- Improvements
  - Support payload endianness selection feature.

#### [2.12.0]

- Improvements
  - Support automatic Remote Response feature.
  - Add API `FLEXCAN_SetRemoteResponseMbConfig()` to configure automatic Remote Response mailbox.

#### [2.11.8]

- Improvements
  - Synchronize flexcan driver update on s32z platform.

#### [2.11.7]

- Bug Fixes
  - Fixed `FLEXCAN_TransferReceiveEnhancedFifoEDMA()` compatibility with edma5.

#### [2.11.6]

- Bug Fixes
  - Fixed ERRATA\_9595 `FLEXCAN_EnterFreezeMode()` may result to bus fault on some platform.

#### [2.11.5]

- Bug Fixes
  - Fixed flexcan\_memset() crash under high optimization compilation.

#### [2.11.4]

- Improvements
  - Update CANFD max bitrate to 10Mbps on MCXNx3x and MCXNx4x.
  - Release peripheral from reset if necessary in init function.

#### [2.11.3]

- Bug Fixes
  - Fixed FLEXCAN\_TransferReceiveEnhancedFifoEDMA() compile error with DMA3.

#### [2.11.2]

- Bug Fixes
  - Fixed bug that timestamp in flexcan\_handle\_t not updated when RX overflow happens.

#### [2.11.1]

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

#### [2.11.0]

- Bug Fixes
  - Fixed wrong base address argument in FLEXCAN2 IRQ Handler.
- Improvements
  - Add API to determine if the instance supports CAN FD mode at run time.

#### [2.10.1]

- Bug Fixes
  - Fixed HIS CCM issue.
  - Fixed RTOS issue by adding protection to read-modify-write operations on interrupt enable/disable API.

#### [2.10.0]

- Improvements
  - Update driver to make it able to support devices which has more than 64 8bytes MBs.
  - Update CAN FD transfer APIs to make them set/get edl bit according to frame content, which can make them compatible with classic CAN.

### [2.9.2]

- Bug Fixes
  - Fixed the issue that FLEXCAN\_CheckUnhandleInterruptEvents() can't detecting the exist enhanced RX FIFO interrupt status.
  - Fixed the issue that FLEXCAN\_ReadPNWakeUpMB() does not return fail even no existing valid wake-up frame.
  - Fixed the issue that FLEXCAN\_ReadEnhancedRxFifo() may clear bits other than the data available bit.
  - Fixed violations of the MISRA C-2012 rules 10.4, 10.8.
- Improvements
  - Return kStatus\_FLEXCAN\_RxFifoDisabled instead of kStatus\_Fail when read FIFO fail during IRQ handler.
  - Remove unreachable code from timing calculates APIs.
  - Update Enhanced Rx FIFO handler to make it deal with underflow/overflow status first.

### [2.9.1]

- Bug Fixes
  - Fixed the issue that FLEXCAN\_TransferReceiveEnhancedFifoBlocking() API clearing Fifo data available flag more than once.
  - Fixed the issue that entering FLEXCAN\_SubHandlerForEnhancedRxFifo() even if Enhanced Rx fifo interrupts are not enabled.
  - Fixed the issue that FLEXCAN\_TransferReceiveEnhancedFifoEDMA() update handle even if previous Rx FIFO receive not finished.
  - Fixed the issue that FLEXCAN\_SetEnhancedRxFifoConfig() not configure the ERFCR[NFE] bits to the correct value.
  - Fixed the issue that FLEXCAN\_ReceiveFifoEDMACallback() can't differentiate between Rx fifo and enhanced rx fifo.
  - Fixed the issue that FLEXCAN\_TransferHandleIRQ() can't report Legacy Rx FIFO warning status.

### [2.9.0]

- Improvements
- Add public set bit rate API to make driver easier to use.
- Update Legacy Rx FIFO transfer APIs to make it support received multiple frames during one API call.
- Optimized FLEXCAN\_SubHandlerForDataTransferred() API in interrupt handling to reduce the probability of packet loss.

### [2.8.7]

- Improvements
- Initialized the EDMA configuration structure in the FLEXCAN EDMA driver.

#### [2.8.6]

- Bug Fixes
  - Fix Coverity overrun issues in fsl\_flexcan\_edma driver.

#### [2.8.5]

- Improvements
  - Make driver aarch64 compatible.

#### [2.8.4]

- Bug Fixes
  - Fixed FlexCan\_Errata\_6032 to disable all interrupts.

#### [2.8.3]

- Bug Fixes
  - Fixed an issue with the FLEXCAN\_EnableInterrupts and FLEXCAN\_DisableInterrupts interrupt enable bits in the CTRL1 register.

#### [2.8.2]

- Bug Fixes
  - Fixed errors in timing calculations and simplify the calculation process.
  - Fixed issue of CBT and FDCBT register may write failure.

#### [2.8.1]

- Bug Fixes
  - Fixed the issue of CAN FD three sampling points.
  - Added macro to support the devices that no MCR[SUPV] bit.
  - Remove unnecessary clear WMB operations.

#### [2.8.0]

- Improvements
  - Update config configuration.
    - \* Added enableSupervisorMode member to support enable/disable Supervisor mode.
  - Simplified the algorithm in CAN FD improved timing APIs.

#### [2.7.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 10.7.

### [2.7.0]

- Improvements
  - Update config configuration.
    - \* Added enablePretendedNetworking member to support enable/disable Pretended Networking feature.
    - \* Added enableTransceiverDelayMeasure member to support enable/disable Transceiver Delay MeasurementPretended feature.
    - \* Added bitRate/bitRateFD member to work as baudRate/baudRateFD member union.
  - Rename all “baud” in code or comments to “bit” to align with the CAN spec.
  - Added Pretended Networking mode related APIs.
    - \* FLEXCAN\_SetPNConfig
    - \* FLEXCAN\_GetPNMatchCount
    - \* FLEXCAN\_ReadPNWakeUpMB
  - Added support for Enhanced Rx FIFO.
  - Removed independent memory error interrupt/status APIs and put all interrupt/status control operation into FLEXCAN\_EnableInterrupts/FLEXCAN\_DisableInterrupts and FLEXCAN\_GetStatusFlags/FLEXCAN\_ClearStatusFlags APIs.
  - Update improved timing APIs to make it calculate improved timing according to CiA doc recommended.
    - \* FLEXCAN\_CalculateImprovedTimingValues.
    - \* FLEXCAN\_FDCalculateImprovedTimingValues.
  - Update FLEXCAN\_SetBitRate/FLEXCAN\_SetFDBitRate to added the use of enhanced timing registers.

### [2.6.2]

- Improvements
  - Add CANFD frame data length enumeration.

### [2.6.1]

- Bug Fixes
  - Fixed the issue of not fully initializing memory in FLEXCAN\_Reset() API.

### [2.6.0]

- Improvements
  - Enable CANFD ISO mode in FLEXCAN\_FDInit API.
  - Enable the transceiver delay compensation feature when enable FD operation and set bitrate switch.
  - Implementation memory error control in FLEXCAN\_Init API.
  - Improve FLEXCAN\_FDCalculateImprovedTimingValues API to get same value for FPRESDIV and PRES DIV.
  - Added memory error configuration for user.

- \* enableMemoryErrorControl
- \* enableNonCorrectableErrorEnterFreeze
- Added memory error related APIs.
  - \* FLEXCAN\_GetMemoryErrorReportStatus
  - \* FLEXCAN\_GetMemoryErrorStatusFlags
  - \* FLEXCAN\_ClearMemoryErrorStatusFlags
  - \* FLEXCAN\_EnableMemoryErrorInterrupts
  - \* FLEXCAN\_DisableMemoryErrorInterrupts
- Bug Fixes
  - Fixed the issue of sent duff CAN frame after call FLEXCAN\_FDInit() API.

### [2.5.2]

- Bug Fixes
  - Fixed the code error issue and simplified the algorithm in improved timing APIs.
    - \* The bit field in CTRL1 register couldn't calculate higher ideal SP, we set it as the lowest one(75%)
      - FLEXCAN\_CalculateImprovedTimingValues
      - FLEXCAN\_FDCalculateImprovedTimingValues
  - Fixed MISRA-C 2012 Rule 17.7 and 14.4.
- Improvements
  - Pass EsrStatus to callback function when kStatus\_FLEXCAN\_ErrorStatus is coming.

### [2.5.1]

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

### [2.5.0]

- Bug Fixes
  - MISRA C-2012 issue check.
    - \* Fixed rules, containing: rule-10.1, rule-10.3, rule-10.4, rule-10.7, rule-10.8, rule-11.8, rule-12.2, rule-13.4, rule-14.4, rule-15.5, rule-15.6, rule-15.7, rule-16.4, rule-17.3, rule-5.8, rule-8.3, rule-8.5.
  - Fixed the issue that API FLEXCAN\_SetFDRxMbConfig lacks inactive message buff.
  - Fixed the issue of Pa082 warning.
  - Fixed the issue of dead lock in the function of interruption handler.
  - Fixed the issue of Legacy Rx Fifo EDMA transfer data fail in evkmimxrt1060 and evk-mimxrt1064.
  - Fixed the issue of setting CANFD Bit Rate Switch.

- Fixed the issue of operating unknown pointer risk.
  - \* when used the pointer “handle->mbFrameBuf[mbIdx]” to update the timestamp in a short-live TX frame, the frame pointer became as unknown, the action of operating it would result in program stack destroyed.
- Added assert to check current CAN clock source affected by other clock gates in current device.
  - \* In some chips, CAN clock sources could be selected by CCM. But for some clock sources affected by other clock gates, if user insisted on using that clock source, they had to open these gates at the same time. However, they should take into consideration the power consumption issue at system level. In RT10xx chips, CAN clock source 2 was affected by the clock gate of lpuart1. ERRATA ID: (ERR050235 in CCM).
- Improvements
  - Implementation for new FLEXCAN with ECC feature able to exit Freeze mode.
  - Optimized the function of interruption handler.
  - Added two APIs for FLEXCAN EDMA driver.
    - \* FLEXCAN\_PrepareTransfConfiguration
    - \* FLEXCAN\_StartTransferDatafromRxFIFO
  - Added new API for FLEXCAN driver.
    - \* FLEXCAN\_GetTimeStamp
      - For TX non-blocking API, we wrote the frame into mailbox only, so no need to register TX frame address to the pointer, and the timestamp could be updated into the new global variable handle->timestamp[mbIdx], the FLEXCAN driver provided a new API for user to get it by handle and index number after TX DONE Success.
    - \* FLEXCAN\_EnterFreezeMode
    - \* FLEXCAN\_ExitFreezeMode
  - Added new configuration for user.
    - \* disableSelfReception
    - \* enableListenOnlyMode
  - Renamed the two clock source enum macros based on CLKSRC bit field value directly.
    - \* The CLKSRC bit value had no property about Oscillator or Peripheral type in lots of devices, it acted as two different clock input source only, but the legacy enum macros name contained such property, that misled user to select incorrect CAN clock source.
  - Created two new enum macros for the FLEXCAN driver.
    - \* kFLEXCAN\_ClkSrc0
    - \* kFLEXCAN\_ClkSrc1
  - Deprecated two legacy enum macros for the FLEXCAN driver.
    - \* kFLEXCAN\_ClkSrcOsc
    - \* kFLEXCAN\_ClkSrcPeri
  - Changed the process flow for Remote request frame response..
    - \* Created a new enum macro for the FLEXCAN driver.
      - kStatus\_FLEXCAN\_RxRemote

- Changed the process flow for kFLEXCAN\_StateRxRemote state in the interrupt handler.
  - \* Should the TX frame not register to the pointer of frame handle, interrupt handler would not be able to read the remote response frame from the mail box to ram, so user should read the frame by manual from mail box after a complete remote frame transfer.

#### [2.4.0]

- Bug Fixes
  - MISRA C-2012 issue check.
    - \* Fixed rules, containing: rule-12.1, rule-17.7, rule-16.4, rule-11.9, rule-8.4, rule-14.4, rule-10.8, rule-10.4, rule-10.3, rule-10.7, rule-10.1, rule-11.6, rule-13.5, rule-11.3, rule-8.3, rule-12.2 and rule-16.1.
  - Fixed the issue that CANFD transfer data fail when bus baudrate is 30Khz.
  - Fixed the issue that ERR009595 does not follow the ERRATA document.
  - Fixed code error for ERR006032 work around solution.
  - Fixed the Coverity issue of BAD\_SHIFT in FLEXCAN.
  - Fixed the Repo build warning issue for variable without initial.
- Improvements
  - Fixed the run fail issue of FlexCAN RemoteRequest UT Case.
  - Implementation all TX and RX transferring Timestamp used in FlexCAN demos.
  - Fixed the issue of UT Test Fail for CANFD payload size changed from 64BperMB to 8PerMB.
  - Implementation for improved timing API by baud rate.

#### [2.3.2]

- Improvements
  - Implementation for ERR005959.
  - Implementation for ERR005829.
  - Implementation for ERR006032.

#### [2.3.1]

- Bug Fixes
  - Added correct handle when kStatus\_FLEXCAN\_TxSwitchToRx is coming.

#### [2.3.0]

- Improvements
  - Added self-wakeup support for STOP mode in the interrupt handling.

#### [2.2.3]

- Bug Fixes
  - Fixed the issue of CANFD data phase's bit rate not set as expected.



**[2.2.2]**

- Improvements
  - Added a time stamp feature and enable it in the interrupt\_transfer example.

**[2.2.1]**

- Improvements
  - Separated CANFD initialization API.
  - In the interrupt handling, fix the issue that the user cannot use the normal CAN API when with an FD.

**[2.2.0]**

- Improvements
  - Added FSL\_FEATURE\_FLEXCAN\_HAS\_SUPPORT\_ENGINE\_CLK\_SEL\_REMOVE feature to support SoCs without CAN Engine Clock selection in FlexCAN module.
  - Added FlexCAN Serial Clock Operation to support i.MX SoCs.

**[2.1.0]**

- Bug Fixes
  - Corrected the spelling error in the function name FLEXCAN\_XXX().
  - Moved Freeze Enable/Disable setting from FLEXCAN\_Enter/ExitFreezeMode() to FLEXCAN\_Init().
  - Corrected wrong helper macro values.
- Improvements
  - Hid FLEXCAN\_Reset() from user.
  - Used NDEBUG macro to wrap FLEXCAN\_IsMbOccupied() function instead of DEBUG macro.

**[2.0.0]**

- Initial version.
- 

**FLEXCAN\_EDMA****[2.12.1]**

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 18.1.

#### [2.12.0]

- Improvements
  - Support high resolution timestamp feature in enhanced Rx FIFO EDMA.
  - Add feature macro for enhanced Rx FIFO because some FlexCAN instance do not have this feature.
- Bug Fixes
  - Fixed array out-of-bounds access when read enhanced Rx FIFO in EDMA.

#### [2.11.7]

- Refer FLEXCAN driver change log 2.7.0 to 2.11.7
- 

### FLEXIO

#### [2.3.0]

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

#### [2.2.3]

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

#### [2.2.2]

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

#### [2.2.1]

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

#### [2.2.0]

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

#### [2.1.0]

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

**[2.0.4]**

- Bug Fixes
  - Fixed MISRA 8.4 issues.

**[2.0.3]**

- Bug Fixes
  - Fixed MISRA 10.4 issues.

**[2.0.2]**

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

**[2.0.1]**

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

**FLEXIO\_I2C****[2.6.2]**

- Improvements
  - Added timeout for while loop in FLEXIO\_I2C\_MasterTransferBlocking().
- Bug Fixes
  - Fixed build issues related to I2C\_RETRY\_TIMES.

**[2.6.1]**

- Bug Fixes
  - Fixed coverity issues

**[2.6.0]**

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

#### [2.5.1]

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

#### [2.5.0]

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

#### [2.4.0]

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

#### [2.3.0]

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

#### [2.2.0]

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

**[2.1.7]**

- Bug Fixes
  - Fixed the issue that FLEXIO\_I2C\_MasterTransferBlocking did not wait for STOP bit sent.
  - Fixed COVERITY issue of useless call in FLEXIO\_I2C\_MasterTransferRunStateMachine.
  - Fixed the issue that I2C master did not check whether bus was busy before transfer.

**[2.1.6]**

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

**[2.1.5]**

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

**[2.1.4]**

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

**[2.1.3]**

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

**[2.1.2]**

- Bug Fixes
  - Fixed the FLEXIO I2C issue where the master could not receive data from I2C slave in high baudrate.
  - Fixed the FLEXIO I2C issue where the master could not receive NAK when master sent non-existent addr.
  - Fixed the FLEXIO I2C issue where the master could not get transfer count successfully.
  - Fixed the FLEXIO I2C issue where the master could not receive data successfully when sending data first.
  - Fixed the Dozen mode configuration error in FLEXIO\_I2C\_MasterInit API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.

- Fixed the issue that FLEXIO\_I2C\_MasterTransferBlocking API called FLEXIO\_I2C\_MasterTransferCreateHandle, which lead to the s\_flexioHandle/s\_flexioIsr/s\_flexioType variable being written. Then, if calling FLEXIO\_I2C\_MasterTransferBlocking API multiple times, the s\_flexioHandle/s\_flexioIsr/s\_flexioType variable would not be written any more due to it being out of range. This lead to the following situation: NonBlocking transfer APIs could not work due to the fail of register IRQ.

#### [2.1.1]

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

#### [2.1.0]

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

### FLEXIO\_I2S

#### [2.2.2]

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

#### [2.2.1]

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

#### [2.2.0]

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

#### [2.1.6]

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

**[2.1.5]**

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

**[2.1.4]**

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

**[2.1.3]**

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

**[2.1.2]**

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

**[2.1.1]**

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

**[2.1.0]**

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

**FLEXIO\_I2S\_EDMA****[2.1.9]**

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

#### [2.1.8]

- Improvements
    - Applied EDMA ERRATA 51327.
- 

### FLEXIO\_SPI

#### [2.4.3]

- Improvements
  - Make SPI\_RETRY\_TIMES configurable by CONFIG\_SPI\_RETRY\_TIMES.

#### [2.4.2]

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

#### [2.4.1]

- Bug Fixes
  - Fixed coverity issues

#### [2.4.0]

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

#### [2.3.5]

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

#### [2.3.4]

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

#### [2.3.3]

- Bugfixes
  - Fixed cs-continuous mode.

#### [2.3.2]

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



### [2.3.1]

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

### [2.3.0]

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

### [2.2.1]

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

### [2.2.0]

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

### [2.1.3]

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

### [2.1.2]

- Bug Fixes
  - The following modification support FlexIO using multiple instances:
    - \* Removed FLEXIO\_Reset API in module Init APIs.

- \* Updated module Deinit APIs to reset the shifter/timer config instead of disabling module/clock.
- \* Updated module Enable APIs to only support enable operation.

#### [2.1.1]

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

#### [2.1.0]

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

## FLEXIO\_UART

#### [2.6.4]

- Improvements
  - Make UART\_RETRY\_TIMES configurable by CONFIG\_UART\_RETRY\_TIMES.

#### [2.6.3]

- Bug Fixes
  - Fixed coverity issues

#### [2.6.2]

- Bug Fixes
  - Fixed coverity issues

**[2.6.1]**

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

**[2.6.0]**

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

**[2.5.1]**

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

**[2.5.0]**

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

**[2.4.0]**

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

**[2.3.0]**

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

**[2.2.0]**

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

#### [2.1.6]

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

#### [2.1.5]

- Improvements
  - Triggered user callback after all the data in ringbuffer were received in FLEXIO\_UART\_TransferReceiveNonBlocking.

#### [2.1.4]

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

#### [2.1.3]

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

#### [2.1.2]

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

#### [2.1.1]

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

### [2.1.0]

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

## FLEXIO\_UART\_EDMA

### [2.3.1]

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

### [2.3.0]

- Refer FLEXIO\_UART driver change log to 2.3.0
- 

## FLEXSPI

### [2.8.1]

- Improvements
  - Updated the LUT configuration parameter checking with flexible way to adapt different Socs.

### [2.8.0]

- Bug Fixes
  - Introduced the **disableAhbReadResume** field in the flexspi\_config\_t structure to provide control over the AHBCR[RESUMEDISABLE] register bit.
  - Implemented a workaround for hardware erratum ERR052733 by setting the default value of **disableAhbReadResume** to **true**.
  - Fixed issue in FLEXSPI\_TransferHandleIRQ where the transfer completion was incorrectly signaled despite pending read/write operations.
- New Features
  - Introduced a new function(FLEXSPI\_UpdateAhbBuffersSettings) that allows users to update the AHB buffer configuration after the FLEXSPI module has been initialized

### [2.7.0]

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

#### [2.6.4]

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

#### [2.6.3]

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

#### [2.6.2]

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

#### [2.6.1]

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

#### [2.6.0]

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

#### [2.5.1]

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

#### [2.5.0]

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

#### [2.4.0]

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

#### [2.3.5]

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

#### [2.3.4]

- Bug Fixes
  - Updated flexspi\_config\_t structure and FlexSPI\_Init to support new feature FSL\_FEATURE\_FLEXSPI\_HAS\_NO\_MCR0\_CONBINATION.

#### [2.3.3]

- Bug Fixes
  - Removed feature FSL\_FEATURE\_FLEXSPI\_DQS\_DELAY\_PS for DLL delay setting. Changed to use feature FSL\_FEATURE\_FLEXSPI\_DQS\_DELAY\_MIN to set slave delay target as 0 for DLL enable and clock frequency higher than 100MHz.

#### [2.3.2]

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

#### [2.3.1]

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

#### [2.3.0]

- New Features
  - Added new API FLEXSPI\_UpdateDllValue for users to update DLL value after updating flexspi root clock.
  - Corrected grammatical issues for comments.
  - Added support for new feature FSL\_FEATURE\_FLEXSPI\_DQS\_DELAY\_PS in DLL configuration.

### [2.2.2]

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

### [2.2.1]

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

### [2.2.0]

- Bug Fixes
  - Fixed flag name typos: `kFLEXSPI_IpTxFifoWatermarkEmptyFlag` to `kFLEXSPI_IpTxFifoWatermarkEmptyFlag`; `kFLEXSPI_IpCommandExcutionDoneFlag` to `kFLEXSPI_IpCommandExecutionDoneFlag`.
  - Fixed comments typos such as `sequencen->sequence`, `levle->level`.
  - Fixed `FLSHCR2[ARDSEQID]` field clean issue.
  - Updated `flexspi_config_t` structure and `FlexSPI_Init` to support new feature `FSL_FEATURE_FLEXSPI_HAS_NO_MCR0_ATDFEN` and `FSL_FEATURE_FLEXSPI_HAS_NO_MCR0_ARDFEN`.
  - Updated `flexspi_flags_t` structure to support new feature `FSL_FEATURE_FLEXSPI_HAS_INTEN_AHBBUSERROREN`.

### [2.1.1]

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

### [2.1.0]

- New Features
  - Added new API `FLEXSPI_UpdateRxSampleClock` for users to update read sample clock source after initialization.
  - Added reset peripheral operation in `FLEXSPI_Init` if required.

### [2.0.5]

- Bug Fixes
  - Fixed `FLEXSPI_UpdateLUT` cannot do partial update issue.



**[2.0.4]**

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

**[2.0.3]**

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

**[2.0.2]**

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

**[2.0.1]**

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

**[2.0.0]**

- Initial version.
- 

**FLEXSPI EDMA Driver****[2.3.3]**

- Bug Fixes
  - Fixed FLEXSPI\_TransferEDMA bug that, the DMA channel not configured correctly when using kFLEXSPI\_Read.

### [2.3.2]

- Bug Fixes
  - Fixed the bug that internal variable `s_edmaPrivateHandle` overflows when using FlexSPI2.

### [2.0.2]

- New Features
  - Provided new API `FLEXSPI_TransferUpdateSizeEDMA` for users to update eDMA transfer size(SSIZE/DSIZE) per DMA transfer.

### [2.0.0]

- Initial version.
- 

## I3C

### [2.14.3]

- Improvements
  - Fixed Coverity CERT-C violations.
  - Used `I3C_RSTS` instead of I3C special feature macro.
  - Adapted the driver to support new platform.

### [2.14.2]

- Improvements
  - Added timeout for ENTDAAs process API.
  - Added build system macro to control the timeout setting.

### [2.14.1]

- Improvements
  - Split the function `I3C_MasterTransferBlocking` to meet the HIS-CCM requirement.

### [2.14.0]

- Improvements
  - Added the choice to set fast start header with push-pull speed when all targets addresses have MSB 0 instead of forcing to set it.
  - Deleted duplicated busy check in `I3C_MasterStart` function.

### [2.13.1]

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

### [2.13.0]

- New features
  - Added the hot-join support for I3C bus initialization API.
- Bug Fixes
  - Set read termination with START at the same time in case unknown issue.
  - Set MCTRL[TYPE] as 0 for DDR force exit.
- Improvements
  - Added the API to reset device count assigned by ENTDAAs.
  - Provided the method to set global macro I3C\_MAX\_DEVCNT to determine how many device addresses ENTDAAs can allocate at one time.
  - Initialized target management static array based on instance number for the case that multiple instances are used at the same time.

### [2.12.0]

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

### [2.11.1]

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

### [2.11.0]

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

#### [2.10.8]

- Improvements
  - Support more instances.

#### [2.10.7]

- Improvements
  - Fixed the potential compile warning.

#### [2.10.6]

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

#### [2.10.5]

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

#### [2.10.4]

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

#### [2.10.3]

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

#### [2.10.2]

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

#### [2.10.1]

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

#### [2.10.0]

- New features
  - Added I3C extended IBI data support.

**[2.9.0]**

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

**[2.8.2]**

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

**[2.8.1]**

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

**[2.8.0]**

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

**[2.7.1]**

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

**[2.7.0]**

- Fixed the CCM problem in file fsl\_i3c.c.
- Fixed the FSL\_FEATURE\_I3C\_HAS\_NO\_SCONFIG\_IDRAND usage issue in I3C\_GetDefaultConfig and I3C\_Init.

**[2.6.0]**

- Fixed the FSL\_FEATURE\_I3C\_HAS\_NO\_SCONFIG\_IDRAND usage issue in fsl\_i3c.h.
- Changed some static functions in fsl\_i3c.c as non-static and define the functions in fsl\_i3c.h to make I3C DMA driver reuse:
  - I3C\_GetIBIType
  - I3C\_GetIBIAddress
  - I3C\_SlaveCheckAndClearError
- Changed the handle pointer parameter in IRQ related functions to void \* type to make it reuse in I3C DMA driver.
- Added new API I3C\_SlaveRequestIBIWithSingleData for slave to request single data byte, this API could be used regardless slave is working in non-blocking interrupt or non-blocking dma.
- Added new API I3C\_MasterGetDeviceListAfterDAA for master application to get the device information list built up in DAA process.

#### [2.5.4]

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

#### [2.5.3]

- Updated driver for new feature FSL\_FEATURE\_I3C\_HAS\_NO\_SCONFIG\_BAMATCH and FSL\_FEATURE\_I3C\_HAS\_NO\_SCONFIG\_IDRAND.

#### [2.5.2]

- Updated driver for new feature FSL\_FEATURE\_I3C\_HAS\_NO\_MERRWARN\_TERM.
- Fixed the issue that call to I3C\_MasterTransferBlocking API did not generate STOP signal when NAK status was returned.

#### [2.5.1]

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

#### [2.5.0]

- Added new API I3C\_MasterRepeatedStartWithRxSize to send repeated start signal with receive terminate size specified.
- Fixed the status used in I3C\_RunTransferStateMachine, changed to use pending interrupts as status to be handled in the state machine.
- Fixed MISRA 2012 violation of rule 10.3, 10.7.

#### [2.4.0]

- Bug Fixes
  - Fixed kI3C\_SlaveMatchedFlag interrupt is not properly handled in I3C\_SlaveTransferHandleIRQ when it comes together with interrupt kI3C\_SlaveBusStartFlag.
  - Fixed the inaccurate I2C baudrate calculation in I3C\_MasterSetBaudRate.
  - Added new API I3C\_MasterGetIBIRules to get registered IBI rules.
  - Added new variable isReadTerm in struct\_i3c\_master\_handle for transfer state routine to check if MCTRL.RDTERM is configured for read transfer.
  - Changed to emit Auto IBI in transfer state routine for slave start flag assertion.
  - Fixed the slave maxWriteLength and maxReadLength does not be configured into SMAXLIMITS register issue.
  - Fixed incorrect state for IBI in I3C master interrupt transfer IRQ handle routine.
  - Added isHotJoin in i3c\_slave\_config\_t to request hot-join event during slave init.

### [2.3.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 8.4, 17.7.
  - Fixed incorrect HotJoin event index in I3C\_GetIBIType.

### [2.3.1]

- Bug Fixes
  - Fixed the issue that call of I3C\_MasterTransferBlocking/I3C\_MasterTransferNonBlocking fails for the case which receive length 1 byte of data.
  - Fixed the issue that STOP signal is not sent when NAK status is detected during execution of I3C\_MasterTransferBlocking function.

### [2.3.0]

- Improvements
  - Added I3C common driver APIs to initialize I3C with both master and slave configuration.
  - Updated I3C master transfer callback to function set structure to include callback invoke for IBI event and slave2master event.
  - Updated I3C master non-blocking transfer model and always enable the interrupts to be able to re-act to the slave start event and handle slave IBI.

### [2.2.0]

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

### [2.1.2]

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

### [2.1.1]

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

### [2.1.0]

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

## [2.0.0]

- Initial version.
- 

## I3C\_EDMA

### [2.2.12]

- Bug Fixes
  - Fixed the issue that EDMA transfer configuration use wrong parameter.

### [2.2.11]

- Improvements
  - Fixed Coverity CERT-C violations.

### [2.2.10]

- Bug Fixes
  - Fixed the issue that slave start event is cleared when it has not been handled.
- Added
  - Supported I3C HDR-DDR transfer with EDMA.
- Changed
  - Used linked EDMA to transfer all I3C subaddress and data without handling of intermediate states, simplifying code logic.
  - Prepare DMA before I3C START to ensure there's no time delay between START and transmitting data.
  - Added the MCTRLDONE flag check after START and STOP request to ensure all states are handled properly.

### [2.2.9]

- Bug Fixes
  - Fixed MISRA issue rule 11.3.
  - Added the master control done flag waiting code after STOP in case the bus is not idle when transfer function finishes.

### [2.2.8]

- Improvements
  - Removed I3C IRQ handler calling in the EDMA callback. Previously driver doesn't use the END byte which can trigger the STOP interrupt for controller sending and receiving, now let I3C event handler deal with all I3C events.
- Bug Fixes
  - Fixed the bug that the END type Tx register is not used when command length or data length is one byte.



**[2.2.7]**

- Bug Fixes
  - Fixed MISRA issue rule 11.6.

**[2.2.6]**

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

**[2.2.5]**

- Improvements
  - Added the workaround for RT1180 I3C EDMA issue ERR052086.

**[2.2.4]**

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

**[2.2.3]**

- Bug Fixes
  - Fixed issue that slave pollutes the last byte when Tx FIFO may be full.

**[2.2.2]**

- Bug Fixes
  - Fixed I3C MISRA issue rule 10.4, 11.3.

**[2.2.1]**

- Bug Fixes
  - Fixed the issue that I3C slave send the last byte data without using the END type register.
- Improvements
  - There's no need to reserve two bytes FIFO for DMA transfer which is for IP issue workaround.

**[2.2.0]**

- Improvements
  - Deleted legacy IBI data request code.

**[2.1.0]**

- Bug Fixes
  - Fixed MISRA issue rule 8.4, 8.6, 11.8.

#### [2.0.1]

- Bug Fixes
  - Fixed MISRA issue rule 9.1.

#### [2.0.0]

- Initial version.
- 

### LPI2C

#### [2.6.3]

- Bug Fixes
  - Fixed static analysis identified issues.

#### [2.6.2]

- Improvements
  - Added timeout for while loop in LPI2C\_TransferStateMachineSendCommand().

#### [2.6.1]

- Bug Fixes
  - Fixed coverity issues.

#### [2.6.0]

- New Feature
  - Added common IRQ handler entry LPI2C\_DriverIRQHandler.

#### [2.5.7]

- Improvements
  - Added support for separated IRQ handlers.

#### [2.5.6]

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

#### [2.5.5]

- Bug Fixes
  - Fixed LPI2C\_SlaveInit() - allow to disable SDA/SCL glitch filter.

**[2.5.4]**

- Bug Fixes
  - Fixed LPI2C\_MasterTransferBlocking() - the return value was sometime affected by call of LPI2C\_MasterStop().

**[2.5.3]**

- Improvements
  - Added handler for LPI2C7 and LPI2C8.

**[2.5.2]**

- Bug Fixes
  - Fixed ERR051119 to ignore the nak flag when IGNACK=1 in LPI2C\_MasterCheckAndClearError.

**[2.5.1]**

- Bug Fixes
  - Added bus stop incase of bus stall in LPI2C\_MasterTransferBlocking.
- Improvements
  - Release peripheral from reset if necessary in init function.

**[2.5.0]**

- New Features
  - Added new function LPI2C\_SlaveEnableAckStall to enable or disable ACKSTALL.

**[2.4.1]**

- Improvements
  - Before master transfer with transactional APIs, enable master function while disable slave function and vise versa for slave transfer to avoid the one affecting the other.

**[2.4.0]**

- Improvements
  - Split some functions, fixed CCM problem in file fsl\_lpi2c.c.
- Bug Fixes
  - Fixed bug in LPI2C\_MasterInit that the MCFGR2's value set in LPI2C\_MasterSetBaudRate may be overwritten by mistake.

**[2.3.2]**

- Improvements
  - Initialized the EDMA configuration structure in the LPI2C EDMA driver.

### [2.3.1]

- Improvements
  - Updated LPI2C\_GetCyclesForWidth to add the parameter of minimum cycle, because for master SDA/SCL filter, master bus idle/pin low timeout and slave SDA/SCL filter configuration, 0 means disabling the feature and cannot be used.
- Bug Fixes
  - Fixed bug in LPI2C\_SlaveTransferHandleIRQ that when restart detect event happens the transfer structure should not be cleared.
  - Fixed bug in LPI2C\_RunTransferStateMachine, that when only slave address is transferred or there is still data remaining in tx FIFO the last byte's nack cannot be ignored.
  - Fixed bug in slave filter doze enable, that when FILTDZ is set it means disable rather than enable.
  - Fixed bug in the usage of LPI2C\_GetCyclesForWidth. First its return value cannot be used directly to configure the slave FILTSDA, FILTSCL, DATAVD or CLKHOLD, because the real cycle width for them should be FILTSDA+3, FILTSCL+3, FILTSCL+DATAVD+3 and CLKHOLD+3. Second when cycle period is not affected by the prescaler value, prescaler value should be passed as 0 rather than 1.
  - Fixed wrong default setting for LPI2C slave. If enabling the slave tx SCL stall, then the default clock hold time should be set to 250ns according to I2C spec for 100kHz standard mode baudrate.
  - Fixed bug that before pushing command to the tx FIFO the FIFO occupation should be checked first in case FIFO overflow.

### [2.3.0]

- New Features
  - Supported reading more than 256 bytes of data in one transfer as master.
  - Added API LPI2C\_GetInstance.
- Bug Fixes
  - Fixed bug in LPI2C\_MasterTransferAbortEDMA, LPI2C\_MasterTransferAbort and LPI2C\_MasterTransferHandleIRQ that before sending stop signal whether master is active and whether stop signal has been sent should be checked, to make sure no FIFO error or bus error will be caused.
  - Fixed bug in LPI2C master EDMA transactional layer that the bus error cannot be caught and returned by user callback, by monitoring bus error events in interrupt handler.
  - Fixed bug in LPI2C\_GetCyclesForWidth that the parameter used to calculate clock cycle should be  $2^{\text{prescaler}}$  rather than prescaler.
  - Fixed bug in LPI2C\_MasterInit that timeout value should be configured after baudrate, since the timeout calculation needs prescaler as parameter which is changed during baudrate configuration.
  - Fixed bug in LPI2C\_MasterTransferHandleIRQ and LPI2C\_RunTransferStateMachine that when master writes with no stop signal, need to first make sure no data remains in the tx FIFO before finishes the transfer.

### [2.2.0]

- Bug Fixes

- Fixed issue that the SCL high time, start hold time and stop setup time do not meet I2C specification, by changing the configuration of data valid delay, setup hold delay, clock high and low parameters.
- MISRA C-2012 issue fixed.
  - \* Fixed rule 8.4, 13.5, 17.7, 20.8.

#### [2.1.12]

- Bug Fixes
  - Fixed MISRA advisory 15.5 issues.

#### [2.1.11]

- Bug Fixes
  - Fixed the bug that, during master non-blocking transfer, after the last byte is sent/received, the kLPI2C\_MasterNackDetectFlag is expected, so master should not check and clear kLPI2C\_MasterNackDetectFlag when remainingBytes is zero, in case FIFO is emptied when stop command has not been sent yet.
  - Fixed the bug that, during non-blocking transfer slave may nack master while master is busy filling tx FIFO, and NDF may not be handled properly.

#### [2.1.10]

- Bug Fixes
  - MISRA C-2012 issue fixed.
    - \* Fixed rule 10.3, 14.4, 15.5.
  - Fixed unaligned access issue in LPI2C\_RunTransferStateMachine.
  - Fixed uninitialized variable issue in LPI2C\_MasterTransferHandleIRQ.
  - Used linked TCD to disable tx and enable rx in read operation to fix the issue that for platform sharing the same DMA request with tx and rx, during LPI2C read operation if interrupt with higher priority happened exactly after command was sent and before tx disabled, potentially both tx and rx could trigger dma and cause trouble.
  - Fixed MISRA issues.
    - \* Fixed rules 10.1, 10.3, 10.4, 11.6, 11.9, 14.4, 17.7.
  - Fixed the waitTimes variable not re-assignment issue for each byte read.
- New Features
  - Added the IRQHandler for LPI2C5 and LPI2C6 instances.
- Improvements
  - Updated the LPI2C\_WAIT\_TIMEOUT macro to unified name I2C\_RETRY\_TIMES.

#### [2.1.9]

- Bug Fixes
  - Fixed Coverity issue of unchecked return value in I2C\_RTOS\_Transfer.
  - Fixed Coverity issue of operands did not affect the result in LPI2C\_SlaveReceive and LPI2C\_SlaveSend.

- Removed STOP signal wait when NAK detected.
- Cleared slave repeat start flag before transmission started in LPI2C\_SlaveSend/LPI2C\_SlaveReceive. The issue was that LPI2C\_SlaveSend/LPI2C\_SlaveReceive did not handle with the reserved repeat start flag. This caused the next slave to send a break, and the master was always in the receive data status, but could not receive data.

#### [2.1.8]

- Bug Fixes
  - Fixed the transfer issue with LPI2C\_MasterTransferNonBlocking, kLPI2C\_TransferNoStopFlag, with the wait transfer done through callback in a way of not doing a blocking transfer.
  - Fixed the issue that STOP signal did not appear in the bus when NAK event occurred.

#### [2.1.7]

- Bug Fixes
  - Cleared the stopflag before transmission started in LPI2C\_SlaveSend/LPI2C\_SlaveReceive. The issue was that LPI2C\_SlaveSend/LPI2C\_SlaveReceive did not handle with the reserved stop flag and caused the next slave to send a break, and the master always stayed in the receive data status but could not receive data.

#### [2.1.6]

- Bug Fixes
  - Fixed driver MISRA build error and C++ build error in LPI2C\_MasterSend and LPI2C\_SlaveSend.
  - Reset FIFO in LPI2C Master Transfer functions to avoid any byte still remaining in FIFO during last transfer.
  - Fixed the issue that LPI2C\_MasterStop did not return the correct NAK status in the bus for second transfer to the non-existing slave address.

#### [2.1.5]

- Bug Fixes
  - Extended the Driver IRQ handler to support LPI2C4.
  - Changed to use ARRAY\_SIZE(kLpi2cBases) instead of FEATURE\_COUNT to decide the array size for handle pointer array.

#### [2.1.4]

- Bug Fixes
  - Fixed the LPI2C\_MasterTransferEDMA receive issue when LPI2C shared same request source with TX/RX DMA request. Previously, the API used scatter-gather method, which handled the command transfer first, then the linked TCD which was pre-set with the receive data transfer. The issue was that the TX DMA request and the RX DMA request were both enabled, so when the DMA finished the first command TCD transfer and handled the receive data TCD, the TX DMA request still happened due to empty TX FIFO. The result was that the RX DMA transfer would start without waiting on the expected RX DMA request.

- Fixed the issue by enabling IntMajor interrupt for the command TCD and checking if there was a linked TCD to disable the TX DMA request in LPI2C\_MasterEDMACallback API.

### [2.1.3]

- Improvements
  - Added LPI2C\_WATI\_TIMEOUT macro to allow the user to specify the timeout times for waiting flags in functional API and blocking transfer API.
  - Added LPI2C\_MasterTransferBlocking API.

### [2.1.2]

- Bug Fixes
  - In LPI2C\_SlaveTransferHandleIRQ, reset the slave status to idle when stop flag was detected.

### [2.1.1]

- Bug Fixes
  - Disabled the auto-stop feature in eDMA driver. Previously, the auto-stop feature was enabled at transfer when transferring with stop flag. Since transfer was without stop flag and the auto-stop feature was enabled, when starting a new transfer with stop flag, the stop flag would be sent before the new transfer started, causing unsuccessful sending of the start flag, so the transfer could not start.
  - Changed default slave configuration with address stall false.

### [2.1.0]

- Improvements
  - API name changed:
    - \* LPI2C\_MasterTransferCreateHandle -> LPI2C\_MasterCreateHandle.
    - \* LPI2C\_MasterTransferGetCount -> LPI2C\_MasterGetTransferCount.
    - \* LPI2C\_MasterTransferAbort -> LPI2C\_MasterAbortTransfer.
    - \* LPI2C\_MasterTransferHandleIRQ -> LPI2C\_MasterHandleInterrupt.
    - \* LPI2C\_SlaveTransferCreateHandle -> LPI2C\_SlaveCreateHandle.
    - \* LPI2C\_SlaveTransferGetCount -> LPI2C\_SlaveGetTransferCount.
    - \* LPI2C\_SlaveTransferAbort -> LPI2C\_SlaveAbortTransfer.
    - \* LPI2C\_SlaveTransferHandleIRQ -> LPI2C\_SlaveHandleInterrupt.

### [2.0.0]

- Initial version.
-

## LPI2C\_EDMA

### [2.4.6]

- Bug Fixes
  - Fixed static analysis identified issues.

### [2.4.5]

- Improvements
  - Added condition to IRQ handler to check whether the interrupt is enabled - kLPI2C\_MasterTxReadyFlag.

### [2.4.4]

- Improvements
  - Added support for 2KB data transfer

### [2.4.3]

- Improvements
  - Added support for separated IRQ handlers.

### [2.4.2]

- Improvements
  - Add EDMA ext API to accommodate more types of EDMA.

### [2.4.1]

- Refer LPI2C driver change log 2.0.0 to 2.4.1
- 

## LPIT

### [2.1.3]

- Bug Fixes
  - Fixed doxygen generation warnings.

### [2.1.2]

- Bug Fixes
  - Fix CERT INT31-C issues.

### [2.1.1]

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



**[2.1.0]**

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

**[2.0.2]**

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

**[2.0.1]**

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

**[2.0.0]**

- Initial version.
- 

**LPSPi****[2.7.4]**

- Bug Fixes
  - Clear WIDTH bits from the TCR register before writing a new value in LPSPi\_MasterTransferBlocking().

**[2.7.3]**

- Improvements
  - Added timeout for while loop in LPSPi\_MasterTransferWriteAllTxData().
  - Make SPI\_RETRY\_TIMES configurable by CONFIG\_SPI\_RETRY\_TIMES.

**[2.7.2]**

- Bug Fixes
  - Fixed coverity issues.

#### [2.7.1]

- Bug Fixes
  - Workaround for errata ERR050607
  - Workaround for errata ERR010655

#### [2.7.0]

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

#### [2.6.10]

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

#### [2.6.9]

- Bug Fixes
  - Fixed reading of TCR register
  - Workaround for errata ERR050606

#### [2.6.8]

- Bug Fixes
  - Fixed build error when SPI\_RETRY\_TIMES is defined to non-zero value.

#### [2.6.7]

- Bug Fixes
  - Fixed the txData from void \* to const void \* in transmit API \_lpspi\_master\_handle and \_lpspi\_slave\_handle.

#### [2.6.6]

- Bug Fixes
  - Added LPSPI register init in LPSPI\_MasterInit incase of LPSPI register exist.

#### [2.6.5]

- Improvements
  - Introduced FSL\_FEATURE\_LPSPI\_HAS\_NO\_PCSCFG and FSL\_FEATURE\_LPSPI\_HAS\_NO\_MULTI\_WIDTH for conditional compile.
  - Release peripheral from reset if necessary in init function.

**[2.6.4]**

- Bug Fixes
  - Added LPSPI6\_DriverIRQHandler for LPSPI6 instance.

**[2.6.3]**

- Hot Fixes
  - Added macro switch in function LPSPI\_Enable about ERRATA051472.

**[2.6.2]**

- Bug Fixes
  - Disabled lpspi before LPSPI\_MasterSetBaudRate incase of LPSPI opened.

**[2.6.1]**

- Bug Fixes
  - Fixed return value while calling LPSPI\_WaitTxFifoEmpty in function LPSPI\_MasterTransferNonBlocking.

**[2.6.0]**

- Feature
  - Added the new feature of multi-IO SPI .

**[2.5.3]**

- Bug Fixes
  - Fixed 3-wire txmask of handle vaule reentrant issue.

**[2.5.2]**

- Bug Fixes
  - Workaround for errata ERR051588 by clearing FIFO after transmit underrun occurs.

**[2.5.1]**

- Bug Fixes
  - Workaround for errata ERR050456 by resetting the entire module using LPSPI\_CR[RST] bit.

**[2.5.0]**

- Bug Fixes
  - Workaround for errata ERR011097 to wait the TX FIFO to go empty when writing TCR register and TCR[TXMSK] value is 1.
  - Added API LPSPI\_WaitTxFifoEmpty for wait the txfifo to go empty.

#### [2.4.7]

- Bug Fixes
  - Fixed bug that the SR[REF] would assert if software disabled or enabled the LPSPI module in LPSPI\_Enable.

#### [2.4.6]

- Improvements
  - Moved the configuration of registers for the 3-wire lpspi mode to the LPSPI\_MasterInit and LPSPI\_SlaveInit function.

#### [2.4.5]

- Improvements
  - Improved LPSPI\_MasterTransferBlocking send performance when frame size is 1-byte.

#### [2.4.4]

- Bug Fixes
  - Fixed LPSPI\_MasterGetDefaultConfig incorrect default inter-transfer delay calculation.

#### [2.4.3]

- Bug Fixes
  - Fixed bug that the ISR response speed is too slow on some platforms, resulting in the first transmission of overflow, Set proper RX watermarks to reduce the ISR response times.

#### [2.4.2]

- Bug Fixes
  - Fixed bug that LPSPI\_MasterTransferBlocking will modify the parameter txbuff and rxbuff pointer.

#### [2.4.1]

- Bug Fixes
  - Fixed bug that LPSPI\_SlaveTransferNonBlocking can't detect RX error.

#### [2.4.0]

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

**[2.3.1]**

- Improvements
  - Initialized the EDMA configuration structure in the LPSPI EDMA driver.
- Bug Fixes
  - Fixed bug that function LPSPI\_MasterTransferBlocking should return after the transfer complete flag is set to make sure the PCS is re-asserted.

**[2.3.0]**

- New Features
  - Supported the master configuration of sampling the input data using a delayed clock to improve slave setup time.

**[2.2.1]**

- Bug Fixes
  - Fixed bug in LPSPI\_SetPCSContinuous when disabling PCS continuous mode.

**[2.2.0]**

- Bug Fixes
  - Fixed bug in 3-wire polling and interrupt transfer that the received data is not correct and the PCS continuous mode is not working.

**[2.1.0]**

- Improvements
  - Improved LPSPI\_SlaveTransferHandleIRQ to fill up TX FIFO instead of write one data to TX register which improves the slave transmit performance.
  - Added new functional APIs LPSPI\_SelectTransferPCS and LPSPI\_SetPCSContinuous to support changing PCS selection and PCS continuous mode.
- Bug Fixes
  - Fixed bug in non-blocking and EDMA transfer APIs that kStatus\_InvalidArgument is returned if user configures 3-wire mode and full-duplex transfer at the same time, but transfer state is already set to kLPSPI\_Busy by mistake causing following transfer can not start.
  - Fixed bug when LPSPI slave using EDMA way to transfer, tx should be masked when tx data is null, otherwise in 3-wire mode which tx/rx use the same pin, the received data will be interfered.

**[2.0.5]**

- Improvements
  - Added timeout mechanism when waiting certain states in transfer driver.
- Bug Fixes
  - Fixed the bug that LPSPI can not transfer large data using EDMA.
  - Fixed MISRA 17.7 issues.

- Fixed variable overflow issue introduced by MISRA fix.
- Fixed issue that `rxFifoMaxBytes` should be calculated according to transfer width rather than FIFO width.
- Fixed issue that completion flag was not cleared after transfer completed.

#### [2.0.4]

- Bug Fixes
  - Fixed in `LPSPi_MasterTransferBlocking` that master `rxfifo` may overflow in stall condition.
  - Eliminated IAR Pa082 warnings.
  - Fixed MISRA issues.
    - \* Fixed rules 10.1, 10.3, 10.4, 10.6, 11.9, 14.2, 14.4, 15.7, 17.7.

#### [2.0.3]

- Bug Fixes
  - Removed `LPSPi_Reset` from `LPSPi_MasterInit` and `LPSPi_SlaveInit`, because this API may glitch the slave select line. If needed, call this function manually.

#### [2.0.2]

- New Features
  - Added dummy data set up API to allow users to configure the dummy data to be transferred.
  - Enabled the 3-wire mode, `SIN` and `SOUT` pins can be configured as input/output pin.

#### [2.0.1]

- Bug Fixes
  - Fixed the bug that the clock source should be divided by the `PRESCALE` setting in `LPSPi_MasterSetDelayTimes` function.
  - Fixed the bug that `LPSPi_MasterTransferBlocking` function would hang in some corner cases.
- Optimization
  - Added `#ifndef/#endif` to allow user to change the default TX value at compile time.

#### [2.0.0]

- Initial version.
- 

## LPSPi\_EDMA

#### [2.4.9]

- Improvements
  - Removed unused code from `LPSPi_SeparateEdmaReadData()`.

**[2.4.8]**

- Improvements
  - Added timeout for while loop in `EDMA_LpspiMasterCallback()` and `EDMA_LpspiSlaveCallback()`.

**[2.4.7]**

- Bug Fixes
  - Add macro `LPSPi_ALIGN_TCD_SIZE_MASK` to align an address to `edma_tcd_t` size.

**[2.4.6]**

- Improvements
  - Increased transmit FIFO watermark to ensure whole transmit FIFO will be used during data transfer.

**[2.4.5]**

- Bug Fixes
  - Fixed reading of TCR register
  - Workaround for errata ERR050606

**[2.4.4]**

- Improvements
  - Add EDMA ext API to accommodate more types of EDMA.

**[2.4.3]**

- Improvements
  - Supported 32K bytes transmit in DMA, improve the max datasize in `LPSPi_MasterTransferEDMALite`.

**[2.4.2]**

- Improvements
  - Added callback status in `EDMA_LpspiMasterCallback` and `EDMA_LpspiSlaveCallback` to check `transferDone`.

**[2.4.1]**

- Improvements
  - Add the `TXMSK` wait after TCR setting.

#### [2.4.0]

- Improvements
    - Separated LPSPI\_MasterTransferEDMA functions to LPSPI\_MasterTransferPrepareEDMA and LPSPI\_MasterTransferEDMALite to optimize the process of transfer.
- 

### LPTMR

#### [2.2.1]

- Bug Fixes
  - Fix CERT INT31-C issues.

#### [2.2.0]

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

#### [2.1.1]

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

#### [2.1.0]

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

#### [2.0.2]

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

#### [2.0.1]

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

#### [2.0.0]

- Initial version.
-



## LPUART

### [2.10.0]

- New Feature
  - Added support to configure RTS watermark.

### [2.9.4]

- Improvements
  - Merged duplicate code.

### [2.9.3]

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

### [2.9.2]

- Bug Fixes
  - Fixed coverity issues.

### [2.9.1]

- Bug Fixes
  - Fixed coverity issues.

### [2.9.0]

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

### [2.8.3]

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

### [2.8.2]

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

### [2.8.1]

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

#### [2.8.0]

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

#### [2.7.7]

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

#### [2.7.6]

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

#### [2.7.5]

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

#### [2.7.4]

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

#### [2.7.3]

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

#### [2.7.2]

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

#### [2.7.1]

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

#### [2.7.0]

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

**[2.6.0]**

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

**[2.5.3]**

- Bug Fixes
  - Fixed comments by replacing unused status flags `kLPUART_NoiseErrorInRxDataRegFlag` and `kLPUART_ParityErrorInRxDataRegFlag` with `kLPUART_NoiseErrorFlag` and `kLPUART_ParityErrorFlag`.

**[2.5.2]**

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

**[2.5.1]**

- Improvements
  - Use separate data for TX and RX in `lpuart_transfer_t`.
- Bug Fixes
  - Fixed bug that when ring buffer is used, if some data is received in ring buffer first before calling `LPUART_TransferReceiveNonBlocking`, the received data count returned by `LPUART_TransferGetReceiveCount` is wrong.

**[2.5.0]**

- Bug Fixes
  - Added missing interrupt enable masks `kLPUART_Match1InterruptEnable` and `kLPUART_Match2InterruptEnable`.
  - Fixed bug in `LPUART_EnableInterrupts`, `LPUART_DisableInterrupts` and `LPUART_GetEnabledInterrupts` that the `BAUD[LBKDIE]` bit field should be soc specific.
  - Fixed bug in `LPUART_TransferHandleIRQ` that idle line interrupt should be disabled when rx data size is zero.
  - Deleted unused status flags `kLPUART_NoiseErrorInRxDataRegFlag` and `kLPUART_ParityErrorInRxDataRegFlag`, since firstly their function are the same as `kLPUART_NoiseErrorFlag` and `kLPUART_ParityErrorFlag`, secondly to obtain them one data word must be read out thus interfering with the receiving process.
  - Fixed bug in `LPUART_GetStatusFlags` that the `STAT[LBKDIF]`, `STAT[MA1F]` and `STAT[MA2F]` should be soc specific.
  - Fixed bug in `LPUART_ClearStatusFlags` that tx/rx FIFO is reset by mistake when clearing flags.

- Fixed bug in LPUART\_TransferHandleIRQ that while clearing idle line flag the other bits should be masked in case other status bits be cleared by accident.
- Fixed bug of race condition during LPUART transfer using transactional APIs, by disabling and re-enabling the global interrupt before and after critical operations on interrupt enable register.
- Fixed DMA/eDMA transfer blocking issue by enabling tx idle interrupt after DMA/eDMA transmission finishes.
- New Features
  - Added APIs LPUART\_GetRxFifoCount/LPUART\_GetTxFifoCount to get rx/tx FIFO data count.
  - Added APIs LPUART\_SetRxFifoWatermark/LPUART\_SetTxFifoWatermark to set rx/tx FIFO water mark.

#### **[2.4.1]**

- Bug Fixes
  - Fixed MISRA advisory 17.7 issues.

#### **[2.4.0]**

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

#### **[2.3.1]**

- Bug Fixes
  - Fixed MISRA advisory 15.5 issues.

#### **[2.3.0]**

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

#### **[2.2.8]**

- Bug Fixes
  - Fixed issue for MISRA-2012 check.
    - \* Fixed rule-10.3, rule-14.4, rule-15.5.
  - Eliminated Pa082 warnings by assigning volatile variables to local variables and using local variables instead.
  - Fixed MISRA issues.
    - \* Fixed rules 10.1, 10.3, 10.4, 10.8, 14.4, 11.6, 17.7.
- Improvements

- Added check for `kLPUART_TransmissionCompleteFlag` in `LPUART_WriteBlocking`, `LPUART_TransferHandleIRQ`, `LPUART_TransferSendDMACallback` and `LPUART_SendEDMACallback` to ensure all the data would be sent out to bus.
- Rounded up the calculated `sbr` value in `LPUART_SetBaudRate` and `LPUART_Init` to achieve more accurate baudrate setting. Changed `osr` from `uint32_t` to `uint8_t` since `osr`'s biggest value is 31.
- Modified `LPUART_ReadBlocking` so that if more than one receiver errors occur, all status flags will be cleared and the most severe error status will be returned.

#### [2.2.7]

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

#### [2.2.6]

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

#### [2.2.5]

- Bug Fixes
  - Do not set or clear the TIE/RIE bits when using `LPUART_EnableTxDMA` and `LPUART_EnableRxDMA`.

#### [2.2.4]

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

#### [2.2.3]

- Improvements
  - Changed parameter type in `LPUART_RTOS_Init` struct from `rtos_lpuart_config` to `lpuart_rtos_config_t`.
- Bug Fixes

- Disabled LPUART receive interrupt instead of all NVICs when reading data from ring buffer. Otherwise when the ring buffer is used, receive nonblocking method will disable all NVICs to protect the ring buffer. This may have a negative effect on other IPs that are using the interrupt.

#### [2.2.2]

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

#### [2.2.1]

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

#### [2.2.0]

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

#### [2.1.1]

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

#### [2.1.0]

- Improvements
    - Update transactional APIs.
- 

### LPUART\_EDMA

#### [2.4.0]

- Refer LPUART driver change log 2.1.0 to 2.4.0
- 

### MCM

#### [2.2.0]

- Improvements
  - Support platforms with less features.

**[2.1.0]**

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

**[2.0.0]**

- Initial version.
- 

**MIPI\_DSI****[2.1.0]**

- Bug Fixes
  - Fixed typo in member of dsi\_transfer\_t structure. The sendDscCmd and dscCmd shall be sendDcsCmd and dcsCmd.

**[2.0.3]**

- Improvements
  - Supported DSI\_ConfigDphy api.
  - Supported DSI\_EnableVpgEnMode to config the video mode pattern generator.

**[2.0.2]**

- Improvements
  - More precise calculation of the values of m & n.
  - Lookup table method to obtain DPHY-related parameters based on bandwidth.

**[2.0.1]**

- Bug Fixes.
  - Fixed MISRA C-2012 issues: 10.1, 10.3, 10.4, 10.8, 21.15

**[2.0.0]**

- Initial version.
- 

**MU****[2.8.1]**

- Bug Fixes
  - Avoid incorrect MU\_BUSY\_POLL\_COUNT macro use.

#### [2.8.0]

- New Features
  - Added MU1\_BUSY\_POLL\_COUNT parameter to prevent infinite polling loops in MU operations.
  - Added timeout mechanism to all polling loops in MU driver code.
- 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
    - \* Added new function MU\_ReceiveMsgTimeout() to include timeout mechanism.
  - Enhanced documentation across all functions to clarify timeout behavior and return values.

#### [2.7.0]

- New Features
  - Added API MU\_GetRxStatusFlags.

#### [2.6.0]

- New Features
  - Added API MU\_GetInterruptsPending.

#### [2.5.1]

- Bug Fixes
  - Fixed the bug that MU\_TriggerGeneralPurposeInterrupts and MU\_TriggerInterrupts may trigger previous triggered general purpose interrupts again by mistake.

#### [2.5.0]

- New Features
  - Supported more than 4 general purpose interrupts.
  - Added separate APIs for general purpose interrupts.

#### [2.4.0]

- Improvements
  - Supported the case that some features only available with specific instances. These features include Hardware Reset, Boot Peer Core, Hold Reset. When using the features with instances which don't support them, driver will report error.

#### [2.3.3]

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



**[2.3.2]**

- Improvements
  - Supported platforms which don't have CCR0[RSTH], CCR0[CLKE], CCR0[HR], CCR0[HRM].

**[2.3.1]**

- Bug Fixes
  - Fixed build error for platforms which have CCR0[RSTH], but no CCR0[NMI].

**[2.3.0]**

- New features
  - Added support for i.MX RT7xx.

**[2.2.1]**

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

**[2.2.0]**

- New features
  - Added support for i.MX RT118x.
- Bug Fixes
  - Fixed general purpose interrupt bug.
- Other Changes
  - Change \_mu\_interrupt\_trigger item value.

**[2.1.2]**

- Bug Fixes
  - Fixed bug that general purpose interrupt can't be configured.

**[2.1.1]**

- Bug Fixes
  - Fixed MISRA C-2012 issues.

**[2.1.0]**

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

**[2.0.0]**

- Initial version.

## OTFAD

### [2.1.4]

- Bug fixes
  - Fixed MISRA 2012 issue: 10.1.

### [2.1.3]

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

### [2.1.2]

- Bug fixes
  - Fixed MISRA 2012 issue: 10.4.

### [2.1.1]

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

### [2.1.0]

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

### [2.0.0]

- Initial version.
- 

## PDM

### [2.9.3]

- 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.

### [2.9.2]

- 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.

### [2.9.1]

- Bug Fixes
  - Fixed the issue that the driver still enters the interrupt after disabling clock.

**[2.9.0]**

- Improvements
- Added feature `FSL_FEATURE_PDM_HAS_DECIMATION_FILTER_BYPASS` to config `CTRL_2[DEC_BYPASS]` field.
- Modify code to make the OSR value is not limited to 16.

**[2.8.1]**

- Improvements
- Added feature `FSL_FEATURE_PDM_HAS_NO_DOZEN` to handle nonexistent `CTRL_1[DOZEN]` field.

**[2.8.0]**

- Improvements
- Added feature `FSL_FEATURE_PDM_HAS_NO_HWVAD` to remove the support of hardware voice activity detector.
- Added feature `FSL_FEATURE_PDM_HAS_NO_FILTER_BUFFER` to remove the support of `FIR_RDY` bitfield in `STAT` register.

**[2.7.4]**

- Bug Fixes
  - Fixed driver can not determine the specific float number of clock divider.
  - Fixed `PDM_ValidateSrcClockRate` calculates PDM channel in wrong method issue.

**[2.7.3]**

- Improvements
- Added feature `FSL_FEATURE_PDM_HAS_NO_VADEF` to remove the support of `VADEF` bitfield in `VAD0_STAT` register.

**[2.7.2]**

- Improvements
- Added feature `FSL_FEATURE_PDM_HAS_NO_MINIMUM_CLKDIV` to decide whether the minimum clock frequency division is required.

**[2.7.1]**

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 8.4, 10.3, 10.1, 10.4, 14.4

#### [2.7.0]

- Improvements
  - Added api PDM\_EnableHwvadInterruptCallback to support handle hwvad IRQ in PDM driver.
  - Corrected the sample rate configuration for non high quality mode.
  - Added api PDM\_SetChannelGain to support adjust the channel gain.

#### [2.6.0]

- Improvements
  - Added new features FSL\_FEATURE\_PDM\_HAS\_STATUS\_LOW\_FREQ/FSL\_FEATURE\_PDM\_HAS\_DC\_OUT

#### [2.5.0]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 8.4, 16.5, 10.4, 10.3, 10.1, 11.9, 17.7, 10.6, 14.4, 11.8, 11.6.

#### [2.4.1]

- Bug Fixes
  - Fixed MDK 66-D warning in pdm driver.

#### [2.4.0]

- Improvements
  - Added api PDM\_TransferSetChannelConfig/PDM\_ReadFifo to support read different width data.
  - Added feature FSL\_FEATURE\_PDM\_HAS\_RANGE\_CTRL and api PDM\_ClearRangeStatus/PDM\_GetRangeStatus for range register.
- Bug Fixes
  - Fixed violation of MISRA C-2012 Rule 14.4, 10.3, 10.4.

#### [2.3.0]

- Improvements
  - Enabled envelope/energy voice detect mode by adding apis PDM\_SetHwvadInEnvelopeBasedMode/PDM\_SetHwvadInEnergyBasedMode.
  - Added feature FSL\_FEATURE\_PDM\_CHANNEL\_NUM for different SOC.

#### [2.2.1]

- Bug Fixes
  - Fixed violation of MISRA C-2012 Rule 10.1, 10.3, 10.4, 10.6, 10.7, 11.3, 11.8, 14.4, 17.7, 18.4.
  - Added medium quality mode support in function PDM\_SetSampleRateConfig.

**[2.2.0]**

- Improvements
  - Added api PDM\_SetSampleRateConfig to improve user experience and marked api PDM\_SetSampleRate as deprecated.

**[2.1.1]**

- Improvements
- Used new SDMA API SDMA\_SetDoneConfig instead of SDMA\_EnableSwDone for PDM SDMA driver.

**[2.1.0]**

- Improvements
  - Added software buffer queue for transactional API.

**[2.0.1]**

- Improvements
  - Improved HWVAD feature.

**[2.0.0]**

- Initial version.
- 

**PDM\_EDMA****[2.6.5]**

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

**[2.6.4]**

- Improvements
  - Add handling for runtime change of number of linked transfers

**[2.6.3]**

- Improvements
  - Add EDMA ext API to accommodate more types of EDMA.

**[2.6.2]**

- Improvements
  - Add macro MCUX\_SDK\_PDM\_EDMA\_PDM\_ENABLE\_INTERNAL to let the user decide whether to enable it when calling PDM\_TransferReceiveEDMA.

### [2.6.1]

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

### [2.6.0]

- Improvements
  - Updated api PDM\_TransferReceiveEDMA to support channel block interleave transfer.
  - Added new api PDM\_TransferSetMultiChannelInterleaveType to support channel interleave type configurations.

### [2.5.0]

- Refer PDM driver change log 2.1.0 to 2.5.0
- 

## RGPIO

### [2.2.0]

- Added RGPIO\_GetPinDirection() API to get the current direction of a RGPIO pin.
- Added FGPIO\_GetPinDirection() API to get the current direction of a FGPIO pin.

### [2.1.0]

- New feature:
  - Added API RGPIO\_EnablePortInput()
  - Added API RGPIO\_SetPinInterruptConfig()
  - Added API RGPIO\_GetPinsInterruptFlags()
  - Added API RGPIO\_ClearPinsInterruptFlags()

### [2.0.3]

- Improvements:
  - Enhanced FGPIO\_PinInit to enable clock internally.

### [2.0.2]

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

### [2.0.1]

- API Interface Change:
  - Refined naming of API while keep all original APIs with marking them as deprecated. The original API will be removed in the next release. The main change is to update API with prefix of \_PinXXX() and \_PortXXX().

**[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.
- 

## SAI\_EDMA

### [2.7.3]

- 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.

### [2.7.2]

- Improvements
  - Add macros `MCUX_SDK_SAI_EDMA_TX_ENABLE_INTERNAL` and `MCUX_SDK_SAI_EDMA_RX_ENABLE_INTERNAL` to let the user decide whether to enable SAI when calling `SAI_TransferSendEDMA/SAI_TransferReceiveEDMA`.

**[2.7.1]**

- Improvements
  - Add EDMA ext API to accommodate more types of EDMA.

**[2.7.0]**

- Improvements
  - Updated api SAI\_TransferReceiveEDMA to support voice channel block interleave transfer.
  - Updated api SAI\_TransferSendEDMA to support voice channel block interleave transfer.
  - Added new api SAI\_TransferSetInterleaveType to support channel interleave type configurations.

**[2.6.0]**

- Improvements
  - Removed deprecated APIs.

**[2.5.1]**

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 20.7.

**[2.5.0]**

- Improvements
  - Added new api SAI\_TransferSendLoopEDMA/SAI\_TransferReceiveLoopEDMA to support loop transfer.
  - Added multi sai channel transfer support.

**[2.4.0]**

- Improvements
  - Added new api SAI\_TransferGetValidTransferSlotsEDMA which can be used to get valid transfer slot count in the sai edma transfer queue.
  - Deprecated the api SAI\_TransferRxSetFormatEDMA and SAI\_TransferTxSetFormatEDMA.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.3,10.4.

**[2.3.2]**

- Refer SAI driver change log 2.1.0 to 2.3.2
-

## SAR\_ADC

### [2.3.0]

- New Feature
  - Added new feature macro a for compatibility with ADCs on some platforms where some instances do not support group3.

### [2.2.0]

- New Feature
  - Added new features to compatible with new platforms.

### [2.1.1]

- Improvement
  - Change ADC sample rate phase duration default value from 0x08 to 0x14.

### [2.1.0]

- New Feature
  - Added ADC\_StopConvChain function to support stop scan in normal conversion scan operation mode.

### [2.0.3]

- Bug Fixes
  - Fixed the array name usage error in function ADC\_GetInstance.

### [2.0.2]

- Bug Fixes
  - Fixed MISRA issues.

### [2.0.1]

- Bug Fixes
  - Fixed the bug that when calling function ADC\_EnableWdgThresholdInt() in function ADC\_SetAnalogWdgConfig(), the parameter was passed incorrectly.

### [2.0.0]

- Initial version.
- 

## SEMA42

### [2.1.1]

- Improvements
  - Updated SEMA42\_TryLock function to avoid unsigned integer operations wrap issue.

**[2.1.0]**

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

**[2.0.4]**

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

**[2.0.3]**

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

**[2.0.2]**

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

**[2.0.1]**

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

**[2.0.0]**

- Initial version.
- 

**TMU****[2.1.0]**

- New feature.
  - Supported rising & falling temp rate critical threshold.
  - Supported polling function to avoid the temp sudden jump issue.

**[2.0.0]**

- Initial version.
    - This module was first developed on i.MX 93.
-

## TPM

### [2.4.1]

- Improvements
  - Add Coverage Justification for uncovered code.

### [2.4.0]

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

### [2.3.6]

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

### [2.3.5]

- New Feature
  - Added IRQ handler entry for TPM2.

### [2.3.4]

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

### [2.3.3]

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

### [2.3.2]

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



**[2.3.1]**

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

**[2.3.0]**

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

**[2.2.4]**

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

**[2.2.3]**

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

**[2.2.2]**

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

**[2.2.1]**

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

**[2.2.0]**

- Improvements
  - Added TPM\_SetChannelPolarity to support select channel input/output polarity.
  - Added TPM\_EnableChannelExtTrigger to support enable external trigger input to be used by channel.
  - Added TPM\_CalculateCounterClkDiv to help calculates the counter clock prescaler.
  - Added TPM\_GetChannelValue to support get TPM channel value.
  - Added new TPM configuration.
    - \* syncGlobalTimeBase
    - \* extTriggerPolarity
    - \* chnlPolarity
  - Added new PWM signal configuration.

- \* secPauseLevel

- Bug Fixes
  - Fixed TPM\_SetupPwm can't configure 0% combined PWM issues.

#### [2.1.1]

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

#### [2.1.0]

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

#### [2.0.8]

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

#### [2.0.7]

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

#### [2.0.6]

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

#### [2.0.5]

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

**[2.0.4]**

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

**[2.0.3]**

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

**[2.0.2]**

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

**[2.0.1]**

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

**[2.0.0]**

- Initial version.
- 

**TRDC****[2.2.2]**

- Improvements
  - Update APIs to check whether the memory access configuration can be updated.
  - Update APIs to mask the MRC address since only high 18 bits are valid.

**[2.2.1]**

- Bug Fixes:
  - Fix MISRA violations.

**[2.2.0]**

- New Features:
  - Supported SoCs that do not have all TRDC modules.

#### [2.1.0]

- Bug Fixes:
  - Fix MISRA violations.
  - Fixed wrong operation of domain mask in TRDC\_MbcNseClearAll and TRDC\_MrcDomainNseClear.

#### [2.0.0]

- Initial version.
- 

### TSTMR

#### [2.1.0]

- New Features
  - Support configured clock frequency.
  - Add TSTMR\_Init and TSTMR\_init APIs.
- Improvements
  - Change TSTMR\_DelayUs from static inline function to normal function.

#### [2.0.4]

- Bugfix
  - Fix MISRA C-2012 Rule 10.4 and 14.4 issues.

#### [2.0.3]

- Bugfix
  - Fix CERT INT30-C that Unsigned integer operation TSTMR\_ReadTimeStamp(base) - startTime may wrap.

#### [2.0.2]

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

#### [2.0.1]

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

**[2.0.0]**

- Initial version.
- 

**WDOG32****[2.2.1]**

- Bug Fixes
  - Fix CERT INT31-C that the bool value shall be converted to unsigned int 0 or 1 then passed to registers.
  - Fix MISRA 2012 20.3 violation.

**[2.2.0]**

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

**[2.1.0]**

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

**[2.0.4]**

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

**[2.0.3]**

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

### [2.0.2]

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

### [2.0.1]

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

### [2.0.0]

- Initial version.
- 

## 1.6 Driver API Reference Manual

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

[MIMX9352](#)

## 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 SDK](#)

### 1.7.2 FreeMASTER

[freemaster](#)

### 1.7.3 FreeRTOS

[FreeRTOS](#)

## 1.7.4 lwIP

*lwIP*





# Chapter 2

## MIMX9352

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

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

Initializes the eDMA peripheral.

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

---

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

---

#### Parameters

- base – eDMA peripheral base address.
- config – A pointer to the configuration structure, see “edma\_config\_t”.

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

Deinitializes the eDMA peripheral.

This function gates the eDMA clock.

#### Parameters

- base – eDMA peripheral base address.

void EDMA\_InstallTCD(*EDMA\_Type* \*base, uint32\_t channel, *edma\_tcd\_t* \*tcd)

Push content of TCD structure into hardware TCD register.

#### Parameters

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

void EDMA\_GetDefaultConfig(*edma\_config\_t* \*config)

Gets the eDMA default configuration structure.

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

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

### Parameters

- config – A pointer to the eDMA configuration structure.

```
void EDMA__InitChannel(EDMA_Type *base, uint32_t channel, edma_channel_config_t
                      *channelConfig)
```

EDMA Channel initialization.

### Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- channelConfig – pointer to user's eDMA4 channel config structure, see *edma\_channel\_config\_t* for detail.

```
static inline void EDMA__SetChannelMemoryAttribute(EDMA_Type *base, uint32_t channel,
                                                    edma_channel_memory_attribute_t
                                                    writeAttribute,
                                                    edma_channel_memory_attribute_t
                                                    readAttribute)
```

Set channel memory attribute.

### Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- writeAttribute – Attributes associated with a write transaction.
- readAttribute – Attributes associated with a read transaction.

```
static inline void EDMA__SetChannelSignExtension(EDMA_Type *base, uint32_t channel, uint8_t
                                                  position)
```

Set channel sign extension.

### Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- position – A non-zero value specifying the sign extend bit position. If 0, sign extension is disabled.

```
static inline void EDMA__SetChannelSwapSize(EDMA_Type *base, uint32_t channel,
                                             edma_channel_swap_size_t swapSize)
```

Set channel swap size.

### Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- swapSize – Swap occurs with respect to the specified transfer size. If 0, swap is disabled.

```
static inline void EDMA_SetChannelAccessType(EDMA_Type *base, uint32_t channel,  
                                             edma_channel_access_type_t  
                                             channelAccessType)
```

Set channel access type.

#### Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- channelAccessType – eDMA4's transactions type on the system bus when the channel is active.

```
static inline void EDMA_SetChannelMux(EDMA_Type *base, uint32_t channel, uint32_t  
                                     channelRequestSource)
```

Set channel request source.

#### Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- channelRequestSource – eDMA hardware service request source for the channel. User need to use the `dma_request_source_t` type as the input parameter. Note that devices may use other enum type to express dma request source and User can find it in SOC header or `fsl_edma_soc.h`.

```
static inline uint32_t EDMA_GetChannelSystemBusInformation(EDMA_Type *base, uint32_t  
                                                         channel)
```

Gets the channel identification and attribute information on the system bus interface.

#### Parameters

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

#### Returns

The mask of the channel system bus information. Users need to use the `_edma_channel_sys_bus_info` type to decode the return variables.

```
static inline void EDMA_EnableChannelMasterIDReplication(EDMA_Type *base, uint32_t  
                                                         channel, bool enable)
```

Set channel master ID replication.

#### Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- enable – true is enable, false is disable.

```
static inline void EDMA_SetChannelSecurityLevel(EDMA_Type *base, uint32_t channel,  
                                                edma_channel_security_level_t level)
```

Set channel security level.

#### Parameters

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

```
static inline void EDMA_SetChannelProtectionLevel(EDMA_Type *base, uint32_t channel,  
                                                edma_channel_protection_level_t level)
```

Set channel security level.

#### Parameters

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

```
void EDMA_ResetChannel(EDMA_Type *base, uint32_t channel)
```

Sets all TCD registers to default values.

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

---

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

---

---

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

---

#### Parameters

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

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

Configures the eDMA transfer attribute.

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

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

---

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

---

#### Parameters

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

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

Configures the eDMA minor offset feature.

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

#### Parameters

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

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

Configures the eDMA channel preemption feature.

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

#### Parameters

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

```
void EDMA_SetChannelLink(EDMA_Type *base, uint32_t channel, edma_channel_link_type_t  
                        type, uint32_t linkedChannel)
```

Sets the channel link for the eDMA transfer.

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

---

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

---

#### Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- type – A channel link type, which can be one of the following:
  - kEDMA\_LinkNone
  - kEDMA\_MinorLink
  - kEDMA\_MajorLink
- linkedChannel – The linked channel number.

```
void EDMA_SetBandWidth(EDMA_Type *base, uint32_t channel, edma_bandwidth_t  
                      bandwidth)
```

Sets the bandwidth for the eDMA transfer.

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

#### Parameters

- base – eDMA peripheral base address.

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

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

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

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

#### Parameters

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

```
static inline void EDMA_EnableAutoStopRequest(EDMA_Type *base, uint32_t channel, bool  
                                              enable)
```

Enables an auto stop request for the eDMA transfer.

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

#### Parameters

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

```
void EDMA_EnableChannelInterrupts(EDMA_Type *base, uint32_t channel, uint32_t mask)
```

Enables the interrupt source for the eDMA transfer.

#### Parameters

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

```
void EDMA_DisableChannelInterrupts(EDMA_Type *base, uint32_t channel, uint32_t mask)
```

Disables the interrupt source for the eDMA transfer.

#### Parameters

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

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

Configures the eDMA channel TCD major offset feature.

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

#### Parameters

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

```
void EDMA__ConfigChannelSoftwareTCD(edma_tcd_t *tcd, const edma_transfer_config_t  
                                    *transfer)
```

Sets TCD fields according to the user's channel transfer configuration structure, *edma\_transfer\_config\_t*.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API *EDMA\_ConfigChannelSoftwareTCDExt*

Application should be careful about the TCD pool buffer storage class,

- For the platform has cache, the software TCD should be put in non cache section
- The TCD pool buffer should have a consistent storage class.

---

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

---

#### Parameters

- tcd – Pointer to the TCD structure.
- transfer – channel transfer configuration pointer.

```
void EDMA__TcdReset(edma_tcd_t *tcd)
```

Sets all fields to default values for the TCD structure.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API *EDMA\_TcdResetExt*

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

---

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

---

#### Parameters

- tcd – Pointer to the TCD structure.

```
void EDMA__TcdSetTransferConfig(edma_tcd_t *tcd, const edma_transfer_config_t *config,  
                                edma_tcd_t *nextTcd)
```

Configures the eDMA TCD transfer attribute.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API *EDMA\_TcdSetTransferConfigExt*

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The TCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address

offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

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

---

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

---

---

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

---

### Parameters

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

```
void EDMA_TcdSetMinorOffsetConfig(edma_tcd_t *tcd, const edma_minor_offset_config_t  
                                *config)
```

Configures the eDMA TCD minor offset feature.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA\_TcdSetMinorOffsetConfigExt

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

### Parameters

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

```
void EDMA_TcdSetChannelLink(edma_tcd_t *tcd, edma_channel_link_type_t type, uint32_t  
                           linkedChannel)
```

Sets the channel link for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA\_TcdSetChannelLinkExt

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

---

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

---

### Parameters

- tcd – Point to the TCD structure.
- type – Channel link type, it can be one of:
  - kEDMA\_LinkNone



- kEDMA\_MinorLink
- kEDMA\_MajorLink
- linkedChannel – The linked channel number.

static inline void EDMA\_TcdSetBandWidth(*edma\_tcd\_t* \*tcd, *edma\_bandwidth\_t* bandWidth)

Sets the bandwidth for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA\_TcdSetBandWidthExt

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

#### Parameters

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

void EDMA\_TcdSetModulo(*edma\_tcd\_t* \*tcd, *edma\_modulo\_t* srcModulo, *edma\_modulo\_t* destModulo)

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

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA\_TcdSetModuloExt

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

#### Parameters

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

static inline void EDMA\_TcdEnableAutoStopRequest(*edma\_tcd\_t* \*tcd, bool enable)

Sets the auto stop request for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA\_TcdEnableAutoStopRequestExt

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

#### Parameters

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

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

Enables the interrupt source for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA\_TcdEnableInterruptsExt

#### Parameters

- `tcd` – Point to the TCD structure.
- `mask` – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

`void EDMA__TcdDisableInterrupts(edma_tcd_t *tcd, uint32_t mask)`

Disables the interrupt source for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA__TcdDisableInterruptsExt`

#### Parameters

- `tcd` – Point to the TCD structure.
- `mask` – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

`void EDMA__TcdSetMajorOffsetConfig(edma_tcd_t *tcd, int32_t sourceOffset, int32_t destOffset)`

Configures the eDMA TCD major offset feature.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA__TcdSetMajorOffsetConfigExt`

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

#### Parameters

- `tcd` – A point to the TCD structure.
- `sourceOffset` – source address offset will be applied to source address after major loop done.
- `destOffset` – destination address offset will be applied to source address after major loop done.

`void EDMA__ConfigChannelSoftwareTCDExt(EDMA_Type *base, edma_tcd_t *tcd, const edma_transfer_config_t *transfer)`

Sets TCD fields according to the user's channel transfer configuration structure, `edma_transfer_config_t`.

Application should be careful about the TCD pool buffer storage class,

- For the platform has cache, the software TCD should be put in non cache section
- The TCD pool buffer should have a consistent storage class.

---

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

---

#### Parameters

- `base` – eDMA peripheral base address.
- `tcd` – Pointer to the TCD structure.
- `transfer` – channel transfer configuration pointer.

`void EDMA__TcdResetExt(EDMA_Type *base, edma_tcd_t *tcd)`

Sets all fields to default values for the TCD structure.

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

---

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

---

#### Parameters

- base – eDMA peripheral base address.
- tcd – Pointer to the TCD structure.

```
void EDMA_TcdSetTransferConfigExt(EDMA_Type *base, edma_tcd_t *tcd, const
                                edma_transfer_config_t *config, edma_tcd_t *nextTcd)
```

Configures the eDMA TCD transfer attribute.

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

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

---

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

---



---

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

---

### Parameters

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

```
void EDMA_TcdSetMinorOffsetConfigExt(EDMA_Type *base, edma_tcd_t *tcd, const
                                    edma_minor_offset_config_t *config)
```

Configures the eDMA TCD minor offset feature.

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

### Parameters

- base – eDMA peripheral base address.
- tcd – A point to the TCD structure.
- config – A pointer to the minor offset configuration structure.

```
void EDMA_TcdSetChannelLinkExt(EDMA_Type *base, edma_tcd_t *tcd,
                              edma_channel_link_type_t type, uint32_t linkedChannel)
```

Sets the channel link for the eDMA TCD.

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

---

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

---

### Parameters

- base – eDMA peripheral base address.
- tcd – Point to the TCD structure.
- type – Channel link type, it can be one of:
  - kEDMA\_LinkNone
  - kEDMA\_MinorLink
  - kEDMA\_MajorLink
- linkedChannel – The linked channel number.

```
static inline void EDMA__TcdSetBandWidthExt(EDMA_Type *base, edma_tcd_t *tcd,  
                                             edma_bandwidth_t bandWidth)
```

Sets the bandwidth for the eDMA TCD.

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

### Parameters

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

```
void EDMA__TcdSetModuloExt(EDMA_Type *base, edma_tcd_t *tcd, edma_modulo_t srcModulo,  
                           edma_modulo_t destModulo)
```

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

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

### Parameters

- base – eDMA peripheral base address.
- tcd – A pointer to the TCD structure.
- srcModulo – A source modulo value.
- destModulo – A destination modulo value.

```
static inline void EDMA__TcdEnableAutoStopRequestExt(EDMA_Type *base, edma_tcd_t *tcd,  
                                                     bool enable)
```

Sets the auto stop request for the eDMA TCD.

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

### Parameters

- base – eDMA peripheral base address.
- tcd – A pointer to the TCD structure.
- enable – The command to enable (true) or disable (false).

void EDMA\_\_TcdEnableInterruptsExt(*EDMA\_Type* \*base, *edma\_tcd\_t* \*tcd, uint32\_t mask)

Enables the interrupt source for the eDMA TCD.

#### Parameters

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

void EDMA\_\_TcdDisableInterruptsExt(*EDMA\_Type* \*base, *edma\_tcd\_t* \*tcd, uint32\_t mask)

Disables the interrupt source for the eDMA TCD.

#### Parameters

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

void EDMA\_\_TcdSetMajorOffsetConfigExt(*EDMA\_Type* \*base, *edma\_tcd\_t* \*tcd, int32\_t sourceOffset, int32\_t destOffset)

Configures the eDMA TCD major offset feature.

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

#### Parameters

- base – eDMA peripheral base address.
- tcd – A point to the TCD structure.
- sourceOffset – source address offset will be applied to source address after major loop done.
- destOffset – destination address offset will be applied to source address after major loop done.

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

Enables the eDMA hardware channel request.

This function enables the hardware channel request.

#### Parameters

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

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

Disables the eDMA hardware channel request.

This function disables the hardware channel request.

#### Parameters

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

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

Starts the eDMA transfer by using the software trigger.

This function starts a minor loop transfer.

#### Parameters

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

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

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

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

---

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

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

#### Parameters

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

#### Returns

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

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

Gets the eDMA channel error status flags.

#### Parameters

- base – eDMA peripheral base address.

#### Returns

The mask of error status flags. Users need to use the `_edma_error_status_flags` type to decode the return variables.

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

Gets the eDMA channel status flags.

#### Parameters

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

#### Returns

The mask of channel status flags. Users need to use the `_edma_channel_status_flags` type to decode the return variables.

`void EDMA_ClearChannelStatusFlags(EDMA_Type *base, uint32_t channel, uint32_t mask)`  
 Clears the eDMA channel status flags.

#### Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of channel status to be cleared. Users need to use the defined `_edma_channel_status_flags` type.

`status_t EDMA_CreateHandle(edma_handle_t *handle, EDMA_Type *base, uint32_t channel)`  
 Creates the eDMA handle.

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

#### Parameters

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

#### Return values

- `kStatus_Success` –
- `kStatus_InvalidArgument` –

`void EDMA_InstallTCDMemory(edma_handle_t *handle, edma_tcd_t *tcdPool, uint32_t tcdSize)`  
 Installs the TCDs memory pool into the eDMA handle.

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

#### Parameters

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

`void EDMA_SetCallback(edma_handle_t *handle, edma_callback callback, void *userData)`  
 Installs a callback function for the eDMA transfer.

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

#### Parameters

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

`void EDMA_PrepareTransferConfig(edma_transfer_config_t *config, void *srcAddr, uint32_t srcWidth, int16_t srcOffset, void *destAddr, uint32_t destWidth, int16_t destOffset, uint32_t bytesEachRequest, uint32_t transferBytes)`

Prepares the eDMA transfer structure configurations.

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

---

**Note:** The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE). User can check if 128 bytes support is available for specific instance by FSL\_FEATURE\_EDMA\_INSTANCE\_SUPPORT\_128\_BYTES\_TRANSFERn.

---

### Parameters

- config – The user configuration structure of type `edma_transfer_t`.
- srcAddr – eDMA transfer source address.
- srcWidth – eDMA transfer source address width(bytes).
- srcOffset – source address offset.
- destAddr – eDMA transfer destination address.
- destWidth – eDMA transfer destination address width(bytes).
- destOffset – destination address offset.
- bytesEachRequest – eDMA transfer bytes per channel request.
- transferBytes – eDMA transfer bytes to be transferred.

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

Prepares the eDMA transfer structure.

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

---

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

---

### Parameters

- config – The user configuration structure of type `edma_transfer_t`.
- srcAddr – eDMA transfer source address.
- srcWidth – eDMA transfer source address width(bytes).
- destAddr – eDMA transfer destination address.
- destWidth – eDMA transfer destination address width(bytes).
- bytesEachRequest – eDMA transfer bytes per channel request.
- transferBytes – eDMA transfer bytes to be transferred.
- type – eDMA transfer type.

```
void EDMA__PrepareTransferTCD(edma_handle_t *handle, edma_tcd_t *tcd, void *srcAddr,  
                              uint32_t srcWidth, int16_t srcOffset, void *destAddr, uint32_t  
                              destWidth, int16_t destOffset, uint32_t bytesEachRequest,  
                              uint32_t transferBytes, edma_tcd_t *nextTcd)
```



Prepares the eDMA transfer content descriptor.

This function prepares the transfer content descriptor structure according to the user input.

---

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

---

### Parameters

- `handle` – eDMA handle pointer.
- `tcd` – Pointer to eDMA transfer content descriptor structure.
- `srcAddr` – eDMA transfer source address.
- `srcWidth` – eDMA transfer source address width(bytes).
- `srcOffset` – source address offset.
- `destAddr` – eDMA transfer destination address.
- `destWidth` – eDMA transfer destination address width(bytes).
- `destOffset` – destination address offset.
- `bytesEachRequest` – eDMA transfer bytes per channel request.
- `transferBytes` – eDMA transfer bytes to be transferred.
- `nextTcd` – eDMA transfer linked TCD address.

*status\_t* EDMA\_SubmitTransferTCD(*edma\_handle\_t* \*handle, *edma\_tcd\_t* \*tcd)

Submits the eDMA transfer content descriptor.

This function submits the eDMA transfer request according to the transfer content descriptor. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function EDMA\_InstallTCDMemory before.

Typical user case:

#### a. submit single transfer

```
edma_tcd_t tcd;
EDMA_PrepareTransferTCD(handle, &tcd, ...)
EDMA_SubmitTransferTCD(handle, tcd)
EDMA_StartTransfer(handle)
```

#### b. submit static link transfer,

```
edma_tcd_t tcd[2];
EDMA_PrepareTransferTCD(handle, &tcd[0], ...)
EDMA_PrepareTransferTCD(handle, &tcd[1], ...)
EDMA_SubmitTransferTCD(handle, &tcd[0])
EDMA_StartTransfer(handle)
```

#### c. submit dynamic link transfer

```
edma_tcd_t tcdpool[2];
EDMA_InstallTCDMemory(&g_DMA_Handle, tcdpool, 2);
edma_tcd_t tcd;
EDMA_PrepareTransferTCD(handle, tcd, ...)
EDMA_SubmitTransferTCD(handle, tcd)
EDMA_PrepareTransferTCD(handle, tcd, ...)
```

(continues on next page)

(continued from previous page)

```
EDMA_SubmitTransferTCD(handle, tcd)
EDMA_StartTransfer(handle)
```

#### d. submit loop transfer

```
edma_tcd_t tcd[2];
EDMA_PrepareTransferTCD(handle, &tcd[0], ..., &tcd[1])
EDMA_PrepareTransferTCD(handle, &tcd[1], ..., &tcd[0])
EDMA_SubmitTransferTCD(handle, &tcd[0])
EDMA_StartTransfer(handle)
```

#### Parameters

- handle – eDMA handle pointer.
- tcd – Pointer to eDMA transfer content descriptor structure.

#### Return values

- kStatus\_EDMA\_Success – It means submit transfer request succeed.
- kStatus\_EDMA\_QueueFull – It means TCD queue is full. Submit transfer request is not allowed.
- kStatus\_EDMA\_Busy – It means the given channel is busy, need to submit request later.

*status\_t* EDMA\_SubmitTransfer(*edma\_handle\_t* \*handle, const *edma\_transfer\_config\_t* \*config)

Submits the eDMA transfer request.

This function submits the eDMA transfer request according to the transfer configuration structure. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function EDMA\_InstallTCMemory before.

#### Parameters

- handle – eDMA handle pointer.
- config – Pointer to eDMA transfer configuration structure.

#### Return values

- kStatus\_EDMA\_Success – It means submit transfer request succeed.
- kStatus\_EDMA\_QueueFull – It means TCD queue is full. Submit transfer request is not allowed.
- kStatus\_EDMA\_Busy – It means the given channel is busy, need to submit request later.

*status\_t* EDMA\_SubmitLoopTransfer(*edma\_handle\_t* \*handle, *edma\_transfer\_config\_t* \*transfer, *uint32\_t* transferLoopCount)

Submits the eDMA scatter gather transfer configurations.

The function is target for submit loop transfer request, the ring transfer request means that the transfer request TAIL is link to HEAD, such as, A->B->C->D->A, or A->A

To use the ring transfer feature, the application should allocate several transfer object, such as

```
edma_channel_transfer_config_t transfer[2];
EDMA_TransferSubmitLoopTransfer(psHandle, &transfer, 2U);
```

Then eDMA driver will link transfer[0] and transfer[1] to each other

---

**Note:** Application should check the return value of this function to avoid transfer request submit failed

---

### Parameters

- `handle` – eDMA handle pointer
- `transfer` – pointer to user's eDMA channel configure structure, see `edma_channel_transfer_config_t` for detail
- `transferLoopCount` – the count of the transfer ring, if loop count is 1, that means that the one will link to itself.

### Return values

- `kStatus_Success` – It means submit transfer request succeed
- `kStatus_EDMA_Busy` – channel is in busy status
- `kStatus_InvalidArgument` – Invalid Argument

`void EDMA_StartTransfer(edma_handle_t *handle)`

eDMA starts transfer.

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

### Parameters

- `handle` – eDMA handle pointer.

`void EDMA_StopTransfer(edma_handle_t *handle)`

eDMA stops transfer.

This function disables the channel request to pause the transfer. Users can call `EDMA_StartTransfer()` again to resume the transfer.

### Parameters

- `handle` – eDMA handle pointer.

`void EDMA_AbortTransfer(edma_handle_t *handle)`

eDMA aborts transfer.

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

### Parameters

- `handle` – DMA handle pointer.

`static inline uint32_t EDMA_GetUnusedTCDNumber(edma_handle_t *handle)`

Get unused TCD slot number.

This function gets current tcd index which is run. If the TCD pool pointer is NULL, it will return 0.

### Parameters

- `handle` – DMA handle pointer.

### Returns

The unused tcd slot number.

`static inline uint32_t EDMA_GetNextTCDAddress(edma_handle_t *handle)`

Get the next tcd address.

This function gets the next tcd address. If this is last TCD, return 0.

**Parameters**

- `handle` – DMA handle pointer.

**Returns**

The next TCD address.

`void EDMA_HandleIRQ(edma_handle_t *handle)`

eDMA IRQ handler for the current major loop transfer completion.

This function clears the channel major interrupt flag and calls the callback function if it is not NULL.

Note: For the case using TCD queue, when the major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Assertion of an optional interrupt request also occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor (if scatter/gather is enabled).

For instance, when the time interrupt of TCD[0] happens, the TCD[1] has already been loaded into the eDMA engine. As `sga` and `sga_index` are calculated based on the `DLAST_SGA` bitfield lies in the `TCD_CSR` register, the `sga_index` in this case should be 2 (`DLAST_SGA` of TCD[1] stores the address of TCD[2]). Thus, the “`tcdUsed`” updated should be (`tcdUsed - 2U`) which indicates the number of TCDs can be loaded in the memory pool (because TCD[0] and TCD[1] have been loaded into the eDMA engine at this point already.).

For the last two continuous ISRs in a scatter/gather process, they both load the last TCD (The last ISR does not load a new TCD) from the memory pool to the eDMA engine when major loop completes. Therefore, ensure that the header and `tcdUsed` updated are identical for them. `tcdUsed` are both 0 in this case as no TCD to be loaded.

See the “eDMA basic data flow” in the eDMA Functional description section of the Reference Manual for further details.

**Parameters**

- `handle` – eDMA handle pointer.

`void EDMA_TcdInit(EDMA_Type *base, edma_tcd_t *tcdRegs)`

Initialize all fields to 0 for the TCD structure.

This function initialize all fields for this TCD structure to 0.

**Parameters**

- `base` – eDMA peripheral base address.
- `tcd` – Pointer to the TCD structure.

`FSL_EDMA_DRIVER_VERSION`

eDMA driver version

Version 2.10.9.

`_edma_transfer_status` eDMA transfer status

*Values:*

enumerator `kStatus_EDMA_QueueFull`

TCD queue is full.

enumerator `kStatus_EDMA_Busy`

Channel is busy and can't handle the transfer request.

enum \_edma\_transfer\_size

eDMA transfer configuration

*Values:*

enumerator kEDMA\_TransferSize1Bytes

Source/Destination data transfer size is 1 byte every time

enumerator kEDMA\_TransferSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kEDMA\_TransferSize4Bytes

Source/Destination data transfer size is 4 bytes every time

enumerator kEDMA\_TransferSize8Bytes

Source/Destination data transfer size is 8 bytes every time

enumerator kEDMA\_TransferSize16Bytes

Source/Destination data transfer size is 16 bytes every time

enumerator kEDMA\_TransferSize32Bytes

Source/Destination data transfer size is 32 bytes every time

enumerator kEDMA\_TransferSize64Bytes

Source/Destination data transfer size is 64 bytes every time

enumerator kEDMA\_TransferSize128Bytes

Source/Destination data transfer size is 128 bytes every time

enum \_edma\_modulo

eDMA modulo configuration

*Values:*

enumerator kEDMA\_ModuloDisable

Disable modulo

enumerator kEDMA\_Modulo2bytes

Circular buffer size is 2 bytes.

enumerator kEDMA\_Modulo4bytes

Circular buffer size is 4 bytes.

enumerator kEDMA\_Modulo8bytes

Circular buffer size is 8 bytes.

enumerator kEDMA\_Modulo16bytes

Circular buffer size is 16 bytes.

enumerator kEDMA\_Modulo32bytes

Circular buffer size is 32 bytes.

enumerator kEDMA\_Modulo64bytes

Circular buffer size is 64 bytes.

enumerator kEDMA\_Modulo128bytes

Circular buffer size is 128 bytes.

enumerator kEDMA\_Modulo256bytes

Circular buffer size is 256 bytes.

enumerator kEDMA\_Modulo512bytes

Circular buffer size is 512 bytes.

enumerator kEDMA\_Modulo1Kbytes  
Circular buffer size is 1 K bytes.

enumerator kEDMA\_Modulo2Kbytes  
Circular buffer size is 2 K bytes.

enumerator kEDMA\_Modulo4Kbytes  
Circular buffer size is 4 K bytes.

enumerator kEDMA\_Modulo8Kbytes  
Circular buffer size is 8 K bytes.

enumerator kEDMA\_Modulo16Kbytes  
Circular buffer size is 16 K bytes.

enumerator kEDMA\_Modulo32Kbytes  
Circular buffer size is 32 K bytes.

enumerator kEDMA\_Modulo64Kbytes  
Circular buffer size is 64 K bytes.

enumerator kEDMA\_Modulo128Kbytes  
Circular buffer size is 128 K bytes.

enumerator kEDMA\_Modulo256Kbytes  
Circular buffer size is 256 K bytes.

enumerator kEDMA\_Modulo512Kbytes  
Circular buffer size is 512 K bytes.

enumerator kEDMA\_Modulo1Mbytes  
Circular buffer size is 1 M bytes.

enumerator kEDMA\_Modulo2Mbytes  
Circular buffer size is 2 M bytes.

enumerator kEDMA\_Modulo4Mbytes  
Circular buffer size is 4 M bytes.

enumerator kEDMA\_Modulo8Mbytes  
Circular buffer size is 8 M bytes.

enumerator kEDMA\_Modulo16Mbytes  
Circular buffer size is 16 M bytes.

enumerator kEDMA\_Modulo32Mbytes  
Circular buffer size is 32 M bytes.

enumerator kEDMA\_Modulo64Mbytes  
Circular buffer size is 64 M bytes.

enumerator kEDMA\_Modulo128Mbytes  
Circular buffer size is 128 M bytes.

enumerator kEDMA\_Modulo256Mbytes  
Circular buffer size is 256 M bytes.

enumerator kEDMA\_Modulo512Mbytes  
Circular buffer size is 512 M bytes.

enumerator kEDMA\_Modulo1Gbytes  
Circular buffer size is 1 G bytes.

enumerator kEDMA\_Modulo2Gbytes

Circular buffer size is 2 G bytes.

enum \_\_edma\_bandwidth

Bandwidth control.

*Values:*

enumerator kEDMA\_BandwidthStallNone

No eDMA engine stalls.

enumerator kEDMA\_BandwidthStall4Cycle

eDMA engine stalls for 4 cycles after each read/write.

enumerator kEDMA\_BandwidthStall8Cycle

eDMA engine stalls for 8 cycles after each read/write.

enum \_\_edma\_channel\_link\_type

Channel link type.

*Values:*

enumerator kEDMA\_LinkNone

No channel link

enumerator kEDMA\_MinorLink

Channel link after each minor loop

enumerator kEDMA\_MajorLink

Channel link while major loop count exhausted

\_edma\_channel\_status\_flags eDMA channel status flags.

*Values:*

enumerator kEDMA\_DoneFlag

DONE flag, set while transfer finished, CITER value exhausted

enumerator kEDMA\_ErrorFlag

eDMA error flag, an error occurred in a transfer

enumerator kEDMA\_InterruptFlag

eDMA interrupt flag, set while an interrupt occurred of this channel

\_edma\_error\_status\_flags eDMA channel error status flags.

*Values:*

enumerator kEDMA\_DestinationBusErrorFlag

Bus error on destination address

enumerator kEDMA\_SourceBusErrorFlag

Bus error on the source address

enumerator kEDMA\_ScatterGatherErrorFlag

Error on the Scatter/Gather address, not 32byte aligned.

enumerator kEDMA\_NbytesErrorFlag

NBYTES/CITER configuration error

enumerator kEDMA\_DestinationOffsetErrorFlag

Destination offset not aligned with destination size

enumerator kEDMA\_DestinationAddressErrorFlag  
Destination address not aligned with destination size

enumerator kEDMA\_SourceOffsetErrorFlag  
Source offset not aligned with source size

enumerator kEDMA\_SourceAddressErrorFlag  
Source address not aligned with source size

enumerator kEDMA\_ErrorChannelFlag  
Error channel number of the cancelled channel number

enumerator kEDMA\_TransferCanceledFlag  
Transfer cancelled

enumerator kEDMA\_ValidFlag  
No error occurred, this bit is 0. Otherwise, it is 1.

\_edma\_interrupt\_enable eDMA interrupt source

*Values:*

enumerator kEDMA\_ErrorInterruptEnable  
Enable interrupt while channel error occurs.

enumerator kEDMA\_MajorInterruptEnable  
Enable interrupt while major count exhausted.

enumerator kEDMA\_HalfInterruptEnable  
Enable interrupt while major count to half value.

enum \_edma\_transfer\_type

eDMA transfer type

*Values:*

enumerator kEDMA\_MemoryToMemory  
Transfer from memory to memory

enumerator kEDMA\_PeripheralToMemory  
Transfer from peripheral to memory

enumerator kEDMA\_MemoryToPeripheral  
Transfer from memory to peripheral

enumerator kEDMA\_PeripheralToPeripheral  
Transfer from Peripheral to peripheral

enum edma\_channel\_memory\_attribute

eDMA channel memory attribute

*Values:*

enumerator kEDMA\_ChannelNoWriteNoReadNoCacheNoBuffer  
No write allocate, no read allocate, non-cacheable, non-bufferable.

enumerator kEDMA\_ChannelNoWriteNoReadNoCacheBufferable  
No write allocate, no read allocate, non-cacheable, bufferable.

enumerator kEDMA\_ChannelNoWriteNoReadCacheableNoBuffer  
No write allocate, no read allocate, cacheable, non-bufferable.



enumerator kEDMA\_ChannelNoWriteNoReadCacheableBufferable

No write allocate, no read allocate, cacheable, bufferable.

enumerator kEDMA\_ChannelNoWriteReadNoCacheNoBuffer

No write allocate, read allocate, non-cacheable, non-bufferable.

enumerator kEDMA\_ChannelNoWriteReadNoCacheBufferable

No write allocate, read allocate, non-cacheable, bufferable.

enumerator kEDMA\_ChannelNoWriteReadCacheableNoBuffer

No write allocate, read allocate, cacheable, non-bufferable.

enumerator kEDMA\_ChannelNoWriteReadCacheableBufferable

No write allocate, read allocate, cacheable, bufferable.

enumerator kEDMA\_ChannelWriteNoReadNoCacheNoBuffer

write allocate, no read allocate, non-cacheable, non-bufferable.

enumerator kEDMA\_ChannelWriteNoReadNoCacheBufferable

write allocate, no read allocate, non-cacheable, bufferable.

enumerator kEDMA\_ChannelWriteNoReadCacheableNoBuffer

write allocate, no read allocate, cacheable, non-bufferable.

enumerator kEDMA\_ChannelWriteNoReadCacheableBufferable

write allocate, no read allocate, cacheable, bufferable.

enumerator kEDMA\_ChannelWriteReadNoCacheNoBuffer

write allocate, read allocate, non-cacheable, non-bufferable.

enumerator kEDMA\_ChannelWriteReadNoCacheBufferable

write allocate, read allocate, non-cacheable, bufferable.

enumerator kEDMA\_ChannelWriteReadCacheableNoBuffer

write allocate, read allocate, cacheable, non-bufferable.

enumerator kEDMA\_ChannelWriteReadCacheableBufferable

write allocate, read allocate, cacheable, bufferable.

enum \_edma\_channel\_swap\_size

eDMA4 channel swap size

*Values:*

enumerator kEDMA\_ChannelSwapDisabled

Swap is disabled.

enumerator kEDMA\_ChannelReadWith8bitSwap

Swap occurs with respect to the read 8bit.

enumerator kEDMA\_ChannelReadWith16bitSwap

Swap occurs with respect to the read 16bit.

enumerator kEDMA\_ChannelReadWith32bitSwap

Swap occurs with respect to the read 32bit.

enumerator kEDMA\_ChannelWriteWith8bitSwap

Swap occurs with respect to the write 8bit.

enumerator kEDMA\_ChannelWriteWith16bitSwap

Swap occurs with respect to the write 16bit.

enumerator kEDMA\_ChannelWriteWith32bitSwap

Swap occurs with respect to the write 32bit.

eDMA channel system bus information, \_edma\_channel\_sys\_bus\_info

*Values:*

enumerator kEDMA\_PrivilegedAccessLevel

Privileged Access Level for DMA transfers. 0b - User protection level; 1b - Privileged protection level.

enumerator kEDMA\_MasterId

DMA's master ID when channel is active and master ID replication is enabled.

enum \_edma\_channel\_access\_type

eDMA4 channel access type

*Values:*

enumerator kEDMA\_ChannelDataAccess

Data access for eDMA4 transfers.

enumerator kEDMA\_ChannelInstructionAccess

Instruction access for eDMA4 transfers.

enum \_edma\_channel\_protection\_level

eDMA4 channel protection level

*Values:*

enumerator kEDMA\_ChannelProtectionLevelUser

user protection level for eDMA transfers.

enumerator kEDMA\_ChannelProtectionLevelPrivileged

Privileged protection level eDMA transfers.

enum \_edma\_channel\_security\_level

eDMA4 channel security level

*Values:*

enumerator kEDMA\_ChannelSecurityLevelNonSecure

non secure level for eDMA transfers.

enumerator kEDMA\_ChannelSecurityLevelSecure

secure level for eDMA transfers.

typedef enum \_edma\_transfer\_size edma\_transfer\_size\_t

eDMA transfer configuration

typedef enum \_edma\_modulo edma\_modulo\_t

eDMA modulo configuration

typedef enum \_edma\_bandwidth edma\_bandwidth\_t

Bandwidth control.

typedef enum \_edma\_channel\_link\_type edma\_channel\_link\_type\_t

Channel link type.

typedef enum \_edma\_transfer\_type edma\_transfer\_type\_t

eDMA transfer type

```
typedef struct _edma_channel_Preemption_config edma_channel_Preemption_config_t
    eDMA channel priority configuration
```

```
typedef struct _edma_minor_offset_config edma_minor_offset_config_t
    eDMA minor offset configuration
```

```
typedef enum edma_channel_memory_attribute edma_channel_memory_attribute_t
    eDMA channel memory attribute
```

```
typedef enum _edma_channel_swap_size edma_channel_swap_size_t
    eDMA4 channel swap size
```

```
typedef enum _edma_channel_access_type edma_channel_access_type_t
    eDMA4 channel access type
```

```
typedef enum _edma_channel_protection_level edma_channel_protection_level_t
    eDMA4 channel protection level
```

```
typedef enum _edma_channel_security_level edma_channel_security_level_t
    eDMA4 channel security level
```

```
typedef struct _edma_channel_config edma_channel_config_t
    eDMA4 channel configuration
```

```
typedef edma_core_tcd_t edma_tcd_t
    eDMA TCD.
```

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

```
typedef struct _edma_transfer_config edma_transfer_config_t
    edma4 channel transfer configuration
```

The transfer configuration structure support full feature configuration of the transfer control descriptor.

1.To perform a simple transfer, below members should be initialized at least .srcAddr - source address .dstAddr - destination address .srcWidthOfEachTransfer - data width of source address .dstWidthOfEachTransfer - data width of destination address, normally it should be as same as srcWidthOfEachTransfer .bytesEachRequest - bytes to be transferred in each DMA request .totalBytes - total bytes to be transferred .srcOffsetOfEachTransfer - offset value in bytes unit to be applied to source address as each source read is completed .dstOffsetOfEachTransfer - offset value in bytes unit to be applied to destination address as each destination write is completed enableChannelRequest - channel request can be enabled together with transfer configure submission

2.The transfer configuration structure also support advance feature: Programmable source/destination address range(MODULO) Programmable minor loop offset Programmable major loop offset Programmable channel chain feature Programmable channel transfer control descriptor link feature

---

**Note:** User should pay attention to the transfer size alignment limitation

- a. the bytesEachRequest should align with the srcWidthOfEachTransfer and the dstWidthOfEachTransfer that is to say bytesEachRequest % srcWidthOfEachTransfer should be 0
- b. the srcOffsetOfEachTransfer and dstOffsetOfEachTransfer must be aligne with transfer width
- c. the totalBytes should align with the bytesEachRequest

- d. the srcAddr should align with the srcWidthOfEachTransfer
  - e. the dstAddr should align with the dstWidthOfEachTransfer
  - f. the srcAddr should align with srcAddrModulo if modulo feature is enabled
  - g. the dstAddr should align with dstAddrModulo if modulo feature is enabled If anyone of above condition can not be satisfied, the edma4 interfaces will generate assert error.
- 

typedef struct *\_edma\_config* edma\_config\_t

eDMA global configuration structure.

typedef void (\*edma\_callback)(struct *\_edma\_handle* \*handle, void \*userData, bool transferDone, uint32\_t tcds)

Define callback function for eDMA.

This callback function is called in the EDMA interrupt handle. In normal mode, run into callback function means the transfer users need is done. In scatter gather mode, run into callback function means a transfer control block (tcd) is finished. Not all transfer finished, users can get the finished tcd numbers using interface EDMA\_GetUnusedTCDNumber.

**Param handle**

EDMA handle pointer, users shall not touch the values inside.

**Param userData**

The callback user parameter pointer. Users can use this parameter to involve things users need to change in EDMA callback function.

**Param transferDone**

If the current loaded transfer done. In normal mode it means if all transfer done. In scatter gather mode, this parameter shows is the current transfer block in EDMA register is done. As the load of core is different, it will be different if the new tcd loaded into EDMA registers while this callback called. If true, it always means new tcd still not loaded into registers, while false means new tcd already loaded into registers.

**Param tcds**

How many tcds are done from the last callback. This parameter only used in scatter gather mode. It tells user how many tcds are finished between the last callback and this.

typedef struct *\_edma\_handle* edma\_handle\_t

eDMA transfer handle structure

FSL\_EDMA\_DRIVER\_EDMA4

eDMA driver name

EDMA\_ALLOCATE\_TCD(name, number)

Macro used for allocate edma TCD.

DMA\_DCHPRI\_INDEX(channel)

Compute the offset unit from DCHPRI3.

struct *\_edma\_channel\_Preemption\_config*

#include <fsl\_edma.h> eDMA channel priority configuration

**Public Members**

bool enableChannelPreemption

If true: a channel can be suspended by other channel with higher priority

bool enablePreemptAbility

If true: a channel can suspend other channel with low priority

uint8\_t channelPriority

Channel priority

struct \_\_edma\_\_minor\_offset\_config

#include <fsl\_edma.h> eDMA minor offset configuration

### Public Members

bool enableSrcMinorOffset

Enable(true) or Disable(false) source minor loop offset.

bool enableDestMinorOffset

Enable(true) or Disable(false) destination minor loop offset.

uint32\_t minorOffset

Offset for a minor loop mapping.

struct \_\_edma\_\_channel\_config

#include <fsl\_edma.h> eDMA4 channel configuration

### Public Members

*edma\_channel\_preemption\_config\_t* channelPreemptionConfig

channel preemption configuration

*edma\_channel\_memory\_attribute\_t* channelReadMemoryAttribute

channel memory read attribute configuration

*edma\_channel\_memory\_attribute\_t* channelWriteMemoryAttribute

channel memory write attribute configuration

*edma\_channel\_swap\_size\_t* channelSwapSize

channel swap size configuration

*edma\_channel\_access\_type\_t* channelAccessType

channel access type configuration

uint8\_t channelDataSignExtensionBitPosition

channel data sign extension bit position configuration

uint32\_t channelRequestSource

hardware service request source for the channel

bool enableMasterIDReplication

enable master ID replication

*edma\_channel\_security\_level\_t* securityLevel

security level

*edma\_channel\_protection\_level\_t* protectionLevel

protection level

struct \_\_edma\_\_transfer\_config

#include <fsl\_edma.h> edma4 channel transfer configuration

The transfer configuration structure support full feature configuration of the transfer control descriptor.

1.To perform a simple transfer, below members should be initialized at least .srcAddr - source address .dstAddr - destination address .srcWidthOfEachTransfer - data width of source address .dstWidthOfEachTransfer - data width of destination address, normally it should be as same as srcWidthOfEachTransfer .bytesEachRequest - bytes to be transferred in each DMA request .totalBytes - total bytes to be transferred .srcOffsetOfEachTransfer - offset value in bytes unit to be applied to source address as each source read is completed .dstOffsetOfEachTransfer - offset value in bytes unit to be applied to destination address as each destination write is completed enablchannelRequest - channel request can be enabled together with transfer configure submission

2.The transfer configuration structure also support advance feature: Programmable source/destination address range(MODULO) Programmable minor loop offset Programmable major loop offset Programmable channel chain feature Programmable channel transfer control descriptor link feature

---

**Note:** User should pay attention to the transfer size alignment limitation

- a. the bytesEachRequest should align with the srcWidthOfEachTransfer and the dstWidthOfEachTransfer that is to say  $\text{bytesEachRequest} \% \text{srcWidthOfEachTransfer}$  should be 0
  - b. the srcOffsetOfEachTransfer and dstOffsetOfEachTransfer must be aligne with transfer width
  - c. the totalBytes should align with the bytesEachRequest
  - d. the srcAddr should align with the srcWidthOfEachTransfer
  - e. the dstAddr should align with the dstWidthOfEachTransfer
  - f. the srcAddr should align with srcAddrModulo if modulo feature is enabled
  - g. the dstAddr should align with dstAddrModulo if modulo feature is enabled If anyone of above condition can not be satisfied, the edma4 interfaces will generate assert error.
- 

## Public Members

uint32\_t srcAddr

Source data address.

uint32\_t destAddr

Destination data address.

*edma\_transfer\_size\_t* srcTransferSize

Source data transfer size.

*edma\_transfer\_size\_t* destTransferSize

Destination data transfer size.

int16\_t srcOffset

Sign-extended offset value in byte unit applied to the current source address to form the next-state value as each source read is completed

int16\_t destOffset

Sign-extended offset value in byte unit applied to the current destination address to form the next-state value as each destination write is completed.

uint32\_t minorLoopBytes

bytes in each minor loop or each request range: 1 - ( $2^{30} - 1$ ) when minor loop mapping is enabled range: 1 - ( $2^{10} - 1$ ) when minor loop mapping is enabled and source or dest minor loop offset is enabled range: 1 - ( $2^{32} - 1$ ) when minor loop mapping is disabled

uint32\_t majorLoopCounts

minor loop counts in each major loop, should be 1 at least for each transfer range: (0 - ( $2^{15} - 1$ )) when minor loop channel link is disabled range: (0 - ( $2^9 - 1$ )) when minor loop channel link is enabled total bytes in a transfer = minorLoopCountsEachMajorLoop \* bytesEachMinorLoop

uint16\_t enabledInterruptMask

channel interrupt to enable, can be OR'ed value of \_edma\_interrupt\_enable

edma\_modulo\_t srcAddrModulo

source circular data queue range

int32\_t srcMajorLoopOffset

source major loop offset

edma\_modulo\_t dstAddrModulo

destination circular data queue range

int32\_t dstMajorLoopOffset

destination major loop offset

bool enableSrcMinorLoopOffset

enable source minor loop offset

bool enableDstMinorLoopOffset

enable dest minor loop offset

int32\_t minorLoopOffset

burst offset, the offset will be applied after minor loop update

bool enableChannelMajorLoopLink

channel link when major loop complete

uint32\_t majorLoopLinkChannel

major loop link channel number

bool enableChannelMinorLoopLink

channel link when minor loop complete

uint32\_t minorLoopLinkChannel

minor loop link channel number

edma\_tcd\_t \*linkTCD

pointer to the link transfer control descriptor

struct \_edma\_config

#include <fsl\_edma.h> eDMA global configuration structure.

## Public Members

bool enableContinuousLinkMode

Enable (true) continuous link mode. Upon minor loop completion, the channel activates again if that channel has a minor loop channel link enabled and the link channel is itself.

bool enableMasterIdReplication

Enable (true) master ID replication. If Master ID replication is disabled, the privileged protection level (supervisor mode) for eDMA4 transfers is used.

bool enableGlobalChannelLink

Enable(true) channel linking is available and controlled by each channel's link settings.

bool enableHaltOnError

Enable (true) transfer halt on error. Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

bool enableDebugMode

Enable(true) eDMA4 debug mode. When in debug mode, the eDMA4 stalls the start of a new channel. Executing channels are allowed to complete.

bool enableRoundRobinArbitration

Enable(true) channel linking is available and controlled by each channel's link settings.

*edma\_channel\_config\_t* \*channelConfig[1]

channel preemption configuration

struct *\_edma\_handle*

*#include <fsl\_edma.h>* eDMA transfer handle structure

### Public Members

*edma\_callback* callback

Callback function for major count exhausted.

void \*userData

Callback function parameter.

*EDMA\_ChannelType* \*channelBase

eDMA peripheral channel base address.

*EDMA\_Type* \*base

eDMA peripheral base address

*EDMA\_TCDType* \*tcdBase

eDMA peripheral tcd base address.

*edma\_tcd\_t* \*tcdPool

Pointer to memory stored TCDs.

uint32\_t channel

eDMA channel number.

volatile int8\_t header

The first TCD index. Should point to the next TCD to be loaded into the eDMA engine.

volatile int8\_t tail

The last TCD index. Should point to the next TCD to be stored into the memory pool.

volatile int8\_t tcdUsed

The number of used TCD slots. Should reflect the number of TCDs can be used/loaded in the memory.

volatile int8\_t tcdSize

The total number of TCD slots in the queue.



## 2.2 eDMA core Driver

```
enum _edma_tcd_type
    eDMA tcd flag type
    Values:
    enumerator kEDMA_EDMA4Flag
        Data access for eDMA4 transfers.
    enumerator kEDMA_EDMA5Flag
        Instruction access for eDMA4 transfers.
typedef struct _edma_core_mp edma_core_mp_t
    edma core channel struture definition
typedef struct _edma_core_channel edma_core_channel_t
    edma core channel struture definition
typedef enum _edma_tcd_type edma_tcd_type_t
    eDMA tcd flag type
typedef struct _edma5_core_tcd edma5_core_tcd_t
    edma5 core TCD struture definition
typedef struct _edma4_core_tcd edma4_core_tcd_t
    edma4 core TCD struture definition
typedef struct _edma_core_tcd edma_core_tcd_t
    edma core TCD struture definition
typedef edma_core_channel_t EDMA_ChannelType
    EDMA typedef.
typedef edma_core_tcd_t EDMA_TCDType
typedef void EDMA_Type
DMA_CORE_MP_CSR_EDBG_MASK
DMA_CORE_MP_CSR_ERCA_MASK
DMA_CORE_MP_CSR_HAE_MASK
DMA_CORE_MP_CSR_HALT_MASK
DMA_CORE_MP_CSR_GCLC_MASK
DMA_CORE_MP_CSR_GMRC_MASK
DMA_CORE_MP_CSR_EDBG(x)
DMA_CORE_MP_CSR_ERCA(x)
DMA_CORE_MP_CSR_HAE(x)
DMA_CORE_MP_CSR_HALT(x)
DMA_CORE_MP_CSR_GCLC(x)
DMA_CORE_MP_CSR_GMRC(x)
DMA_CSR_INTMAJOR_MASK
```

DMA\_CSR\_INTHALF\_MASK  
DMA\_CSR\_DREQ\_MASK  
DMA\_CSR\_ESG\_MASK  
DMA\_CSR\_BWC\_MASK  
DMA\_CSR\_BWC(x)  
DMA\_CSR\_START\_MASK  
DMA\_CITER\_ELINKNO\_CITER\_MASK  
DMA\_BITER\_ELINKNO\_BITER\_MASK  
DMA\_CITER\_ELINKNO\_CITER\_SHIFT  
DMA\_CITER\_ELINKYES\_CITER\_MASK  
DMA\_CITER\_ELINKYES\_CITER\_SHIFT  
DMA\_ATTR\_SMOD\_MASK  
DMA\_ATTR\_DMOD\_MASK  
DMA\_CITER\_ELINKNO\_ELINK\_MASK  
DMA\_CSR\_MAJORELINK\_MASK  
DMA\_BITER\_ELINKYES\_ELINK\_MASK  
DMA\_CITER\_ELINKYES\_ELINK\_MASK  
DMA\_CSR\_MAJORLINKCH\_MASK  
DMA\_BITER\_ELINKYES\_LINKCH\_MASK  
DMA\_CITER\_ELINKYES\_LINKCH\_MASK  
DMA\_NBYTES\_MLOFFYES\_MLOFF\_MASK  
DMA\_NBYTES\_MLOFFYES\_DMLOE\_MASK  
DMA\_NBYTES\_MLOFFYES\_SMLOE\_MASK  
DMA\_NBYTES\_MLOFFNO\_NBYTES\_MASK  
DMA\_ATTR\_DMOD(x)  
DMA\_ATTR\_SMOD(x)  
DMA\_BITER\_ELINKYES\_LINKCH(x)  
DMA\_CITER\_ELINKYES\_LINKCH(x)  
DMA\_NBYTES\_MLOFFYES\_MLOFF(x)  
DMA\_NBYTES\_MLOFFYES\_DMLOE(x)  
DMA\_NBYTES\_MLOFFYES\_SMLOE(x)  
DMA\_NBYTES\_MLOFFNO\_NBYTES(x)  
DMA\_NBYTES\_MLOFFYES\_NBYTES(x)

DMA\_ATTR\_DSIZE(x)  
DMA\_ATTR\_SSIZE(x)  
DMA\_CSR\_DREQ(x)  
DMA\_CSR\_MAJORLINKCH(x)  
DMA\_CH\_MATTR\_WCACHE(x)  
DMA\_CH\_MATTR\_RCACHE(x)  
DMA\_CH\_CSR\_SIGNEXT\_MASK  
DMA\_CH\_CSR\_SIGNEXT\_SHIFT  
DMA\_CH\_CSR\_SWAP\_MASK  
DMA\_CH\_CSR\_SWAP\_SHIFT  
DMA\_CH\_SBR\_INSTR\_MASK  
DMA\_CH\_SBR\_INSTR\_SHIFT  
DMA\_CH\_SBR\_EMI\_MASK  
DMA\_CH\_SBR\_EMI\_SHIFT  
DMA\_CH\_MUX\_SOURCE(x)  
DMA\_ERR\_DBE\_FLAG  
    **DMA error flag.**  
DMA\_ERR\_SBE\_FLAG  
DMA\_ERR\_SGE\_FLAG  
DMA\_ERR\_NCE\_FLAG  
DMA\_ERR\_DOE\_FLAG  
DMA\_ERR\_DAE\_FLAG  
DMA\_ERR\_SOE\_FLAG  
DMA\_ERR\_SAE\_FLAG  
DMA\_ERR\_ERRCHAN\_FLAG  
DMA\_ERR\_ECX\_FLAG  
DMA\_ERR\_FLAG  
DMA\_CLEAR\_DONE\_STATUS(base, channel)  
    **get/clear DONE bit**  
DMA\_GET\_DONE\_STATUS(base, channel)  
DMA\_ENABLE\_ERROR\_INT(base, channel)  
    **enable/disable error interrupt**  
DMA\_DISABLE\_ERROR\_INT(base, channel)  
DMA\_CLEAR\_ERROR\_STATUS(base, channel)  
    **get/clear error status**

DMA\_GET\_ERROR\_STATUS(base, channel)  
DMA\_CLEAR\_INT\_STATUS(base, channel)  
    get/clear INT status  
DMA\_GET\_INT\_STATUS(base, channel)  
DMA\_ENABLE\_MAJOR\_INT(base, channel)  
    enable/disable MAJOR/HALF INT  
DMA\_ENABLE\_HALF\_INT(base, channel)  
DMA\_DISABLE\_MAJOR\_INT(base, channel)  
DMA\_DISABLE\_HALF\_INT(base, channel)  
EDMA\_TCD\_ALIGN\_SIZE  
    EDMA tcd align size.  
EDMA\_CORE\_BASE(base)  
    EDMA base address convert macro.  
EDMA\_MP\_BASE(base)  
EDMA\_CHANNEL\_BASE(base, channel)  
EDMA\_TCD\_BASE(base, channel)  
EDMA\_TCD\_TYPE(x)  
    EDMA TCD type macro.  
EDMA\_TCD\_SADDR(tcd, flag)  
    EDMA TCD address convert macro.  
EDMA\_TCD\_SOFF(tcd, flag)  
EDMA\_TCD\_ATTR(tcd, flag)  
EDMA\_TCD\_NBYTES(tcd, flag)  
EDMA\_TCD\_SLAST(tcd, flag)  
EDMA\_TCD\_DADDR(tcd, flag)  
EDMA\_TCD\_DOFF(tcd, flag)  
EDMA\_TCD\_CITER(tcd, flag)  
EDMA\_TCD\_DLAST\_SGA(tcd, flag)  
EDMA\_TCD\_CSR(tcd, flag)  
EDMA\_TCD\_BITER(tcd, flag)  
struct \_edma\_core\_mp  
    #include <fsl\_edma\_core.h> edma core channel structure definition

### Public Members

\_\_IO uint32\_t MP\_CSR  
    Channel Control and Status, array offset: 0x10000, array step: 0x10000

```

__IO uint32_t MP_ES
    Channel Error Status, array offset: 0x10004, array step: 0x10000
struct __edma_core_channel
    #include <fsl_edma_core.h> edma core channel struture definition

```

### Public Members

```

__IO uint32_t CH_CSR
    Channel Control and Status, array offset: 0x10000, array step: 0x10000
__IO uint32_t CH_ES
    Channel Error Status, array offset: 0x10004, array step: 0x10000
__IO uint32_t CH_INT
    Channel Interrupt Status, array offset: 0x10008, array step: 0x10000
__IO uint32_t CH_SBR
    Channel System Bus, array offset: 0x1000C, array step: 0x10000
__IO uint32_t CH_PRI
    Channel Priority, array offset: 0x10010, array step: 0x10000
struct __edma5_core_tcd
    #include <fsl_edma_core.h> edma5 core TCD struture definition

```

### Public Members

```

__IO uint32_t SADDR
    SADDR register, used to save source address
__IO uint32_t SADDR_HIGH
    SADDR HIGH register, used to save source address
__IO uint16_t SOFF
    SOFF register, save offset bytes every transfer
__IO uint16_t ATTR
    ATTR register, source/destination transfer size and modulo
__IO uint32_t NBYTES
    Nbytes register, minor loop length in bytes
__IO uint32_t SLAST
    SLAST register
__IO uint32_t SLAST_SDA_HIGH
    SLAST SDA HIGH register
__IO uint32_t DADDR
    DADDR register, used for destination address
__IO uint32_t DADDR_HIGH
    DADDR HIGH register, used for destination address
__IO uint32_t DLAST_SGA
    DLASTSGA register, next tcd address used in scatter-gather mode
__IO uint32_t DLAST_SGA_HIGH
    DLASTSGA HIGH register, next tcd address used in scatter-gather mode

```

```
__IO uint16_t DOFF
    DOFF register, used for destination offset
__IO uint16_t CITER
    CITER register, current minor loop numbers, for unfinished minor loop.
__IO uint16_t CSR
    CSR register, for TCD control status
__IO uint16_t BITER
    BITER register, begin minor loop count.
uint8_t RESERVED[16]
    Aligned 64 bytes
struct _edma4_core_tcd
    #include <fsl_edma_core.h> edma4 core TCD struture definition
```

### Public Members

```
__IO uint32_t SADDR
    SADDR register, used to save source address
__IO uint16_t SOFF
    SOFF register, save offset bytes every transfer
__IO uint16_t ATTR
    ATTR register, source/destination transfer size and modulo
__IO uint32_t NBYTES
    Nbytes register, minor loop length in bytes
__IO uint32_t SLAST
    SLAST register
__IO uint32_t DADDR
    DADDR register, used for destination address
__IO uint16_t DOFF
    DOFF register, used for destination offset
__IO uint16_t CITER
    CITER register, current minor loop numbers, for unfinished minor loop.
__IO uint32_t DLAST_SGA
    DLASTSGA register, next tcd address used in scatter-gather mode
__IO uint16_t CSR
    CSR register, for TCD control status
__IO uint16_t BITER
    BITER register, begin minor loop count.
struct _edma_core_tcd
    #include <fsl_edma_core.h> edma core TCD struture definition
union MP_REGS
```

**Public Members**

```
struct _edma_core_mp EDMA5_REG
```

```
struct EDMA5_REG
```

**Public Members**

```
__IO uint32_t MP_INT_LOW
```

Channel Control and Status, array offset: 0x10008, array step: 0x10000

```
__IO uint32_t MP_INT_HIGH
```

Channel Control and Status, array offset: 0x1000C, array step: 0x10000

```
__IO uint32_t MP_HRS_LOW
```

Channel Control and Status, array offset: 0x10010, array step: 0x10000

```
__IO uint32_t MP_HRS_HIGH
```

Channel Control and Status, array offset: 0x10014, array step: 0x10000

```
__IO uint32_t MP_STOPCH
```

Channel Control and Status, array offset: 0x10020, array step: 0x10000

```
__IO uint32_t MP_SSR_LOW
```

Channel Control and Status, array offset: 0x10030, array step: 0x10000

```
__IO uint32_t MP_SSR_HIGH
```

Channel Control and Status, array offset: 0x10034, array step: 0x10000

```
__IO uint32_t CH_GRPRI [64]
```

Channel Control and Status, array offset: 0x10100, array step: 0x10000

```
__IO uint32_t CH_MUX [64]
```

Channel Control and Status, array offset: 0x10200, array step: 0x10000

```
__IO uint32_t CH_PROT [64]
```

Channel Control and Status, array offset: 0x10400, array step: 0x10000

```
union CH_REGS
```

**Public Members**

```
struct _edma_core_channel EDMA5_REG
```

```
struct _edma_core_channel EDMA4_REG
```

```
struct EDMA5_REG
```

**Public Members**

```
__IO uint32_t CH_MATTR
```

Memory Attributes Register, array offset: 0x10018, array step: 0x8000

```
struct EDMA4_REG
```

### Public Members

`__IO uint32_t CH_MUX`  
Channel Multiplexor Configuration, array offset: 0x10014, array step: 0x10000

`__IO uint16_t CH_MATTR`  
Memory Attributes Register, array offset: 0x10018, array step: 0x8000

`union TCD_REGS`

### Public Members

`edma4_core_tcd_t edma4_tcd`

## 2.3 eDMA soc Driver

`FSL_EDMA_SOC_DRIVER_VERSION`  
Driver version 2.0.0.

`FSL_EDMA_SOC_IP_DMA3`  
DMA IP version.

`FSL_EDMA_SOC_IP_DMA4`

`EDMA_BASE_PTRS`  
DMA base table.

`EDMA_CHN_IRQS`

`FSL_FEATURE_EDMA_HAS_CHANNEL_MUX`  
Verify dma base and request source

`EDMA_CHANNEL_HAS_REQUEST_SOURCE(base, source)`

`EDMA_CHANNEL_OFFSET`  
EDMA base address convert macro.

`EDMA_CHANNEL_ARRAY_STEP(base)`

## 2.4 ENET: Ethernet MAC Driver

`void ENET_GetDefaultConfig(enet_config_t *config)`

Gets the ENET default configuration structure.

The purpose of this API is to get the default ENET MAC controller configure structure for `ENET_Init()`. User may use the initialized structure unchanged in `ENET_Init()`, or modify some fields of the structure before calling `ENET_Init()`. Example:

```
enet_config_t config;
ENET_GetDefaultConfig(&config);
```

### Parameters

- `config` – The ENET mac controller configuration structure pointer.



*status\_t* ENET\_Up(ENET\_Type \*base, *enet\_handle\_t* \*handle, const *enet\_config\_t* \*config, const *enet\_buffer\_config\_t* \*bufferConfig, uint8\_t \*macAddr, uint32\_t srcClock\_Hz)

Initializes the ENET module.

This function initializes the module with the ENET configuration.

---

**Note:** ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining “ENET\_ENHANCEDBUFFERDESCRIPTOR\_MODE” and calling ENET\_Ptp1588Configure() to configure the 1588 feature and related buffers after calling ENET\_Up().

---

#### Parameters

- base – ENET peripheral base address.
- handle – ENET handler pointer.
- config – ENET mac configuration structure pointer. The “enet\_config\_t” type mac configuration return from ENET\_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.
- bufferConfig – ENET buffer configuration structure pointer. The buffer configuration should be prepared for ENET Initialization. It is the start address of “ringNum” enet\_buffer\_config structures. To support added multi-ring features in some soc and compatible with the previous enet driver version. For single ring supported, this bufferConfig is a buffer configure structure pointer, for multi-ring supported and used case, this bufferConfig pointer should be a buffer configure structure array pointer.
- macAddr – ENET mac address of Ethernet device. This MAC address should be provided.
- srcClock\_Hz – The internal module clock source for MII clock.

#### Return values

- kStatus\_Success – Succeed to initialize the ethernet driver.
- kStatus\_ENET\_InitMemoryFail – Init fails since buffer memory is not enough.

*status\_t* ENET\_Init(ENET\_Type \*base, *enet\_handle\_t* \*handle, const *enet\_config\_t* \*config, const *enet\_buffer\_config\_t* \*bufferConfig, uint8\_t \*macAddr, uint32\_t srcClock\_Hz)

Initializes the ENET module.

This function ungates the module clock and initializes it with the ENET configuration.

---

**Note:** ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining “ENET\_ENHANCEDBUFFERDESCRIPTOR\_MODE” and calling ENET\_Ptp1588Configure() to configure the 1588 feature and related buffers after calling ENET\_Init().

---

#### Parameters

- base – ENET peripheral base address.
- handle – ENET handler pointer.

- `config` – ENET mac configuration structure pointer. The “enet\_config\_t” type mac configuration return from `ENET_GetDefaultConfig` can be used directly. It is also possible to verify the Mac configuration using other methods.
- `bufferConfig` – ENET buffer configuration structure pointer. The buffer configuration should be prepared for ENET Initialization. It is the start address of “ringNum” `enet_buffer_config` structures. To support added multi-ring features in some soc and compatible with the previous enet driver version. For single ring supported, this `bufferConfig` is a buffer configure structure pointer, for multi-ring supported and used case, this `bufferConfig` pointer should be a buffer configure structure array pointer.
- `macAddr` – ENET mac address of Ethernet device. This MAC address should be provided.
- `srcClock_Hz` – The internal module clock source for MII clock.

#### Return values

- `kStatus_Success` – Succeed to initialize the ethernet driver.
- `kStatus_ENET_InitMemoryFail` – Init fails since buffer memory is not enough.

`void ENET_Down(ENET_Type *base)`

Stops the ENET module.

This function disables the ENET module.

#### Parameters

- `base` – ENET peripheral base address.

`void ENET_Deinit(ENET_Type *base)`

Deinitializes the ENET module.

This function gates the module clock, clears ENET interrupts, and disables the ENET module.

#### Parameters

- `base` – ENET peripheral base address.

`static inline void ENET_Reset(ENET_Type *base)`

Resets the ENET module.

This function restores the ENET module to reset state. Note that this function sets all registers to reset state. As a result, the ENET module can't work after calling this function.

#### Parameters

- `base` – ENET peripheral base address.

`void ENET_SetMII(ENET_Type *base, enet_mii_speed_t speed, enet_mii_duplex_t duplex)`

Sets the ENET MII speed and duplex.

This API is provided to dynamically change the speed and duplex for MAC.

#### Parameters

- `base` – ENET peripheral base address.
- `speed` – The speed of the RMII mode.
- `duplex` – The duplex of the RMII mode.

`void ENET_SetSMI(ENET_Type *base, uint32_t srcClock_Hz, bool isPreambleDisabled)`  
Sets the ENET SMI(serial management interface)- MII management interface.

**Parameters**

- `base` – ENET peripheral base address.
- `srcClock_Hz` – This is the ENET module clock frequency. See clock distribution.
- `isPreambleDisabled` – The preamble disable flag.
  - `true` Enables the preamble.
  - `false` Disables the preamble.

`static inline bool ENET_GetSMI(ENET_Type *base)`  
Gets the ENET SMI- MII management interface configuration.

This API is used to get the SMI configuration to check whether the MII management interface has been set.

**Parameters**

- `base` – ENET peripheral base address.

**Returns**

The SMI setup status true or false.

`static inline uint32_t ENET_ReadSMIData(ENET_Type *base)`  
Reads data from the PHY register through an SMI interface.

**Parameters**

- `base` – ENET peripheral base address.

**Returns**

The data read from PHY

`static inline void ENET_StartSMIWrite(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, enet_mii_write_t operation, uint16_t data)`

Sends the MDIO IEEE802.3 Clause 22 format write command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated `ENET_MDIOWrite()` can be called. For customized requirements, implement with combining separated APIs.

**Parameters**

- `base` – ENET peripheral base address.
- `phyAddr` – The PHY address. Range from 0 ~ 31.
- `regAddr` – The PHY register address. Range from 0 ~ 31.
- `operation` – The write operation.
- `data` – The data written to PHY.

`static inline void ENET_StartSMIRead(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, enet_mii_read_t operation)`

Sends the MDIO IEEE802.3 Clause 22 format read command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated `ENET_MDIORead()` can be called. For customized requirements, implement with combining separated APIs.

**Parameters**

- `base` – ENET peripheral base address.

- phyAddr – The PHY address. Range from 0 ~ 31.
- regAddr – The PHY register address. Range from 0 ~ 31.
- operation – The read operation.

*status\_t* ENET\_MDIOWrite(ENET\_Type \*base, uint8\_t phyAddr, uint8\_t regAddr, uint16\_t data)  
MDIO write with IEEE802.3 Clause 22 format.

#### Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address. Range from 0 ~ 31.
- regAddr – The PHY register. Range from 0 ~ 31.
- data – The data written to PHY.

#### Returns

kStatus\_Success MDIO access succeeds.

#### Returns

kStatus\_Timeout MDIO access timeout.

*status\_t* ENET\_MDIORead(ENET\_Type \*base, uint8\_t phyAddr, uint8\_t regAddr, uint16\_t \*pData)

MDIO read with IEEE802.3 Clause 22 format.

#### Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address. Range from 0 ~ 31.
- regAddr – The PHY register. Range from 0 ~ 31.
- pData – The data read from PHY.

#### Returns

kStatus\_Success MDIO access succeeds.

#### Returns

kStatus\_Timeout MDIO access timeout.

static inline void ENET\_StartExtC45SMIWriteReg(ENET\_Type \*base, uint8\_t portAddr, uint8\_t devAddr, uint16\_t regAddr)

Sends the MDIO IEEE802.3 Clause 45 format write register command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated ENET\_MDIOC45Write()/ENET\_MDIOC45Read() can be called. For customized requirements, implement with combining separated APIs.

#### Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.

static inline void ENET\_StartExtC45SMIWriteData(ENET\_Type \*base, uint8\_t portAddr, uint8\_t devAddr, uint16\_t data)

Sends the MDIO IEEE802.3 Clause 45 format write data command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated ENET\_MDIOC45Write() can be called. For customized requirements, implement with combining separated APIs.

**Parameters**

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- data – The data written to PHY.

```
static inline void ENET_StartExtC45SMIReadData(ENET_Type *base, uint8_t portAddr, uint8_t devAddr)
```

Sends the MDIO IEEE802.3 Clause 45 format read data command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated ENET\_MDIOC45Read() can be called. For customized requirements, implement with combining separated APIs.

**Parameters**

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.

```
status_t ENET_MDIOC45Write(ENET_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr, uint16_t data)
```

MDIO write with IEEE802.3 Clause 45 format.

**Parameters**

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.
- data – The data written to PHY.

**Returns**

kStatus\_Success MDIO access succeeds.

**Returns**

kStatus\_Timeout MDIO access timeout.

```
status_t ENET_MDIOC45Read(ENET_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr, uint16_t *pData)
```

MDIO read with IEEE802.3 Clause 45 format.

**Parameters**

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.
- pData – The data read from PHY.

**Returns**

kStatus\_Success MDIO access succeeds.

**Returns**

kStatus\_Timeout MDIO access timeout.

```
static inline void ENET_SetRGMIIClockDelay(ENET_Type *base, bool txEnabled, bool rxEnabled)
```

Control the usage of the delayed tx/rx RGMII clock.

#### Parameters

- base – ENET peripheral base address.
- txEnabled – Enable or disable to generate the delayed version of RGMII\_TXC.
- rxEnabled – Enable or disable to use the delayed version of RGMII\_RXC.

```
void ENET_SetMacAddr(ENET_Type *base, uint8_t *macAddr)
```

Sets the ENET module Mac address.

#### Parameters

- base – ENET peripheral base address.
- macAddr – The six-byte Mac address pointer. The pointer is allocated by application and input into the API.

```
void ENET_GetMacAddr(ENET_Type *base, uint8_t *macAddr)
```

Gets the ENET module Mac address.

#### Parameters

- base – ENET peripheral base address.
- macAddr – The six-byte Mac address pointer. The pointer is allocated by application and input into the API.

```
void ENET_AddMulticastGroup(ENET_Type *base, uint8_t *address)
```

Adds the ENET device to a multicast group.

#### Parameters

- base – ENET peripheral base address.
- address – The six-byte multicast group address which is provided by application.

```
void ENET_LeaveMulticastGroup(ENET_Type *base, uint8_t *address)
```

Moves the ENET device from a multicast group.

#### Parameters

- base – ENET peripheral base address.
- address – The six-byte multicast group address which is provided by application.

```
static inline void ENET_ActiveRead(ENET_Type *base)
```

Activates frame reception for multiple rings.

This function is to active the enet read process.

---

**Note:** This must be called after the MAC configuration and state are ready. It must be called after the ENET\_Init(). This should be called when the frame reception is required.

---

#### Parameters

- base – ENET peripheral base address.

```
static inline void ENET_EnableSleepMode(ENET_Type *base, bool enable)
```

Enables/disables the MAC to enter sleep mode. This function is used to set the MAC enter sleep mode. When entering sleep mode, the magic frame wakeup interrupt should be enabled to wake up MAC from the sleep mode and reset it to normal mode.

#### Parameters

- base – ENET peripheral base address.
- enable – True enable sleep mode, false disable sleep mode.

```
static inline void ENET_GetAccelFunction(ENET_Type *base, uint32_t *txAccelOption, uint32_t *rxAccelOption)
```

Gets ENET transmit and receive accelerator functions from MAC controller.

#### Parameters

- base – ENET peripheral base address.
- txAccelOption – The transmit accelerator option. The “enet\_tx\_accelerator\_t” is recommended to be used to as the mask to get the exact the accelerator option.
- rxAccelOption – The receive accelerator option. The “enet\_rx\_accelerator\_t” is recommended to be used to as the mask to get the exact the accelerator option.

```
static inline void ENET_EnableInterrupts(ENET_Type *base, uint32_t mask)
```

Enables the ENET interrupt.

This function enables the ENET interrupt according to the provided mask. The mask is a logical OR of enumeration members. See `enet_interrupt_enable_t`. For example, to enable the TX frame interrupt and RX frame interrupt, do the following.

```
ENET_EnableInterrupts(ENET, kENET_TxFrameInterrupt | kENET_RxFrameInterrupt);
```

#### Parameters

- base – ENET peripheral base address.
- mask – ENET interrupts to enable. This is a logical OR of the enumeration `enet_interrupt_enable_t`.

```
static inline void ENET_DisableInterrupts(ENET_Type *base, uint32_t mask)
```

Disables the ENET interrupt.

This function disables the ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See `enet_interrupt_enable_t`. For example, to disable the TX frame interrupt and RX frame interrupt, do the following.

```
ENET_DisableInterrupts(ENET, kENET_TxFrameInterrupt | kENET_RxFrameInterrupt);
```

#### Parameters

- base – ENET peripheral base address.
- mask – ENET interrupts to disable. This is a logical OR of the enumeration `enet_interrupt_enable_t`.

```
static inline uint32_t ENET_GetInterruptStatus(ENET_Type *base)
```

Gets the ENET interrupt status flag.

#### Parameters

- base – ENET peripheral base address.

**Returns**

The event status of the interrupt source. This is the logical OR of members of the enumeration `enet_interrupt_enable_t`.

`static inline void ENET_ClearInterruptStatus(ENET_Type *base, uint32_t mask)`

Clears the ENET interrupt events status flag.

This function clears enabled ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See the `enet_interrupt_enable_t`. For example, to clear the TX frame interrupt and RX frame interrupt, do the following.

```
ENET_ClearInterruptStatus(ENET, kENET_TxFrameInterrupt | kENET_RxFrameInterrupt);
```

**Parameters**

- `base` – ENET peripheral base address.
- `mask` – ENET interrupt source to be cleared. This is the logical OR of members of the enumeration `enet_interrupt_enable_t`.

`void ENET_SetRxISRHandler(ENET_Type *base, enet_isr_t ISRHandler)`

Set the second level Rx IRQ handler.

**Parameters**

- `base` – ENET peripheral base address.
- `ISRHandler` – The handler to install.

`void ENET_SetTxISRHandler(ENET_Type *base, enet_isr_t ISRHandler)`

Set the second level Tx IRQ handler.

**Parameters**

- `base` – ENET peripheral base address.
- `ISRHandler` – The handler to install.

`void ENET_SetErrISRHandler(ENET_Type *base, enet_isr_t ISRHandler)`

Set the second level Err IRQ handler.

**Parameters**

- `base` – ENET peripheral base address.
- `ISRHandler` – The handler to install.

`void ENET_GetRxErrBeforeReadFrame(enet_handle_t *handle, enet_data_error_stats_t *eErrorStatic, uint8_t ringId)`

Gets the error statistics of a received frame for ENET specified ring.

This API must be called after the `ENET_GetRxFrameSize` and before the `ENET_ReadFrame()`. If the `ENET_GetRxFrameSize` returns `kStatus_ENET_RxFrameError`, the `ENET_GetRxErrBeforeReadFrame` can be used to get the exact error statistics. This is an example.

```
status = ENET_GetRxFrameSize(&g_handle, &length, 0);
if (status == kStatus_ENET_RxFrameError)
{
    Comments: Get the error information of the received frame.
    ENET_GetRxErrBeforeReadFrame(&g_handle, &eErrStatic, 0);
    Comments: update the receive buffer.
    ENET_ReadFrame(EXAMPLE_ENET, &g_handle, NULL, 0);
}
```

**Parameters**



- `handle` – The ENET handler structure pointer. This is the same handler pointer used in the `ENET_Init`.
- `eErrorStatic` – The error statistics structure pointer.
- `ringId` – The ring index, range from 0 ~ `(FSL_FEATURE_ENET_INSTANCE_QUEUEn(x) - 1)`.

`void ENET_EnableStatistics(ENET_Type *base, bool enable)`

Enables/disables collection of transfer statistics.

Note that this function does not reset any of the already collected data, use the function `ENET_ResetStatistics` to clear the transfer statistics if needed.

#### Parameters

- `base` – ENET peripheral base address.
- `enable` – True enable statistics collection, false disable statistics collection.

`void ENET_GetStatistics(ENET_Type *base, enet_transfer_stats_t *statistics)`

Gets transfer statistics.

Copies the actual value of hardware counters into the provided structure. Calling this function does not reset the counters in hardware.

#### Parameters

- `base` – ENET peripheral base address.
- `statistics` – The statistics structure pointer.

`void ENET_ResetStatistics(ENET_Type *base)`

Resets transfer statistics.

Sets the value of hardware transfer counters to zero.

#### Parameters

- `base` – ENET peripheral base address.

`status_t ENET_GetRxFramSize(enet_handle_t *handle, uint32_t *length, uint8_t ringId)`

Gets the size of the read frame for specified ring.

This function gets a received frame size from the ENET buffer descriptors.

---

**Note:** The FCS of the frame is automatically removed by MAC and the size is the length without the FCS. After calling `ENET_GetRxFramSize`, `ENET_ReadFrame()` should be called to receive frame and update the BD if the result is not “`kStatus_ENET_RxFrameEmpty`”.

---

#### Parameters

- `handle` – The ENET handler structure. This is the same handler pointer used in the `ENET_Init`.
- `length` – The length of the valid frame received.
- `ringId` – The ring index or ring number.

#### Return values

- `kStatus_ENET_RxFrameEmpty` – No frame received. Should not call `ENET_ReadFrame` to read frame.
- `kStatus_ENET_RxFrameError` – Data error happens. `ENET_ReadFrame` should be called with NULL data and NULL length to update the receive buffers.

- `kStatus_Success` – Receive a frame Successfully then the `ENET_ReadFrame` should be called with the right data buffer and the captured data length input.

`status_t` `ENET_ReadFrame`(`ENET_Type` \*base, `enet_handle_t` \*handle, `uint8_t` \*data, `uint32_t` length, `uint8_t` ringId, `uint32_t` \*ts)

Reads a frame from the ENET device. This function reads a frame (both the data and the length) from the ENET buffer descriptors. User can get timestamp through ts pointer if the ts is not NULL.

---

**Note:** It doesn't store the timestamp in the receive timestamp queue. The `ENET_GetRxFramSize` should be used to get the size of the prepared data buffer. This API uses `memcpy` to copy data from DMA buffer to application buffer, 4 bytes aligned data buffer in 32 bits platforms provided by user may let compiler use optimization instruction to reduce time consumption. This is an example:

```
uint32_t length;
enet_handle_t g_handle;
Comments: Get the received frame size firstly.
status = ENET_GetRxFramSize(&g_handle, &length, 0);
if (length != 0)
{
    Comments: Allocate memory here with the size of "length"
    uint8_t *data = memory allocate interface;
    if (!data)
    {
        ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0, NULL);
        Comments: Add the console warning log.
    }
    else
    {
        status = ENET_ReadFrame(ENET, &g_handle, data, length, 0, NULL);
        Comments: Call stack input API to deliver the data to stack
    }
}
else if (status == kStatus_ENET_RxFramError)
{
    Comments: Update the received buffer when a error frame is received.
    ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0, NULL);
}
```

---

### Parameters

- base – ENET peripheral base address.
- handle – The ENET handler structure. This is the same handler pointer used in the `ENET_Init`.
- data – The data buffer provided by user to store the frame which memory size should be at least "length".
- length – The size of the data buffer which is still the length of the received frame.
- ringId – The ring index or ring number.
- ts – The timestamp address to store received timestamp.

### Returns

The execute status, successful or failure.

`status_t` `ENET_SendFrame`(`ENET_Type` \*base, `enet_handle_t` \*handle, const `uint8_t` \*data, `uint32_t` length, `uint8_t` ringId, bool tsFlag, void \*context)

Transmits an ENET frame for specified ring.

---

**Note:** The CRC is automatically appended to the data. Input the data to send without the CRC. This API uses memcpy to copy data from DMA buffer to application buffer, 4 bytes aligned data buffer in 32 bits platforms provided by user may let compiler use optimization instruction to reduce time consumption.

---

#### Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer. This is the same handler pointer used in the ENET\_Init.
- data – The data buffer provided by user to send.
- length – The length of the data to send.
- ringId – The ring index or ring number.
- tsFlag – Timestamp enable flag.
- context – Used by user to handle some events after transmit over.

#### Return values

- kStatus\_Success – Send frame succeed.
- kStatus\_ENET\_TxFrameBusy – Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity. The waiting mechanism is recommended to be added after each call return with kStatus\_ENET\_TxFrameBusy.

*status\_t* ENET\_SetTxReclaim(*enet\_handle\_t* \*handle, bool isEnabled, uint8\_t ringId)

Enable or disable tx descriptors reclaim mechanism.

---

**Note:** This function must be called when no pending send frame action. Set enable if you want to reclaim context or timestamp in interrupt.

---

#### Parameters

- handle – The ENET handler pointer. This is the same handler pointer used in the ENET\_Init.
- isEnabled – Enable or disable flag.
- ringId – The ring index or ring number.

#### Return values

- kStatus\_Success – Succeed to enable/disable Tx reclaim.
- kStatus\_Fail – Fail to enable/disable Tx reclaim.

void ENET\_ReclaimTxDescriptor(ENET\_Type \*base, *enet\_handle\_t* \*handle, uint8\_t ringId)

Reclaim tx descriptors. This function is used to update the tx descriptor status and store the tx timestamp when the 1588 feature is enabled. This is called by the transmit interrupt IRQ handler after the complete of a frame transmission.

#### Parameters

- base – ENET peripheral base address.

- `handle` – The ENET handler pointer. This is the same handler pointer used in the `ENET_Init`.
- `ringId` – The ring index or ring number.

`status_t` `ENET_GetRxFrame(ENET_Type *base, enet_handle_t *handle, enet_rx_frame_struct_t *rxFrame, uint8_t ringId)`

Receives one frame in specified BD ring with zero copy.

This function uses the user-defined allocation and free callbacks. Every time application gets one frame through this function, driver stores the buffer address(es) in `enet_buffer_struct_t` and allocate new buffer(s) for the BD(s). If there's no memory buffer in the pool, this function drops current one frame to keep the Rx frame in BD ring is as fresh as possible.

---

**Note:** Application must provide a memory pool including at least BD number + n buffers in order for this function to work properly, because each BD must always take one buffer while driver is running, then other extra n buffer(s) can be taken by application. Here n is the `ceil(max_frame_length(set by RCR) / bd_rx_size(set by MRBR))`. Application must also provide an array structure in `rxFrame->rxBuffArray` with n index to receive one complete frame in any case.

---

#### Parameters

- `base` – ENET peripheral base address.
- `handle` – The ENET handler pointer. This is the same handler pointer used in the `ENET_Init`.
- `rxFrame` – The received frame information structure provided by user.
- `ringId` – The ring index or ring number.

#### Return values

- `kStatus_Success` – Succeed to get one frame and allocate new memory for Rx buffer.
- `kStatus_ENET_RxFrameEmpty` – There's no Rx frame in the BD.
- `kStatus_ENET_RxFrameError` – There's issue in this receiving.
- `kStatus_ENET_RxFrameDrop` – There's no new buffer memory for BD, drop this frame.

`status_t` `ENET_StartTxFrame(ENET_Type *base, enet_handle_t *handle, enet_tx_frame_struct_t *txFrame, uint8_t ringId)`

Sends one frame in specified BD ring with zero copy.

This function supports scattered buffer transmit, user needs to provide the buffer array.

---

**Note:** Tx reclaim should be enabled to ensure the Tx buffer ownership can be given back to application after Tx is over.

---

#### Parameters

- `base` – ENET peripheral base address.
- `handle` – The ENET handler pointer. This is the same handler pointer used in the `ENET_Init`.
- `txFrame` – The Tx frame structure.

- ringId – The ring index or ring number.

**Return values**

- kStatus\_Success – Succeed to send one frame.
- kStatus\_ENET\_TxFrameBusy – The BD is not ready for Tx or the reclaim operation still not finishes.
- kStatus\_ENET\_TxFrameOverLen – The Tx frame length is over max ethernet frame length.

void ENET\_TransmitIRQHandler(ENET\_Type \*base, *enet\_handle\_t* \*handle)

The transmit IRQ handler.

**Parameters**

- base – ENET peripheral base address.
- handle – The ENET handler pointer.

void ENET\_ReceiveIRQHandler(ENET\_Type \*base, *enet\_handle\_t* \*handle)

The receive IRQ handler.

**Parameters**

- base – ENET peripheral base address.
- handle – The ENET handler pointer.

void ENET\_ErrorIRQHandler(ENET\_Type \*base, *enet\_handle\_t* \*handle)

Some special IRQ handler including the error, mii, wakeup irq handler.

**Parameters**

- base – ENET peripheral base address.
- handle – The ENET handler pointer.

void ENET\_Ptp1588IRQHandler(ENET\_Type \*base)

the common IRQ handler for the 1588 irq handler.

This is used for the 1588 timer interrupt.

**Parameters**

- base – ENET peripheral base address.

void ENET\_CommonFrame0IRQHandler(ENET\_Type \*base)

the common IRQ handler for the tx/rx/error etc irq handler.

This is used for the combined tx/rx/error interrupt for single/multi-ring (frame 0).

**Parameters**

- base – ENET peripheral base address.

FSL\_ENET\_DRIVER\_VERSION

Defines the driver version.

ENET\_BUFFDESCRIPTOR\_RX\_EMPTY\_MASK

Empty bit mask.

ENET\_BUFFDESCRIPTOR\_RX\_SOFTOWNER1\_MASK

Software owner one mask.

ENET\_BUFFDESCRIPTOR\_RX\_WRAP\_MASK

Next buffer descriptor is the start address.

ENET\_BUFFDESCRIPTOR\_RX\_SOFTOWNER2\_Mask

Software owner two mask.

ENET\_BUFFDESCRIPTOR\_RX\_LAST\_MASK

Last BD of the frame mask.

ENET\_BUFFDESCRIPTOR\_RX\_MISS\_MASK

Received because of the promiscuous mode.

ENET\_BUFFDESCRIPTOR\_RX\_BROADCAST\_MASK

Broadcast packet mask.

ENET\_BUFFDESCRIPTOR\_RX\_MULTICAST\_MASK

Multicast packet mask.

ENET\_BUFFDESCRIPTOR\_RX\_LENVIOLATE\_MASK

Length violation mask.

ENET\_BUFFDESCRIPTOR\_RX\_NOOCTET\_MASK

Non-octet aligned frame mask.

ENET\_BUFFDESCRIPTOR\_RX\_CRC\_MASK

CRC error mask.

ENET\_BUFFDESCRIPTOR\_RX\_OVERRUN\_MASK

FIFO overrun mask.

ENET\_BUFFDESCRIPTOR\_RX\_TRUNC\_MASK

Frame is truncated mask.

ENET\_BUFFDESCRIPTOR\_TX\_READY\_MASK

Ready bit mask.

ENET\_BUFFDESCRIPTOR\_TX\_SOFTOWNER1\_MASK

Software owner one mask.

ENET\_BUFFDESCRIPTOR\_TX\_WRAP\_MASK

Wrap buffer descriptor mask.

ENET\_BUFFDESCRIPTOR\_TX\_SOFTOWNER2\_MASK

Software owner two mask.

ENET\_BUFFDESCRIPTOR\_TX\_LAST\_MASK

Last BD of the frame mask.

ENET\_BUFFDESCRIPTOR\_TX\_TRANMITCRC\_MASK

Transmit CRC mask.

ENET\_FRAME\_MAX\_FRAMELEN

Default maximum Ethernet frame size without VLAN tag.

ENET\_FRAME\_VLAN\_TAGLEN

Ethernet single VLAN tag size.

ENET\_FRAME\_CRC\_LEN

CRC size in a frame.

ENET\_FRAME\_TX\_LEN\_LIMITATION(x)

ENET\_FIFO\_MIN\_RX\_FULL

ENET minimum receive FIFO full.

ENET\_RX\_MIN\_BUFFERSIZE

ENET minimum buffer size.

ENET\_PHY\_MAXADDRESS

Maximum PHY address.

ENET\_TX\_INTERRUPT

Enet Tx interrupt flag.

ENET\_RX\_INTERRUPT

Enet Rx interrupt flag.

ENET\_TS\_INTERRUPT

Enet timestamp interrupt flag.

ENET\_ERR\_INTERRUPT

Enet error interrupt flag.

Defines the status return codes for transaction.

*Values:*

enumerator kStatus\_ENET\_InitMemoryFail

Init fails since buffer memory is not enough.

enumerator kStatus\_ENET\_RxFrameError

A frame received but data error happen.

enumerator kStatus\_ENET\_RxFrameFail

Failed to receive a frame.

enumerator kStatus\_ENET\_RxFrameEmpty

No frame arrive.

enumerator kStatus\_ENET\_RxFrameDrop

Rx frame is dropped since no buffer memory.

enumerator kStatus\_ENET\_TxFrameOverLen

Tx frame over length.

enumerator kStatus\_ENET\_TxFrameBusy

Tx buffer descriptors are under process.

enumerator kStatus\_ENET\_TxFrameFail

Transmit frame fail.

enum \_enet\_mii\_mode

Defines the MII/RMII/RGMII mode for data interface between the MAC and the PHY.

*Values:*

enumerator kENET\_MiiMode

MII mode for data interface.

enumerator kENET\_RmiiMode

RMII mode for data interface.

enumerator kENET\_RgmiiMode

RGMII mode for data interface.

enum \_enet\_mii\_speed

Defines the 10/100/1000 Mbps speed for the MII data interface.

Notice: “kENET\_MiiSpeed1000M” only supported when mii mode is “kENET\_RgmiiMode”.

*Values:*

enumerator kENET\_MiiSpeed10M

Speed 10 Mbps.

enumerator kENET\_MiiSpeed100M

Speed 100 Mbps.

enumerator kENET\_MiiSpeed1000M

Speed 1000M bps.

enum \_enet\_mii\_duplex

Defines the half or full duplex for the MII data interface.

*Values:*

enumerator kENET\_MiiHalfDuplex

Half duplex mode.

enumerator kENET\_MiiFullDuplex

Full duplex mode.

enum \_enet\_mii\_write

Define the MII opcode for normal MDIO\_CLAUSES\_22 Frame.

*Values:*

enumerator kENET\_MiiWriteNoCompliant

Write frame operation, but not MII-compliant.

enumerator kENET\_MiiWriteValidFrame

Write frame operation for a valid MII management frame.

enum \_enet\_mii\_read

Defines the read operation for the MII management frame.

*Values:*

enumerator kENET\_MiiReadValidFrame

Read frame operation for a valid MII management frame.

enumerator kENET\_MiiReadNoCompliant

Read frame operation, but not MII-compliant.

enum \_enet\_mii\_extend\_opcode

Define the MII opcode for extended MDIO\_CLAUSES\_45 Frame.

*Values:*

enumerator kENET\_MiiAddrWrite\_C45

Address Write operation.

enumerator kENET\_MiiWriteFrame\_C45

Write frame operation for a valid MII management frame.

enumerator kENET\_MiiReadFrame\_C45

Read frame operation for a valid MII management frame.



enum \_enet\_special\_control\_flag

Defines a special configuration for ENET MAC controller.

These control flags are provided for special user requirements. Normally, these control flags are unused for ENET initialization. For special requirements, set the flags to mac-SpecialConfig in the enet\_config\_t. The kENET\_ControlStoreAndFwdDisable is used to disable the FIFO store and forward. FIFO store and forward means that the FIFO read/send is started when a complete frame is stored in TX/RX FIFO. If this flag is set, configure rxFifo-FullThreshold and txFifoWatermark in the enet\_config\_t.

*Values:*

enumerator kENET\_ControlFlowControlEnable

Enable ENET flow control: pause frame.

enumerator kENET\_ControlRxPayloadCheckEnable

Enable ENET receive payload length check.

enumerator kENET\_ControlRxPadRemoveEnable

Padding is removed from received frames.

enumerator kENET\_ControlRxBroadCastRejectEnable

Enable broadcast frame reject.

enumerator kENET\_ControlMacAddrInsert

Enable MAC address insert.

enumerator kENET\_ControlStoreAndFwdDisable

Enable FIFO store and forward.

enumerator kENET\_ControlSMIPreambleDisable

Enable SMI preamble.

enumerator kENET\_ControlPromiscuousEnable

Enable promiscuous mode.

enumerator kENET\_ControlMIILoopEnable

Enable ENET MII loop back.

enumerator kENET\_ControlVLANTagEnable

Enable normal VLAN (single vlan tag).

enumerator kENET\_ControlSVLANEnable

Enable S-VLAN.

enumerator kENET\_ControlVLANUseSecondTag

Enable extracting the second vlan tag for further processing.

enum \_enet\_interrupt\_enable

List of interrupts supported by the peripheral. This enumeration uses one-bit encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

*Values:*

enumerator kENET\_BabrInterrupt

Babbling receive error interrupt source

enumerator kENET\_BabtInterrupt

Babbling transmit error interrupt source

enumerator kENET\_GraceStopInterrupt

Graceful stop complete interrupt source

enumerator kENET\_TxFrameInterrupt  
TX FRAME interrupt source

enumerator kENET\_TxBufferInterrupt  
TX BUFFER interrupt source

enumerator kENET\_RxFrameInterrupt  
RX FRAME interrupt source

enumerator kENET\_RxBufferInterrupt  
RX BUFFER interrupt source

enumerator kENET\_MiiInterrupt  
MII interrupt source

enumerator kENET\_EBusERInterrupt  
Ethernet bus error interrupt source

enumerator kENET\_LateCollisionInterrupt  
Late collision interrupt source

enumerator kENET\_RetryLimitInterrupt  
Collision Retry Limit interrupt source

enumerator kENET\_UnderrunInterrupt  
Transmit FIFO underrun interrupt source

enumerator kENET\_PayloadRxInterrupt  
Payload Receive error interrupt source

enumerator kENET\_WakeupInterrupt  
WAKEUP interrupt source

enumerator kENET\_TsAvailInterrupt  
TS AVAIL interrupt source for PTP

enumerator kENET\_TsTimerInterrupt  
TS WRAP interrupt source for PTP

enum \_enet\_event

Defines the common interrupt event for callback use.

*Values:*

enumerator kENET\_RxEvent  
Receive event.

enumerator kENET\_TxEvent  
Transmit event.

enumerator kENET\_ErrEvent  
Error event: BABR/BABT/EBERR/LC/RL/UN/PLR .

enumerator kENET\_WakeUpEvent  
Wake up from sleep mode event.

enumerator kENET\_TimeStampEvent  
Time stamp event.

enumerator kENET\_TimeStampAvailEvent  
Time stamp available event.

enum \_enet\_idle\_slope

Defines certain idle slope for bandwidth fraction.

*Values:*

enumerator kENET\_IdleSlope1

The bandwidth fraction is about 0.002.

enumerator kENET\_IdleSlope2

The bandwidth fraction is about 0.003.

enumerator kENET\_IdleSlope4

The bandwidth fraction is about 0.008.

enumerator kENET\_IdleSlope8

The bandwidth fraction is about 0.02.

enumerator kENET\_IdleSlope16

The bandwidth fraction is about 0.03.

enumerator kENET\_IdleSlope32

The bandwidth fraction is about 0.06.

enumerator kENET\_IdleSlope64

The bandwidth fraction is about 0.11.

enumerator kENET\_IdleSlope128

The bandwidth fraction is about 0.20.

enumerator kENET\_IdleSlope256

The bandwidth fraction is about 0.33.

enumerator kENET\_IdleSlope384

The bandwidth fraction is about 0.43.

enumerator kENET\_IdleSlope512

The bandwidth fraction is about 0.50.

enumerator kENET\_IdleSlope640

The bandwidth fraction is about 0.56.

enumerator kENET\_IdleSlope768

The bandwidth fraction is about 0.60.

enumerator kENET\_IdleSlope896

The bandwidth fraction is about 0.64.

enumerator kENET\_IdleSlope1024

The bandwidth fraction is about 0.67.

enumerator kENET\_IdleSlope1152

The bandwidth fraction is about 0.69.

enumerator kENET\_IdleSlope1280

The bandwidth fraction is about 0.71.

enumerator kENET\_IdleSlope1408

The bandwidth fraction is about 0.73.

enumerator kENET\_IdleSlope1536

The bandwidth fraction is about 0.75.

enum `_enet_tx_accelerator`

Defines the transmit accelerator configuration.

Note that the hardware does not insert ICMPv6 protocol checksums as mentioned in errata ERR052152.

*Values:*

enumerator `kENET_TxAccelIsShift16Enabled`

Transmit FIFO shift-16.

enumerator `kENET_TxAccelIpCheckEnabled`

Insert IP header checksum.

enumerator `kENET_TxAccelProtoCheckEnabled`

Insert protocol checksum (TCP, UDP, ICMPv4).

enum `_enet_rx_accelerator`

Defines the receive accelerator configuration.

Note that the hardware does not validate ICMPv6 protocol checksums as mentioned in errata ERR052152.

*Values:*

enumerator `kENET_RxAccelPadRemoveEnabled`

Padding removal for short IP frames.

enumerator `kENET_RxAccelIpCheckEnabled`

Discard with wrong IP header checksum.

enumerator `kENET_RxAccelProtoCheckEnabled`

Discard with wrong protocol checksum (TCP, UDP, ICMPv4).

enumerator `kENET_RxAccelMacCheckEnabled`

Discard with Mac layer errors.

enumerator `kENET_RxAccelIsShift16Enabled`

Receive FIFO shift-16.

typedef enum `_enet_mii_mode` `enet_mii_mode_t`

Defines the MII/RMII/RGMII mode for data interface between the MAC and the PHY.

typedef enum `_enet_mii_speed` `enet_mii_speed_t`

Defines the 10/100/1000 Mbps speed for the MII data interface.

Notice: “kENET\_MiiSpeed1000M” only supported when mii mode is “kENET\_RgmiiMode”.

typedef enum `_enet_mii_duplex` `enet_mii_duplex_t`

Defines the half or full duplex for the MII data interface.

typedef enum `_enet_mii_write` `enet_mii_write_t`

Define the MII opcode for normal MDIO\_CLAUSES\_22 Frame.

typedef enum `_enet_mii_read` `enet_mii_read_t`

Defines the read operation for the MII management frame.

typedef enum `_enet_mii_extend_opcode` `enet_mii_extend_opcode`

Define the MII opcode for extended MDIO\_CLAUSES\_45 Frame.

typedef enum *\_enet\_special\_control\_flag* enet\_special\_control\_flag\_t

Defines a special configuration for ENET MAC controller.

These control flags are provided for special user requirements. Normally, these control flags are unused for ENET initialization. For special requirements, set the flags to mac-SpecialConfig in the enet\_config\_t. The kENET\_ControlStoreAndFwdDisable is used to disable the FIFO store and forward. FIFO store and forward means that the FIFO read/send is started when a complete frame is stored in TX/RX FIFO. If this flag is set, configure rxFifo-FullThreshold and txFifoWatermark in the enet\_config\_t.

typedef enum *\_enet\_interrupt\_enable* enet\_interrupt\_enable\_t

List of interrupts supported by the peripheral. This enumeration uses one-bit encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

typedef enum *\_enet\_event* enet\_event\_t

Defines the common interrupt event for callback use.

typedef enum *\_enet\_idle\_slope* enet\_idle\_slope\_t

Defines certain idle slope for bandwidth fraction.

typedef enum *\_enet\_tx\_accelerator* enet\_tx\_accelerator\_t

Defines the transmit accelerator configuration.

Note that the hardware does not insert ICMPv6 protocol checksums as mentioned in errata ERR052152.

typedef enum *\_enet\_rx\_accelerator* enet\_rx\_accelerator\_t

Defines the receive accelerator configuration.

Note that the hardware does not validate ICMPv6 protocol checksums as mentioned in errata ERR052152.

typedef struct *\_enet\_rx\_bd\_struct* enet\_rx\_bd\_struct\_t

Defines the receive buffer descriptor structure for the little endian system.

typedef struct *\_enet\_tx\_bd\_struct* enet\_tx\_bd\_struct\_t

Defines the enhanced transmit buffer descriptor structure for the little endian system.

typedef struct *\_enet\_data\_error\_stats* enet\_data\_error\_stats\_t

Defines the ENET data error statistics structure.

typedef struct *\_enet\_rx\_frame\_error* enet\_rx\_frame\_error\_t

Defines the Rx frame error structure.

typedef struct *\_enet\_transfer\_stats* enet\_transfer\_stats\_t

Defines the ENET transfer statistics structure.

typedef struct *enet\_frame\_info* enet\_frame\_info\_t

Defines the frame info structure.

typedef struct *\_enet\_tx\_dirty\_ring* enet\_tx\_dirty\_ring\_t

Defines the ENET transmit dirty addresses ring/queue structure.

typedef void (\*enet\_rx\_alloc\_callback\_t)(ENET\_Type \*base, void \*userData, uint8\_t ringId)

Defines the ENET Rx memory buffer alloc function pointer.

typedef void (\*enet\_rx\_free\_callback\_t)(ENET\_Type \*base, void \*buffer, void \*userData, uint8\_t ringId)

Defines the ENET Rx memory buffer free function pointer.

```
typedef struct _enet_buffer_config enet_buffer_config_t
```

Defines the receive buffer descriptor configuration structure.

Note that for the internal DMA requirements, the buffers have a corresponding alignment requirements.

- a. The aligned receive and transmit buffer size must be evenly divisible by ENET\_BUFF\_ALIGNMENT. when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of “ENET\_BUFF\_ALIGNMENT” and the cache line size.
- b. The aligned transmit and receive buffer descriptor start address must be at least 64 bit aligned. However, it's recommended to be evenly divisible by ENET\_BUFF\_ALIGNMENT. buffer descriptors should be put in non-cacheable region when cache is enabled.
- c. The aligned transmit and receive data buffer start address must be evenly divisible by ENET\_BUFF\_ALIGNMENT. Receive buffers should be continuous with the total size equal to “rxBdNumber \* rxBuffSizeAlign”. Transmit buffers should be continuous with the total size equal to “txBdNumber \* txBuffSizeAlign”. when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of “ENET\_BUFF\_ALIGNMENT” and the cache line size.

```
typedef struct _enet_intcoalesce_config enet_intcoalesce_config_t
```

Defines the interrupt coalescing configure structure.

```
typedef struct _enet_avb_config enet_avb_config_t
```

Defines the ENET AVB Configure structure.

This is used for to configure the extended ring 1 and ring 2.

- a. The classification match format is  $(CMP3 \ll 12) \mid (CMP2 \ll 8) \mid (CMP1 \ll 4) \mid CMP0$ . composed of four 3-bit compared VLAN priority field cmp0~cmp3, cm0 ~ cmp3 are used in parallel.

If CMP1,2,3 are not unused, please set them to the same value as CMP0.

- a. The idleSlope is used to calculate the Band Width fraction, BW fraction =  $1 / (1 + 512/idleSlope)$ . For avb configuration, the BW fraction of Class 1 and Class 2 combined must not exceed 0.75.

```
typedef struct _enet_handle enet_handle_t
```

```
typedef void (*enet_callback_t)(ENET_Type *base, enet_handle_t *handle, enet_event_t event, enet_frame_info_t *frameInfo, void *userData)
```

ENET callback function.

```
typedef struct _enet_config enet_config_t
```

Defines the basic configuration structure for the ENET device.

Note:

- a. macSpecialConfig is used for a special control configuration, A logical OR of “enet\_special\_control\_flag\_t”. For a special configuration for MAC, set this parameter to 0.
- b. txWatermark is used for a cut-through operation. It is in steps of 64 bytes: 0/1 - 64 bytes written to TX FIFO before transmission of a frame begins. 2 - 128 bytes written to TX FIFO .... 3 - 192 bytes written to TX FIFO .... The maximum of txWatermark is 0x2F - 4032 bytes written to TX FIFO .... txWatermark allows minimizing the transmit latency to set the txWatermark to 0 or 1 or for larger bus access latency 3 or larger due to contention for the system bus.
- c. rxFifoFullThreshold is similar to the txWatermark for cut-through operation in RX. It is in 64-bit words. The minimum is ENET\_FIFO\_MIN\_RX\_FULL and the maximum is

0xFF. If the end of the frame is stored in FIFO and the frame size is smaller than the txWatermark, the frame is still transmitted. The rule is the same for rxFifoFullThreshold in the receive direction.

- d. When “kENET\_ControlFlowControlEnable” is set in the macSpecialConfig, ensure that the pauseDuration, rxFifoEmptyThreshold, and rxFifoStatEmptyThreshold are set for flow control enabled case.
- e. When “kENET\_ControlStoreAndFwdDisabled” is set in the macSpecialConfig, ensure that the rxFifoFullThreshold and txFifoWatermark are set for store and forward disable.
- f. The rxAccelerConfig and txAccelerConfig default setting with 0 - accelerator are disabled. The “enet\_tx\_accelerator\_t” and “enet\_rx\_accelerator\_t” are recommended to be used to enable the transmit and receive accelerator. After the accelerators are enabled, the store and forward feature should be enabled. As a result, kENET\_ControlStoreAndFwdDisabled should not be set.
- g. The intCoalesceCfg can be used in the rx or tx enabled cases to decrease the CPU loading.

```
typedef struct _enet_tx_bd_ring enet_tx_bd_ring_t
```

Defines the ENET transmit buffer descriptor ring/queue structure.

```
typedef struct _enet_rx_bd_ring enet_rx_bd_ring_t
```

Defines the ENET receive buffer descriptor ring/queue structure.

```
typedef struct _enet_buffer_struct enet_buffer_struct_t
```

```
typedef struct _enet_rx_frame_attribute_struct enet_rx_frame_attribute_t
```

```
typedef struct _enet_rx_frame_struct enet_rx_frame_struct_t
```

```
typedef struct _enet_tx_frame_struct enet_tx_frame_struct_t
```

```
typedef void (*enet_isr_t)(ENET_Type *base, enet_handle_t *handle)
```

Define interrupt IRQ handler.

```
const clock_ip_name_t s_enetClock[]
```

Pointers to enet clocks for each instance.

```
const clock_ip_name_t s_enetExtraClock[]
```

```
uint32_t ENET_GetInstance(ENET_Type *base)
```

Get the ENET instance from peripheral base address.

### Parameters

- base – ENET peripheral base address.

### Returns

ENET instance.

```
ENET_BUFFDESCRIPTOR_RX_ERR_MASK
```

Defines the receive error status flag mask.

```
struct _enet_rx_bd_struct
```

#include <fsl\_enet.h> Defines the receive buffer descriptor structure for the little endian system.

### Public Members

```
uint16_t length
```

Buffer descriptor data length.

uint16\_t control

Buffer descriptor control and status.

uint32\_t buffer

Data buffer pointer.

struct \_\_enet\_tx\_bd\_struct

*#include <fsl\_enet.h>* Defines the enhanced transmit buffer descriptor structure for the little endian system.

### Public Members

uint16\_t length

Buffer descriptor data length.

uint16\_t control

Buffer descriptor control and status.

uint32\_t buffer

Data buffer pointer.

struct \_\_enet\_data\_error\_stats

*#include <fsl\_enet.h>* Defines the ENET data error statistics structure.

### Public Members

uint32\_t statsRxLenGreaterErr

Receive length greater than RCR[MAX\_FL].

uint32\_t statsRxAlignErr

Receive non-octet alignment/

uint32\_t statsRxFcsErr

Receive CRC error.

uint32\_t statsRxOverRunErr

Receive over run.

uint32\_t statsRxTruncateErr

Receive truncate.

struct \_\_enet\_rx\_frame\_error

*#include <fsl\_enet.h>* Defines the Rx frame error structure.

### Public Members

bool statsRxTruncateErr

Receive truncate.

bool statsRxOverRunErr

Receive over run.

bool statsRxFcsErr

Receive CRC error.

bool statsRxAlignErr

Receive non-octet alignment.



bool statsRxLenGreaterErr  
Receive length greater than RCR[MAX\_FL].

struct \_\_enet\_\_transfer\_stats  
*#include <fsl\_enet.h>* Defines the ENET transfer statistics structure.

### Public Members

uint32\_t statsRxFrameCount  
Rx frame number.

uint32\_t statsRxFrameOk  
Good Rx frame number.

uint32\_t statsRxCrcErr  
Rx frame number with CRC error.

uint32\_t statsRxAlignErr  
Rx frame number with alignment error.

uint32\_t statsRxDropInvalidSFD  
Dropped frame number due to invalid SFD.

uint32\_t statsRxFifoOverflowErr  
Rx FIFO overflow count.

uint32\_t statsTxFrameCount  
Tx frame number.

uint32\_t statsTxFrameOk  
Good Tx frame number.

uint32\_t statsTxCrcAlignErr  
The transmit frame is error.

uint32\_t statsTxFifoUnderRunErr  
Tx FIFO underrun count.

struct enet\_\_frame\_\_info  
*#include <fsl\_enet.h>* Defines the frame info structure.

### Public Members

void \*context  
User specified data

struct \_\_enet\_\_tx\_\_dirty\_\_ring  
*#include <fsl\_enet.h>* Defines the ENET transmit dirty addresses ring/queue structure.

### Public Members

enet\_frame\_info\_t \*txDirtyBase  
Dirty buffer descriptor base address pointer.

uint16\_t txGenIdx  
tx generate index.

uint16\_t txConsumeIdx  
tx consume index.

uint16\_t txRingLen

tx ring length.

bool isFull

tx ring is full flag.

struct \_\_enet\_\_buffer\_\_config

#include <fsl\_enet.h> Defines the receive buffer descriptor configuration structure.

Note that for the internal DMA requirements, the buffers have a corresponding alignment requirements.

- The aligned receive and transmit buffer size must be evenly divisible by ENET\_BUFF\_ALIGNMENT. when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET\_BUFF\_ALIGNMENT" and the cache line size.
- The aligned transmit and receive buffer descriptor start address must be at least 64 bit aligned. However, it's recommended to be evenly divisible by ENET\_BUFF\_ALIGNMENT. buffer descriptors should be put in non-cacheable region when cache is enabled.
- The aligned transmit and receive data buffer start address must be evenly divisible by ENET\_BUFF\_ALIGNMENT. Receive buffers should be continuous with the total size equal to "rxBdNumber \* rxBuffSizeAlign". Transmit buffers should be continuous with the total size equal to "txBdNumber \* txBuffSizeAlign". when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET\_BUFF\_ALIGNMENT" and the cache line size.

## Public Members

uint16\_t rxBdNumber

Receive buffer descriptor number.

uint16\_t txBdNumber

Transmit buffer descriptor number.

uint16\_t rxBuffSizeAlign

Aligned receive data buffer size.

uint16\_t txBuffSizeAlign

Aligned transmit data buffer size.

volatile *enet\_rx\_bd\_struct\_t* \*rxBdStartAddrAlign

Aligned receive buffer descriptor start address: should be non-cacheable.

volatile *enet\_tx\_bd\_struct\_t* \*txBdStartAddrAlign

Aligned transmit buffer descriptor start address: should be non-cacheable.

uint8\_t \*rxBufferAlign

Receive data buffer start address.

uint8\_t \*txBufferAlign

Transmit data buffer start address.

bool rxMaintainEnable

Receive buffer cache maintain.

bool txMaintainEnable

Transmit buffer cache maintain.

*enet\_frame\_info\_t* \*txFrameInfo

Transmit frame information start address.

struct \_\_enet\_intcoalesce\_config

#include <fsl\_enet.h> Defines the interrupt coalescing configure structure.

### Public Members

uint8\_t txCoalesceFrameCount[1]

Transmit interrupt coalescing frame count threshold.

uint16\_t txCoalesceTimeCount[1]

Transmit interrupt coalescing timer count threshold.

uint8\_t rxCoalesceFrameCount[1]

Receive interrupt coalescing frame count threshold.

uint16\_t rxCoalesceTimeCount[1]

Receive interrupt coalescing timer count threshold.

struct \_\_enet\_avb\_config

#include <fsl\_enet.h> Defines the ENET AVB Configure structure.

This is used for to configure the extended ring 1 and ring 2.

- a. The classification match format is  $(CMP3 \ll 12) \mid (CMP2 \ll 8) \mid (CMP1 \ll 4) \mid CMP0$ . composed of four 3-bit compared VLAN priority field cmp0~cmp3, cm0 ~ cmp3 are used in parallel.

If CMP1,2,3 are not unused, please set them to the same value as CMP0.

- a. The idleSlope is used to calculate the Band Width fraction,  $BW \text{ fraction} = 1 / (1 + 512/idleSlope)$ . For avb configuration, the BW fraction of Class 1 and Class 2 combined must not exceed 0.75.

### Public Members

uint16\_t rxClassifyMatch[1 - 1]

The classification match value for the ring.

*enet\_idle\_slope\_t* idleSlope[1 - 1]

The idle slope for certian bandwidth fraction.

struct \_\_enet\_config

#include <fsl\_enet.h> Defines the basic configuration structure for the ENET device.

Note:

- a. macSpecialConfig is used for a special control configuration, A logical OR of "enet\_special\_control\_flag\_t". For a special configuration for MAC, set this parameter to 0.
- b. txWatermark is used for a cut-through operation. It is in steps of 64 bytes: 0/1 - 64 bytes written to TX FIFO before transmission of a frame begins. 2 - 128 bytes written to TX FIFO .... 3 - 192 bytes written to TX FIFO .... The maximum of txWatermark is 0x2F - 4032 bytes written to TX FIFO .... txWatermark allows minimizing the transmit latency to set the txWatermark to 0 or 1 or for larger bus access latency 3 or larger due to contention for the system bus.
- c. rxFifoFullThreshold is similar to the txWatermark for cut-through operation in RX. It is in 64-bit words. The minimum is ENET\_FIFO\_MIN\_RX\_FULLL and the maximum is 0xFF. If the end of the frame is stored in FIFO and the frame size if smaller than the

txWatermark, the frame is still transmitted. The rule is the same for rxFifoFullThreshold in the receive direction.

- d. When “kENET\_ControlFlowControlEnable” is set in the macSpecialConfig, ensure that the pauseDuration, rxFifoEmptyThreshold, and rxFifoStatEmptyThreshold are set for flow control enabled case.
- e. When “kENET\_ControlStoreAndFwdDisabled” is set in the macSpecialConfig, ensure that the rxFifoFullThreshold and txFifoWatermark are set for store and forward disable.
- f. The rxAccelerConfig and txAccelerConfig default setting with 0 - accelerator are disabled. The “enet\_tx\_accelerator\_t” and “enet\_rx\_accelerator\_t” are recommended to be used to enable the transmit and receive accelerator. After the accelerators are enabled, the store and forward feature should be enabled. As a result, kENET\_ControlStoreAndFwdDisabled should not be set.
- g. The intCoalesceCfg can be used in the rx or tx enabled cases to decrease the CPU loading.

## Public Members

uint32\_t macSpecialConfig

Mac special configuration. A logical OR of “enet\_special\_control\_flag\_t”.

uint32\_t interrupt

Mac interrupt source. A logical OR of “enet\_interrupt\_enable\_t”.

uint16\_t rxMaxFrameLen

Receive maximum frame length.

enet\_mii\_mode\_t miiMode

MII mode.

enet\_mii\_speed\_t miiSpeed

MII Speed.

enet\_mii\_duplex\_t miiDuplex

MII duplex.

uint8\_t rxAccelerConfig

Receive accelerator, A logical OR of “enet\_rx\_accelerator\_t”.

uint8\_t txAccelerConfig

Transmit accelerator, A logical OR of “enet\_tx\_accelerator\_t”.

uint16\_t pauseDuration

For flow control enabled case: Pause duration.

uint8\_t rxFifoEmptyThreshold

For flow control enabled case: when RX FIFO level reaches this value, it makes MAC generate XOFF pause frame.

uint8\_t rxFifoStatEmptyThreshold

For flow control enabled case: number of frames in the receive FIFO, independent of size, that can be accept. If the limit is reached, reception continues and a pause frame is triggered.

uint8\_t rxFifoFullThreshold

For store and forward disable case, the data required in RX FIFO to notify the MAC receive ready status.

uint8\_t txFifoWatermark

For store and forward disable case, the data required in TX FIFO before a frame transmit start.

enet\_intcoalesce\_config\_t \*intCoalesceCfg

If the interrupt coalesce is not required in the ring n(0,1,2), please set to NULL.

uint8\_t ringNum

Number of used rings. default with 1 &#8212; single ring.

enet\_rx\_alloc\_callback\_t rxBuffAlloc

Callback function to alloc memory, must be provided for zero-copy Rx.

enet\_rx\_free\_callback\_t rxBuffFree

Callback function to free memory, must be provided for zero-copy Rx.

enet\_callback\_t callback

General callback function.

void \*userData

Callback function parameter.

struct \_\_enet\_tx\_bd\_ring

#include <fsl\_enet.h> Defines the ENET transmit buffer descriptor ring/queue structure.

### Public Members

volatile enet\_tx\_bd\_struct\_t \*txBdBase

Buffer descriptor base address pointer.

uint16\_t txGenIdx

The current available transmit buffer descriptor pointer.

uint16\_t txConsumeIdx

Transmit consume index.

volatile uint16\_t txDescUsed

Transmit descriptor used number.

uint16\_t txRingLen

Transmit ring length.

struct \_\_enet\_rx\_bd\_ring

#include <fsl\_enet.h> Defines the ENET receive buffer descriptor ring/queue structure.

### Public Members

volatile enet\_rx\_bd\_struct\_t \*rxBdBase

Buffer descriptor base address pointer.

uint16\_t rxGenIdx

The current available receive buffer descriptor pointer.

uint16\_t rxRingLen

Receive ring length.

struct \_\_enet\_handle

#include <fsl\_enet.h> Defines the ENET handler structure.

**Public Members**

*enet\_rx\_bd\_ring\_t* rxBdRing[1]  
Receive buffer descriptor.

*enet\_tx\_bd\_ring\_t* txBdRing[1]  
Transmit buffer descriptor.

uint16\_t rxBuffSizeAlign[1]  
Receive buffer size alignment.

uint16\_t txBuffSizeAlign[1]  
Transmit buffer size alignment.

bool rxMaintainEnable[1]  
Receive buffer cache maintain.

bool txMaintainEnable[1]  
Transmit buffer cache maintain.

uint8\_t ringNum  
Number of used rings.

*enet\_callback\_t* callback  
Callback function.

void \*userData  
Callback function parameter.

*enet\_tx\_dirty\_ring\_t* txDirtyRing[1]  
Ring to store tx frame information.

bool txReclaimEnable[1]  
Tx reclaim enable flag.

*enet\_rx\_alloc\_callback\_t* rxBuffAlloc  
Callback function to alloc memory for zero copy Rx.

*enet\_rx\_free\_callback\_t* rxBuffFree  
Callback function to free memory for zero copy Rx.

uint8\_t multicastCount[64]  
Multicast collisions counter

uint32\_t enetClock  
The clock of enet peripheral, to caculate core cycles for PTP timestamp.

uint32\_t tsDelayCount  
The count of core cycles for PTP timestamp capture delay.

struct \_\_enet\_buffer\_struct  
*#include <fsl\_enet.h>*

**Public Members**

void \*buffer  
The buffer store the whole or partial frame.

uint16\_t length  
The byte length of this buffer.

struct \_\_enet\_rx\_frame\_attribute\_struct  
*#include <fsl\_enet.h>*

### Public Members

bool promiscuous

This frame is received because of promiscuous mode.

```
struct __enet_rx_frame_struct
#include <fsl_enet.h>
```

### Public Members

*enet\_buffer\_struct\_t* \*rxBuffArray  
Rx frame buffer structure.

uint16\_t totLen  
Rx frame total length.

*enet\_rx\_frame\_attribute\_t* rxAttribute  
Rx frame attribute structure.

*enet\_rx\_frame\_error\_t* rxFrameError  
Rx frame error.

```
struct __enet_tx_frame_struct
#include <fsl_enet.h>
```

### Public Members

*enet\_buffer\_struct\_t* \*txBuffArray  
Tx frame buffer structure.

uint32\_t txBuffNum  
Buffer number of this Tx frame.

void \*context  
Driver reclaims and gives it in Tx over callback, usually store network packet header.

## 2.5 EQOS-TSN: Ethernet QoS with TSN Driver

### 2.6 Enet\_qos\_qos

void ENET\_QOS\_GetDefaultConfig(*enet\_qos\_config\_t* \*config)  
Gets the ENET default configuration structure.

The purpose of this API is to get the default ENET configure structure for ENET\_QOS\_Init(). User may use the initialized structure unchanged in ENET\_QOS\_Init(), or modify some fields of the structure before calling ENET\_QOS\_Init(). Example:

```
enet_qos_config_t config;
ENET_QOS_GetDefaultConfig(&config);
```

#### Parameters

- config – The ENET mac controller configuration structure pointer.

```
status_t ENET_QOS_Up(ENET_QOS_Type *base, const enet_qos_config_t *config, uint8_t
                    *macAddr, uint8_t macCount, uint32_t refclkSrc_Hz)
```

Initializes the ENET module.

This function initializes it with the ENET basic configuration.

#### Parameters

- base – ENET peripheral base address.
- config – ENET mac configuration structure pointer. The “*enet\_qos\_config\_t*” type mac configuration return from `ENET_QOS_GetDefaultConfig` can be used directly. It is also possible to verify the Mac configuration using other methods.
- macAddr – Pointer to ENET mac address array of Ethernet device. This MAC address should be provided.
- macCount – Count of macAddr in the ENET mac address array
- refclkSrc\_Hz – ENET input reference clock.

```
status_t ENET_QOS_Init(ENET_QOS_Type *base, const enet_qos_config_t *config, uint8_t
                      *macAddr, uint8_t macCount, uint32_t refclkSrc_Hz)
```

Initializes the ENET module.

This function ungates the module clock and initializes it with the ENET basic configuration.

#### Parameters

- base – ENET peripheral base address.
- config – ENET mac configuration structure pointer. The “*enet\_qos\_config\_t*” type mac configuration return from `ENET_QOS_GetDefaultConfig` can be used directly. It is also possible to verify the Mac configuration using other methods.
- macAddr – Pointer to ENET mac address array of Ethernet device. This MAC address should be provided.
- macCount – Count of macAddr in the ENET mac address array
- refclkSrc\_Hz – ENET input reference clock.

```
void ENET_QOS_Down(ENET_QOS_Type *base)
```

Stops the ENET module.

This function disables the ENET module.

#### Parameters

- base – ENET peripheral base address.

```
void ENET_QOS_Deinit(ENET_QOS_Type *base)
```

Deinitializes the ENET module.

This function gates the module clock and disables the ENET module.

#### Parameters

- base – ENET peripheral base address.

```
uint32_t ENET_QOS_GetInstance(ENET_QOS_Type *base)
```

Get the ENET instance from peripheral base address.

#### Parameters

- base – ENET peripheral base address.



**Returns**

ENET instance.

*status\_t* ENET\_QOS\_DescriptorInit(ENET\_QOS\_Type \*base, *enet\_qos\_config\_t* \*config,  
*enet\_qos\_buffer\_config\_t* \*bufferConfig)

Initialize for all ENET descriptors.

---

**Note:** This function is do all tx/rx descriptors initialization. Because this API read all interrupt registers first and then set the interrupt flag for all descriptors, if the interrupt register is set. so the descriptor initialization should be called after ENET\_QOS\_Init(), ENET\_QOS\_EnableInterrupts() and ENET\_QOS\_CreateHandle()(if transactional APIs are used).

---

**Parameters**

- base – ENET peripheral base address.
- config – The configuration for ENET.
- bufferConfig – All buffers configuration.

*status\_t* ENET\_QOS\_RxBufferAllocAll(ENET\_QOS\_Type \*base, *enet\_qos\_handle\_t* \*handle)

Allocates Rx buffers for all BDs. It's used for zero copy Rx. In zero copy Rx case, Rx buffers are dynamic. This function will populate initial buffers in all BDs for receiving. Then ENET\_QOS\_GetRxFrame() is used to get Rx frame with zero copy, it will allocate new buffer to replace the buffer in BD taken by application application should free those buffers after they're used.

---

**Note:** This function should be called after ENET\_QOS\_CreateHandler() and buffer allocating callback function should be ready.

---

**Parameters**

- base – ENET\_QOS peripheral base address.
- handle – The ENET\_QOS handler structure. This is the same handler pointer used in the ENET\_QOS\_Init.

void ENET\_QOS\_RxBufferFreeAll(ENET\_QOS\_Type \*base, *enet\_qos\_handle\_t* \*handle)

Frees Rx buffers in all BDs. It's used for zero copy Rx. In zero copy Rx case, Rx buffers are dynamic. This function will free left buffers in all BDs.

**Parameters**

- base – ENET\_QOS peripheral base address.
- handle – The ENET\_QOS handler structure. This is the same handler pointer used in the ENET\_QOS\_Init.

void ENET\_QOS\_StartRxTx(ENET\_QOS\_Type \*base, uint8\_t txRingNum, uint8\_t rxRingNum)

Starts the ENET rx/tx. This function enable the tx/rx and starts the rx/tx DMA. This shall be set after ENET initialization and before starting to receive the data.

---

**Note:** This must be called after all the ENET initialization. And should be called when the ENET receive/transmit is required.

---

**Parameters**

- base – ENET peripheral base address.

- rxRingNum – The number of the used rx rings. It shall not be larger than the ENET\_QOS\_RING\_NUM\_MAX(2). If the ringNum is set with 1, the ring 0 will be used.
- txRingNum – The number of the used tx rings. It shall not be larger than the ENET\_QOS\_RING\_NUM\_MAX(2). If the ringNum is set with 1, the ring 0 will be used.

*status\_t* ENET\_QOS\_SetMII(ENET\_QOS\_Type \*base, *enet\_qos\_mii\_speed\_t* speed, *enet\_qos\_mii\_duplex\_t* duplex)

Sets the ENET MII speed and duplex.

This API is provided to dynamically change the speed and duplex for MAC.

#### Parameters

- base – ENET peripheral base address.
- speed – The speed of the RMII mode.
- duplex – The duplex of the RMII mode.

#### Returns

kStatus\_Success The ENET MII speed and duplex has been set successfully.

#### Returns

kStatus\_InvalidArgument Could not set the desired ENET MII speed and duplex combination.

*void* ENET\_QOS\_SetSMI(ENET\_QOS\_Type \*base, *uint32\_t* csrClock\_Hz)

Sets the ENET SMI(serial management interface)- MII management interface.

#### Parameters

- base – ENET peripheral base address.
- csrClock\_Hz – CSR clock frequency in HZ

*static inline bool* ENET\_QOS\_IsSMIBusy(ENET\_QOS\_Type \*base)

Checks if the SMI is busy.

#### Parameters

- base – ENET peripheral base address.

#### Returns

The status of MII Busy status.

*static inline uint16\_t* ENET\_QOS\_ReadSMIData(ENET\_QOS\_Type \*base)

Reads data from the PHY register through SMI interface.

#### Parameters

- base – ENET peripheral base address.

#### Returns

The data read from PHY

*void* ENET\_QOS\_StartSMIWrite(ENET\_QOS\_Type \*base, *uint8\_t* phyAddr, *uint8\_t* regAddr, *uint16\_t* data)

Sends the MDIO IEEE802.3 Clause 22 format write command. After send command, user needs to check whether the transmission is over with ENET\_QOS\_IsSMIBusy().

#### Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address.
- regAddr – The PHY register address.

- data – The data written to PHY.

`void ENET_QOS_StartSMIRRead(ENET_QOS_Type *base, uint8_t phyAddr, uint8_t regAddr)`

Sends the MDIO IEEE802.3 Clause 22 format read command. After send command, user needs to check whether the transmission is over with `ENET_QOS_IsSMIBusy()`.

#### Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address.
- regAddr – The PHY register address.

`void ENET_QOS_StartExtC45SMIWrite(ENET_QOS_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr, uint16_t data)`

Sends the MDIO IEEE802.3 Clause 45 format write command. After send command, user needs to check whether the transmission is over with `ENET_QOS_IsSMIBusy()`.

#### Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.
- data – The data written to PHY.

`void ENET_QOS_StartExtC45SMIRRead(ENET_QOS_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr)`

Sends the MDIO IEEE802.3 Clause 45 format read command. After send command, user needs to check whether the transmission is over with `ENET_QOS_IsSMIBusy()`.

#### Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.

`status_t ENET_QOS_MDIOWrite(ENET_QOS_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t data)`

MDIO write with IEEE802.3 MDIO Clause 22 format.

#### Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address.
- regAddr – The PHY register.
- data – The data written to PHY.

#### Returns

`kStatus_Success` MDIO access succeeds.

#### Returns

`kStatus_Timeout` MDIO access timeout.

`status_t ENET_QOS_MDIORRead(ENET_QOS_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t *pData)`

MDIO read with IEEE802.3 MDIO Clause 22 format.

#### Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address.
- regAddr – The PHY register.
- pData – The data read from PHY.

**Returns**

kStatus\_Success MDIO access succeeds.

**Returns**

kStatus\_Timeout MDIO access timeout.

*status\_t* ENET\_QOS\_MDIO\_C45\_Write(ENET\_QOS\_Type \*base, uint8\_t portAddr, uint8\_t devAddr, uint16\_t regAddr, uint16\_t data)

MDIO write with IEEE802.3 Clause 45 format.

**Parameters**

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.
- data – The data written to PHY.

**Returns**

kStatus\_Success MDIO access succeeds.

**Returns**

kStatus\_Timeout MDIO access timeout.

*status\_t* ENET\_QOS\_MDIO\_C45\_Read(ENET\_QOS\_Type \*base, uint8\_t portAddr, uint8\_t devAddr, uint16\_t regAddr, uint16\_t \*pData)

MDIO read with IEEE802.3 Clause 45 format.

**Parameters**

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.
- pData – The data read from PHY.

**Returns**

kStatus\_Success MDIO access succeeds.

**Returns**

kStatus\_Timeout MDIO access timeout.

static inline void ENET\_QOS\_SetMacAddr(ENET\_QOS\_Type \*base, uint8\_t \*macAddr, uint8\_t index)

Sets the ENET module Mac address.

**Parameters**

- base – ENET peripheral base address.
- macAddr – The six-byte Mac address pointer. The pointer is allocated by application and input into the API.
- index – Configure macAddr to MAC\_ADDRESS[index] register.

`void ENET_QOS_GetMacAddr(ENET_QOS_Type *base, uint8_t *macAddr, uint8_t index)`

Gets the ENET module Mac address.

**Parameters**

- `base` – ENET peripheral base address.
- `macAddr` – The six-byte Mac address pointer. The pointer is allocated by application and input into the API.
- `index` – Get `macAddr` from `MAC_ADDRESS[index]` register.

`void ENET_QOS_AddMulticastGroup(ENET_QOS_Type *base, uint8_t *address)`

Adds the ENET\_QOS device to a multicast group.

**Parameters**

- `base` – ENET\_QOS peripheral base address.
- `address` – The six-byte multicast group address which is provided by application.

`void ENET_QOS_LeaveMulticastGroup(ENET_QOS_Type *base, uint8_t *address)`

Moves the ENET\_QOS device from a multicast group.

**Parameters**

- `base` – ENET\_QOS peripheral base address.
- `address` – The six-byte multicast group address which is provided by application.

`static inline void ENET_QOS_AcceptAllMulticast(ENET_QOS_Type *base)`

Enable ENET device to accept all multicast frames.

**Parameters**

- `base` – ENET peripheral base address.

`static inline void ENET_QOS_RejectAllMulticast(ENET_QOS_Type *base)`

ENET device reject to accept all multicast frames.

**Parameters**

- `base` – ENET peripheral base address.

`void ENET_QOS_EnterPowerDown(ENET_QOS_Type *base, uint32_t *wakeFilter)`

Set the MAC to enter into power down mode. the remote power wake up frame and magic frame can wake up the ENET from the power down mode.

**Parameters**

- `base` – ENET peripheral base address.
- `wakeFilter` – The `wakeFilter` provided to configure the wake up frame filter. Set the `wakeFilter` to NULL is not required. But if you have the filter requirement, please make sure the `wakeFilter` pointer shall be eight continuous 32-bits configuration.

`static inline void ENET_QOS_ExitPowerDown(ENET_QOS_Type *base)`

Set the MAC to exit power down mode. Exit from the power down mode and recover to normal work mode.

**Parameters**

- `base` – ENET peripheral base address.

*status\_t* ENET\_QOS\_EnableRxParser(ENET\_QOS\_Type \*base, bool enable)

Enable/Disable Rx parser, please notice that for enable/disable Rx Parser, should better disable Receive first.

#### Parameters

- base – ENET\_QOS peripheral base address.
- enable – Enable/Disable Rx parser function

#### Return values

- kStatus\_Success – Configure rx parser success.
- kStatus\_ENET\_QOS\_Timeout – Poll status flag timeout.

void ENET\_QOS\_EnableInterrupts(ENET\_QOS\_Type \*base, uint32\_t mask)

Enables the ENET DMA and MAC interrupts.

This function enables the ENET interrupt according to the provided mask. The mask is a logical OR of `enet_qos_dma_interrupt_enable_t` and `enet_qos_mac_interrupt_enable_t`. For example, to enable the dma and mac interrupt, do the following.

```
ENET_QOS_EnableInterrupts(ENET, kENET_QOS_DmaRx | kENET_QOS_DmaTx | kENET_
↪QOS_MacPmt);
```

#### Parameters

- base – ENET peripheral base address.
- mask – ENET interrupts to enable. This is a logical OR of both enumeration :: `enet_qos_dma_interrupt_enable_t` and `enet_qos_mac_interrupt_enable_t`.

void ENET\_QOS\_DisableInterrupts(ENET\_QOS\_Type \*base, uint32\_t mask)

Disables the ENET DMA and MAC interrupts.

This function disables the ENET interrupt according to the provided mask. The mask is a logical OR of `enet_qos_dma_interrupt_enable_t` and `enet_qos_mac_interrupt_enable_t`. For example, to disable the dma and mac interrupt, do the following.

```
ENET_QOS_DisableInterrupts(ENET, kENET_QOS_DmaRx | kENET_QOS_DmaTx | kENET_
↪QOS_MacPmt);
```

#### Parameters

- base – ENET peripheral base address.
- mask – ENET interrupts to disables. This is a logical OR of both enumeration :: `enet_qos_dma_interrupt_enable_t` and `enet_qos_mac_interrupt_enable_t`.

static inline uint32\_t ENET\_QOS\_GetDmaInterruptStatus(ENET\_QOS\_Type \*base, uint8\_t channel)

Gets the ENET DMA interrupt status flag.

#### Parameters

- base – ENET peripheral base address.
- channel – The DMA Channel. Shall not be larger than `ENET_QOS_RING_NUM_MAX`.

#### Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration :: `enet_qos_dma_interrupt_enable_t`.

```
static inline void ENET_QOS_ClearDmaInterruptStatus(ENET_QOS_Type *base, uint8_t channel,
                                                    uint32_t mask)
```

Clear the ENET DMA interrupt status flag.

#### Parameters

- base – ENET peripheral base address.
- channel – The DMA Channel. Shall not be larger than ENET\_QOS\_RING\_NUM\_MAX.
- mask – The interrupt status to be cleared. This is the logical OR of members of the enumeration :: `enet_qos_dma_interrupt_enable_t`.

```
static inline uint32_t ENET_QOS_GetMacInterruptStatus(ENET_QOS_Type *base)
```

Gets the ENET MAC interrupt status flag.

#### Parameters

- base – ENET peripheral base address.

#### Returns

The event status of the interrupt source. Use the enum in `enet_qos_mac_interrupt_enable_t` and right shift ENET\_QOS\_MACINT\_ENUM\_OFFSET to mask the returned value to get the exact interrupt status.

```
void ENET_QOS_ClearMacInterruptStatus(ENET_QOS_Type *base, uint32_t mask)
```

Clears the ENET mac interrupt events status flag.

This function clears enabled ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See the `enet_qos_mac_interrupt_enable_t`. For example, to clear the TX frame interrupt and RX frame interrupt, do the following.

```
ENET_QOS_ClearMacInterruptStatus(ENET, kENET_QOS_MacPmt);
```

#### Parameters

- base – ENET peripheral base address.
- mask – ENET interrupt source to be cleared. This is the logical OR of members of the enumeration :: `enet_qos_mac_interrupt_enable_t`.

```
static inline bool ENET_QOS_IsTxDescriptorDmaOwn(enet_qos_tx_bd_struct_t *txDesc)
```

Get the tx descriptor DMA Own flag.

#### Parameters

- txDesc – The given tx descriptor.

#### Return values

True – the dma own tx descriptor, false application own tx descriptor.

```
void ENET_QOS_SetupTxDescriptor(enet_qos_tx_bd_struct_t *txDesc, void *buffer1, uint32_t
                                bytes1, void *buffer2, uint32_t bytes2, uint32_t framelen,
                                bool intEnable, bool tsEnable, enet_qos_desc_flag flag,
                                uint8_t slotNum)
```

Setup a given tx descriptor. This function is a low level functional API to setup or prepare a given tx descriptor.

---

**Note:** This must be called after all the ENET initialization. And should be called when the ENET receive/transmit is required. Transmit buffers are ‘zero-copy’ buffers, so the buffer must remain in memory until the packet has been fully transmitted. The buffers should be free or requeued in the transmit interrupt irq handler.

---

### Parameters

- txDesc – The given tx descriptor.
- buffer1 – The first buffer address in the descriptor.
- bytes1 – The bytes in the first buffer.
- buffer2 – The second buffer address in the descriptor.
- bytes2 – The bytes in the second buffer.
- framelen – The length of the frame to be transmitted.
- intEnable – Interrupt enable flag.
- tsEnable – The timestamp enable.
- flag – The flag of this tx descriptor, `enet_qos_desc_flag`.
- slotNum – The slot num used for AV only.

```
static inline void ENET_QOS_UpdateTxDescriptorTail(ENET_QOS_Type *base, uint8_t channel,  
                                                  uint32_t txDescTailAddrAlign)
```

Update the tx descriptor tail pointer. This function is a low level functional API to update the tx descriptor tail. This is called after you setup a new tx descriptor to update the tail pointer to make the new descriptor accessible by DMA.

### Parameters

- base – ENET peripheral base address.
- channel – The tx DMA channel.
- txDescTailAddrAlign – The new tx tail pointer address.

```
static inline void ENET_QOS_UpdateRxDescriptorTail(ENET_QOS_Type *base, uint8_t channel,  
                                                  uint32_t rxDescTailAddrAlign)
```

Update the rx descriptor tail pointer. This function is a low level functional API to update the rx descriptor tail. This is called after you setup a new rx descriptor to update the tail pointer to make the new descriptor accessible by DMA and to anounce the rx poll command for DMA.

### Parameters

- base – ENET peripheral base address.
- channel – The rx DMA channel.
- rxDescTailAddrAlign – The new rx tail pointer address.

```
static inline uint32_t ENET_QOS_GetRxDescriptor(enet_qos_rx_bd_struct_t *rxDesc)
```

Gets the context in the ENET rx descriptor. This function is a low level functional API to get the status flag from a given rx descriptor.

---

**Note:** This must be called after all the ENET initialization. And should be called when the ENET receive/transmit is required.

---

### Parameters

- rxDesc – The given rx descriptor.

### Return values

The – RDES3 regions for write-back format rx buffer descriptor.



```
void ENET_QOS_UpdateRxDescriptor(enet_qos_rx_bd_struct_t *rxDesc, void *buffer1, void  
                                *buffer2, bool intEnable, bool doubleBuffEnable)
```

Updates the buffers and the own status for a given rx descriptor. This function is a low level functional API to Updates the buffers and the own status for a given rx descriptor.

---

**Note:** This must be called after all the ENET initialization. And should be called when the ENET receive/transmit is required.

---

#### Parameters

- rxDesc – The given rx descriptor.
- buffer1 – The first buffer address in the descriptor.
- buffer2 – The second buffer address in the descriptor.
- intEnable – Interrupt enable flag.
- doubleBuffEnable – The double buffer enable flag.

```
status_t ENET_QOS_ConfigureRxParser(ENET_QOS_Type *base, enet_qos_rxp_config_t  
                                   *rxpConfig, uint16_t entryCount)
```

Configure flexible rx parser.

This function is used to configure the flexible rx parser table.

#### Parameters

- base – ENET peripheral base address..
- rxpConfig – The rx parser configuration pointer.
- entryCount – The rx parser entry count.

#### Return values

- kStatus\_Success – Configure rx parser success.
- kStatus\_ENET\_QOS\_Timeout – Poll status flag timeout.

```
status_t ENET_QOS_ReadRxParser(ENET_QOS_Type *base, enet_qos_rxp_config_t *rxpConfig,  
                               uint16_t entryIndex)
```

Read flexible rx parser configuration at specified index.

This function is used to read flexible rx parser configuration at specified index.

#### Parameters

- base – ENET peripheral base address..
- rxpConfig – The rx parser configuration pointer.
- entryIndex – The rx parser entry index to read, start from 0.

#### Return values

- kStatus\_Success – Configure rx parser success.
- kStatus\_ENET\_QOS\_Timeout – Poll status flag timeout.

```
status_t ENET_QOS_EstProgramGcl(ENET_QOS_Type *base, enet_qos_est_gcl_t *gcl, uint32_t  
                                ptpClk_Hz)
```

Program Gate Control List.

This function is used to program the Enhanced Scheduled Transmisson. (IEEE802.1Qbv)

#### Parameters

- base – ENET peripheral base address..

- `gcl` – Pointer to the Gate Control List structure.
- `ptpClk_Hz` – frequency of the PTP clock.

`status_t` ENET\_QOS\_EstReadGcl(ENET\_QOS\_Type \*base, *enet\_qos\_est\_gcl\_t* \*gcl, uint32\_t listLen, bool hwList)

Read Gate Control List.

This function is used to read the Enhanced Scheduled Transmisson list. (IEEE802.1Qbv)

#### Parameters

- `base` – ENET peripheral base address..
- `gcl` – Pointer to the Gate Control List structure.
- `listLen` – length of the provided opList array in `gcl` structure.
- `hwList` – Boolean if True read HW list, false read SW list.

static inline void ENET\_QOS\_FpeEnable(ENET\_QOS\_Type \*base)

Enable Frame Preemption.

This function is used to enable frame preemption. (IEEE802.1Qbu)

#### Parameters

- `base` – ENET peripheral base address..

static inline void ENET\_QOS\_FpeDisable(ENET\_QOS\_Type \*base)

Disable Frame Preemption.

This function is used to disable frame preemption. (IEEE802.1Qbu)

#### Parameters

- `base` – ENET peripheral base address..

static inline void ENET\_QOS\_FpeConfigPreemptable(ENET\_QOS\_Type \*base, uint8\_t queueMask)

Configure preemptable transmit queues.

This function is used to configure the preemptable queues. (IEEE802.1Qbu)

#### Parameters

- `base` – ENET peripheral base address..
- `queueMask` – bitmask representing queues to set in preemptable mode. The N-th bit represents the queue N.

void ENET\_QOS\_AVBConfigure(ENET\_QOS\_Type \*base, const *enet\_qos\_cbs\_config\_t* \*config, uint8\_t queueIndex)

Sets the ENET AVB feature.

ENET\_QOS AVB feature configuration, set transmit bandwidth. This API is called when the AVB feature is required.

#### Parameters

- `base` – ENET\_QOS peripheral base address.
- `config` – The ENET\_QOS AVB feature configuration structure.
- `queueIndex` – ENET\_QOS queue index.

void ENET\_QOS\_GetStatistics(ENET\_QOS\_Type \*base, *enet\_qos\_transfer\_stats\_t* \*statistics)

Gets statistical data in transfer.

#### Parameters

- base – ENET\_QOS peripheral base address.
- statistics – The statistics structure pointer.

```
void ENET_QOS_CreateHandler(ENET_QOS_Type *base, enet_qos_handle_t *handle,  
                           enet_qos_config_t *config, enet_qos_buffer_config_t  
                           *bufferConfig, enet_qos_callback_t callback, void *userData)
```

Create ENET Handler.

This is a transactional API and it's provided to store all data which are needed during the whole transactional process. This API should not be used when you use functional APIs to do data tx/rx. This is function will store many data/flag for transactional use, so all configure API such as ENET\_QOS\_Init(), ENET\_QOS\_DescriptorInit(), ENET\_QOS\_EnableInterrupts() etc.

---

**Note:** as our transactional transmit API use the zero-copy transmit buffer. so there are two thing we emphasize here:

- a. tx buffer free/requeue for application should be done in the tx interrupt handler. Please set callback: kENET\_QOS\_TxIntEvent with tx buffer free/requeue process APIs.
  - b. the tx interrupt is forced to open.
- 

#### Parameters

- base – ENET peripheral base address.
- handle – ENET handler.
- config – ENET configuration.
- bufferConfig – ENET buffer configuration.
- callback – The callback function.
- userData – The application data.

```
status_t ENET_QOS_GetRxFramSize(ENET_QOS_Type *base, enet_qos_handle_t *handle,  
                                uint32_t *length, uint8_t channel)
```

Gets the size of the read frame. This function gets a received frame size from the ENET buffer descriptors.

---

**Note:** The FCS of the frame is automatically removed by MAC and the size is the length without the FCS. After calling ENET\_QOS\_GetRxFramSize, ENET\_QOS\_ReadFrame() should be called to update the receive buffers If the result is not "kStatus\_ENET\_QOS\_RxFramEmpty".

---

#### Parameters

- base – ENET peripheral base address.
- handle – The ENET handler structure. This is the same handler pointer used in the ENET\_QOS\_Init.
- length – The length of the valid frame received.
- channel – The DMAC channel for the rx.

#### Return values

- kStatus\_ENET\_QOS\_RxFramEmpty – No frame received. Should not call ENET\_QOS\_ReadFrame to read frame.

- `kStatus_ENET_QOS_RxFrameError` – Data error happens. `ENET_QOS_ReadFrame` should be called with NULL data and NULL length to update the receive buffers.
- `kStatus_Success` – Receive a frame Successfully then the `ENET_QOS_ReadFrame` should be called with the right data buffer and the captured data length input.

`status_t` `ENET_QOS_ReadFrame`(`ENET_QOS_Type` \*base, `enet_qos_handle_t` \*handle, `uint8_t` \*data, `uint32_t` length, `uint8_t` channel, `enet_qos_ptp_time_t` \*ts)

Reads a frame from the ENET device. This function reads a frame from the ENET DMA descriptors. The `ENET_QOS_GetRxFrameSize` should be used to get the size of the prepared data buffer. For example use rx dma channel 0:

```
uint32_t length;
enet_qos_handle_t g_handle;
status = ENET_QOS_GetRxFrameSize(&g_handle, &length, 0);
if (length != 0)
{
    uint8_t *data = memory allocate interface;
    if (!data)
    {
        ENET_QOS_ReadFrame(ENET, &g_handle, NULL, 0, 0);
    }
    else
    {
        status = ENET_QOS_ReadFrame(ENET, &g_handle, data, length, 0);
    }
}
else if (status == kStatus_ENET_QOS_RxFrameError)
{
    ENET_QOS_ReadFrame(ENET, &g_handle, NULL, 0, 0);
}
```

### Parameters

- base – ENET peripheral base address.
- handle – The ENET handler structure. This is the same handler pointer used in the `ENET_QOS_Init`.
- data – The data buffer provided by user to store the frame which memory size should be at least “length”.
- length – The size of the data buffer which is still the length of the received frame.
- channel – The rx DMA channel. shall not be larger than 2.
- ts – Pointer to the structure `enet_qos_ptp_time_t` to save frame timestamp.

### Returns

The execute status, successful or failure.

`status_t` `ENET_QOS_SendFrame`(`ENET_QOS_Type` \*base, `enet_qos_handle_t` \*handle, `uint8_t` \*data, `uint32_t` length, `uint8_t` channel, `bool` isNeedTs, `void` \*context, `enet_qos_tx_offload_t` txOffloadOps)

Transmits an ENET frame.

---

**Note:** The CRC is automatically appended to the data. Input the data to send without the

---

CRC.

---

### Parameters

- `base` – ENET peripheral base address.
- `handle` – The ENET handler pointer. This is the same handler pointer used in the `ENET_QOS_Init`.
- `data` – The data buffer provided by user to be send.
- `length` – The length of the data to be send.
- `channel` – Channel to send the frame, same with queue index.
- `isNeedTs` – True to enable timestamp save for the frame
- `context` – pointer to user context to be kept in the tx dirty frame information.
- `txOffloadOps` – The Tx frame checksum offload option.

### Return values

- `kStatus_Success` – Send frame succeed.
- `kStatus_ENET_QOS_TxFrameBusy` – Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity. The waiting mechanism is recommended to be added after each call return with `kStatus_ENET_QOS_TxFrameBusy`.

```
void ENET_QOS_ReclaimTxDescriptor(ENET_QOS_Type *base, enet_qos_handle_t *handle,
                                uint8_t channel)
```

Reclaim tx descriptors. This function is used to update the tx descriptor status and store the tx timestamp when the 1588 feature is enabled. This is called by the transmit interrupt IRQ handler after the complete of a frame transmission.

### Parameters

- `base` – ENET peripheral base address.
- `handle` – The ENET handler pointer. This is the same handler pointer used in the `ENET_QOS_Init`.
- `channel` – The tx DMA channel.

```
void ENET_QOS_CommonIRQHandler(ENET_QOS_Type *base, enet_qos_handle_t *handle)
```

The ENET IRQ handler.

### Parameters

- `base` – ENET peripheral base address.
- `handle` – The ENET handler pointer.

```
void ENET_QOS_SetISRHandler(ENET_QOS_Type *base, enet_qos_isr_t ISRHandler)
```

Set the second level IRQ handler, allow user to overwrite the default second level weak IRQ handler.

### Parameters

- `base` – ENET peripheral base address.
- `ISRHandler` – The handler to install.

```
status_t ENET_QOS_Ptp1588CorrectTimerInCoarse(ENET_QOS_Type *base, enet_qos_systime_op
                                                operation, uint32_t second, uint32_t
                                                nanosecond)
```

Correct the ENET PTP 1588 timer in coarse method.

**Parameters**

- base – ENET peripheral base address.
- operation – The system time operation, refer to “enet\_qos\_systime\_op”
- second – The correction second.
- nanosecond – The correction nanosecond.

*status\_t* ENET\_QOS\_Ptp1588CorrectTimerInFine(ENET\_QOS\_Type \*base, uint32\_t addend)  
Correct the ENET PTP 1588 timer in fine method.

---

**Note:** Should take refer to the chapter “System time correction” and see the description for the “fine correction method”.

---

**Parameters**

- base – ENET peripheral base address.
- addend – The addend value to be set in the fine method

static inline uint32\_t ENET\_QOS\_Ptp1588GetAddend(ENET\_QOS\_Type \*base)  
Get the ENET Time stamp current addend value.

**Parameters**

- base – ENET peripheral base address.

**Returns**

The addend value.

void ENET\_QOS\_Ptp1588GetTimerNoIRQDisable(ENET\_QOS\_Type \*base, uint64\_t \*second,  
uint32\_t \*nanosecond)

Gets the current ENET time from the PTP 1588 timer without IRQ disable.

**Parameters**

- base – ENET peripheral base address.
- second – The PTP 1588 system timer second.
- nanosecond – The PTP 1588 system timer nanosecond. For the unit of the nanosecond is 1ns. so the nanosecond is the real nanosecond.

static inline *status\_t* ENET\_Ptp1588PpsControl(ENET\_QOS\_Type \*base,  
enet\_qos\_ptp\_pps\_instance\_t instance,  
enet\_qos\_ptp\_pps\_trgt\_mode\_t trgtMode,  
enet\_qos\_ptp\_pps\_cmd\_t cmd)

Sets the ENET PTP 1588 PPS control. All channels operate in flexible PPS output mode.

**Parameters**

- base – ENET peripheral base address.
- instance – The ENET QOS PTP PPS instance.
- trgtMode – The target time register mode.
- cmd – The target flexible PPS output control command.

*status\_t* ENET\_QOS\_Ptp1588PpsSetTrgtTime(ENET\_QOS\_Type \*base,  
enet\_qos\_ptp\_pps\_instance\_t instance, uint32\_t  
seconds, uint32\_t nanoseconds)

Sets the ENET QOS PTP 1588 PPS target time registers.

**Parameters**

- base – ENET QOS peripheral base address.
- instance – The ENET QOS PTP PPS instance.
- seconds – The target seconds.
- nanoseconds – The target nanoseconds.

```
static inline void ENET_QOS_Ptp1588PpsSetWidth(ENET_QOS_Type *base,  
                                                enet_qos_ptp_pps_instance_t instance,  
                                                uint32_t width)
```

Sets the ENET QOS PTP 1588 PPS output signal interval.

#### Parameters

- base – ENET QOS peripheral base address.
- instance – The ENET QOS PTP PPS instance.
- width – Signal Width. It is stored in terms of number of units of sub-second increment value. The width value must be lesser than interval value.

```
static inline void ENET_QOS_Ptp1588PpsSetInterval(ENET_QOS_Type *base,  
                                                  enet_qos_ptp_pps_instance_t instance,  
                                                  uint32_t interval)
```

Sets the ENET QOS PTP 1588 PPS output signal width.

#### Parameters

- base – ENET QOS peripheral base address.
- instance – The ENET QOS PTP PPS instance.
- interval – Signal Interval. It is stored in terms of number of units of sub-second increment value.

```
void ENET_QOS_Ptp1588GetTimer(ENET_QOS_Type *base, uint64_t *second, uint32_t  
                             *nanosecond)
```

Gets the current ENET time from the PTP 1588 timer.

#### Parameters

- base – ENET peripheral base address.
- second – The PTP 1588 system timer second.
- nanosecond – The PTP 1588 system timer nanosecond. For the unit of the nanosecond is 1ns.so the nanosecond is the real nanosecond.

```
void ENET_QOS_GetTxFrame(enet_qos_handle_t *handle, enet_qos_frame_info_t *txFrame,  
                        uint8_t channel)
```

Gets the time stamp of the transmit frame.

This function is used for PTP stack to get the timestamp captured by the ENET driver.

#### Parameters

- handle – The ENET handler pointer.This is the same state pointer used in ENET\_QOS\_Init.
- txFrame – Input parameter, pointer to enet\_qos\_frame\_info\_t for saving read out frame information.
- channel – Channel for searching the tx frame.

```
status_t ENET_QOS_GetRxFrame(ENET_QOS_Type *base, enet_qos_handle_t *handle,  
                             enet_qos_rx_frame_struct_t *rxFrame, uint8_t channel)
```

Receives one frame in specified BD ring with zero copy.

This function will use the user-defined allocate and free callback. Every time application gets one frame through this function, driver will allocate new buffers for the BDs whose buffers have been taken by application.

---

**Note:** This function will drop current frame and update related BDs as available for DMA if new buffers allocating fails. Application must provide a memory pool including at least BD number + 1 buffers(+2 if enable double buffer) to make this function work normally. If user calls this function in Rx interrupt handler, be careful that this function makes Rx BD ready with allocating new buffer(normal) or updating current BD(out of memory). If there's always new Rx frame input, Rx interrupt will be triggered forever. Application need to disable Rx interrupt according to specific design in this case.

---

### Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer. This is the same handler pointer used in the ENET\_Init.
- rxFrame – The received frame information structure provided by user.
- channel – Channel for searching the rx frame.

### Return values

- kStatus\_Success – Succeed to get one frame and allocate new memory for Rx buffer.
- kStatus\_ENET\_QOS\_RxFrameEmpty – There's no Rx frame in the BD.
- kStatus\_ENET\_QOS\_RxFrameError – There's issue in this receiving.
- kStatus\_ENET\_QOS\_RxFrameDrop – There's no new buffer memory for BD, drop this frame.

FSL\_ENET\_QOS\_DRIVER\_VERSION

Defines the driver version.

ENET\_QOS\_RXDESCRIP\_RD\_BUFF1VALID\_MASK

Defines for read format.

Buffer1 address valid.

ENET\_QOS\_RXDESCRIP\_RD\_BUFF2VALID\_MASK

Buffer2 address valid.

ENET\_QOS\_RXDESCRIP\_RD\_IOC\_MASK

Interrupt enable on complete.

ENET\_QOS\_RXDESCRIP\_RD\_OWN\_MASK

Own bit.

ENET\_QOS\_RXDESCRIP\_WR\_ERR\_MASK

Defines for write back format.

ENET\_QOS\_RXDESCRIP\_WR\_PYLOAD\_MASK

ENET\_QOS\_RXDESCRIP\_WR\_PTPMSGTYPE\_MASK

ENET\_QOS\_RXDESCRIP\_WR\_PTPTYPE\_MASK



ENET\_QOS\_RXDESCRIP\_WR\_PTPVERSION\_MASK  
ENET\_QOS\_RXDESCRIP\_WR\_PTPTSA\_MASK  
ENET\_QOS\_RXDESCRIP\_WR\_PACKETLEN\_MASK  
ENET\_QOS\_RXDESCRIP\_WR\_ERRSUM\_MASK  
ENET\_QOS\_RXDESCRIP\_WR\_TYPE\_MASK  
ENET\_QOS\_RXDESCRIP\_WR\_DE\_MASK  
ENET\_QOS\_RXDESCRIP\_WR\_RE\_MASK  
ENET\_QOS\_RXDESCRIP\_WR\_OE\_MASK  
ENET\_QOS\_RXDESCRIP\_WR\_RWT\_MASK  
ENET\_QOS\_RXDESCRIP\_WR\_GP\_MASK  
ENET\_QOS\_RXDESCRIP\_WR\_CRC\_MASK  
ENET\_QOS\_RXDESCRIP\_WR\_RS0V\_MASK  
ENET\_QOS\_RXDESCRIP\_WR\_RS1V\_MASK  
ENET\_QOS\_RXDESCRIP\_WR\_RS2V\_MASK  
ENET\_QOS\_RXDESCRIP\_WR\_LD\_MASK  
ENET\_QOS\_RXDESCRIP\_WR\_FD\_MASK  
ENET\_QOS\_RXDESCRIP\_WR\_CTXT\_MASK  
ENET\_QOS\_RXDESCRIP\_WR\_OWN\_MASK  
ENET\_QOS\_RXDESCRIP\_WR\_SA\_FAILURE\_MASK  
ENET\_QOS\_RXDESCRIP\_WR\_DA\_FAILURE\_MASK  
ENET\_QOS\_TXDESCRIP\_RD\_BL1\_MASK

Defines for read format.

ENET\_QOS\_TXDESCRIP\_RD\_BL2\_MASK  
ENET\_QOS\_TXDESCRIP\_RD\_BL1(n)  
ENET\_QOS\_TXDESCRIP\_RD\_BL2(n)  
ENET\_QOS\_TXDESCRIP\_RD\_TTSE\_MASK  
ENET\_QOS\_TXDESCRIP\_RD\_IOC\_MASK  
ENET\_QOS\_TXDESCRIP\_RD\_FL\_MASK  
ENET\_QOS\_TXDESCRIP\_RD\_FL(n)  
ENET\_QOS\_TXDESCRIP\_RD\_CIC(n)  
ENET\_QOS\_TXDESCRIP\_RD\_TSE\_MASK  
ENET\_QOS\_TXDESCRIP\_RD\_SLOT(n)  
ENET\_QOS\_TXDESCRIP\_RD\_SAIC(n)

ENET\_QOS\_TXDESCRIP\_RD\_CPC(n)

ENET\_QOS\_TXDESCRIP\_RD\_LDFD(n)

ENET\_QOS\_TXDESCRIP\_RD\_LD\_MASK

ENET\_QOS\_TXDESCRIP\_RD\_FD\_MASK

ENET\_QOS\_TXDESCRIP\_RD\_CTXT\_MASK

ENET\_QOS\_TXDESCRIP\_RD\_OWN\_MASK

ENET\_QOS\_TXDESCRIP\_WB\_TTSS\_MASK

Defines for write back format.

ENET\_QOS\_ABNORM\_INT\_MASK

ENET\_QOS\_NORM\_INT\_MASK

ENET\_QOS\_RING\_NUM\_MAX

The Maximum number of tx/rx descriptor rings.

ENET\_QOS\_FRAME\_MAX\_FRAMELEN

Default maximum Ethernet frame size.

ENET\_QOS\_FCS\_LEN

Ethernet FCS length.

ENET\_QOS\_ADDR\_ALIGNMENT

Recommended Ethernet buffer alignment.

ENET\_QOS\_BUFF\_ALIGNMENT

Receive buffer alignment shall be 4bytes-aligned.

ENET\_QOS\_MTL\_RXFIFOSIZE

The rx fifo size.

ENET\_QOS\_MTL\_TXFIFOSIZE

The tx fifo size.

ENET\_QOS\_MACINT\_ENUM\_OFFSET

The offset for mac interrupt in enum type.

ENET\_QOS\_RXP\_ENTRY\_COUNT

RXP table entry count, implied by FRPES in MAC\_HW\_FEATURE3

ENET\_QOS\_RXP\_BUFFER\_SIZE

RXP Buffer size, implied by FRPBS in MAC\_HW\_FEATURE3

ENET\_QOS\_EST\_WID

Width of the time interval in Gate Control List

ENET\_QOS\_EST\_DEP

Maximum depth of Gate Control List

Defines the status return codes for transaction.

*Values:*

enumerator kStatus\_ENET\_QOS\_InitMemoryFail

Init fails since buffer memory is not enough.

enumerator kStatus\_ENET\_QOS\_RxFrameError

A frame received but data error happen.

enumerator kStatus\_ENET\_QOS\_RxFrameFail

Failed to receive a frame.

enumerator kStatus\_ENET\_QOS\_RxFrameEmpty

No frame arrive.

enumerator kStatus\_ENET\_QOS\_RxFrameDrop

Rx frame is dropped since no buffer memory.

enumerator kStatus\_ENET\_QOS\_TxFrameBusy

Transmit descriptors are under process.

enumerator kStatus\_ENET\_QOS\_TxFrameFail

Transmit frame fail.

enumerator kStatus\_ENET\_QOS\_TxFrameOverLen

Transmit oversize.

enumerator kStatus\_ENET\_QOS\_Est\_SwListBusy

SW Gcl List not yet processed by HW.

enumerator kStatus\_ENET\_QOS\_Est\_SwListWriteAbort

SW Gcl List write aborted .

enumerator kStatus\_ENET\_QOS\_Est\_InvalidParameter

Invalid parameter in Gcl List .

enumerator kStatus\_ENET\_QOS\_Est\_BtrError

Base Time Error when loading list.

enumerator kStatus\_ENET\_QOS\_TrgtBusy

Target time register busy.

enumerator kStatus\_ENET\_QOS\_Timeout

Target time register busy.

enumerator kStatus\_ENET\_QOS\_PpsBusy

Pps command busy.

enum \_enet\_qos\_mii\_mode

Defines the MII/RGMII mode for data interface between the MAC and the PHY.

*Values:*

enumerator kENET\_QOS\_MiiMode

MII mode for data interface.

enumerator kENET\_QOS\_RgmiiMode

RGMII mode for data interface.

enumerator kENET\_QOS\_RmiiMode

RMII mode for data interface.

enum \_enet\_qos\_mii\_speed

Defines the 10/100/1000 Mbps speed for the MII data interface.

*Values:*

enumerator kENET\_QOS\_MiiSpeed10M

Speed 10 Mbps.

enumerator kENET\_QOS\_MiiSpeed100M

Speed 100 Mbps.

enumerator kENET\_QOS\_MiiSpeed1000M

Speed 1000 Mbps.

enumerator kENET\_QOS\_MiiSpeed2500M

Speed 2500 Mbps.

enum \_enet\_qos\_mii\_duplex

Defines the half or full duplex for the MII data interface.

*Values:*

enumerator kENET\_QOS\_MiiHalfDuplex

Half duplex mode.

enumerator kENET\_QOS\_MiiFullDuplex

Full duplex mode.

enum \_enet\_qos\_mii\_normal\_opcode

Define the MII opcode for normal MDIO\_CLAUSES\_22 Frame.

*Values:*

enumerator kENET\_QOS\_MiiWriteFrame

Write frame operation for a valid MII management frame.

enumerator kENET\_QOS\_MiiReadFrame

Read frame operation for a valid MII management frame.

enum \_enet\_qos\_dma\_burstlen

Define the DMA maximum transmit burst length.

*Values:*

enumerator kENET\_QOS\_BurstLen1

DMA burst length 1.

enumerator kENET\_QOS\_BurstLen2

DMA burst length 2.

enumerator kENET\_QOS\_BurstLen4

DMA burst length 4.

enumerator kENET\_QOS\_BurstLen8

DMA burst length 8.

enumerator kENET\_QOS\_BurstLen16

DMA burst length 16.

enumerator kENET\_QOS\_BurstLen32

DMA burst length 32.

enumerator kENET\_QOS\_BurstLen64

DMA burst length 64. eight times enabled.

enumerator kENET\_QOS\_BurstLen128

DMA burst length 128. eight times enabled.

enumerator kENET\_QOS\_BurstLen256

DMA burst length 256. eight times enabled.

enum \_enet\_qos\_desc\_flag

Define the flag for the descriptor.

*Values:*

enumerator kENET\_QOS\_MiddleFlag

It's a middle descriptor of the frame.

enumerator kENET\_QOS\_LastFlagOnly

It's the last descriptor of the frame.

enumerator kENET\_QOS\_FirstFlagOnly

It's the first descriptor of the frame.

enumerator kENET\_QOS\_FirstLastFlag

It's the first and last descriptor of the frame.

enum \_enet\_qos\_systime\_op

Define the system time adjust operation control.

*Values:*

enumerator kENET\_QOS\_SystimeAdd

System time add to.

enumerator kENET\_QOS\_SystimeSubtract

System time subtract.

enum \_enet\_qos\_ts\_rollover\_type

Define the system time rollover control.

*Values:*

enumerator kENET\_QOS\_BinaryRollover

System time binary rollover.

enumerator kENET\_QOS\_DigitalRollover

System time digital rollover.

enum \_enet\_qos\_special\_config

Defines some special configuration for ENET.

These control flags are provided for special user requirements. Normally, there is no need to set these control flags for ENET initialization. But if you have some special requirements, set the flags to specialControl in the enet\_qos\_config\_t.

---

**Note:** “kENET\_QOS\_StoreAndForward” is recommended to be set.

---

*Values:*

enumerator kENET\_QOS\_DescDoubleBuffer

The double buffer is used in the tx/rx descriptor.

enumerator kENET\_QOS\_StoreAndForward

The rx/tx store and forward enable.

enumerator kENET\_QOS\_PromiscuousEnable

The promiscuous enabled.

enumerator kENET\_QOS\_FlowControlEnable

The flow control enabled.

enumerator kENET\_QOS\_BroadCastRxDisable

The broadcast disabled.

enumerator kENET\_QOS\_MulticastAllEnable

All multicast are passed.

enumerator kENET\_QOS\_8023AS2KPacket

8023as support for 2K packets.

enumerator kENET\_QOS\_HashMulticastEnable

The multicast packets are filtered through hash table.

enumerator kENET\_QOS\_RxChecksumOffloadEnable

The Rx checksum offload enabled.

enum \_enet\_qos\_dma\_interrupt\_enable

List of DMA interrupts supported by the ENET interrupt. This enumeration uses one-bot encoding to allow a logical OR of multiple members.

*Values:*

enumerator kENET\_QOS\_DmaTx

Tx interrupt.

enumerator kENET\_QOS\_DmaTxStop

Tx stop interrupt.

enumerator kENET\_QOS\_DmaTxBuffUnavail

Tx buffer unavailable.

enumerator kENET\_QOS\_DmaRx

Rx interrupt.

enumerator kENET\_QOS\_DmaRxBuffUnavail

Rx buffer unavailable.

enumerator kENET\_QOS\_DmaRxStop

Rx stop.

enumerator kENET\_QOS\_DmaRxWatchdogTimeout

Rx watchdog timeout.

enumerator kENET\_QOS\_DmaEarlyTx

Early transmit.

enumerator kENET\_QOS\_DmaEarlyRx

Early receive.

enumerator kENET\_QOS\_DmaBusErr

Fatal bus error.

enum \_enet\_qos\_mac\_interrupt\_enable

List of mac interrupts supported by the ENET interrupt. This enumeration uses one-bot encoding to allow a logical OR of multiple members.

*Values:*

enumerator kENET\_QOS\_MacTimestamp

enum \_enet\_qos\_event

Defines the common interrupt event for callback use.

*Values:*

enumerator kENET\_QOS\_RxIntEvent  
Receive interrupt event.

enumerator kENET\_QOS\_TxIntEvent  
Transmit interrupt event.

enumerator kENET\_QOS\_WakeUpIntEvent  
Wake up interrupt event.

enumerator kENET\_QOS\_TimeStampIntEvent  
Time stamp interrupt event.

enum \_enet\_qos\_queue\_mode  
Define the MTL mode for multiple queues/rings.

*Values:*

enumerator kENET\_QOS\_AVB\_Mode  
Enable queue in AVB mode.

enumerator kENET\_QOS\_DCB\_Mode  
Enable queue in DCB mode.

enum \_enet\_qos\_mtl\_multiqueue\_txsche  
Define the MTL tx scheduling algorithm for multiple queues/rings.

*Values:*

enumerator kENET\_QOS\_txWeightRR  
Tx weight round-robin.

enumerator kENET\_QOS\_txWeightFQ  
Tx weight fair queuing.

enumerator kENET\_QOS\_txDefictWeightRR  
Tx deficit weighted round-robin.

enumerator kENET\_QOS\_txStrPrio  
Tx strict priority.

enum \_enet\_qos\_mtl\_multiqueue\_rxsche  
Define the MTL rx scheduling algorithm for multiple queues/rings.

*Values:*

enumerator kENET\_QOS\_rxStrPrio  
Rx strict priority, Queue 0 has the lowest priority.

enumerator kENET\_QOS\_rxWeightStrPrio  
Weighted Strict Priority.

enum \_enet\_qos\_mtl\_rxqueuemap  
Define the MTL rx queue and DMA channel mapping.

*Values:*

enumerator kENET\_QOS\_StaticDirctMap  
The received fame in rx Qn(n = 0,1) directly map to dma channel n.

enumerator kENET\_QOS\_DynamicMap  
The received frame in rx Qn(n = 0,1) map to the dma channel m(m = 0,1) related with the same Mac.

enum \_enet\_qos\_rx\_queue\_route

Defines the package type for receive queue routing.

*Values:*

enumerator kENET\_QOS\_PacketNoQ

enumerator kENET\_QOS\_PacketAVCPQ

enumerator kENET\_QOS\_PacketPTPQ

enumerator kENET\_QOS\_PacketUPQ

enumerator kENET\_QOS\_PacketMCBCQ

enum \_enet\_qos\_ptp\_event\_type

Defines the ENET PTP message related constant.

*Values:*

enumerator kENET\_QOS\_PtpEventMsgType

PTP event message type.

enumerator kENET\_QOS\_PtpSrcPortIdLen

PTP message sequence id length.

enumerator kENET\_QOS\_PtpEventPort

PTP event port number.

enumerator kENET\_QOS\_PtpGnrlPort

PTP general port number.

enum \_enet\_qos\_ptp\_pps\_instance

Defines the PPS instance numbers.

*Values:*

enumerator kENET\_QOS\_PtpPpsInstance0

PPS instance 0.

enumerator kENET\_QOS\_PtpPpsInstance1

PPS instance 1.

enumerator kENET\_QOS\_PtpPpsInstance2

PPS instance 2.

enumerator kENET\_QOS\_PtpPpsInstance3

PPS instance 3.

enum \_enet\_qos\_ptp\_pps\_trgt\_mode

Defines the Target Time register mode.

*Values:*

enumerator kENET\_QOS\_PtpPpsTrgtModeOnlyInt

Only interrupts.

enumerator kENET\_QOS\_PtpPpsTrgtModeIntSt

Both interrupt and output signal.

enumerator kENET\_QOS\_PtpPpsTrgtModeOnlySt

Only output signal.



enum \_enet\_qos\_ptp\_pps\_cmd

Defines commands for ppscmd register.

*Values:*

enumerator kENET\_QOS\_PtpPpsCmdNC  
No Command.

enumerator kENET\_QOS\_PtpPpsCmdSSP  
Start Single Pulse.

enumerator kENET\_QOS\_PtpPpsCmdSPT  
Start Pulse Train.

enumerator kENET\_QOS\_PtpPpsCmdCS  
Cancel Start.

enumerator kENET\_QOS\_PtpPpsCmdSPTAT  
Stop Pulse Train At Time.

enumerator kENET\_QOS\_PtpPpsCmdSPTI  
Stop Pulse Train Immediately.

enumerator kENET\_QOS\_PtpPpsCmdCSPT  
Cancel Stop Pulse Train.

enum \_enet\_qos\_ets\_list\_length

Defines the enumeration of ETS list length.

*Values:*

enumerator kENET\_QOS\_Ets\_List\_64  
List length of 64

enumerator kENET\_QOS\_Ets\_List\_128  
List length of 128

enumerator kENET\_QOS\_Ets\_List\_256  
List length of 256

enumerator kENET\_QOS\_Ets\_List\_512  
List length of 512

enumerator kENET\_QOS\_Ets\_List\_1024  
List length of 1024

enum \_enet\_qos\_ets\_gccr\_addr

Defines the enumeration of ETS gate control address.

*Values:*

enumerator kENET\_QOS\_Ets\_btr\_low  
BTR Low

enumerator kENET\_QOS\_Ets\_btr\_high  
BTR High

enumerator kENET\_QOS\_Ets\_ctr\_low  
CTR Low

enumerator kENET\_QOS\_Ets\_ctr\_high  
CTR High

enumerator kENET\_QOS\_Ets\_ter  
TER

enumerator kENET\_QOS\_Ets\_llr  
LLR

enum \_enet\_qos\_rxp\_dma\_chn

Defines the enumeration of DMA channel used for rx parser entry.

*Values:*

enumerator kENET\_QOS\_Rxp\_DMACHn0  
DMA Channel 0 used for RXP entry match

enumerator kENET\_QOS\_Rxp\_DMACHn1  
DMA Channel 1 used for RXP entry match

enumerator kENET\_QOS\_Rxp\_DMACHn2  
DMA Channel 2 used for RXP entry match

enumerator kENET\_QOS\_Rxp\_DMACHn3  
DMA Channel 3 used for RXP entry match

enumerator kENET\_QOS\_Rxp\_DMACHn4  
DMA Channel 4 used for RXP entry match

enum \_enet\_qos\_tx\_offload

Define the Tx checksum offload options.

*Values:*

enumerator kENET\_QOS\_TxOffloadDisable  
Disable Tx checksum offload.

enumerator kENET\_QOS\_TxOffloadIPHeader  
Enable IP header checksum calculation and insertion.

enumerator kENET\_QOS\_TxOffloadIPHeaderPlusPayload  
Enable IP header and payload checksum calculation and insertion.

enumerator kENET\_QOS\_TxOffloadAll  
Enable IP header, payload and pseudo header checksum calculation and insertion.

typedef enum \_enet\_qos\_mii\_mode enet\_qos\_mii\_mode\_t

Defines the MII/RGMII mode for data interface between the MAC and the PHY.

typedef enum \_enet\_qos\_mii\_speed enet\_qos\_mii\_speed\_t

Defines the 10/100/1000 Mbps speed for the MII data interface.

typedef enum \_enet\_qos\_mii\_duplex enet\_qos\_mii\_duplex\_t

Defines the half or full duplex for the MII data interface.

typedef enum \_enet\_qos\_mii\_normal\_opcode enet\_qos\_mii\_normal\_opcode

Define the MII opcode for normal MDIO\_CLAUSES\_22 Frame.

typedef enum \_enet\_qos\_dma\_burstlen enet\_qos\_dma\_burstlen

Define the DMA maximum transmit burst length.

typedef enum \_enet\_qos\_desc\_flag enet\_qos\_desc\_flag

Define the flag for the descriptor.

typedef enum \_enet\_qos\_systime\_op enet\_qos\_systime\_op

Define the system time adjust operation control.

typedef enum *\_enet\_qos\_ts\_rollover\_type* enet\_qos\_ts\_rollover\_type

Define the system time rollover control.

typedef enum *\_enet\_qos\_special\_config* enet\_qos\_special\_config\_t

Defines some special configuration for ENET.

These control flags are provided for special user requirements. Normally, there is no need to set these control flags for ENET initialization. But if you have some special requirements, set the flags to specialControl in the enet\_qos\_config\_t.

---

**Note:** “kENET\_QOS\_StoreAndForward” is recommended to be set.

---

typedef enum *\_enet\_qos\_dma\_interrupt\_enable* enet\_qos\_dma\_interrupt\_enable\_t

List of DMA interrupts supported by the ENET interrupt. This enumeration uses one-hot encoding to allow a logical OR of multiple members.

typedef enum *\_enet\_qos\_mac\_interrupt\_enable* enet\_qos\_mac\_interrupt\_enable\_t

List of mac interrupts supported by the ENET interrupt. This enumeration uses one-hot encoding to allow a logical OR of multiple members.

typedef enum *\_enet\_qos\_event* enet\_qos\_event\_t

Defines the common interrupt event for callback use.

typedef enum *\_enet\_qos\_queue\_mode* enet\_qos\_queue\_mode\_t

Define the MTL mode for multiple queues/rings.

typedef enum *\_enet\_qos\_mtl\_multiqueue\_txsche* enet\_qos\_mtl\_multiqueue\_txsche

Define the MTL tx scheduling algorithm for multiple queues/rings.

typedef enum *\_enet\_qos\_mtl\_multiqueue\_rxsche* enet\_qos\_mtl\_multiqueue\_rxsche

Define the MTL rx scheduling algorithm for multiple queues/rings.

typedef enum *\_enet\_qos\_mtl\_rxqueueuemap* enet\_qos\_mtl\_rxqueueuemap\_t

Define the MTL rx queue and DMA channel mapping.

typedef enum *\_enet\_qos\_rx\_queue\_route* enet\_qos\_rx\_queue\_route\_t

Defines the package type for receive queue routing.

typedef enum *\_enet\_qos\_ptp\_event\_type* enet\_qos\_ptp\_event\_type\_t

Defines the ENET PTP message related constant.

typedef enum *\_enet\_qos\_ptp\_pps\_instance* enet\_qos\_ptp\_pps\_instance\_t

Defines the PPS instance numbers.

typedef enum *\_enet\_qos\_ptp\_pps\_trgt\_mode* enet\_qos\_ptp\_pps\_trgt\_mode\_t

Defines the Target Time register mode.

typedef enum *\_enet\_qos\_ptp\_pps\_cmd* enet\_qos\_ptp\_pps\_cmd\_t

Defines commands for ppscmd register.

typedef enum *\_enet\_qos\_ets\_list\_length* enet\_qos\_ets\_list\_length\_t

Defines the enumeration of ETS list length.

typedef enum *\_enet\_qos\_ets\_gccr\_addr* enet\_qos\_ets\_gccr\_addr\_t

Defines the enumeration of ETS gate control address.

typedef enum *\_enet\_qos\_rxp\_dma\_chn* enet\_qos\_rxp\_dma\_chn\_t

Defines the enumeration of DMA channel used for rx parser entry.

typedef enum *\_enet\_qos\_tx\_offload* enet\_qos\_tx\_offload\_t

Define the Tx checksum offload options.

```
typedef struct _enet_qos_rx_bd_struct enet_qos_rx_bd_struct_t
```

Defines the receive descriptor structure has the read-format and write-back format structure. They both has the same size with different region definition. so we define the read-format region as the receive descriptor structure Use the read-format region mask bits in the descriptor initialization Use the write-back format region mask bits in the receive data process.

```
typedef struct _enet_qos_tx_bd_struct enet_qos_tx_bd_struct_t
```

Defines the transmit descriptor structure has the read-format and write-back format structure. They both has the same size with different region definition. so we define the read-format region as the transmit descriptor structure Use the read-format region mask bits in the descriptor initialization Use the write-back format region mask bits in the transmit data process.

```
typedef struct _enet_qos_tx_bd_config_struct enet_qos_tx_bd_config_struct_t
```

Defines the Tx BD configuration structure.

```
typedef struct _enet_qos_ptp_time enet_qos_ptp_time_t
```

Defines the ENET PTP time stamp structure.

```
typedef struct enet_qos_frame_info enet_qos_frame_info_t
```

Defines the frame info structure.

```
typedef struct _enet_qos_tx_dirty_ring enet_qos_tx_dirty_ring_t
```

Defines the ENET transmit dirty addresses ring/queue structure.

```
typedef struct _enet_qos_ptp_config enet_qos_ptp_config_t
```

Defines the ENET PTP configuration structure.

```
typedef struct _enet_qos_est_gate_op enet_qos_est_gate_op_t
```

Defines the EST gate operation structure.

```
typedef struct _enet_qos_est_gcl enet_qos_est_gcl_t
```

Defines the EST gate control list structure.

```
typedef struct _enet_qos_rxp_config enet_qos_rxp_config_t
```

Defines the ENET\_QOS Rx parser configuration structure.

```
typedef struct _enet_qos_buffer_config enet_qos_buffer_config_t
```

Defines the buffer descriptor configure structure.

---

**Note:**

- a. The receive and transmit descriptor start address pointer and tail pointer must be word-aligned.
  - b. The recommended minimum tx/rx ring length is 4.
  - c. The tx/rx descriptor tail address shall be the address pointer to the address just after the end of the last last descriptor. because only the descriptors between the start address and the tail address will be used by DMA.
  - d. The descriptor address is the start address of all used contiguous memory. for example, the rxDescStartAddrAlign is the start address of rxRingLen contiguous descriptor memories for rx descriptor ring 0.
  - e. The “\*rxBufferstartAddr” is the first element of rxRingLen (2\*rxRingLen for double buffers) rx buffers. It means the \*rxBufferStartAddr is the rx buffer for the first descriptor the \*rxBufferStartAddr + 1 is the rx buffer for the second descriptor or the rx buffer for the second buffer in the first descriptor. so please make sure the rxBufferStartAddr is the address of a rxRingLen or 2\*rxRingLen array.
-

```
typedef struct _enet_qos_cbs_config enet_qos_cbs_config_t
```

Defines the CBS configuration for queue.

```
typedef struct _enet_qos_tx_queue_config enet_qos_queue_tx_config_t
```

Defines the queue configuration structure.

```
typedef struct _enet_qos_rx_queue_config enet_qos_queue_rx_config_t
```

Defines the queue configuration structure.

```
typedef struct _enet_qos_multiqueue_config enet_qos_multiqueue_config_t
```

Defines the configuration when multi-queue is used.

```
typedef void (*enet_qos_rx_alloc_callback_t)(ENET_QOS_Type *base, void *userData, uint8_t channel)
```

Defines the Rx memory buffer alloc function pointer.

```
typedef void (*enet_qos_rx_free_callback_t)(ENET_QOS_Type *base, void *buffer, void *userData, uint8_t channel)
```

Defines the Rx memory buffer free function pointer.

```
typedef struct _enet_qos_config enet_qos_config_t
```

Defines the basic configuration structure for the ENET device.

---

**Note:** Default the signal queue is used so the “\*multiqueueCfg” is set default with NULL. Set the pointer with a valid configuration pointer if the multiple queues are required. If multiple queue is enabled, please make sure the buffer configuration for all are prepared also.

---

```
typedef struct _enet_qos_handle enet_qos_handle_t
```

```
typedef void (*enet_qos_callback_t)(ENET_QOS_Type *base, enet_qos_handle_t *handle, enet_qos_event_t event, uint8_t channel, void *userData)
```

ENET callback function.

```
typedef struct _enet_qos_tx_bd_ring enet_qos_tx_bd_ring_t
```

Defines the ENET transmit buffer descriptor ring/queue structure.

```
typedef struct _enet_qos_rx_bd_ring enet_qos_rx_bd_ring_t
```

Defines the ENET receive buffer descriptor ring/queue structure.

```
typedef struct _enet_qos_state enet_qos_state_t
```

Defines the ENET state structure.

---

**Note:** The structure contains saved state for the instance. It could be stored in *enet\_qos\_handle\_t*, but that's used only with the transactional API.

---

```
typedef struct _enet_qos_buffer_struct enet_qos_buffer_struct_t
```

Defines the frame buffer structure.

```
typedef struct _enet_qos_rx_frame_error enet_qos_rx_frame_error_t
```

Defines the Rx frame error structure.

```
typedef struct _enet_qos_rx_frame_attribute_struct enet_qos_rx_frame_attribute_t
```

```
typedef struct _enet_qos_rx_frame_struct enet_qos_rx_frame_struct_t
```

Defines the Rx frame data structure.

```
typedef struct _enet_qos_transfer_stats enet_qos_transfer_stats_t
```

Defines the ENET QOS transfer statistics structure.

```
typedef void (*enet_qos_isr_t)(ENET_QOS_Type *base, enet_qos_handle_t *handle)
```

```
const clock_ip_name_t s_enetqosClock[]
```

Pointers to enet clocks for each instance.

```
void ENET_QOS_SetSYSControl(enet_qos_mii_mode_t miiMode)
```

Set ENET system configuration.

---

**Note:** User needs to provide the implementation because the implementation is SoC specific. This function set the phy selection and enable clock. It should be called before any other ethernet operation.

---

### Parameters

- miiMode – The MII/RGMII/RMII mode for interface between the phy and Ethernet.

```
void ENET_QOS_EnableClock(bool enable)
```

Enable/Disable ENET qos clock.

---

**Note:** User needs to provide the implementation because the implementation is SoC specific. This function should be called before config RMII mode.

---

```
struct _enet_qos_rx_bd_struct
```

*#include <fsl\_enet\_qos.h>* Defines the receive descriptor structure has the read-format and write-back format structure. They both has the same size with different region definition. so we define the read-format region as the receive descriptor structure Use the read-format region mask bits in the descriptor initialization Use the write-back format region mask bits in the receive data process.

### Public Members

```
__IO uint32_t buff1Addr
```

Buffer 1 address

```
__IO uint32_t reserved
```

Reserved

```
__IO uint32_t buff2Addr
```

Buffer 2 or next descriptor address

```
__IO uint32_t control
```

Buffer 1/2 byte counts and control

```
struct _enet_qos_tx_bd_struct
```

*#include <fsl\_enet\_qos.h>* Defines the transmit descriptor structure has the read-format and write-back format structure. They both has the same size with different region definition. so we define the read-format region as the transmit descriptor structure Use the read-format region mask bits in the descriptor initialization Use the write-back format region mask bits in the transmit data process.

### Public Members

```
__IO uint32_t buff1Addr
```

Buffer 1 address

\_\_IO uint32\_t buff2Addr  
    Buffer 2 address

\_\_IO uint32\_t buffLen  
    Buffer 1/2 byte counts

\_\_IO uint32\_t controlStat  
    TDDES control and status word

struct \_\_enet\_qos\_tx\_bd\_config\_struct  
    *#include <fsl\_enet\_qos.h>* Defines the Tx BD configuration structure.

### Public Members

void \*buffer1  
    The first buffer address in the descriptor.

uint32\_t bytes1  
    The bytes in the fist buffer.

void \*buffer2  
    The second buffer address in the descriptor.

uint32\_t bytes2  
    The bytes in the second buffer.

uint32\_t framelen  
    The length of the frame to be transmitted.

bool intEnable  
    Interrupt enable flag.

bool tsEnable  
    The timestamp enable.

enet\_qos\_tx\_offload\_t txOffloadOps  
    The Tx checksum offload option.

enet\_qos\_desc\_flag flag  
    The flag of this tx desciriptor, see “enet\_qos\_desc\_flag”.

struct \_\_enet\_qos\_ptp\_time  
    *#include <fsl\_enet\_qos.h>* Defines the ENET PTP time stamp structure.

### Public Members

uint64\_t second  
    Second.

uint32\_t nanosecond  
    Nanosecond.

struct enet\_qos\_frame\_info  
    *#include <fsl\_enet\_qos.h>* Defines the frame info structure.

### Public Members

void \*context

User specified data, could be buffer address for free

bool isTsAvail

Flag indicates timestamp available status

enet\_qos\_ptp\_time\_t timeStamp

Timestamp of frame

struct \_\_enet\_qos\_tx\_dirty\_ring

#include <fsl\_enet\_qos.h> Defines the ENET transmit dirty addresses ring/queue structure.

### Public Members

enet\_qos\_frame\_info\_t \*txDirtyBase

Dirty buffer descriptor base address pointer.

uint16\_t txGenIdx

tx generate index.

uint16\_t txConsumIdx

tx consume index.

uint16\_t txRingLen

tx ring length.

bool isFull

tx ring is full flag, add this parameter to avoid waste one element.

struct \_\_enet\_qos\_ptp\_config

#include <fsl\_enet\_qos.h> Defines the ENET PTP configuration structure.

### Public Members

bool fineUpdateEnable

Use the fine update.

uint32\_t defaultAddend

Default addend value when fine update is enable, could be  $2^{32} / (\text{refClk\_Hz} / \text{ENET\_QOS\_MICRSECS\_ONESECOND} / \text{ENET\_QOS\_SYSTIME\_REQUIRED\_CLK\_MHZ})$ .

bool ptp1588V2Enable

The desired system time frequency. Must be lower than reference clock. (Only used with fine correction method). ptp 1588 version 2 is used.

enet\_qos\_ts\_rollover\_type tsRollover

1588 time nanosecond rollover.

struct \_\_enet\_qos\_est\_gate\_op

#include <fsl\_enet\_qos.h> Defines the EST gate operation structure.

struct \_\_enet\_qos\_est\_gcl

#include <fsl\_enet\_qos.h> Defines the EST gate control list structure.



### Public Members

bool enable

Enable or disable EST

uint64\_t cycleTime

Base Time 32 bits seconds 32 bits nanoseconds

uint32\_t extTime

Cycle Time 32 bits seconds 32 bits nanoseconds

uint32\_t numEntries

Time Extension 32 bits seconds 32 bits nanoseconds

enet\_qos\_est\_gate\_op\_t \*opList

Number of entries

struct \_\_enet\_qos\_rxp\_config

#include <fsl\_enet\_qos.h> Defines the ENET\_QOS Rx parser configuration structure.

### Public Members

uint32\_t matchEnable

4-byte match data used for comparing with incoming packet

uint8\_t acceptFrame

When matchEnable is set to 1, the matchData is used for comparing

uint8\_t rejectFrame

When acceptFrame = 1 and data is matched, the frame will be sent to DMA channel

uint8\_t inverseMatch

When rejectFrame = 1 and data is matched, the frame will be dropped

uint8\_t nextControl

Inverse match

uint8\_t reserved

Next instruction indexing control

uint8\_t frameOffset

Reserved control fields

uint8\_t okIndex

Frame offset in the packet data to be compared for match, in terms of 4 bytes.

uint8\_t dmaChannel

Memory Index to be used next.

uint32\_t reserved2

The DMA channel enet\_qos\_rxp\_dma\_chn\_t used for receiving the frame when frame match and acceptFrame = 1

struct \_\_enet\_qos\_buffer\_config

#include <fsl\_enet\_qos.h> Defines the buffer descriptor configure structure.

---

### Note:

- The receive and transmit descriptor start address pointer and tail pointer must be word-aligned.
- The recommended minimum tx/rx ring length is 4.

- c. The tx/rx descriptor tail address shall be the address pointer to the address just after the end of the last last descriptor. because only the descriptors between the start address and the tail address will be used by DMA.
  - d. The descriptor address is the start address of all used contiguous memory. for example, the rxDescStartAddrAlign is the start address of rxRingLen contiguous descriptor memories for rx descriptor ring 0.
  - e. The “\*rxBufferstartAddr” is the first element of rxRingLen (2\*rxRingLen for double buffers) rx buffers. It means the \*rxBufferStartAddr is the rx buffer for the first descriptor the \*rxBufferStartAddr + 1 is the rx buffer for the second descriptor or the rx buffer for the second buffer in the first descriptor. so please make sure the rxBufferStartAddr is the address of a rxRingLen or 2\*rxRingLen array.
- 

### Public Members

uint8\_t rxRingLen

The length of receive buffer descriptor ring.

uint8\_t txRingLen

The length of transmit buffer descriptor ring.

enet\_qos\_tx\_bd\_struct\_t \*txDescStartAddrAlign

Aligned transmit descriptor start address.

enet\_qos\_tx\_bd\_struct\_t \*txDescTailAddrAlign

Aligned transmit descriptor tail address.

enet\_qos\_frame\_info\_t \*txDirtyStartAddr

Start address of the dirty tx frame information.

enet\_qos\_rx\_bd\_struct\_t \*rxDescStartAddrAlign

Aligned receive descriptor start address.

enet\_qos\_rx\_bd\_struct\_t \*rxDescTailAddrAlign

Aligned receive descriptor tail address.

uint32\_t \*rxBufferStartAddr

Start address of the rx buffers.

uint32\_t rxBuffSizeAlign

Aligned receive data buffer size.

bool rxBuffNeedMaintain

Whether receive data buffer need cache maintain.

struct \_\_enet\_qos\_cbs\_config

#include <fsl\_enet\_qos.h> Defines the CBS configuration for queue.

### Public Members

uint16\_t sendSlope

Send slope configuration.

uint16\_t idleSlope

Idle slope configuration.

uint32\_t highCredit

High credit.

uint32\_t lowCredit  
Low credit.

struct enet\_qos\_tx\_queue\_config  
*#include <fsl\_enet\_qos.h>* Defines the queue configuration structure.

### Public Members

enet\_qos\_queue\_mode\_t mode  
tx queue mode configuration.

uint32\_t weight  
Refer to the MTL TxQ Quantum Weight register.

uint32\_t priority  
Refer to Transmit Queue Priority Mapping register.

enet\_qos\_cbs\_config\_t \*cbsConfig  
CBS configuration if queue use AVB mode.

struct enet\_qos\_rx\_queue\_config  
*#include <fsl\_enet\_qos.h>* Defines the queue configuration structure.

### Public Members

enet\_qos\_queue\_mode\_t mode  
rx queue mode configuration.

uint8\_t mapChannel  
tx queue map dma channel.

uint32\_t priority  
Rx queue priority.

enet\_qos\_rx\_queue\_route\_t packetRoute  
Receive packet routing.

struct enet\_qos\_multiqueue\_config  
*#include <fsl\_enet\_qos.h>* Defines the configuration when multi-queue is used.

### Public Members

enet\_qos\_dma\_burstlen burstLen  
Burst len for the multi-queue.

uint8\_t txQueueUse  
Used Tx queue count.

enet\_qos\_mtl\_multiqueue\_txsche mtltxSche  
Transmit schedule for multi-queue.

enet\_qos\_queue\_tx\_config\_t txQueueConfig[ENET\_QOS\_DMA\_CH\_COUNT]  
Tx Queue configuration.

uint8\_t rxQueueUse  
Used Rx queue count.

enet\_qos\_mtl\_multiqueue\_rxsche mtlrxSche  
Receive schedule for multi-queue.

*enet\_qos\_queue\_rx\_config\_t* rxQueueConfig[ENET\_QOS\_DMA\_CH\_COUNT]

Rx Queue configuration.

struct *\_enet\_qos\_config*

*#include <fsl\_enet\_qos.h>* Defines the basic configuration structure for the ENET device.

---

**Note:** Default the signal queue is used so the “\*multiqueueCfg” is set default with NULL. Set the pointer with a valid configuration pointer if the multiple queues are required. If multiple queue is enabled, please make sure the buffer configuration for all are prepared also.

---

### Public Members

*uint16\_t* specialControl

The logic or of *enet\_qos\_special\_config\_t*

*enet\_qos\_multiqueue\_config\_t* \*multiqueueCfg

Use multi-queue.

*enet\_qos\_mii\_mode\_t* miiMode

MII mode.

*enet\_qos\_mii\_speed\_t* miiSpeed

MII Speed.

*enet\_qos\_mii\_duplex\_t* miiDuplex

MII duplex.

*uint16\_t* pauseDuration

Used in the tx flow control frame, only valid when *kENET\_QOS\_FlowControlEnable* is set.

*enet\_qos\_ptp\_config\_t* \*ptpConfig

PTP 1588 feature configuration

*uint32\_t* csrClock\_Hz

CSR clock frequency in HZ.

*enet\_qos\_rx\_alloc\_callback\_t* rxBuffAlloc

Callback to alloc memory, must be provided for zero-copy Rx.

*enet\_qos\_rx\_free\_callback\_t* rxBuffFree

Callback to free memory, must be provided for zero-copy Rx.

struct *\_enet\_qos\_tx\_bd\_ring*

*#include <fsl\_enet\_qos.h>* Defines the ENET transmit buffer descriptor ring/queue structure.

### Public Members

*enet\_qos\_tx\_bd\_struct\_t* \*txBdBase

Buffer descriptor base address pointer.

*uint16\_t* txGenIdx

tx generate index.

*uint16\_t* txConsumIdx

tx consume index.

volatile uint16\_t txDescUsed  
tx descriptor used number.

uint16\_t txRingLen  
tx ring length.

struct \_\_enet\_qos\_rx\_bd\_ring  
*#include <fsl\_enet\_qos.h>* Defines the ENET receive buffer descriptor ring/queue structure.

### Public Members

enet\_qos\_rx\_bd\_struct\_t \*rxBdBase  
Buffer descriptor base address pointer.

uint16\_t rxGenIdx  
The current available receive buffer descriptor pointer.

uint16\_t rxRingLen  
Receive ring length.

uint32\_t rxBuffSizeAlign  
Receive buffer size.

struct \_\_enet\_qos\_handle  
*#include <fsl\_enet\_qos.h>* Defines the ENET handler structure.

### Public Members

uint8\_t txQueueUse  
Used tx queue count.

uint8\_t rxQueueUse  
Used rx queue count.

bool doubleBuffEnable  
The double buffer is used in the descriptor.

bool rxintEnable  
Rx interrupt enabled.

bool rxMaintainEnable[ENET\_QOS\_DMA\_CH\_COUNT]  
Rx buffer cache maintain enabled.

enet\_qos\_rx\_bd\_ring\_t rxBdRing[ENET\_QOS\_DMA\_CH\_COUNT]  
Receive buffer descriptor.

enet\_qos\_tx\_bd\_ring\_t txBdRing[ENET\_QOS\_DMA\_CH\_COUNT]  
Transmit buffer descriptor.

enet\_qos\_tx\_dirty\_ring\_t txDirtyRing[ENET\_QOS\_DMA\_CH\_COUNT]  
Transmit dirty buffers addresses.

uint32\_t \*rxBufferStartAddr[ENET\_QOS\_DMA\_CH\_COUNT]  
Rx buffer start address for reInitialize.

enet\_qos\_callback\_t callback  
Callback function.

void \*userData  
Callback function parameter.

uint8\_t multicastCount[64]

Multicast collisions counter

enet\_qos\_rx\_alloc\_callback\_t rxBuffAlloc

Callback to alloc memory, must be provided for zero-copy Rx.

enet\_qos\_rx\_free\_callback\_t rxBuffFree

Callback to free memory, must be provided for zero-copy Rx.

struct \_\_enet\_qos\_state

#include <fsl\_enet\_qos.h> Defines the ENET state structure.

---

**Note:** The structure contains saved state for the instance. It could be stored in enet\_qos\_handle\_t, but that's used only with the transactional API.

---

### Public Members

enet\_qos\_mii\_mode\_t miiMode

MII mode.

struct \_\_enet\_qos\_buffer\_struct

#include <fsl\_enet\_qos.h> Defines the frame buffer structure.

### Public Members

void \*buffer

The buffer store the whole or partial frame.

uint16\_t length

The byte length of this buffer.

struct \_\_enet\_qos\_rx\_frame\_error

#include <fsl\_enet\_qos.h> Defines the Rx frame error structure.

### Public Members

bool rxDstAddrFilterErr

Destination Address Filter Fail.

bool rxSrcAddrFilterErr

SA Address Filter Fail.

bool rxDribbleErr

Dribble error.

bool rxReceiveErr

Receive error.

bool rxOverflowErr

Receive over flow.

bool rxWatchDogErr

Watch dog timeout.

bool rxGaintPacketErr

Receive gaint packet.

bool rxCrcErr

Receive CRC error.

struct \_\_enet\_qos\_rx\_frame\_attribute\_struct

*#include <fsl\_enet\_qos.h>*

### Public Members

bool isTsAvail

Rx frame timestamp is available or not.

enet\_qos\_ptp\_time\_t timestamp

The nanosecond part timestamp of this Rx frame.

struct \_\_enet\_qos\_rx\_frame\_struct

*#include <fsl\_enet\_qos.h>* Defines the Rx frame data structure.

### Public Members

enet\_qos\_buffer\_struct\_t \*rxBuffArray

Rx frame buffer structure.

uint16\_t totLen

Rx frame total length.

enet\_qos\_rx\_frame\_attribute\_t rxAttribute

Rx frame attribute structure.

enet\_qos\_rx\_frame\_error\_t rxFrameError

Rx frame error.

struct \_\_enet\_qos\_transfer\_stats

*#include <fsl\_enet\_qos.h>* Defines the ENET QOS transfer statistics structure.

### Public Members

uint32\_t statsRxFrameCount

Rx frame number.

uint32\_t statsRxCrcErr

Rx frame number with CRC error.

uint32\_t statsRxAlignErr

Rx frame number with alignment error.

uint32\_t statsRxLengthErr

Rx frame length field doesn't equal to packet size.

uint32\_t statsRxFifoOverflowErr

Rx FIFO overflow count.

uint32\_t statsTxFrameCount

Tx frame number.

uint32\_t statsTxFifoUnderRunErr

Tx FIFO underrun count.

## 2.7 FGPIO Driver

## 2.8 FlexCAN: Flex Controller Area Network Driver

### 2.9 FlexCAN Driver

`bool FLEXCAN_IsInstanceHasFDMode(CAN_Type *base)`

Determine whether the FlexCAN instance support CAN FD mode at run time.

---

**Note:** Use this API only if different soc parts share the SOC part name macro define. Otherwise, a different SOC part name can be used to determine at compile time whether the FlexCAN instance supports CAN FD mode or not. If need use this API to determine if CAN FD mode is supported, the FLEXCAN\_Init function needs to be executed first, and then call this API and use the return to value determines whether to supports CAN FD mode, if return true, continue calling FLEXCAN\_FDInit to enable CAN FD mode.

---

#### Parameters

- `base` – FlexCAN peripheral base address.

#### Returns

return TRUE if instance support CAN FD mode, FALSE if instance only support classic CAN (2.0) mode.

`uint32_t FLEXCAN_GetFDMailboxOffset(CAN_Type *base, uint8_t mbIdx)`

Get Mailbox offset number by dword.

This function gets the offset number of the specified mailbox. Mailbox is not consecutive between memory regions when payload is not 8 bytes so need to calculate the specified mailbox address. For example, in the first memory region, MB[0].CS address is 0x4002\_4080. For 32 bytes payload frame, the second mailbox is  $((1/12)*512 + 1*12*40)/4 = 10$ , meaning 10 dword after the 0x4002\_4080, which is actually the address of mailbox MB[1].CS.

#### Parameters

- `base` – FlexCAN peripheral base address.
- `mbIdx` – Mailbox index.

#### Returns

Mailbox address offset in word.

`status_t FLEXCAN_EnterFreezeMode(CAN_Type *base)`

Enter FlexCAN Freeze Mode.

This function makes the FlexCAN work under Freeze Mode.

#### Parameters

- `base` – FlexCAN peripheral base address.

#### Returns

`kStatus_Success` Enter Freeze Mode successful `kStatus_Timeout` Timeout when wait for Freeze Mode Acknowledge

`status_t FLEXCAN_ExitFreezeMode(CAN_Type *base)`

Exit FlexCAN Freeze Mode.

This function makes the FlexCAN leave Freeze Mode.

#### Parameters



- base – FlexCAN peripheral base address.

### Returns

kStatus\_Success Enter Freeze Mode successful kStatus\_Timeout Timeout when wait for Freeze Mode Acknowledge

```
uint32_t FLEXCAN_GetInstance(CAN_Type *base)
```

Get the FlexCAN instance from peripheral base address.

### Parameters

- base – FlexCAN peripheral base address.

### Returns

FlexCAN instance.

```
bool FLEXCAN_CalculateImprovedTimingValues(CAN_Type *base, uint32_t bitRate, uint32_t
                                           sourceClock_Hz, flexcan_timing_config_t
                                           *pTimingConfig)
```

Calculates the improved timing values by specific bit Rates for classical CAN.

This function use to calculates the Classical CAN timing values according to the given bit rate. The Calculated timing values will be set in CTRL1/CBT/ENCBT register. The calculation is based on the recommendation of the CiA 301 v4.2.0 and previous version document.

### Parameters

- base – FlexCAN peripheral base address.
- bitRate – The classical CAN speed in bps defined by user, should be less than or equal to 1Mbps.
- sourceClock\_Hz – The Source clock frequency in Hz.
- pTimingConfig – Pointer to the FlexCAN timing configuration structure.

### Returns

TRUE if timing configuration found, FALSE if failed to find configuration.

```
void FLEXCAN_Init(CAN_Type *base, const flexcan_config_t *pConfig, uint32_t sourceClock_Hz)
```

Initializes a FlexCAN instance.

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the flexcan\_config\_t parameters and how to call the FLEXCAN\_Init function by passing in these parameters.

```
flexcan_config_t flexcanConfig;
flexcanConfig.clkSrc      = kFLEXCAN_ClkSrc0;
flexcanConfig.bitRate     = 1000000U;
flexcanConfig.maxMbNum    = 16;
flexcanConfig.enableLoopBack = false;
flexcanConfig.enableSelfWakeup = false;
flexcanConfig.enableIndividMask = false;
flexcanConfig.enableDoze     = false;
flexcanConfig.disableSelfReception = false;
flexcanConfig.enableListenOnlyMode = false;
flexcanConfig.timingConfig  = timingConfig;
FLEXCAN_Init(CAN0, &flexcanConfig, 4000000UL);
```

### Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the user-defined configuration structure.
- sourceClock\_Hz – FlexCAN Protocol Engine clock source frequency in Hz.

```
bool FLEXCAN_FDCalculateImprovedTimingValues(CAN_Type *base, uint32_t bitRate, uint32_t
                                             bitRateFD, uint32_t sourceClock_Hz,
                                             flexcan_timing_config_t *pTimingConfig)
```

Calculates the improved timing values by specific bit rates for CANFD.

This function use to calculates the CANFD timing values according to the given nominal phase bit rate and data phase bit rate. The Calculated timing values will be set in CBT/ENCBT and FDCBT/EDCBT registers. The calculation is based on the recommendation of the CiA 1301 v1.0.0 document.

#### Parameters

- base – FlexCAN peripheral base address.
- bitRate – The CANFD bus control speed in bps defined by user.
- bitRateFD – The CAN FD data phase speed in bps defined by user. Equal to bitRate means disable bit rate switching.
- sourceClock\_Hz – The Source clock frequency in Hz.
- pTimingConfig – Pointer to the FlexCAN timing configuration structure.

#### Returns

TRUE if timing configuration found, FALSE if failed to find configuration

```
void FLEXCAN_FDInit(CAN_Type *base, const flexcan_config_t *pConfig, uint32_t
                   sourceClock_Hz, flexcan_mb_size_t dataSize, bool brs)
```

Initializes a FlexCAN instance.

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the flexcan\_config\_t parameters and how to call the FLEXCAN\_FDInit function by passing in these parameters.

```
flexcan_config_t flexcanConfig;
flexcanConfig.clkSrc      = kFLEXCAN_ClkSrc0;
flexcanConfig.bitRate     = 1000000U;
flexcanConfig.bitRateFD   = 2000000U;
flexcanConfig.maxMbNum    = 16;
flexcanConfig.enableLoopBack = false;
flexcanConfig.enableSelfWakeup = false;
flexcanConfig.enableIndividMask = false;
flexcanConfig.disableSelfReception = false;
flexcanConfig.enableListenOnlyMode = false;
flexcanConfig.enableDoze    = false;
flexcanConfig.timingConfig  = timingConfig;
FLEXCAN_FDInit(CAN0, &flexcanConfig, 8000000UL, kFLEXCAN_16BperMB, true);
```

#### Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the user-defined configuration structure.
- sourceClock\_Hz – FlexCAN Protocol Engine clock source frequency in Hz.
- dataSize – FlexCAN Message Buffer payload size. The actual transmitted or received CAN FD frame data size needs to be less than or equal to this value.
- brs – True if bit rate switch is enabled in FD mode.

```
void FLEXCAN_Deinit(CAN_Type *base)
```

De-initializes a FlexCAN instance.

This function disables the FlexCAN module clock and sets all register values to the reset value.

### Parameters

- base – FlexCAN peripheral base address.

void FLEXCAN\_GetDefaultConfig(*flexcan\_config\_t* \*pConfig)

Gets the default configuration structure.

This function initializes the FlexCAN configuration structure to default values. The default values are as follows. `flexcanConfig->clkSrc = kFLEXCAN_ClkSrc0; flexcanConfig->bitRate = 1000000U; flexcanConfig->bitRateFD = 2000000U; flexcanConfig->maxMbNum = 16; flexcanConfig->enableLoopBack = false; flexcanConfig->enableSelfWakeup = false; flexcanConfig->enableIndividMask = false; flexcanConfig->disableSelfReception = false; flexcanConfig->enableListenOnlyMode = false; flexcanConfig->enableDoze = false; flexcanConfig->enablePretendedeNetworking = false; flexcanConfig->enableMemoryErrorControl = true; flexcanConfig->enableNonCorrectableErrorEnterFreeze = true; flexcanConfig->enableTransceiverDelayMeasure = true; flexcanConfig->enableRemoteRequestFrameStored = true; flexcanConfig->payloadEndianness = kFLEXCAN_bigEndian; flexcanConfig.timingConfig = timingConfig;`

### Parameters

- pConfig – Pointer to the FlexCAN configuration structure.

void FLEXCAN\_SetTimingConfig(CAN\_Type \*base, const *flexcan\_timing\_config\_t* \*pConfig)

Sets the FlexCAN classical CAN protocol timing characteristic.

This function gives user settings to classical CAN or CAN FD nominal phase timing characteristic. The function is for an experienced user. For less experienced users, call the FLEXCAN\_SetBitRate() instead.

---

**Note:** Calling FLEXCAN\_SetTimingConfig() overrides the bit rate set in FLEXCAN\_Init() or FLEXCAN\_SetBitRate().

---

### Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the timing configuration structure.

*status\_t* FLEXCAN\_SetBitRate(CAN\_Type \*base, uint32\_t sourceClock\_Hz, uint32\_t bitRate\_Bps)

Set bit rate of FlexCAN classical CAN frame or CAN FD frame nominal phase.

This function set the bit rate of classical CAN frame or CAN FD frame nominal phase base on FLEXCAN\_CalculateImprovedTimingValues() API calculated timing values.

---

**Note:** Calling FLEXCAN\_SetBitRate() overrides the bit rate set in FLEXCAN\_Init().

---

### Parameters

- base – FlexCAN peripheral base address.
- sourceClock\_Hz – Source Clock in Hz.
- bitRate\_Bps – Bit rate in Bps.

### Returns

kStatus\_Success - Set CAN baud rate (only Nominal phase) successfully.

```
void FLEXCAN_SetFDTimingConfig(CAN_Type *base, const flexcan_timing_config_t *pConfig)
```

Sets the FlexCAN CANFD data phase timing characteristic.

This function gives user settings to CANFD data phase timing characteristic. The function is for an experienced user. For less experienced users, call the FLEXCAN\_SetFDBitRate() to set both Nominal/Data bit Rate instead.

---

**Note:** Calling FLEXCAN\_SetFDTimingConfig() overrides the data phase bit rate set in FLEXCAN\_FDInit()/FLEXCAN\_SetFDBitRate().

---

#### Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the timing configuration structure.

```
status_t FLEXCAN_SetFDBitRate(CAN_Type *base, uint32_t sourceClock_Hz, uint32_t  
                             bitRateN_Bps, uint32_t bitRateD_Bps)
```

Set bit rate of FlexCAN FD frame.

This function set the baud rate of FLEXCAN FD base on FLEXCAN\_FDCalculateImprovedTimingValues() API calculated timing values.

#### Parameters

- base – FlexCAN peripheral base address.
- sourceClock\_Hz – Source Clock in Hz.
- bitRateN\_Bps – Nominal bit Rate in Bps.
- bitRateD\_Bps – Data bit Rate in Bps.

#### Returns

kStatus\_Success - Set CAN FD bit rate (include Nominal and Data phase) successfully.

```
void FLEXCAN_SetRxMbGlobalMask(CAN_Type *base, uint32_t mask)
```

Sets the FlexCAN receive message buffer global mask.

This function sets the global mask for the FlexCAN message buffer in a matching process. The configuration is only effective when the Rx individual mask is disabled in the FLEXCAN\_Init().

#### Parameters

- base – FlexCAN peripheral base address.
- mask – Rx Message Buffer Global Mask value.

```
void FLEXCAN_SetRxFifoGlobalMask(CAN_Type *base, uint32_t mask)
```

Sets the FlexCAN receive FIFO global mask.

This function sets the global mask for FlexCAN FIFO in a matching process.

#### Parameters

- base – FlexCAN peripheral base address.
- mask – Rx Fifo Global Mask value.

```
void FLEXCAN_SetRxIndividualMask(CAN_Type *base, uint8_t maskIdx, uint32_t mask)
```

Sets the FlexCAN receive individual mask.

This function sets the individual mask for the FlexCAN matching process. The configuration is only effective when the Rx individual mask is enabled in the FLEXCAN\_Init(). If the Rx FIFO is disabled, the individual mask is applied to the corresponding Message Buffer. If the

Rx FIFO is enabled, the individual mask for Rx FIFO occupied Message Buffer is applied to the Rx Filter with the same index. Note that only the first 32 individual masks can be used as the Rx FIFO filter mask.

#### Parameters

- base – FlexCAN peripheral base address.
- maskIdx – The Index of individual Mask.
- mask – Rx Individual Mask value.

void FLEXCAN\_SetTxMbConfig(CAN\_Type \*base, uint8\_t mbIdx, bool enable)

Configures a FlexCAN transmit message buffer.

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

#### Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- enable – Enable/disable Tx Message Buffer.
  - true: Enable Tx Message Buffer.
  - false: Disable Tx Message Buffer.

void FLEXCAN\_SetRxMbConfig(CAN\_Type \*base, uint8\_t mbIdx, const flexcan\_rx\_mb\_config\_t \*pRxMbConfig, bool enable)

Configures a FlexCAN Receive Message Buffer.

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer. User should invoke this API when CTRL2[RRS]=1. When CTRL2[RRS]=1, frame's ID is compared to the IDs of the receive mailboxes with the CODE field configured as kFLEXCAN\_RxMbEmpty, kFLEXCAN\_RxMbFull or kFLEXCAN\_RxMbOverrun. Message buffer will store the remote frame in the same fashion of a data frame. No automatic remote response frame will be generated. User need to setup another message buffer to respond remote request.

#### Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pRxMbConfig – Pointer to the FlexCAN Message Buffer configuration structure.
- enable – Enable/disable Rx Message Buffer.
  - true: Enable Rx Message Buffer.
  - false: Disable Rx Message Buffer.

static inline void FLEXCAN\_SetMbID(CAN\_Type \*base, uint8\_t mbIdx, uint32\_t id)

Configures a FlexCAN Message Buffer identifier.

#### Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- id – CAN Message Buffer Identifier, should use FLEXCAN\_ID\_EXT() or FLEXCAN\_ID\_STD() macro.

```
void FLEXCAN_SetFDTxMbConfig(CAN_Type *base, uint8_t mbIdx, bool enable)
```

Configures a FlexCAN transmit message buffer.

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

#### Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- enable – Enable/disable Tx Message Buffer.
  - true: Enable Tx Message Buffer.
  - false: Disable Tx Message Buffer.

```
void FLEXCAN_SetFDRxMbConfig(CAN_Type *base, uint8_t mbIdx, const  
                             flexcan_rx_mb_config_t *pRxMbConfig, bool enable)
```

Configures a FlexCAN Receive Message Buffer.

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer.

#### Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pRxMbConfig – Pointer to the FlexCAN Message Buffer configuration structure.
- enable – Enable/disable Rx Message Buffer.
  - true: Enable Rx Message Buffer.
  - false: Disable Rx Message Buffer.

```
static inline void FLEXCAN_SetFDMbID(CAN_Type *base, uint8_t mbIdx, uint32_t id)
```

Configures a FlexCAN Message Buffer identifier.

#### Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- id – CAN Message Buffer Identifier; should use FLEXCAN\_ID\_EXT() or FLEXCAN\_ID\_STD() macro.

```
void FLEXCAN_SetRemoteResponseMbConfig(CAN_Type *base, uint8_t mbIdx, const  
                                       flexcan_frame_t *pFrame)
```

Configures a FlexCAN Remote Response Message Buffer.

User should invoke this API when CTRL2[RRS]=0. When CTRL2[RRS]=0, frame's ID is compared to the IDs of the receive mailboxes with the CODE field configured as kFLEXCAN\_RxMbRanswer. If there is a matching ID, then this mailbox content will be transmitted as response. The received remote request frame is not stored in receive buffer. It is only used to trigger a transmission of a frame in response.

#### Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pFrame – Pointer to CAN message frame structure for response.

```
void FLEXCAN_SetRxFifoConfig(CAN_Type *base, const flexcan_rx_fifo_config_t *pRxFifoConfig,
                             bool enable)
```

Configures the FlexCAN Legacy Rx FIFO.

This function configures the FlexCAN Rx FIFO with given configuration.

---

**Note:** Legacy Rx FIFO only can receive classic CAN message.

---

#### Parameters

- base – FlexCAN peripheral base address.
- pRxFifoConfig – Pointer to the FlexCAN Legacy Rx FIFO configuration structure. Can be NULL when enable parameter is false.
- enable – Enable/disable Legacy Rx FIFO.
  - true: Enable Legacy Rx FIFO.
  - false: Disable Legacy Rx FIFO.

```
void FLEXCAN_SetEnhancedRxFifoConfig(CAN_Type *base, const
                                     flexcan_enhanced_rx_fifo_config_t *pConfig, bool
                                     enable)
```

Configures the FlexCAN Enhanced Rx FIFO.

This function configures the Enhanced Rx FIFO with given configuration.

---

**Note:** Enhanced Rx FIFO support receive classic CAN or CAN FD messages, Legacy Rx FIFO and Enhanced Rx FIFO cannot be enabled at the same time.

---

#### Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the FlexCAN Enhanced Rx FIFO configuration structure. Can be NULL when enable parameter is false.
- enable – Enable/disable Enhanced Rx FIFO.
  - true: Enable Enhanced Rx FIFO.
  - false: Disable Enhanced Rx FIFO.

```
void FLEXCAN_SetPNConfig(CAN_Type *base, const flexcan_pn_config_t *pConfig)
```

Configures the FlexCAN Pretended Networking mode.

This function configures the FlexCAN Pretended Networking mode with given configuration.

#### Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the FlexCAN Rx FIFO configuration structure.

```
static inline uint64_t FLEXCAN_GetStatusFlags(CAN_Type *base)
```

Gets the FlexCAN module interrupt flags.

This function gets all FlexCAN status flags. The flags are returned as the logical OR value of the enumerators `_flexcan_flags`. To check the specific status, compare the return value with enumerators in `_flexcan_flags`.

#### Parameters

- base – FlexCAN peripheral base address.

**Returns**

FlexCAN status flags which are ORed by the enumerators in the `_flexcan_flags`.

```
static inline void FLEXCAN_ClearStatusFlags(CAN_Type *base, uint64_t mask)
```

Clears status flags with the provided mask.

This function clears the FlexCAN status flags with a provided mask. An automatically cleared flag can't be cleared by this function.

**Parameters**

- base – FlexCAN peripheral base address.
- mask – The status flags to be cleared, it is logical OR value of `_flexcan_flags`.

```
static inline void FLEXCAN_GetBusErrCount(CAN_Type *base, uint8_t *txErrBuf, uint8_t *rxErrBuf)
```

Gets the FlexCAN Bus Error Counter value.

This function gets the FlexCAN Bus Error Counter value for both Tx and Rx direction. These values may be needed in the upper layer error handling.

**Parameters**

- base – FlexCAN peripheral base address.
- txErrBuf – Buffer to store Tx Error Counter value.
- rxErrBuf – Buffer to store Rx Error Counter value.

```
static inline uint64_t FLEXCAN_GetMbStatusFlags(CAN_Type *base, uint64_t mask)
```

Gets the FlexCAN low 64 Message Buffer interrupt flags.

This function gets the interrupt flags of a given Message Buffers.

**Parameters**

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

**Returns**

The status of given Message Buffers.

```
static inline uint64_t FLEXCAN_GetHigh64MbStatusFlags(CAN_Type *base, uint64_t mask)
```

Gets the FlexCAN High 64 Message Buffer interrupt flags.

Valid only if the number of available MBs exceeds 64.

**Parameters**

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

**Returns**

The status of given Message Buffers.

```
static inline void FLEXCAN_ClearMbStatusFlags(CAN_Type *base, uint64_t mask)
```

Clears the FlexCAN low 64 Message Buffer interrupt flags.

This function clears the interrupt flags of a given Message Buffers.

**Parameters**

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.



```
static inline void FLEXCAN_ClearHigh64MbStatusFlags(CAN_Type *base, uint64_t mask)
```

Clears the FlexCAN High 64 Message Buffer interrupt flags.

Valid only if the number of available MBs exceeds 64.

#### Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

```
void FLEXCAN_GetMemoryErrorReportStatus(CAN_Type *base,  
                                         flexcan_memory_error_report_status_t  
                                         *errorStatus)
```

Gets the FlexCAN Memory Error Report registers status.

This function gets the FlexCAN Memory Error Report registers status.

#### Parameters

- base – FlexCAN peripheral base address.
- errorStatus – Pointer to FlexCAN Memory Error Report registers status structure.

```
static inline uint8_t FLEXCAN_GetPNMatchCount(CAN_Type *base)
```

Gets the FlexCAN Number of Matches when in Pretended Networking.

This function gets the number of times a given message has matched the predefined filtering criteria for ID and/or PL before a wakeup event.

#### Parameters

- base – FlexCAN peripheral base address.

#### Returns

The number of received wake up messages.

```
static inline uint32_t FLEXCAN_GetEnhancedFifoDataCount(CAN_Type *base)
```

Gets the number of FlexCAN Enhanced Rx FIFO available frames.

This function gets the number of CAN messages stored in the Enhanced Rx FIFO.

#### Parameters

- base – FlexCAN peripheral base address.

#### Returns

The number of available CAN messages stored in the Enhanced Rx FIFO.

```
static inline void FLEXCAN_EnableInterrupts(CAN_Type *base, uint64_t mask)
```

Enables FlexCAN interrupts according to the provided mask.

This function enables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see `_flexcan_interrupt_enable`.

#### Parameters

- base – FlexCAN peripheral base address.
- mask – The interrupts to enable. Logical OR of `_flexcan_interrupt_enable`.

```
static inline void FLEXCAN_DisableInterrupts(CAN_Type *base, uint64_t mask)
```

Disables FlexCAN interrupts according to the provided mask.

This function disables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see `_flexcan_interrupt_enable`.

#### Parameters

- base – FlexCAN peripheral base address.

- mask – The interrupts to disable. Logical OR of `_flexcan_interrupt_enable`.

`static inline void FLEXCAN_EnableMbInterrupts(CAN_Type *base, uint64_t mask)`

Enables FlexCAN low 64 Message Buffer interrupts.

This function enables the interrupts of given Message Buffers.

#### Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

`static inline void FLEXCAN_EnableHigh64MbInterrupts(CAN_Type *base, uint64_t mask)`

Enables FlexCAN high 64 Message Buffer interrupts.

Valid only if the number of available MBs exceeds 64.

#### Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

`static inline void FLEXCAN_DisableMbInterrupts(CAN_Type *base, uint64_t mask)`

Disables FlexCAN low 64 Message Buffer interrupts.

This function disables the interrupts of given Message Buffers.

#### Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

`static inline void FLEXCAN_DisableHigh64MbInterrupts(CAN_Type *base, uint64_t mask)`

Disables FlexCAN high 64 Message Buffer interrupts.

Valid only if the number of available MBs exceeds 64.

#### Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

`void FLEXCAN_EnableRxFifoDMA(CAN_Type *base, bool enable)`

Enables or disables the FlexCAN Rx FIFO DMA request.

This function enables or disables the DMA feature of FlexCAN build-in Rx FIFO.

#### Parameters

- base – FlexCAN peripheral base address.
- enable – true to enable, false to disable.

`static inline uintptr_t FLEXCAN_GetRxFifoHeadAddr(CAN_Type *base)`

Gets the Rx FIFO Head address.

This function returns the FlexCAN Rx FIFO Head address, which is mainly used for the DMA/eDMA use case.

#### Parameters

- base – FlexCAN peripheral base address.

#### Returns

FlexCAN Rx FIFO Head address.

static inline *status\_t* FLEXCAN\_Enable(CAN\_Type \*base, bool enable)

Enables or disables the FlexCAN module operation.

This function enables or disables the FlexCAN module.

#### Parameters

- base – FlexCAN base pointer.
- enable – true to enable, false to disable.

#### Returns

kStatus\_Success Enable FlexCAN module successful  
kStatus\_Timeout Timeout when wait for Low-Power Mode Acknowledge

*status\_t* FLEXCAN\_WriteTxMb(CAN\_Type \*base, uint8\_t mbIdx, const *flexcan\_frame\_t* \*pTxFrame)

Writes a FlexCAN Message to the Transmit Message Buffer.

This function writes a CAN Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN Message transmit. After that the function returns immediately.

#### Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN Message Buffer index.
- pTxFrame – Pointer to CAN message frame to be sent.

#### Return values

- kStatus\_Success – - Write Tx Message Buffer Successfully.
- kStatus\_Fail – - Tx Message Buffer is currently in use.

*status\_t* FLEXCAN\_ReadRxMb(CAN\_Type \*base, uint8\_t mbIdx, *flexcan\_frame\_t* \*pRxFrame)

Reads a FlexCAN Message from Receive Message Buffer.

This function reads a CAN message from a specified Receive Message Buffer. The function fills a receive CAN message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

#### Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN Message Buffer index.
- pRxFrame – Pointer to CAN message frame structure for reception.

#### Return values

- kStatus\_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus\_FLEXCAN\_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus\_Fail – - Rx Message Buffer is empty.

*status\_t* FLEXCAN\_WriteFDTxMb(CAN\_Type \*base, uint8\_t mbIdx, const *flexcan\_fd\_frame\_t* \*pTxFrame)

Writes a FlexCAN FD Message to the Transmit Message Buffer.

This function writes a CAN FD Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN FD Message transmit. After that the function returns immediately.

#### Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN FD Message Buffer index.
- pTxFrame – Pointer to CAN FD message frame to be sent.

**Return values**

- kStatus\_Success – - Write Tx Message Buffer Successfully.
- kStatus\_Fail – - Tx Message Buffer is currently in use.

*status\_t* FLEXCAN\_ReadFDRxMb(CAN\_Type \*base, uint8\_t mbIdx, *flexcan\_fd\_frame\_t* \*pRxFrame)

Reads a FlexCAN FD Message from Receive Message Buffer.

This function reads a CAN FD message from a specified Receive Message Buffer. The function fills a receive CAN FD message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

**Parameters**

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN FD Message Buffer index.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

**Return values**

- kStatus\_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus\_FLEXCAN\_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus\_Fail – - Rx Message Buffer is empty.

*status\_t* FLEXCAN\_ReadRxFifo(CAN\_Type \*base, *flexcan\_frame\_t* \*pRxFrame)

Reads a FlexCAN Message from Legacy Rx FIFO.

This function reads a CAN message from the FlexCAN Legacy Rx FIFO.

**Parameters**

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN message frame structure for reception.

**Return values**

- kStatus\_Success – - Read Message from Rx FIFO successfully.
- kStatus\_Fail – - Rx FIFO is not enabled.

*status\_t* FLEXCAN\_ReadEnhancedRxFifo(CAN\_Type \*base, *flexcan\_fd\_frame\_t* \*pRxFrame)

Reads a FlexCAN Message from Enhanced Rx FIFO.

This function reads a CAN or CAN FD message from the FlexCAN Enhanced Rx FIFO.

**Parameters**

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

**Return values**

- kStatus\_Success – - Read Message from Rx FIFO successfully.
- kStatus\_Fail – - Rx FIFO is not enabled.

*status\_t* FLEXCAN\_ReadPNWakeUpMB(CAN\_Type \*base, uint8\_t mbIdx, *flexcan\_frame\_t* \*pRxFrame)

Reads a FlexCAN Message from Wake Up MB.

This function reads a CAN message from the FlexCAN Wake up Message Buffers. There are four Wake up Message Buffers (WMBs) used to store incoming messages in Pretended Networking mode. The WMB index indicates the arrival order. The last message is stored in WMB3.

#### Parameters

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN message frame structure for reception.
- mbIdx – The FlexCAN Wake up Message Buffer index. Range in 0x0 ~ 0x3.

#### Return values

- kStatus\_Success – - Read Message from Wake up Message Buffer successfully.
- kStatus\_Fail – - Wake up Message Buffer has no valid content.

*status\_t* FLEXCAN\_TransferFDSendBlocking(CAN\_Type \*base, uint8\_t mbIdx, *flexcan\_fd\_frame\_t* \*pTxFrame)

Performs a polling send transaction on the CAN bus.

---

**Note:** A transfer handle does not need to be created before calling this API.

---

#### Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN FD Message Buffer index.
- pTxFrame – Pointer to CAN FD message frame to be sent.

#### Return values

- kStatus\_Success – - Write Tx Message Buffer Successfully.
- kStatus\_Fail – - Tx Message Buffer is currently in use.
- kStatus\_Timeout – - Failed to send frames within specific time.

*status\_t* FLEXCAN\_TransferFDReceiveBlocking(CAN\_Type \*base, uint8\_t mbIdx, *flexcan\_fd\_frame\_t* \*pRxFrame)

Performs a polling receive transaction on the CAN bus.

---

**Note:** A transfer handle does not need to be created before calling this API.

---

#### Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN FD Message Buffer index.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

#### Return values

- kStatus\_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus\_FLEXCAN\_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.

- kStatus\_Fail – Rx Message Buffer is empty.
- kStatus\_Timeout – Failed to receive frames within specific time.

*status\_t* FLEXCAN\_TransferFDSendNonBlocking(*CAN\_Type* \*base, *flexcan\_handle\_t* \*handle, *flexcan\_mb\_transfer\_t* \*pMbXfer)

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

#### Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN FD Message Buffer transfer structure. See the flexcan\_mb\_transfer\_t.

#### Return values

- kStatus\_Success – Start Tx Message Buffer sending process successfully.
- kStatus\_Fail – Write Tx Message Buffer failed.
- kStatus\_FLEXCAN\_TxBusy – Tx Message Buffer is in use.

*status\_t* FLEXCAN\_TransferFDReceiveNonBlocking(*CAN\_Type* \*base, *flexcan\_handle\_t* \*handle, *flexcan\_mb\_transfer\_t* \*pMbXfer)

Receives a message using IRQ.

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

#### Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN FD Message Buffer transfer structure. See the flexcan\_mb\_transfer\_t.

#### Return values

- kStatus\_Success – Start Rx Message Buffer receiving process successfully.
- kStatus\_FLEXCAN\_RxBusy – Rx Message Buffer is in use.

*void* FLEXCAN\_TransferFDAbortSend(*CAN\_Type* \*base, *flexcan\_handle\_t* \*handle, *uint8\_t* mbIdx)

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

#### Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN FD Message Buffer index.

*void* FLEXCAN\_TransferFDAbortReceive(*CAN\_Type* \*base, *flexcan\_handle\_t* \*handle, *uint8\_t* mbIdx)

Aborts the interrupt driven message receive process.

This function aborts the interrupt driven message receive process.

#### Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN FD Message Buffer index.

*status\_t* FLEXCAN\_TransferSendBlocking(CAN\_Type \*base, uint8\_t mbIdx, *flexcan\_frame\_t* \*pTxFrame)

Performs a polling send transaction on the CAN bus.

---

**Note:** A transfer handle does not need to be created before calling this API.

---

#### Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN Message Buffer index.
- pTxFrame – Pointer to CAN message frame to be sent.

#### Return values

- kStatus\_Success – - Write Tx Message Buffer Successfully.
- kStatus\_Fail – - Tx Message Buffer is currently in use.
- kStatus\_Timeout – - Failed to send frames within specific time.

*status\_t* FLEXCAN\_TransferReceiveBlocking(CAN\_Type \*base, uint8\_t mbIdx, *flexcan\_frame\_t* \*pRxFrame)

Performs a polling receive transaction on the CAN bus.

---

**Note:** A transfer handle does not need to be created before calling this API.

---

#### Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN Message Buffer index.
- pRxFrame – Pointer to CAN message frame structure for reception.

#### Return values

- kStatus\_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus\_FLEXCAN\_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus\_Fail – - Rx Message Buffer is empty.
- kStatus\_Timeout – - Failed to receive frames within specific time.

*status\_t* FLEXCAN\_TransferReceiveFifoBlocking(CAN\_Type \*base, *flexcan\_frame\_t* \*pRxFrame)

Performs a polling receive transaction from Legacy Rx FIFO on the CAN bus.

---

**Note:** A transfer handle does not need to be created before calling this API.

---

#### Parameters

- base – FlexCAN peripheral base pointer.
- pRxFrame – Pointer to CAN message frame structure for reception.

**Return values**

- kStatus\_Success -- Read Message from Rx FIFO successfully.
- kStatus\_Fail -- Rx FIFO is not enabled.
- kStatus\_Timeout -- Failed to receive frames within specific time.

*status\_t* FLEXCAN\_TransferReceiveEnhancedFifoBlocking(CAN\_Type \*base, *flexcan\_fd\_frame\_t* \*pRxFrame)

Performs a polling receive transaction from Enhanced Rx FIFO on the CAN bus.

---

**Note:** A transfer handle does not need to be created before calling this API.

---

**Parameters**

- base – FlexCAN peripheral base pointer.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

**Return values**

- kStatus\_Success -- Read Message from Rx FIFO successfully.
- kStatus\_Fail -- Rx FIFO is not enabled.
- kStatus\_Timeout -- Failed to receive frames within specific time.

*void* FLEXCAN\_TransferCreateHandle(CAN\_Type \*base, *flexcan\_handle\_t* \*handle, *flexcan\_transfer\_callback\_t* callback, *void* \*userData)

Initializes the FlexCAN handle.

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

**Parameters**

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- callback – The callback function.
- userData – The parameter of the callback function.

*status\_t* FLEXCAN\_TransferSendNonBlocking(CAN\_Type \*base, *flexcan\_handle\_t* \*handle, *flexcan\_mb\_transfer\_t* \*pMbXfer)

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

**Parameters**

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the *flexcan\_mb\_transfer\_t*.

**Return values**

- kStatus\_Success – Start Tx Message Buffer sending process successfully.
- kStatus\_Fail – Write Tx Message Buffer failed.
- kStatus\_FLEXCAN\_TxBusy – Tx Message Buffer is in use.



*status\_t* FLEXCAN\_\_TransferReceiveNonBlocking(CAN\_Type \*base, *flexcan\_handle\_t* \*handle, *flexcan\_mb\_transfer\_t* \*pMbXfer)

Receives a message using IRQ.

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

#### Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the *flexcan\_mb\_transfer\_t*.

#### Return values

- kStatus\_Success – Start Rx Message Buffer receiving process successfully.
- kStatus\_FLEXCAN\_RxBusy – Rx Message Buffer is in use.

*status\_t* FLEXCAN\_\_TransferReceiveFifoNonBlocking(CAN\_Type \*base, *flexcan\_handle\_t* \*handle, *flexcan\_fifo\_transfer\_t* \*pFifoXfer)

Receives a message from Rx FIFO using IRQ.

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

#### Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pFifoXfer – FlexCAN Rx FIFO transfer structure. See the *flexcan\_fifo\_transfer\_t*.

#### Return values

- kStatus\_Success – Start Rx FIFO receiving process successfully.
- kStatus\_FLEXCAN\_RxFifoBusy – Rx FIFO is currently in use.

*status\_t* FLEXCAN\_\_TransferGetReceiveFifoCount(CAN\_Type \*base, *flexcan\_handle\_t* \*handle, *size\_t* \*count)

Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.

#### Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

#### Return values

- kStatus\_InvalidArgument – count is Invalid.
- kStatus\_Success – Successfully return the count.

*status\_t* FLEXCAN\_\_TransferReceiveEnhancedFifoNonBlocking(CAN\_Type \*base, *flexcan\_handle\_t* \*handle, *flexcan\_fifo\_transfer\_t* \*pFifoXfer)

Receives a message from Enhanced Rx FIFO using IRQ.

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

**Parameters**

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pFifoXfer – FlexCAN Rx FIFO transfer structure. See the ref flex-can\_fifo\_transfer\_t.@

**Return values**

- kStatus\_Success – Start Rx FIFO receiving process successfully.
- kStatus\_FLEXCAN\_RxFifoBusy – Rx FIFO is currently in use.

```
static inline status_t FLEXCAN_TransferGetReceiveEnhancedFifoCount(CAN_Type *base,  
                                                                    flexcan_handle_t *handle,  
                                                                    size_t *count)
```

Gets the Enhanced Rx Fifo transfer status during a interrupt non-blocking receive.

**Parameters**

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

**Return values**

- kStatus\_InvalidArgument – count is Invalid.
- kStatus\_Success – Successfully return the count.

```
uint32_t FLEXCAN_GetTimeStamp(flexcan_handle_t *handle, uint8_t mbIdx)
```

Gets the detail index of Mailbox's Timestamp by handle.

Then function can only be used when calling non-blocking Data transfer (TX/RX) API, After TX/RX data transfer done (User can get the status by handler's callback function), we can get the detail index of Mailbox's timestamp by handle, Detail non-blocking data transfer API (TX/RX) contain. -FLEXCAN\_TransferSendNonBlocking -FLEXCAN\_TransferFDSendNonBlocking -FLEXCAN\_TransferReceiveNonBlocking -FLEXCAN\_TransferFDReceiveNonBlocking -FLEXCAN\_TransferReceiveFifoNonBlocking

**Parameters**

- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

**Return values**

the – index of mailbox 's timestamp stored in the handle.

```
static inline uint32_t FLEXCAN_GetHighResolutionTimeStamp(CAN_Type *base, uint8_t mbIdx)
```

```
void FLEXCAN_TransferAbortSend(CAN_Type *base, flexcan_handle_t *handle, uint8_t mbIdx)
```

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

**Parameters**

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

```
void FLEXCAN_TransferAbortReceive(CAN_Type *base, flexcan_handle_t *handle, uint8_t mbIdx)
```

Aborts the interrupt driven message receive process.

This function aborts the interrupt driven message receive process.

**Parameters**

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

```
void FLEXCAN_TransferAbortReceiveFifo(CAN_Type *base, flexcan_handle_t *handle)
```

Aborts the interrupt driven message receive from Rx FIFO process.

This function aborts the interrupt driven message receive from Rx FIFO process.

**Parameters**

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

```
void FLEXCAN_TransferAbortReceiveEnhancedFifo(CAN_Type *base, flexcan_handle_t *handle)
```

Aborts the interrupt driven message receive from Enhanced Rx FIFO process.

This function aborts the interrupt driven message receive from Enhanced Rx FIFO process.

**Parameters**

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

```
void FLEXCAN_TransferHandleIRQ(CAN_Type *base, flexcan_handle_t *handle)
```

FlexCAN IRQ handle function.

This function handles the FlexCAN Error, the Message Buffer, and the Rx FIFO IRQ request.

**Parameters**

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

```
void FLEXCAN_MbHandleIRQ(CAN_Type *base, flexcan_handle_t *handle, uint32_t startMbIdx, uint32_t endMbIdx)
```

FlexCAN Message Buffer IRQ handle function.

**Parameters**

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- startMbIdx – First Message Buffer to handle.
- endMbIdx – Last Message Buffer to handle.

```
void FLEXCAN_EhancedRxFifoHandleIRQ(CAN_Type *base, flexcan_handle_t *handle)
```

FlexCAN Enhanced Rx FIFO IRQ handle function.

**Parameters**

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN\_BusoffErrorHandleIRQ(CAN\_Type \*base, flexcan\_handle\_t \*handle)

FlexCAN Bus Off, Error and Warning IRQ handle function.

**Parameters**

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN\_PNWakeUpHandleIRQ(CAN\_Type \*base, flexcan\_handle\_t \*handle)

FlexCAN Pretended Networking Wake-up IRQ handle function.

**Parameters**

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN\_MemoryErrorHandleIRQ(CAN\_Type \*base, flexcan\_handle\_t \*handle)

FlexCAN Memory Error IRQ handle function.

**Parameters**

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

FSL\_FLEXCAN\_DRIVER\_VERSION

FlexCAN driver version.

FlexCAN transfer status.

*Values:*

enumerator kStatus\_FLEXCAN\_TxBusy

Tx Message Buffer is Busy.

enumerator kStatus\_FLEXCAN\_TxIdle

Tx Message Buffer is Idle.

enumerator kStatus\_FLEXCAN\_TxSwitchToRx

Remote Message is send out and Message buffer changed to Receive one.

enumerator kStatus\_FLEXCAN\_RxBusy

Rx Message Buffer is Busy.

enumerator kStatus\_FLEXCAN\_RxIdle

Rx Message Buffer is Idle.

enumerator kStatus\_FLEXCAN\_RxOverflow

Rx Message Buffer is Overflowed.

enumerator kStatus\_FLEXCAN\_RxFifoBusy

Rx Message FIFO is Busy.

enumerator kStatus\_FLEXCAN\_RxFifoIdle

Rx Message FIFO is Idle.

enumerator kStatus\_FLEXCAN\_RxFifoOverflow

Rx Message FIFO is overflowed.

enumerator kStatus\_FLEXCAN\_RxFifoWarning

Rx Message FIFO is almost overflowed.

enumerator kStatus\_FLEXCAN\_RxFifoDisabled  
Rx Message FIFO is disabled during reading.

enumerator kStatus\_FLEXCAN\_ErrorStatus  
FlexCAN Module Error and Status.

enumerator kStatus\_FLEXCAN\_WakeUp  
FlexCAN is waken up from STOP mode.

enumerator kStatus\_FLEXCAN\_UnHandled  
UnHandled Interrupt asserted.

enumerator kStatus\_FLEXCAN\_RxRemote  
Rx Remote Message Received in Mail box.

enumerator kStatus\_FLEXCAN\_RxFifoUnderflow  
Enhanced Rx Message FIFO is underflow.

enumerator kStatus\_FLEXCAN\_MemoryError  
FlexCAN Memory Error.

enum \_flexcan\_frame\_format  
FlexCAN frame format.

*Values:*

enumerator kFLEXCAN\_FrameFormatStandard  
Standard frame format attribute.

enumerator kFLEXCAN\_FrameFormatExtend  
Extend frame format attribute.

enum \_flexcan\_frame\_type  
FlexCAN frame type.

*Values:*

enumerator kFLEXCAN\_FrameTypeData  
Data frame type attribute.

enumerator kFLEXCAN\_FrameTypeRemote  
Remote frame type attribute.

enum \_flexcan\_clock\_source  
FlexCAN clock source.

*Deprecated:*

Do not use the kFLEXCAN\_ClkSrcOs. It has been superceded kFLEXCAN\_ClkSrc0

Do not use the kFLEXCAN\_ClkSrcPeri. It has been superceded kFLEXCAN\_ClkSrc1

*Values:*

enumerator kFLEXCAN\_ClkSrcOsc  
FlexCAN Protocol Engine clock from Oscillator.

enumerator kFLEXCAN\_ClkSrcPeri  
FlexCAN Protocol Engine clock from Peripheral Clock.

enumerator kFLEXCAN\_ClkSrc0  
FlexCAN Protocol Engine clock selected by user as SRC == 0.

enumerator kFLEXCAN\_ClkSrc1

FlexCAN Protocol Engine clock selected by user as SRC == 1.

enum \_flexcan\_wake\_up\_source

FlexCAN wake up source.

*Values:*

enumerator kFLEXCAN\_WakeupSrcUnfiltered

FlexCAN uses unfiltered Rx input to detect edge.

enumerator kFLEXCAN\_WakeupSrcFiltered

FlexCAN uses filtered Rx input to detect edge.

enum \_flexcan\_endianness

FlexCAN payload endianness.

*Values:*

enumerator kFLEXCAN\_bigEndian

Transmit frame with MSB first, receive frame with big-endian format.

enumerator kFLEXCAN\_littleEndian

Transmit frame with LSB first, receive frame with little-endian format.

enum \_flexcan\_MB\_timestamp\_base

FlexCAN timebase used for capturing 16-bit TIME\_STAMP field of message buffer.

*Values:*

enumerator kFLEXCAN\_CANTimer

FlexCAN free-running timer.

enumerator kFLEXCAN\_Lower16bitsHRTimer

Lower 16 bits of high-resolution on-chip timer.

enumerator kFLEXCAN\_Upper16bitsHRTimer

Upper 16 bits of high-resolution on-chip timer.

enum \_flexcan\_capture\_point

FlexCAN capture point of 32-bit high resolution timebase during a CAN frame.

*Values:*

enumerator kFLEXCAN\_CANFrameID2ndBit

Second bit of identifier field of any frame is on the CAN bus. HR\_TIME\_STAMPn register will not capture 32-bit counter value.

enumerator kFLEXCAN\_CANFrameEnd

End of the CAN frame.

enumerator kFLEXCAN\_CANFrameStart

Start of the CAN frame.

enumerator kFLEXCAN\_CANFDFrameRes

Start of frame for classical CAN frames; res bit for CAN FD frames.

enum \_flexcan\_rx\_fifo\_filter\_type

FlexCAN Rx Fifo Filter type.

*Values:*

enumerator kFLEXCAN\_RxFifoFilterTypeA

One full ID (standard and extended) per ID Filter element.

enumerator kFLEXCAN\_RxFifoFilterTypeB

Two full standard IDs or two partial 14-bit ID slices per ID Filter Table element.

enumerator kFLEXCAN\_RxFifoFilterTypeC

Four partial 8-bit Standard or extended ID slices per ID Filter Table element.

enumerator kFLEXCAN\_RxFifoFilterTypeD

All frames rejected.

enum \_flexcan\_mb\_size

FlexCAN Message Buffer Payload size.

*Values:*

enumerator kFLEXCAN\_8BperMB

Selects 8 bytes per Message Buffer.

enumerator kFLEXCAN\_16BperMB

Selects 16 bytes per Message Buffer.

enumerator kFLEXCAN\_32BperMB

Selects 32 bytes per Message Buffer.

enumerator kFLEXCAN\_64BperMB

Selects 64 bytes per Message Buffer.

enum \_flexcan\_fd\_frame\_length

FlexCAN CAN FD frame supporting data length (available DLC values).

For Tx, when the Data size corresponding to DLC value stored in the MB selected for transmission is larger than the MB Payload size, FlexCAN adds the necessary number of bytes with constant 0xCC pattern to complete the expected DLC. For Rx, when the Data size corresponding to DLC value received from the CAN bus is larger than the MB Payload size, the high order bytes that do not fit the Payload size will lose.

*Values:*

enumerator kFLEXCAN\_0BperFrame

Frame contains 0 valid data bytes.

enumerator kFLEXCAN\_1BperFrame

Frame contains 1 valid data bytes.

enumerator kFLEXCAN\_2BperFrame

Frame contains 2 valid data bytes.

enumerator kFLEXCAN\_3BperFrame

Frame contains 3 valid data bytes.

enumerator kFLEXCAN\_4BperFrame

Frame contains 4 valid data bytes.

enumerator kFLEXCAN\_5BperFrame

Frame contains 5 valid data bytes.

enumerator kFLEXCAN\_6BperFrame

Frame contains 6 valid data bytes.

enumerator kFLEXCAN\_7BperFrame

Frame contains 7 valid data bytes.

enumerator kFLEXCAN\_8BperFrame

Frame contains 8 valid data bytes.

enumerator kFLEXCAN\_12BperFrame  
Frame contains 12 valid data bytes.

enumerator kFLEXCAN\_16BperFrame  
Frame contains 16 valid data bytes.

enumerator kFLEXCAN\_20BperFrame  
Frame contains 20 valid data bytes.

enumerator kFLEXCAN\_24BperFrame  
Frame contains 24 valid data bytes.

enumerator kFLEXCAN\_32BperFrame  
Frame contains 32 valid data bytes.

enumerator kFLEXCAN\_48BperFrame  
Frame contains 48 valid data bytes.

enumerator kFLEXCAN\_64BperFrame  
Frame contains 64 valid data bytes.

enum \_flexcan\_efifo\_dma\_per\_read\_length  
FlexCAN Enhanced Rx Fifo DMA transfer per read length enumerations.

*Values:*

enumerator kFLEXCAN\_1WordPerRead  
Transfer 1 32-bit words (CS).

enumerator kFLEXCAN\_2WordPerRead  
Transfer 2 32-bit words (CS + ID).

enumerator kFLEXCAN\_3WordPerRead  
Transfer 3 32-bit words (CS + ID + 1~4 bytes data).

enumerator kFLEXCAN\_4WordPerRead  
Transfer 4 32-bit words (CS + ID + 5~8 bytes data).

enumerator kFLEXCAN\_5WordPerRead  
Transfer 5 32-bit words (CS + ID + 9~12 bytes data).

enumerator kFLEXCAN\_6WordPerRead  
Transfer 6 32-bit words (CS + ID + 13~16 bytes data).

enumerator kFLEXCAN\_7WordPerRead  
Transfer 7 32-bit words (CS + ID + 17~20 bytes data).

enumerator kFLEXCAN\_8WordPerRead  
Transfer 8 32-bit words (CS + ID + 21~24 bytes data).

enumerator kFLEXCAN\_9WordPerRead  
Transfer 9 32-bit words (CS + ID + 25~28 bytes data).

enumerator kFLEXCAN\_10WordPerRead  
Transfer 10 32-bit words (CS + ID + 29~32 bytes data).

enumerator kFLEXCAN\_11WordPerRead  
Transfer 11 32-bit words (CS + ID + 33~36 bytes data).

enumerator kFLEXCAN\_12WordPerRead  
Transfer 12 32-bit words (CS + ID + 37~40 bytes data).



enumerator kFLEXCAN\_13WordPerRead

Transfer 13 32-bit words (CS + ID + 41~44 bytes data).

enumerator kFLEXCAN\_14WordPerRead

Transfer 14 32-bit words (CS + ID + 45~48 bytes data).

enumerator kFLEXCAN\_15WordPerRead

Transfer 15 32-bit words (CS + ID + 49~52 bytes data).

enumerator kFLEXCAN\_16WordPerRead

Transfer 16 32-bit words (CS + ID + 53~56 bytes data).

enumerator kFLEXCAN\_17WordPerRead

Transfer 17 32-bit words (CS + ID + 57~60 bytes data).

enumerator kFLEXCAN\_18WordPerRead

Transfer 18 32-bit words (CS + ID + 61~64 bytes data).

enumerator kFLEXCAN\_19WordPerRead

Transfer 19 32-bit words (CS + ID + 64 bytes data + ID HIT).

enumerator kFLEXCAN\_20WordPerRead

Transfer 20 32-bit words (CS + ID + 64 bytes data + ID HIT + HR timestamp).

enum \_flexcan\_rx\_fifo\_priority

FlexCAN Enhanced/Legacy Rx FIFO priority.

The matching process starts from the Rx MB(or Enhanced/Legacy Rx FIFO) with higher priority. If no MB(or Enhanced/Legacy Rx FIFO filter) is satisfied, the matching process goes on with the Enhanced/Legacy Rx FIFO(or Rx MB) with lower priority.

*Values:*

enumerator kFLEXCAN\_RxFifoPrioLow

Matching process start from Rx Message Buffer first.

enumerator kFLEXCAN\_RxFifoPrioHigh

Matching process start from Enhanced/Legacy Rx FIFO first.

enum \_flexcan\_interrupt\_enable

FlexCAN interrupt enable enumerations.

This provides constants for the FlexCAN interrupt enable enumerations for use in the FlexCAN functions.

---

**Note:** FlexCAN Message Buffers and Legacy Rx FIFO interrupts not included in.

---

*Values:*

enumerator kFLEXCAN\_BusOffInterruptEnable

Bus Off interrupt, use bit 15.

enumerator kFLEXCAN\_ErrorInterruptEnable

CAN Error interrupt, use bit 14.

enumerator kFLEXCAN\_TxWarningInterruptEnable

Tx Warning interrupt, use bit 11.

enumerator kFLEXCAN\_RxWarningInterruptEnable

Rx Warning interrupt, use bit 10.

enumerator kFLEXCAN\_FDErrorInterruptEnable

CAN FD Error interrupt, use bit 31.

enumerator kFLEXCAN\_PNMatchWakeUpInterruptEnable

PN Match Wake Up interrupt, use high word bit 17.

enumerator kFLEXCAN\_PNTimeoutWakeUpInterruptEnable

PN Timeout Wake Up interrupt, use high word bit 16. Enhanced Rx FIFO Underflow interrupt, use high word bit 31.

enumerator kFLEXCAN\_ERxFifoUnderflowInterruptEnable

Enhanced Rx FIFO Overflow interrupt, use high word bit 30.

enumerator kFLEXCAN\_ERxFifoOverflowInterruptEnable

Enhanced Rx FIFO Watermark interrupt, use high word bit 29.

enumerator kFLEXCAN\_ERxFifoWatermarkInterruptEnable

Enhanced Rx FIFO Data Available interrupt, use high word bit 28.

enumerator kFLEXCAN\_ERxFifoDataAvlInterruptEnable

enumerator kFLEXCAN\_HostAccessNCErrInterruptEnable

Host Access With Non-Correctable Errors interrupt, use high word bit 0.

enumerator kFLEXCAN\_FlexCanAccessNCErrInterruptEnable

FlexCAN Access With Non-Correctable Errors interrupt, use high word bit 2.

enumerator kFLEXCAN\_HostOrFlexCanCErrInterruptEnable

Host or FlexCAN Access With Correctable Errors interrupt, use high word bit 3.

enum \_flexcan\_flags

FlexCAN status flags.

This provides constants for the FlexCAN status flags for use in the FlexCAN functions.

---

**Note:** The CPU read action clears the bits corresponding to the FLEXCAN\_ErrorFlag macro, therefore user need to read status flags and distinguish which error is occur using \_flexcan\_error\_flags enumerations.

---

*Values:*

enumerator kFLEXCAN\_ErrorOverrunFlag

Error Overrun Status.

enumerator kFLEXCAN\_FDErrorIntFlag

CAN FD Error Interrupt Flag.

enumerator kFLEXCAN\_BusoffDoneIntFlag

Bus Off process completed Interrupt Flag.

enumerator kFLEXCAN\_SynchFlag

CAN Synchronization Status.

enumerator kFLEXCAN\_TxWarningIntFlag

Tx Warning Interrupt Flag.

enumerator kFLEXCAN\_RxWarningIntFlag

Rx Warning Interrupt Flag.

enumerator kFLEXCAN\_IdleFlag

FlexCAN In IDLE Status.

enumerator kFLEXCAN\_FaultConfinementFlag  
FlexCAN Fault Confinement State.

enumerator kFLEXCAN\_TransmittingFlag  
FlexCAN In Transmission Status.

enumerator kFLEXCAN\_ReceivingFlag  
FlexCAN In Reception Status.

enumerator kFLEXCAN\_BusOffIntFlag  
Bus Off Interrupt Flag.

enumerator kFLEXCAN\_ErrorIntFlag  
CAN Error Interrupt Flag.

enumerator kFLEXCAN\_ErrorFlag

enumerator kFLEXCAN\_PNMatchIntFlag  
PN Matching Event Interrupt Flag.

enumerator kFLEXCAN\_PNTimeoutIntFlag  
PN Timeout Event Interrupt Flag.

enumerator kFLEXCAN\_ERxFifoUnderflowIntFlag  
Enhanced Rx FIFO underflow Interrupt Flag.

enumerator kFLEXCAN\_ERxFifoOverflowIntFlag  
Enhanced Rx FIFO overflow Interrupt Flag.

enumerator kFLEXCAN\_ERxFifoWatermarkIntFlag  
Enhanced Rx FIFO watermark Interrupt Flag.

enumerator kFLEXCAN\_ERxFifoDataAvlIntFlag  
Enhanced Rx FIFO data available Interrupt Flag.

enumerator kFLEXCAN\_ERxFifoEmptyFlag  
Enhanced Rx FIFO empty status.

enumerator kFLEXCAN\_ERxFifoFullFlag  
Enhanced Rx FIFO full status.

enumerator kFLEXCAN\_HostAccessNonCorrectableErrorIntFlag  
Host Access With Non-Correctable Error Interrupt Flag.

enumerator kFLEXCAN\_FlexCanAccessNonCorrectableErrorIntFlag  
FlexCAN Access With Non-Correctable Error Interrupt Flag.

enumerator kFLEXCAN\_CorrectableErrorIntFlag  
Correctable Error Interrupt Flag.

enumerator kFLEXCAN\_HostAccessNonCorrectableErrorOverrunFlag  
Host Access With Non-Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN\_FlexCanAccessNonCorrectableErrorOverrunFlag  
FlexCAN Access With Non-Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN\_CorrectableErrorOverrunFlag  
Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN\_AllMemoryErrorIntFlag  
All Memory Error Interrupt Flags.

enumerator kFLEXCAN\_AllMemoryErrorFlag

All Memory Error Flags.

enum \_flexcan\_error\_flags

FlexCAN error status flags.

The FlexCAN Error Status enumerations is used to report current error of the FlexCAN bus. This enumerations should be used with KFLEXCAN\_ErrorFlag in \_flexcan\_flags enumerations to determine which error is generated.

*Values:*

enumerator kFLEXCAN\_FDStuffingError

Stuffing Error.

enumerator kFLEXCAN\_FDFormError

Form Error.

enumerator kFLEXCAN\_FDCrcError

Cyclic Redundancy Check Error.

enumerator kFLEXCAN\_FDBit0Error

Unable to send dominant bit.

enumerator kFLEXCAN\_FDBit1Error

Unable to send recessive bit.

enumerator kFLEXCAN\_TxErrorWarningFlag

Tx Error Warning Status.

enumerator kFLEXCAN\_RxErrorWarningFlag

Rx Error Warning Status.

enumerator kFLEXCAN\_StuffingError

Stuffing Error.

enumerator kFLEXCAN\_FormError

Form Error.

enumerator kFLEXCAN\_CrcError

Cyclic Redundancy Check Error.

enumerator kFLEXCAN\_AckError

Received no ACK on transmission.

enumerator kFLEXCAN\_Bit0Error

Unable to send dominant bit.

enumerator kFLEXCAN\_Bit1Error

Unable to send recessive bit.

FlexCAN Legacy Rx FIFO status flags.

The FlexCAN Legacy Rx FIFO Status enumerations are used to determine the status of the Rx FIFO. Because Rx FIFO occupy the MB0 ~ MB7 (Rx Fifo filter also occupies more Message Buffer space), Rx FIFO status flags are mapped to the corresponding Message Buffer status flags.

*Values:*

enumerator kFLEXCAN\_RxFifoOverflowFlag

Rx FIFO overflow flag.

enumerator kFLEXCAN\_RxFifoWarningFlag  
Rx FIFO almost full flag.

enumerator kFLEXCAN\_RxFifoFrameAvlFlag  
Frames available in Rx FIFO flag.

enum \_flexcan\_memory\_error\_type  
FlexCAN Memory Error Type.

*Values:*

enumerator kFLEXCAN\_CorrectableError  
The memory error is correctable which means on bit error.

enumerator kFLEXCAN\_NonCorrectableError  
The memory error is non-correctable which means two bit errors.

enum \_flexcan\_memory\_access\_type  
FlexCAN Memory Access Type.

*Values:*

enumerator kFLEXCAN\_MoveOutFlexCanAccess  
The memory error was detected during move-out FlexCAN access.

enumerator kFLEXCAN\_MoveInAccess  
The memory error was detected during move-in FlexCAN access.

enumerator kFLEXCAN\_TxArbitrationAccess  
The memory error was detected during Tx Arbitration FlexCAN access.

enumerator kFLEXCAN\_RxMatchingAccess  
The memory error was detected during Rx Matching FlexCAN access.

enumerator kFLEXCAN\_MoveOutHostAccess  
The memory error was detected during Rx Matching Host (CPU) access.

enum \_flexcan\_byte\_error\_syndrome  
FlexCAN Memory Error Byte Syndrome.

*Values:*

enumerator kFLEXCAN\_NoError  
No bit error in this byte.

enumerator kFLEXCAN\_ParityBits0Error  
Parity bit 0 error in this byte.

enumerator kFLEXCAN\_ParityBits1Error  
Parity bit 1 error in this byte.

enumerator kFLEXCAN\_ParityBits2Error  
Parity bit 2 error in this byte.

enumerator kFLEXCAN\_ParityBits3Error  
Parity bit 3 error in this byte.

enumerator kFLEXCAN\_ParityBits4Error  
Parity bit 4 error in this byte.

enumerator kFLEXCAN\_DataBits0Error  
Data bit 0 error in this byte.

enumerator kFLEXCAN\_DataBits1Error  
Data bit 1 error in this byte.

enumerator kFLEXCAN\_DataBits2Error  
Data bit 2 error in this byte.

enumerator kFLEXCAN\_DataBits3Error  
Data bit 3 error in this byte.

enumerator kFLEXCAN\_DataBits4Error  
Data bit 4 error in this byte.

enumerator kFLEXCAN\_DataBits5Error  
Data bit 5 error in this byte.

enumerator kFLEXCAN\_DataBits6Error  
Data bit 6 error in this byte.

enumerator kFLEXCAN\_DataBits7Error  
Data bit 7 error in this byte.

enumerator kFLEXCAN\_AllZeroError  
All-zeros non-correctable error in this byte.

enumerator kFLEXCAN\_AllOneError  
All-ones non-correctable error in this byte.

enumerator kFLEXCAN\_NonCorrectableErrors  
Non-correctable error in this byte.

enum \_flexcan\_pn\_match\_source  
FlexCAN Pretended Networking match source selection.  
*Values:*

enumerator kFLEXCAN\_PNMatSrcID  
Message match with ID filtering.

enumerator kFLEXCAN\_PNMatSrcIDAndData  
Message match with ID filtering and payload filtering.

enum \_flexcan\_pn\_match\_mode  
FlexCAN Pretended Networking mode match type.  
*Values:*

enumerator kFLEXCAN\_PNMatModeEqual  
Match upon ID/Payload contents against an exact target value.

enumerator kFLEXCAN\_PNMatModeGreater  
Match upon an ID/Payload value greater than or equal to a specified target value.

enumerator kFLEXCAN\_PNMatModeSmaller  
Match upon an ID/Payload value smaller than or equal to a specified target value.

enumerator kFLEXCAN\_PNMatModeRange  
Match upon an ID/Payload value inside a range, greater than or equal to a specified lower limit, and smaller than or equal to a specified upper limit

typedef enum \_flexcan\_frame\_format flexcan\_frame\_format\_t  
FlexCAN frame format.

typedef enum \_flexcan\_frame\_type flexcan\_frame\_type\_t  
FlexCAN frame type.

typedef enum *\_flexcan\_clock\_source* flexcan\_clock\_source\_t  
FlexCAN clock source.

*Deprecated:*

Do not use the kFLEXCAN\_ClkSrcOs. It has been superceded kFLEXCAN\_ClkSrc0

Do not use the kFLEXCAN\_ClkSrcPeri. It has been superceded kFLEXCAN\_ClkSrc1

typedef enum *\_flexcan\_wake\_up\_source* flexcan\_wake\_up\_source\_t  
FlexCAN wake up source.

typedef enum *\_flexcan\_endianness* flexcan\_endianness\_t  
FlexCAN payload endianness.

typedef enum *\_flexcan\_MB\_timestamp\_base* flexcan\_MB\_timestamp\_base\_t  
FlexCAN timebase used for capturing 16-bit TIME\_STAMP field of message buffer.

typedef enum *\_flexcan\_capture\_point* flexcan\_capture\_point\_t  
FlexCAN capture point of 32-bit high resolution timebase during a CAN frame.

typedef enum *\_flexcan\_rx\_fifo\_filter\_type* flexcan\_rx\_fifo\_filter\_type\_t  
FlexCAN Rx Fifo Filter type.

typedef enum *\_flexcan\_mb\_size* flexcan\_mb\_size\_t  
FlexCAN Message Buffer Payload size.

typedef enum *\_flexcan\_efifo\_dma\_per\_read\_length* flexcan\_efifo\_dma\_per\_read\_length\_t  
FlexCAN Enhanced Rx Fifo DMA transfer per read length enumerations.

typedef enum *\_flexcan\_rx\_fifo\_priority* flexcan\_rx\_fifo\_priority\_t  
FlexCAN Enhanced/Legacy Rx FIFO priority.

The matching process starts from the Rx MB(or Enhanced/Legacy Rx FIFO) with higher priority. If no MB(or Enhanced/Legacy Rx FIFO filter) is satisfied, the matching process goes on with the Enhanced/Legacy Rx FIFO(or Rx MB) with lower priority.

typedef enum *\_flexcan\_memory\_error\_type* flexcan\_memory\_error\_type\_t  
FlexCAN Memory Error Type.

typedef enum *\_flexcan\_memory\_access\_type* flexcan\_memory\_access\_type\_t  
FlexCAN Memory Access Type.

typedef enum *\_flexcan\_byte\_error\_syndrome* flexcan\_byte\_error\_syndrome\_t  
FlexCAN Memory Error Byte Syndrome.

typedef struct *\_flexcan\_memory\_error\_report\_status* flexcan\_memory\_error\_report\_status\_t  
FlexCAN memory error register status structure.

This structure contains the memory access properties that caused a memory error access. It is used as the parameter of FLEXCAN\_GetMemoryErrorReportStatus() function. And user can use FLEXCAN\_GetMemoryErrorReportStatus to get the status of the last memory error access.

typedef struct *\_flexcan\_frame* flexcan\_frame\_t  
FlexCAN message frame structure.

typedef struct *\_flexcan\_fd\_frame* flexcan\_fd\_frame\_t  
CAN FD message frame structure.

The CAN FD message supporting up to sixty four bytes can be used for a data frame, depending on the length selected for the message buffers. The length should be a enumeration member, see *\_flexcan\_fd\_frame\_length*.

typedef struct *flexcan\_timing\_config* flexcan\_timing\_config\_t

FlexCAN protocol timing characteristic configuration structure.

typedef struct *flexcan\_config* flexcan\_config\_t

FlexCAN module configuration structure.

*Deprecated:*

Do not use the baudRate. It has been superceded bitRate

Do not use the baudRateFD. It has been superceded bitRateFD

typedef struct *flexcan\_rx\_mb\_config* flexcan\_rx\_mb\_config\_t

FlexCAN Receive Message Buffer configuration structure.

This structure is used as the parameter of FLEXCAN\_SetRxMbConfig() function. The FLEXCAN\_SetRxMbConfig() function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

typedef enum *flexcan\_pn\_match\_source* flexcan\_pn\_match\_source\_t

FlexCAN Pretended Networking match source selection.

typedef enum *flexcan\_pn\_match\_mode* flexcan\_pn\_match\_mode\_t

FlexCAN Pretended Networking mode match type.

typedef struct *flexcan\_pn\_config* flexcan\_pn\_config\_t

FlexCAN Pretended Networking configuration structure.

This structure is used as the parameter of FLEXCAN\_SetPNConfig() function. The FLEXCAN\_SetPNConfig() function is used to configure FlexCAN Networking work mode.

typedef struct *flexcan\_rx\_fifo\_config* flexcan\_rx\_fifo\_config\_t

FlexCAN Legacy Rx FIFO configuration structure.

typedef struct *flexcan\_enhanced\_rx\_fifo\_std\_id\_filter* flexcan\_enhanced\_rx\_fifo\_std\_id\_filter\_t

FlexCAN Enhanced Rx FIFO Standard ID filter element structure.

typedef struct *flexcan\_enhanced\_rx\_fifo\_ext\_id\_filter* flexcan\_enhanced\_rx\_fifo\_ext\_id\_filter\_t

FlexCAN Enhanced Rx FIFO Extended ID filter element structure.

typedef struct *flexcan\_enhanced\_rx\_fifo\_config* flexcan\_enhanced\_rx\_fifo\_config\_t

FlexCAN Enhanced Rx FIFO configuration structure.

typedef struct *flexcan\_mb\_transfer* flexcan\_mb\_transfer\_t

FlexCAN Message Buffer transfer.

typedef struct *flexcan\_fifo\_transfer* flexcan\_fifo\_transfer\_t

FlexCAN Rx FIFO transfer.

typedef struct *flexcan\_handle* flexcan\_handle\_t

FlexCAN handle structure definition.

typedef void (\*flexcan\_transfer\_callback\_t)(CAN\_Type \*base, flexcan\_handle\_t \*handle, status\_t status, uint64\_t result, void \*userData)

FLEXCAN\_WAIT\_TIMEOUT

FLEXCAN\_POLLING\_TIMEOUT

Max loops to wait for polling transfer.

FLEXCAN\_MODULE\_TIMEOUT

Max loops to wait for FlexCAN register access complete.



DLC\_LENGTH\_DECODE(dlc)

FlexCAN frame length helper macro.

FLEXCAN\_ID\_STD(id)

FlexCAN Frame ID helper macro.

Standard Frame ID helper macro.

FLEXCAN\_ID\_EXT(id)

Extend Frame ID helper macro.

FLEXCAN\_RX\_MB\_STD\_MASK(id, rtr, ide)

FlexCAN Rx Message Buffer Mask helper macro.

Standard Rx Message Buffer Mask helper macro.

FLEXCAN\_RX\_MB\_EXT\_MASK(id, rtr, ide)

Extend Rx Message Buffer Mask helper macro.

FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_A(id, rtr, ide)

FlexCAN Legacy Rx FIFO Mask helper macro.

Standard Rx FIFO Mask helper macro Type A helper macro.

FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_B\_HIGH(id, rtr, ide)

Standard Rx FIFO Mask helper macro Type B upper part helper macro.

FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_B\_LOW(id, rtr, ide)

Standard Rx FIFO Mask helper macro Type B lower part helper macro.

FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_HIGH(id)

Standard Rx FIFO Mask helper macro Type C upper part helper macro.

FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_HIGH(id)

Standard Rx FIFO Mask helper macro Type C mid-upper part helper macro.

FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_LOW(id)

Standard Rx FIFO Mask helper macro Type C mid-lower part helper macro.

FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_LOW(id)

Standard Rx FIFO Mask helper macro Type C lower part helper macro.

FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_A(id, rtr, ide)

Extend Rx FIFO Mask helper macro Type A helper macro.

FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_B\_HIGH(id, rtr, ide)

Extend Rx FIFO Mask helper macro Type B upper part helper macro.

FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_B\_LOW(id, rtr, ide)

Extend Rx FIFO Mask helper macro Type B lower part helper macro.

FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_HIGH(id)

Extend Rx FIFO Mask helper macro Type C upper part helper macro.

FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_HIGH(id)

Extend Rx FIFO Mask helper macro Type C mid-upper part helper macro.

FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_LOW(id)

Extend Rx FIFO Mask helper macro Type C mid-lower part helper macro.

FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_LOW(id)

Extend Rx FIFO Mask helper macro Type C lower part helper macro.

`FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(id, rtr, ide)`

FlexCAN Rx FIFO Filter helper macro.

Standard Rx FIFO Filter helper macro Type A helper macro.

`FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH(id, rtr, ide)`

Standard Rx FIFO Filter helper macro Type B upper part helper macro.

`FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW(id, rtr, ide)`

Standard Rx FIFO Filter helper macro Type B lower part helper macro.

`FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH(id)`

Standard Rx FIFO Filter helper macro Type C upper part helper macro.

`FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH(id)`

Standard Rx FIFO Filter helper macro Type C mid-upper part helper macro.

`FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW(id)`

Standard Rx FIFO Filter helper macro Type C mid-lower part helper macro.

`FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW(id)`

Standard Rx FIFO Filter helper macro Type C lower part helper macro.

`FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A(id, rtr, ide)`

Extend Rx FIFO Filter helper macro Type A helper macro.

`FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH(id, rtr, ide)`

Extend Rx FIFO Filter helper macro Type B upper part helper macro.

`FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW(id, rtr, ide)`

Extend Rx FIFO Filter helper macro Type B lower part helper macro.

`FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH(id)`

Extend Rx FIFO Filter helper macro Type C upper part helper macro.

`FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH(id)`

Extend Rx FIFO Filter helper macro Type C mid-upper part helper macro.

`FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW(id)`

Extend Rx FIFO Filter helper macro Type C mid-lower part helper macro.

`FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW(id)`

Extend Rx FIFO Filter helper macro Type C lower part helper macro.

`ENHANCED_RX_FIFO_FSCH(x)`

FlexCAN Enhanced Rx FIFO Filter and Mask helper macro.

`RTR_STD_HIGH(x)`

`RTR_STD_LOW(x)`

`RTR_EXT(x)`

`ID_STD_LOW(id)`

`ID_STD_HIGH(id)`

`ID_EXT(id)`

`FLEXCAN_ENHANCED_RX_FIFO_STD_MASK_AND_FILTER(id, rtr, id_mask, rtr_mask)`

Standard ID filter element with filter + mask scheme.

FLEXCAN\_ENHANCED\_RX\_FIFO\_STD\_FILTER\_WITH\_RANGE(id\_upper, rtr, id\_lower,  
rtr\_mask)

Standard ID filter element with filter range.

FLEXCAN\_ENHANCED\_RX\_FIFO\_STD\_TWO\_FILTERS(id1, rtr1, id2, rtr2)

Standard ID filter element with two filters without masks.

FLEXCAN\_ENHANCED\_RX\_FIFO\_EXT\_MASK\_AND\_FILTER\_LOW(id, rtr)

Extended ID filter element with filter + mask scheme low word.

FLEXCAN\_ENHANCED\_RX\_FIFO\_EXT\_MASK\_AND\_FILTER\_HIGH(id\_mask, rtr\_mask)

Extended ID filter element with filter + mask scheme high word.

FLEXCAN\_ENHANCED\_RX\_FIFO\_EXT\_FILTER\_WITH\_RANGE\_LOW(id\_upper, rtr)

Extended ID filter element with range scheme low word.

FLEXCAN\_ENHANCED\_RX\_FIFO\_EXT\_FILTER\_WITH\_RANGE\_HIGH(id\_lower, rtr\_mask)

Extended ID filter element with range scheme high word.

FLEXCAN\_ENHANCED\_RX\_FIFO\_EXT\_TWO\_FILTERS\_LOW(id2, rtr2)

Extended ID filter element with two filters without masks low word.

FLEXCAN\_ENHANCED\_RX\_FIFO\_EXT\_TWO\_FILTERS\_HIGH(id1, rtr1)

Extended ID filter element with two filters without masks high word.

FLEXCAN\_PN\_STD\_MASK(id, rtr)

FlexCAN Pretended Networking ID Mask helper macro.

Standard Rx Message Buffer Mask helper macro.

FLEXCAN\_PN\_EXT\_MASK(id, rtr)

Extend Rx Message Buffer Mask helper macro.

FLEXCAN\_PN\_INT\_MASK(x)

FlexCAN interrupt/status flag helper macro.

FLEXCAN\_PN\_INT\_UNMASK(x)

FLEXCAN\_PN\_STATUS\_MASK(x)

FLEXCAN\_PN\_STATUS\_UNMASK(x)

FLEXCAN\_EFIFO\_INT\_MASK(x)

FLEXCAN\_EFIFO\_INT\_UNMASK(x)

FLEXCAN\_EFIFO\_STATUS\_MASK(x)

FLEXCAN\_EFIFO\_STATUS\_UNMASK(x)

FLEXCAN\_MECR\_INT\_MASK(x)

FLEXCAN\_MECR\_INT\_UNMASK(x)

FLEXCAN\_MECR\_STATUS\_MASK(x)

FLEXCAN\_MECR\_STATUS\_UNMASK(x)

FLEXCAN\_ERROR\_AND\_STATUS\_INT\_FLAG

FLEXCAN\_PNWAKE\_UP\_FLAG

FLEXCAN\_WAKE\_UP\_FLAG

FLEXCAN\_MEMORY\_ERROR\_INT\_FLAG

FLEXCAN\_ENHANCED\_RX\_FIFO\_INT\_FLAG

FlexCAN Enhanced Rx FIFO base address helper macro.

E\_RX\_FIFO(base)

FLEXCAN\_CALLBACK(x)

FlexCAN transfer callback function.

The FlexCAN transfer callback returns a value from the underlying layer. If the status equals to `kStatus_FLEXCAN_ErrorStatus`, the result parameter is the Content of FlexCAN status register which can be used to get the working status(or error status) of FlexCAN module. If the status equals to other FlexCAN Message Buffer transfer status, the result is the index of Message Buffer that generate transfer event. If the status equals to other FlexCAN Message Buffer transfer status, the result is meaningless and should be Ignored.

struct `_flexcan_memory_error_report_status`

*#include <fsl\_flexcan.h>* FlexCAN memory error register status structure.

This structure contains the memory access properties that caused a memory error access. It is used as the parameter of `FLEXCAN_GetMemoryErrorReportStatus()` function. And user can use `FLEXCAN_GetMemoryErrorReportStatus` to get the status of the last memory error access.

### Public Members

*flexcan\_memory\_error\_type\_t* errorType

The type of memory error that giving rise to the report.

*flexcan\_memory\_access\_type\_t* accessType

The type of memory access that giving rise to the memory error.

*uint16\_t* accessAddress

The address where memory error detected.

*uint32\_t* errorData

The raw data word read from memory with error.

struct `_flexcan_frame`

*#include <fsl\_flexcan.h>* FlexCAN message frame structure.

struct `_flexcan_fd_frame`

*#include <fsl\_flexcan.h>* CAN FD message frame structure.

The CAN FD message supporting up to sixty four bytes can be used for a data frame, depending on the length selected for the message buffers. The length should be a enumeration member, see `_flexcan_fd_frame_length`.

### Public Members

*uint32\_t* idhit

---

**Note:** ID HIT offset is changed dynamically according to data length code (DLC), when DLC is 15, they will be located below. Using `FLEXCAN_FixEnhancedRxFifoFrameIdHit` API is recommended to ensure this idhit value is correct. CAN Enhanced Rx FIFO filter hit id (This value is only used in Enhanced Rx FIFO receive mode).

---

uint32\_t hrtimestamp

---

**Note:** HR timestamp offset is changed dynamically according to data length code (DLC). External 32-bit on-chip timer high-resolution timestamp.

---

struct \_flexcan\_timing\_config

*#include <fsl\_flexcan.h>* FlexCAN protocol timing characteristic configuration structure.

### Public Members

uint32\_t preDivider

Classic CAN or CAN FD nominal phase bit rate prescaler.

uint32\_t rJumpwidth

Classic CAN or CAN FD nominal phase Re-sync Jump Width.

uint32\_t phaseSeg1

Classic CAN or CAN FD nominal phase Segment 1.

uint32\_t phaseSeg2

Classic CAN or CAN FD nominal phase Segment 2.

uint32\_t propSeg

Classic CAN or CAN FD nominal phase Propagation Segment.

uint32\_t fpreDivider

CAN FD data phase bit rate prescaler.

uint32\_t frJumpwidth

CAN FD data phase Re-sync Jump Width.

uint32\_t fphaseSeg1

CAN FD data phase Phase Segment 1.

uint32\_t fphaseSeg2

CAN FD data phase Phase Segment 2.

uint32\_t fpropSeg

CAN FD data phase Propagation Segment.

struct \_flexcan\_config

*#include <fsl\_flexcan.h>* FlexCAN module configuration structure.

### Deprecated:

Do not use the baudRate. It has been superceded bitRate

Do not use the baudRateFD. It has been superceded bitRateFD

### Public Members

*flexcan\_clock\_source\_t* clkSrc

Clock source for FlexCAN Protocol Engine.

*flexcan\_wake\_up\_source\_t* wakeupSrc

Wake up source selection.

uint8\_t maxMbNum

The maximum number of Message Buffers used by user.

bool enableLoopBack

Enable or Disable Loop Back Self Test Mode.

bool enableTimerSync

Enable or Disable Timer Synchronization.

bool enableIndividMask

Enable or Disable Rx Individual Mask and Queue feature.

bool disableSelfReception

Enable or Disable Self Reflection.

bool enableListenOnlyMode

Enable or Disable Listen Only Mode.

bool enableDoze

Enable or Disable Doze Mode.

bool enablePretendedeNetworking

Enable or Disable the Pretended Networking mode.

bool enableMemoryErrorControl

Enable or Disable the memory errors detection and correction mechanism.

bool enableNonCorrectableErrorEnterFreeze

Enable or Disable Non-Correctable Errors In FlexCAN Access Put Device In Freeze Mode.

bool enableTransceiverDelayMeasure

Enable or Disable the transceiver delay measurement, when it is enabled, then the secondary sample point position is determined by the sum of the transceiver delay measurement plus the enhanced TDC offset.

bool enableRemoteRequestFrameStored

true: Store Remote Request Frame in the same fashion of data frame. false: Generate an automatic Remote Response Frame.

*flexcan\_endianness\_t* payloadEndianness

Selects the byte order for the payload of transmit and receive frames, see *flexcan\_endianness\_t*.

bool enableExternalTimeTick

true: External time tick clocks the free-running timer. false: FlexCAN bit clock clocks the free-running timer.

*flexcan\_MB\_timestamp\_base\_t* captureTimeBase

Timebase of message buffer 16-bit TIME\_STAMP field.

*flexcan\_capture\_point\_t* capturePoint

Point in time when 32-bit timebase is captured during CAN frame.

struct *\_flexcan\_rx\_mb\_config*

*#include <fsl\_flexcan.h>* FlexCAN Receive Message Buffer configuration structure.

This structure is used as the parameter of FLEXCAN\_SetRxMbConfig() function. The FLEXCAN\_SetRxMbConfig() function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

## Public Members

`uint32_t id`

CAN Message Buffer Frame Identifier, should be set using `FLEXCAN_ID_EXT()` or `FLEXCAN_ID_STD()` macro.

*`flexcan_frame_format_t`* format

CAN Frame Identifier format(Standard of Extend).

*`flexcan_frame_type_t`* type

CAN Frame Type(Data or Remote for classical CAN only).

`struct _flexcan_pn_config`

*`#include <fsl_flexcan.h>`* FlexCAN Pretended Networking configuration structure.

This structure is used as the parameter of `FLEXCAN_SetPNConfig()` function. The `FLEXCAN_SetPNConfig()` function is used to configure FlexCAN Networking work mode.

## Public Members

`bool enableTimeout`

Enable or Disable timeout event trigger wakeup.

`uint16_t timeoutValue`

The timeout value that generates a wakeup event, the counter timer is incremented based on 64 times the CAN Bit Time unit.

`bool enableMatch`

Enable or Disable match event trigger wakeup.

*`flexcan_pn_match_source_t`* matchSrc

Selects the match source (ID and/or data match) to trigger wakeup.

`uint8_t matchNum`

The number of times a given message must match the predefined ID and/or data before generating a wakeup event, range in 0x1 ~ 0xFF.

*`flexcan_pn_match_mode_t`* idMatchMode

The ID match type.

*`flexcan_pn_match_mode_t`* dataMatchMode

The data match type.

`uint32_t idLower`

The ID target values 1 which used either for ID match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in ID match “range detection”.

`uint32_t idUpper`

The ID target values 2 which used only as the upper limit value in ID match “range detection” or used to store the ID mask in “equal to”.

`uint8_t lengthLower`

The lower limit for length of data bytes which used only in data match “range detection”. Range in 0x0 ~ 0x8.

`uint8_t lengthUpper`

The upper limit for length of data bytes which used only in data match “range detection”. Range in 0x0 ~ 0x8.

struct `_flexcan_rx_fifo_config`

*#include <fsl\_flexcan.h>* FlexCAN Legacy Rx FIFO configuration structure.

### Public Members

uint32\_t \*idFilterTable

Pointer to the FlexCAN Legacy Rx FIFO identifier filter table.

uint8\_t idFilterNum

The FlexCAN Legacy Rx FIFO Filter elements quantity.

*flexcan\_rx\_fifo\_filter\_type\_t* idFilterType

The FlexCAN Legacy Rx FIFO Filter type.

*flexcan\_rx\_fifo\_priority\_t* priority

The FlexCAN Legacy Rx FIFO receive priority.

struct `_flexcan_enhanced_rx_fifo_std_id_filter`

*#include <fsl\_flexcan.h>* FlexCAN Enhanced Rx FIFO Standard ID filter element structure.

### Public Members

uint32\_t filterType

FlexCAN internal Free-Running Counter Time Stamp.

uint32\_t rtr1

CAN FD frame data length code (DLC), range see `_flexcan_fd_frame_length`, When the length <= 8, it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use `DLC_LENGTH_DECODE(length)` macro to get the number of valid data bytes.

uint32\_t std1

CAN Frame Type(DATA or REMOTE).

uint32\_t rtr2

CAN Frame Identifier(STD or EXT format).

uint32\_t std2

Substitute Remote request.

struct `_flexcan_enhanced_rx_fifo_ext_id_filter`

*#include <fsl\_flexcan.h>* FlexCAN Enhanced Rx FIFO Extended ID filter element structure.

### Public Members

uint32\_t filterType

FlexCAN internal Free-Running Counter Time Stamp.

uint32\_t rtr1

CAN FD frame data length code (DLC), range see `_flexcan_fd_frame_length`, When the length <= 8, it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use `DLC_LENGTH_DECODE(length)` macro to get the number of valid data bytes.

uint32\_t std1

CAN Frame Type(DATA or REMOTE).



uint32\_t rtr2

CAN Frame Identifier(STD or EXT format).

uint32\_t std2

Substitute Remote request.

struct `_flexcan_enhanced_rx_fifo_config`

`#include <fsl_flexcan.h>` FlexCAN Enhanced Rx FIFO configuration structure.

### Public Members

uint32\_t \*idFilterTable

Pointer to the FlexCAN Enhanced Rx FIFO identifier filter table, each table member occupies 32 bit word, table size should be equal to idFilterNum. There are two types of Enhanced Rx FIFO filter elements that can be stored in table : extended-ID filter element (1 word, occupies 1 table members) and standard-ID filter element (2 words, occupies 2 table members), the extended-ID filter element needs to be placed in front of the table.

uint8\_t idFilterPairNum

idFilterPairNum is the Enhanced Rx FIFO identifier filter element pair numbers, each pair of filter elements occupies 2 words and can consist of one extended ID filter element or two standard ID filter elements.

uint8\_t extendIdFilterNum

The number of extended ID filter element items in the FlexCAN enhanced Rx FIFO identifier filter table, each extended-ID filter element occupies 2 words, extendIdFilterNum need less than or equal to idFilterPairNum.

uint8\_t fifoWatermark

(fifoWatermark + 1) is the minimum number of CAN messages stored in the Enhanced RX FIFO which can trigger FIFO watermark interrupt or a DMA request.

*flexcan\_efifo\_dma\_per\_read\_length\_t* dmaPerReadLength

Define the length of each read of the Enhanced RX FIFO element by the DAM, see `_flexcan_fd_frame_length`.

*flexcan\_rx\_fifo\_priority\_t* priority

The FlexCAN Enhanced Rx FIFO receive priority.

struct `_flexcan_mb_transfer`

`#include <fsl_flexcan.h>` FlexCAN Message Buffer transfer.

### Public Members

*flexcan\_frame\_t* \*frame

The buffer of CAN Message to be transfer.

uint8\_t mbIdx

The index of Message buffer used to transfer Message.

struct `_flexcan_fifo_transfer`

`#include <fsl_flexcan.h>` FlexCAN Rx FIFO transfer.

### Public Members

*flexcan\_fd\_frame\_t* \*framefd

The buffer of CAN Message to be received from Enhanced Rx FIFO.

*flexcan\_frame\_t* \*frame

The buffer of CAN Message to be received from Legacy Rx FIFO.

size\_t frameNum

Number of CAN Message need to be received from Legacy or Enhanced Rx FIFO.

struct *\_flexcan\_handle*

*#include <fsl\_flexcan.h>* FlexCAN handle structure.

### Public Members

*flexcan\_transfer\_callback\_t* callback

Callback function.

void \*userData

FlexCAN callback function parameter.

*flexcan\_frame\_t* \*volatile mbFrameBuf[CAN\_WORD1\_COUNT]

The buffer for received CAN data from Message Buffers.

*flexcan\_fd\_frame\_t* \*volatile mbFDFrameBuf[CAN\_WORD1\_COUNT]

The buffer for received CAN FD data from Message Buffers.

*flexcan\_frame\_t* \*volatile rxFifoFrameBuf

The buffer for received CAN data from Legacy Rx FIFO.

*flexcan\_fd\_frame\_t* \*volatile rxFifoFDFrameBuf

The buffer for received CAN FD data from Enhanced Rx FIFO.

size\_t rxFifoFrameNum

The number of CAN messages remaining to be received from Legacy or Enhanced Rx FIFO.

size\_t rxFifoTransferTotalNum

Total CAN Message number need to be received from Legacy or Enhanced Rx FIFO.

volatile uint8\_t mbState[CAN\_WORD1\_COUNT]

Message Buffer transfer state.

volatile uint8\_t rxFifoState

Rx FIFO transfer state.

volatile uint32\_t timestamp[CAN\_WORD1\_COUNT]

Mailbox transfer timestamp.

struct byteStatus

### Public Members

bool byteIsRead

The byte n (0~3) was read or not. The type of error and which bit in byte (n) is affected by the error.

struct \_\_\_\_unnamed18\_\_\_\_

**Public Members**

uint32\_t timestamp  
FlexCAN internal Free-Running Counter Time Stamp.

uint32\_t length  
CAN frame data length in bytes (Range: 0~8).

uint32\_t type  
CAN Frame Type(DATA or REMOTE).

uint32\_t format  
CAN Frame Identifier(STD or EXT format).

uint32\_t \_\_pad0\_\_  
Reserved.

uint32\_t idhit  
CAN Rx FIFO filter hit id(This value is only used in Rx FIFO receive mode).

struct \_\_unnamed20\_\_

**Public Members**

uint32\_t id  
CAN Frame Identifier, should be set using FLEXCAN\_ID\_EXT() or FLEXCAN\_ID\_STD() macro.

uint32\_t \_\_pad0\_\_  
Reserved.

union \_\_unnamed22\_\_

**Public Members**

struct \_flexcan\_frame

struct \_flexcan\_frame

struct \_\_unnamed24\_\_

**Public Members**

uint32\_t dataWord0  
CAN Frame payload word0.

uint32\_t dataWord1  
CAN Frame payload word1.

struct \_\_unnamed26\_\_

**Public Members**

uint8\_t dataByte3  
CAN Frame payload byte3.

uint8\_t dataByte2  
CAN Frame payload byte2.

```
uint8_t dataByte1
    CAN Frame payload byte1.
uint8_t dataByte0
    CAN Frame payload byte0.
uint8_t dataByte7
    CAN Frame payload byte7.
uint8_t dataByte6
    CAN Frame payload byte6.
uint8_t dataByte5
    CAN Frame payload byte5.
uint8_t dataByte4
    CAN Frame payload byte4.
struct ____unnamed28____
```

### Public Members

```
uint32_t timestamp
    FlexCAN internal Free-Running Counter Time Stamp.
uint32_t length
    CAN FD frame data length code (DLC), range see _flexcan_fd_frame_length, When the
    length <= 8, it equal to the data length, otherwise the number of valid frame data is
    not equal to the length value. user can use DLC_LENGTH_DECODE(length) macro to
    get the number of valid data bytes.
uint32_t type
    CAN Frame Type(DATA only).
uint32_t format
    CAN Frame Identifier(STD or EXT format).
uint32_t srr
    Substitute Remote request.
uint32_t esi
    Error State Indicator.
uint32_t brs
    Bit Rate Switch.
uint32_t edl
    Extended Data Length.
struct ____unnamed30____
```

### Public Members

```
uint32_t id
    CAN Frame Identifier, should be set using FLEXCAN_ID_EXT() or FLEXCAN_ID_STD()
    macro.
uint32_t ____pad0____
    Reserved.
union ____unnamed32____
```

**Public Members**

struct \_\_flexcan\_fd\_frame

struct \_\_flexcan\_fd\_frame

struct \_\_\_\_unnamed34\_\_\_\_

**Public Members**

uint32\_t dataWord[16]

CAN FD Frame payload, 16 double word maximum.

struct \_\_\_\_unnamed36\_\_\_\_

**Public Members**

uint8\_t dataByte3

CAN Frame payload byte3.

uint8\_t dataByte2

CAN Frame payload byte2.

uint8\_t dataByte1

CAN Frame payload byte1.

uint8\_t dataByte0

CAN Frame payload byte0.

uint8\_t dataByte7

CAN Frame payload byte7.

uint8\_t dataByte6

CAN Frame payload byte6.

uint8\_t dataByte5

CAN Frame payload byte5.

uint8\_t dataByte4

CAN Frame payload byte4.

union \_\_\_\_unnamed38\_\_\_\_

**Public Members**

struct \_\_flexcan\_config

struct \_\_flexcan\_config

struct \_\_\_\_unnamed40\_\_\_\_

**Public Members**

uint32\_t baudRate

FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.

uint32\_t baudRateFD

FlexCAN FD bit rate in bps, for CANFD data phase.

struct \_\_\_\_unnamed42\_\_\_\_

**Public Members**

uint32\_t bitRate

FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.

uint32\_t bitRateFD

FlexCAN FD bit rate in bps, for CANFD data phase.

union \_\_\_\_unnamed44\_\_\_\_

**Public Members**

struct \_\_flexcan\_pn\_config

< The data target values 1 which used either for data match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in data match “range detection”.

struct \_\_flexcan\_pn\_config

struct \_\_\_\_unnamed48\_\_\_\_

< The data target values 1 which used either for data match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in data match “range detection”.

**Public Members**

uint32\_t lowerWord0

CAN Frame payload word0.

uint32\_t lowerWord1

CAN Frame payload word1.

struct \_\_\_\_unnamed50\_\_\_\_

**Public Members**

uint8\_t lowerByte3

CAN Frame payload byte3.

uint8\_t lowerByte2

CAN Frame payload byte2.

uint8\_t lowerByte1

CAN Frame payload byte1.

uint8\_t lowerByte0

CAN Frame payload byte0.

uint8\_t lowerByte7

CAN Frame payload byte7.

uint8\_t lowerByte6

CAN Frame payload byte6.

uint8\_t lowerByte5

CAN Frame payload byte5.

```
uint8_t lowerByte4
    CAN Frame payload byte4.
union __unnamed46__
```

### Public Members

```
struct __flexcan_pn_config
    < The data target values 2 which used only as the upper limit value in data match
    "range
    detection" or used to store the data mask in "equal to".
struct __flexcan_pn_config
struct __unnamed52__
    < The data target values 2 which used only as the upper limit value in data match "range
    detection" or used to store the data mask in "equal to".
```

### Public Members

```
uint32_t upperWord0
    CAN Frame payload word0.
uint32_t upperWord1
    CAN Frame payload word1.
struct __unnamed54__
```

### Public Members

```
uint8_t upperByte3
    CAN Frame payload byte3.
uint8_t upperByte2
    CAN Frame payload byte2.
uint8_t upperByte1
    CAN Frame payload byte1.
uint8_t upperByte0
    CAN Frame payload byte0.
uint8_t upperByte7
    CAN Frame payload byte7.
uint8_t upperByte6
    CAN Frame payload byte6.
uint8_t upperByte5
    CAN Frame payload byte5.
uint8_t upperByte4
    CAN Frame payload byte4.
```

## 2.10 FlexCAN eDMA Driver

```
void FLEXCAN_TransferCreateHandleEDMA(CAN_Type *base, flexcan_edma_handle_t *handle,
                                       flexcan_edma_transfer_callback_t callback, void
                                       *userData, edma_handle_t *rxFifoEdmaHandle)
```

Initializes the FlexCAN handle, which is used in transactional functions.

### Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan\_edma\_handle\_t structure.
- callback – The callback function.
- userData – The parameter of the callback function.
- rxFifoEdmaHandle – User-requested DMA handle for Rx FIFO DMA transfer.

```
void FLEXCAN_PrepareTransfConfiguration(CAN_Type *base, flexcan_fifo_transfer_t *pFifoXfer,
                                       edma_transfer_config_t *pEdmaConfig)
```

Prepares the eDMA transfer configuration for FLEXCAN Legacy RX FIFO.

This function prepares the eDMA transfer configuration structure according to FLEXCAN Legacy RX FIFO.

### Parameters

- base – FlexCAN peripheral base address.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see flexcan\_fifo\_transfer\_t.
- pEdmaConfig – The user configuration structure of type edma\_transfer\_t.

```
status_t FLEXCAN_StartTransferDatafromRxFIFO(CAN_Type *base, flexcan_edma_handle_t
                                             *handle, edma_transfer_config_t
                                             *pEdmaConfig)
```

Start Transfer Data from the FLEXCAN Legacy Rx FIFO using eDMA.

This function to Update edma transfer configuration and Start eDMA transfer

### Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan\_edma\_handle\_t structure.
- pEdmaConfig – The user configuration structure of type edma\_transfer\_t.

### Return values

- kStatus\_Success – if succeed, others failed.
- kStatus\_FLEXCAN\_RxFifoBusy – Previous transfer ongoing.

```
status_t FLEXCAN_TransferReceiveFifoEDMA(CAN_Type *base, flexcan_edma_handle_t *handle,
                                          flexcan_fifo_transfer_t *pFifoXfer)
```

Receives the CAN Message from the Legacy Rx FIFO using eDMA.

This function receives the CAN Message using eDMA. This is a non-blocking function, which returns right away. After the CAN Message is received, the receive callback function is called.

### Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan\_edma\_handle\_t structure.



- `pFifoXfer` – FlexCAN Rx FIFO EDMA transfer structure, see `flexcan_fifo_transfer_t`.

#### Return values

- `kStatus_Success` – if succeed, others failed.
- `kStatus_FLEXCAN_RxFifoBusy` – Previous transfer ongoing.

`status_t FLEXCAN_TransferGetReceiveFifoCountEMDA(CAN_Type *base, flexcan_edma_handle_t *handle, size_t *count)`

Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.

#### Parameters

- `base` – FlexCAN peripheral base address.
- `handle` – FlexCAN handle pointer.
- `count` – Number of CAN messages receive so far by the non-blocking transaction.

#### Return values

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

`void FLEXCAN_TransferAbortReceiveFifoEDMA(CAN_Type *base, flexcan_edma_handle_t *handle)`

Aborts the receive Legacy/Enhanced Rx FIFO process which used eDMA.

This function aborts the receive Legacy/Enhanced Rx FIFO process which used eDMA.

#### Parameters

- `base` – FlexCAN peripheral base address.
- `handle` – Pointer to `flexcan_edma_handle_t` structure.

`status_t FLEXCAN_TransferReceiveEnhancedFifoEDMA(CAN_Type *base, flexcan_edma_handle_t *handle, flexcan_fifo_transfer_t *pFifoXfer)`

Receives the CAN FD Message from the Enhanced Rx FIFO using eDMA.

This function receives the CAN FD Message using eDMA. This is a non-blocking function, which returns right away. After the CAN Message is received, the receive callback function is called.

#### Parameters

- `base` – FlexCAN peripheral base address.
- `handle` – Pointer to `flexcan_edma_handle_t` structure.
- `pFifoXfer` – FlexCAN Rx FIFO EDMA transfer structure, see `flexcan_fifo_transfer_t`.

#### Return values

- `kStatus_Success` – if succeed, others failed.
- `kStatus_FLEXCAN_RxFifoBusy` – Previous transfer ongoing.

`static inline status_t FLEXCAN_TransferGetReceiveEnhancedFifoCountEMDA(CAN_Type *base, flexcan_edma_handle_t *handle, size_t *count)`

Gets the Enhanced Rx Fifo transfer status during a interrupt non-blocking receive.

**Parameters**

- `base` – FlexCAN peripheral base address.
- `handle` – FlexCAN handle pointer.
- `count` – Number of CAN messages receive so far by the non-blocking transaction.

**Return values**

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

FSL\_FLEXCAN\_EDMA\_DRIVER\_VERSION

FlexCAN EDMA driver version.

`typedef struct flexcan_edma_handle flexcan_edma_handle_t`

`typedef void (*flexcan_edma_transfer_callback_t)(CAN_Type *base, flexcan_edma_handle_t *handle, status_t status, void *userData)`

FlexCAN transfer callback function.

`struct flexcan_edma_handle`

`#include <fsl_flexcan_edma.h>` FlexCAN eDMA handle.

**Public Members**

*flexcan\_edma\_transfer\_callback\_t* callback

Callback function.

`void *userData`

FlexCAN callback function parameter.

*edma\_handle\_t* \*rxFifoEdmaHandle

The EDMA handler for Rx FIFO.

`volatile uint8_t` rxFifoState

Rx FIFO transfer state.

`size_t` frameNum

The number of messages that need to be received.

*flexcan\_fd\_frame\_t* \*framefd

Point to the buffer of CAN Message to be received from Enhanced Rx FIFO.

## 2.11 FlexIO: FlexIO Driver

## 2.12 FlexIO Driver

`void FLEXIO_GetDefaultConfig(flexio_config_t *userConfig)`

Gets the default configuration to configure the FlexIO module. The configuration can used directly to call the FLEXIO\_Configure().

Example:

```
flexio_config_t config;
FLEXIO_GetDefaultConfig(&config);
```

**Parameters**

- userConfig – pointer to flexio\_config\_t structure

void FLEXIO\_Init(FLEXIO\_Type \*base, const flexio\_config\_t \*userConfig)

Configures the FlexIO with a FlexIO configuration. The configuration structure can be filled by the user or be set with default values by FLEXIO\_GetDefaultConfig().

**Example**

```
flexio_config_t config = {  
    .enableFlexio = true,  
    .enableInDoze = false,  
    .enableInDebug = true,  
    .enableFastAccess = false  
};  
FLEXIO_Configure(base, &config);
```

**Parameters**

- base – FlexIO peripheral base address
- userConfig – pointer to flexio\_config\_t structure

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

Gates the FlexIO clock. Call this API to stop the FlexIO clock.

---

**Note:** After calling this API, call the FLEXIO\_Init to use the FlexIO module.

---

**Parameters**

- base – FlexIO peripheral base address

uint32\_t FLEXIO\_GetInstance(FLEXIO\_Type \*base)

Get instance number for FLEXIO module.

**Parameters**

- base – FLEXIO peripheral base address.

void FLEXIO\_Reset(FLEXIO\_Type \*base)

Resets the FlexIO module.

**Parameters**

- base – FlexIO peripheral base address

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

Enables the FlexIO module operation.

**Parameters**

- base – FlexIO peripheral base address
- enable – true to enable, false to disable.

static inline uint32\_t FLEXIO\_ReadPinInput(FLEXIO\_Type \*base)

Reads the input data on each of the FlexIO pins.

**Parameters**

- base – FlexIO peripheral base address

**Returns**

FlexIO pin input data

```
static inline uint8_t FLEXIO_GetShifterState(FLEXIO_Type *base)
```

Gets the current state pointer for state mode use.

#### Parameters

- base – FlexIO peripheral base address

#### Returns

current State pointer

```
void FLEXIO_SetShifterConfig(FLEXIO_Type *base, uint8_t index, const flexio_shifter_config_t *shifterConfig)
```

Configures the shifter with the shifter configuration. The configuration structure covers both the SHIFTCTL and SHIFTCFG registers. To configure the shifter to the proper mode, select which timer controls the shifter to shift, whether to generate start bit/stop bit, and the polarity of start bit and stop bit.

#### Example

```
flexio_shifter_config_t config = {  
.timerSelect = 0,  
.timerPolarity = kFLEXIO_ShifterTimerPolarityOnPositive,  
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,  
.pinPolarity = kFLEXIO_PinActiveLow,  
.shifterMode = kFLEXIO_ShifterModeTransmit,  
.inputSource = kFLEXIO_ShifterInputFromPin,  
.shifterStop = kFLEXIO_ShifterStopBitHigh,  
.shifterStart = kFLEXIO_ShifterStartBitLow  
};  
FLEXIO_SetShifterConfig(base, &config);
```

#### Parameters

- base – FlexIO peripheral base address
- index – Shifter index
- shifterConfig – Pointer to flexio\_shifter\_config\_t structure

```
void FLEXIO_SetTimerConfig(FLEXIO_Type *base, uint8_t index, const flexio_timer_config_t *timerConfig)
```

Configures the timer with the timer configuration. The configuration structure covers both the TIMCTL and TIMCFG registers. To configure the timer to the proper mode, select trigger source for timer and the timer pin output and the timing for timer.

#### Example

```
flexio_timer_config_t config = {  
.triggerSelect = FLEXIO_TIMER_TRIGGER_SEL_SHIFToNSTAT(0),  
.triggerPolarity = kFLEXIO_TimerTriggerPolarityActiveLow,  
.triggerSource = kFLEXIO_TimerTriggerSourceInternal,  
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,  
.pinSelect = 0,  
.pinPolarity = kFLEXIO_PinActiveHigh,  
.timerMode = kFLEXIO_TimerModeDual8BitBaudBit,  
.timerOutput = kFLEXIO_TimerOutputZeroNotAffectedByReset,  
.timerDecrement = kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput,  
.timerReset = kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput,  
.timerDisable = kFLEXIO_TimerDisableOnTimerCompare,  
.timerEnable = kFLEXIO_TimerEnableOnTriggerHigh,  
.timerStop = kFLEXIO_TimerStopBitEnableOnTimerDisable,  
.timerStart = kFLEXIO_TimerStartBitEnabled  
};  
FLEXIO_SetTimerConfig(base, &config);
```

**Parameters**

- base – FlexIO peripheral base address
- index – Timer index
- timerConfig – Pointer to the flexio\_timer\_config\_t structure

```
static inline void FLEXIO_SetClockMode(FLEXIO_Type *base, uint8_t index,  
                                       flexio_timer_decrement_source_t clocksource)
```

This function set the value of the prescaler on flexio channels.

**Parameters**

- base – Pointer to the FlexIO simulated peripheral type.
- index – Timer index
- clocksource – Set clock value

```
static inline void FLEXIO_EnableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
```

Enables the shifter status interrupt. The interrupt generates when the corresponding SSF is set.

---

**Note:** For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

---

**Parameters**

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by  $(1 \ll \text{shifter index})$

```
static inline void FLEXIO_DisableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
```

Disables the shifter status interrupt. The interrupt won't generate when the corresponding SSF is set.

---

**Note:** For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

---

**Parameters**

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by  $(1 \ll \text{shifter index})$

```
static inline void FLEXIO_EnableShifterErrorInterrupts(FLEXIO_Type *base, uint32_t mask)
```

Enables the shifter error interrupt. The interrupt generates when the corresponding SEF is set.

---

**Note:** For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

---

**Parameters**

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by  $(1 \ll \text{shifter index})$

static inline void FLEXIO\_DisableShifterErrorInterrupts(FLEXIO\_Type \*base, uint32\_t mask)  
Disables the shifter error interrupt. The interrupt won't generate when the corresponding SEF is set.

---

**Note:** For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

---

#### Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by  $(1 \ll \text{shifter index})$

static inline void FLEXIO\_EnableTimerStatusInterrupts(FLEXIO\_Type \*base, uint32\_t mask)  
Enables the timer status interrupt. The interrupt generates when the corresponding SSF is set.

---

**Note:** For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using  $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

---

#### Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by  $(1 \ll \text{timer index})$

static inline void FLEXIO\_DisableTimerStatusInterrupts(FLEXIO\_Type \*base, uint32\_t mask)  
Disables the timer status interrupt. The interrupt won't generate when the corresponding SSF is set.

---

**Note:** For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using  $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

---

#### Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by  $(1 \ll \text{timer index})$

static inline uint32\_t FLEXIO\_GetShifterStatusFlags(FLEXIO\_Type \*base)  
Gets the shifter status flags.

#### Parameters

- base – FlexIO peripheral base address

#### Returns

Shifter status flags

static inline void FLEXIO\_ClearShifterStatusFlags(FLEXIO\_Type \*base, uint32\_t mask)  
Clears the shifter status flags.

---

**Note:** For clearing multiple shifter status flags, for example, two shifter status flags, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

---

#### Parameters

- base – FlexIO peripheral base address

- mask – The shifter status mask which can be calculated by  $(1 \ll \text{shifter index})$

static inline uint32\_t FLEXIO\_GetShifterErrorFlags(FLEXIO\_Type \*base)

Gets the shifter error flags.

#### Parameters

- base – FlexIO peripheral base address

#### Returns

Shifter error flags

static inline void FLEXIO\_ClearShifterErrorFlags(FLEXIO\_Type \*base, uint32\_t mask)

Clears the shifter error flags.

---

**Note:** For clearing multiple shifter error flags, for example, two shifter error flags, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

---

#### Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by  $(1 \ll \text{shifter index})$

static inline uint32\_t FLEXIO\_GetTimerStatusFlags(FLEXIO\_Type \*base)

Gets the timer status flags.

#### Parameters

- base – FlexIO peripheral base address

#### Returns

Timer status flags

static inline void FLEXIO\_ClearTimerStatusFlags(FLEXIO\_Type \*base, uint32\_t mask)

Clears the timer status flags.

---

**Note:** For clearing multiple timer status flags, for example, two timer status flags, can calculate the mask by using  $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

---

#### Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by  $(1 \ll \text{timer index})$

static inline void FLEXIO\_EnableShifterStatusDMA(FLEXIO\_Type \*base, uint32\_t mask, bool enable)

Enables/disables the shifter status DMA. The DMA request generates when the corresponding SSF is set.

---

**Note:** For multiple shifter status DMA enables, for example, calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

---

#### Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by  $(1 \ll \text{shifter index})$

- enable – True to enable, false to disable.

uint32\_t FLEXIO\_GetShifterBufferAddress(FLEXIO\_Type \*base, flexio\_shifter\_buffer\_type\_t type, uint8\_t index)

Gets the shifter buffer address for the DMA transfer usage.

#### Parameters

- base – FlexIO peripheral base address
- type – Shifter type of flexio\_shifter\_buffer\_type\_t
- index – Shifter index

#### Returns

Corresponding shifter buffer index

status\_t FLEXIO\_RegisterHandleIRQ(void \*base, void \*handle, flexio\_isr\_t isr)

Registers the handle and the interrupt handler for the FlexIO-simulated peripheral.

#### Parameters

- base – Pointer to the FlexIO simulated peripheral type.
- handle – Pointer to the handler for FlexIO simulated peripheral.
- isr – FlexIO simulated peripheral interrupt handler.

#### Return values

- kStatus\_Success – Successfully create the handle.
- kStatus\_OutOfRange – The FlexIO type/handle/ISR table out of range.

status\_t FLEXIO\_UnregisterHandleIRQ(void \*base)

Unregisters the handle and the interrupt handler for the FlexIO-simulated peripheral.

#### Parameters

- base – Pointer to the FlexIO simulated peripheral type.

#### Return values

- kStatus\_Success – Successfully create the handle.
- kStatus\_OutOfRange – The FlexIO type/handle/ISR table out of range.

static inline void FLEXIO\_ClearPortOutput(FLEXIO\_Type \*base, uint32\_t mask)

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

#### Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

static inline void FLEXIO\_SetPortOutput(FLEXIO\_Type \*base, uint32\_t mask)

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

#### Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

static inline void FLEXIO\_TogglePortOutput(FLEXIO\_Type \*base, uint32\_t mask)

Reverses the current output logic of the multiple FLEXIO pins.

#### Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask



static inline void FLEXIO\_PinWrite(FLEXIO\_Type \*base, uint32\_t pin, uint8\_t output)

Sets the output level of the FLEXIO pins to the logic 1 or 0.

**Parameters**

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.
- output – FLEXIO pin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

static inline void FLEXIO\_EnablePinOutput(FLEXIO\_Type \*base, uint32\_t pin)

Enables the FLEXIO output pin function.

**Parameters**

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

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

Reads the current input value of the FLEXIO pin.

**Parameters**

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

**Return values**

FLEXIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

static inline uint32\_t FLEXIO\_GetPinStatus(FLEXIO\_Type \*base, uint32\_t pin)

Gets the FLEXIO input pin status.

**Parameters**

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

**Return values**

FLEXIO – port input status

- 0: corresponding pin input capture no status.
- 1: corresponding pin input capture rising or falling edge.

static inline void FLEXIO\_SetPinLevel(FLEXIO\_Type \*base, uint8\_t pin, bool level)

Sets the FLEXIO output pin level.

**Parameters**

- base – FlexIO peripheral base address
- pin – FlexIO pin number.
- level – FlexIO output pin level to set, can be either 0 or 1.

static inline bool FLEXIO\_GetPinOverride(const FLEXIO\_Type \*const base, uint8\_t pin)

Gets the enabled status of a FLEXIO output pin.

**Parameters**

- base – FlexIO peripheral base address

- pin – FlexIO pin number.

**Return values**

FlexIO – port enabled status

- 0: corresponding output pin is in disabled state.
- 1: corresponding output pin is in enabled state.

static inline void FLEXIO\_ConfigPinOverride(FLEXIO\_Type \*base, uint8\_t pin, bool enabled)  
Enables or disables a FLEXIO output pin.

**Parameters**

- base – FlexIO peripheral base address
- pin – Flexio pin number.
- enabled – Enable or disable the FlexIO pin.

static inline void FLEXIO\_ClearPortStatus(FLEXIO\_Type \*base, uint32\_t mask)  
Clears the multiple FLEXIO input pins status.

**Parameters**

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

FSL\_FLEXIO\_DRIVER\_VERSION  
FlexIO driver version.

enum \_flexio\_timer\_trigger\_polarity  
Define time of timer trigger polarity.

*Values:*

enumerator kFLEXIO\_TimerTriggerPolarityActiveHigh  
Active high.

enumerator kFLEXIO\_TimerTriggerPolarityActiveLow  
Active low.

enum \_flexio\_timer\_trigger\_source  
Define type of timer trigger source.

*Values:*

enumerator kFLEXIO\_TimerTriggerSourceExternal  
External trigger selected.

enumerator kFLEXIO\_TimerTriggerSourceInternal  
Internal trigger selected.

enum \_flexio\_pin\_config  
Define type of timer/shifter pin configuration.

*Values:*

enumerator kFLEXIO\_PinConfigOutputDisabled  
Pin output disabled.

enumerator kFLEXIO\_PinConfigOpenDrainOrBidirection  
Pin open drain or bidirectional output enable.

enumerator kFLEXIO\_PinConfigBidirectionOutputData  
Pin bidirectional output data.

enumerator kFLEXIO\_PinConfigOutput

Pin output.

enum \_flexio\_pin\_polarity

Definition of pin polarity.

*Values:*

enumerator kFLEXIO\_PinActiveHigh

Active high.

enumerator kFLEXIO\_PinActiveLow

Active low.

enum \_flexio\_timer\_mode

Define type of timer work mode.

*Values:*

enumerator kFLEXIO\_TimerModeDisabled

Timer Disabled.

enumerator kFLEXIO\_TimerModeDual8BitBaudBit

Dual 8-bit counters baud/bit mode.

enumerator kFLEXIO\_TimerModeDual8BitPWM

Dual 8-bit counters PWM mode.

enumerator kFLEXIO\_TimerModeSingle16Bit

Single 16-bit counter mode.

enumerator kFLEXIO\_TimerModeDual8BitPWMLow

Dual 8-bit counters PWM Low mode.

enum \_flexio\_timer\_output

Define type of timer initial output or timer reset condition.

*Values:*

enumerator kFLEXIO\_TimerOutputOneNotAffectedByReset

Logic one when enabled and is not affected by timer reset.

enumerator kFLEXIO\_TimerOutputZeroNotAffectedByReset

Logic zero when enabled and is not affected by timer reset.

enumerator kFLEXIO\_TimerOutputOneAffectedByReset

Logic one when enabled and on timer reset.

enumerator kFLEXIO\_TimerOutputZeroAffectedByReset

Logic zero when enabled and on timer reset.

enum \_flexio\_timer\_decrement\_source

Define type of timer decrement.

*Values:*

enumerator kFLEXIO\_TimerDecSrcOnFlexIOClockShiftTimerOutput

Decrement counter on FlexIO clock, Shift clock equals Timer output.

enumerator kFLEXIO\_TimerDecSrcOnTriggerInputShiftTimerOutput

Decrement counter on Trigger input (both edges), Shift clock equals Timer output.

enumerator kFLEXIO\_TimerDecSrcOnPinInputShiftPinInput

Decrement counter on Pin input (both edges), Shift clock equals Pin input.

enumerator kFLEXIO\_\_TimerDecSrcOnTriggerInputShiftTriggerInput

Decrement counter on Trigger input (both edges), Shift clock equals Trigger input.

enum \_flexio\_timer\_reset\_condition

Define type of timer reset condition.

*Values:*

enumerator kFLEXIO\_\_TimerResetNever

Timer never reset.

enumerator kFLEXIO\_\_TimerResetOnTimerPinEqualToTimerOutput

Timer reset on Timer Pin equal to Timer Output.

enumerator kFLEXIO\_\_TimerResetOnTimerTriggerEqualToTimerOutput

Timer reset on Timer Trigger equal to Timer Output.

enumerator kFLEXIO\_\_TimerResetOnTimerPinRisingEdge

Timer reset on Timer Pin rising edge.

enumerator kFLEXIO\_\_TimerResetOnTimerTriggerRisingEdge

Timer reset on Trigger rising edge.

enumerator kFLEXIO\_\_TimerResetOnTimerTriggerBothEdge

Timer reset on Trigger rising or falling edge.

enum \_flexio\_timer\_disable\_condition

Define type of timer disable condition.

*Values:*

enumerator kFLEXIO\_\_TimerDisableNever

Timer never disabled.

enumerator kFLEXIO\_\_TimerDisableOnPreTimerDisable

Timer disabled on Timer N-1 disable.

enumerator kFLEXIO\_\_TimerDisableOnTimerCompare

Timer disabled on Timer compare.

enumerator kFLEXIO\_\_TimerDisableOnTimerCompareTriggerLow

Timer disabled on Timer compare and Trigger Low.

enumerator kFLEXIO\_\_TimerDisableOnPinBothEdge

Timer disabled on Pin rising or falling edge.

enumerator kFLEXIO\_\_TimerDisableOnPinBothEdgeTriggerHigh

Timer disabled on Pin rising or falling edge provided Trigger is high.

enumerator kFLEXIO\_\_TimerDisableOnTriggerFallingEdge

Timer disabled on Trigger falling edge.

enum \_flexio\_timer\_enable\_condition

Define type of timer enable condition.

*Values:*

enumerator kFLEXIO\_\_TimerEnabledAlways

Timer always enabled.

enumerator kFLEXIO\_\_TimerEnableOnPrevTimerEnable

Timer enabled on Timer N-1 enable.

enumerator kFLEXIO\_\_TimerEnableOnTriggerHigh  
Timer enabled on Trigger high.

enumerator kFLEXIO\_\_TimerEnableOnTriggerHighPinHigh  
Timer enabled on Trigger high and Pin high.

enumerator kFLEXIO\_\_TimerEnableOnPinRisingEdge  
Timer enabled on Pin rising edge.

enumerator kFLEXIO\_\_TimerEnableOnPinRisingEdgeTriggerHigh  
Timer enabled on Pin rising edge and Trigger high.

enumerator kFLEXIO\_\_TimerEnableOnTriggerRisingEdge  
Timer enabled on Trigger rising edge.

enumerator kFLEXIO\_\_TimerEnableOnTriggerBothEdge  
Timer enabled on Trigger rising or falling edge.

enum \_flexio\_timer\_stop\_bit\_condition  
Define type of timer stop bit generate condition.

*Values:*

enumerator kFLEXIO\_\_TimerStopBitDisabled  
Stop bit disabled.

enumerator kFLEXIO\_\_TimerStopBitEnableOnTimerCompare  
Stop bit is enabled on timer compare.

enumerator kFLEXIO\_\_TimerStopBitEnableOnTimerDisable  
Stop bit is enabled on timer disable.

enumerator kFLEXIO\_\_TimerStopBitEnableOnTimerCompareDisable  
Stop bit is enabled on timer compare and timer disable.

enum \_flexio\_timer\_start\_bit\_condition  
Define type of timer start bit generate condition.

*Values:*

enumerator kFLEXIO\_\_TimerStartBitDisabled  
Start bit disabled.

enumerator kFLEXIO\_\_TimerStartBitEnabled  
Start bit enabled.

enum \_flexio\_timer\_output\_state  
FlexIO as PWM channel output state.

*Values:*

enumerator kFLEXIO\_\_PwmLow  
The output state of PWM channel is low

enumerator kFLEXIO\_\_PwmHigh  
The output state of PWM channel is high

enum \_flexio\_shifter\_timer\_polarity  
Define type of timer polarity for shifter control.

*Values:*

enumerator kFLEXIO\_\_ShifterTimerPolarityOnPositive  
Shift on positive edge of shift clock.

enumerator kFLEXIO\_\_ShifterTimerPolarityOnNegative  
Shift on negative edge of shift clock.

enum \_flexio\_\_shifter\_\_mode

Define type of shifter working mode.

*Values:*

enumerator kFLEXIO\_\_ShifterDisabled  
Shifter is disabled.

enumerator kFLEXIO\_\_ShifterModeReceive  
Receive mode.

enumerator kFLEXIO\_\_ShifterModeTransmit  
Transmit mode.

enumerator kFLEXIO\_\_ShifterModeMatchStore  
Match store mode.

enumerator kFLEXIO\_\_ShifterModeMatchContinuous  
Match continuous mode.

enumerator kFLEXIO\_\_ShifterModeState  
SHIFTBUF contents are used for storing programmable state attributes.

enumerator kFLEXIO\_\_ShifterModeLogic  
SHIFTBUF contents are used for implementing programmable logic look up table.

enum \_flexio\_\_shifter\_\_input\_\_source

Define type of shifter input source.

*Values:*

enumerator kFLEXIO\_\_ShifterInputFromPin  
Shifter input from pin.

enumerator kFLEXIO\_\_ShifterInputFromNextShifterOutput  
Shifter input from Shifter N+1.

enum \_flexio\_\_shifter\_\_stop\_\_bit

Define of STOP bit configuration.

*Values:*

enumerator kFLEXIO\_\_ShifterStopBitDisable  
Disable shifter stop bit.

enumerator kFLEXIO\_\_ShifterStopBitLow  
Set shifter stop bit to logic low level.

enumerator kFLEXIO\_\_ShifterStopBitHigh  
Set shifter stop bit to logic high level.

enum \_flexio\_\_shifter\_\_start\_\_bit

Define type of START bit configuration.

*Values:*

enumerator kFLEXIO\_\_ShifterStartBitDisabledLoadDataOnEnable  
Disable shifter start bit, transmitter loads data on enable.

enumerator kFLEXIO\_\_ShifterStartBitDisabledLoadDataOnShift  
Disable shifter start bit, transmitter loads data on first shift.

enumerator kFLEXIO\_\_ShifterStartBitLow  
Set shifter start bit to logic low level.

enumerator kFLEXIO\_\_ShifterStartBitHigh  
Set shifter start bit to logic high level.

enum \_\_flexio\_shifter\_buffer\_type  
Define FlexIO shifter buffer type.

*Values:*

enumerator kFLEXIO\_\_ShifterBuffer  
Shifter Buffer N Register.

enumerator kFLEXIO\_\_ShifterBufferBitSwapped  
Shifter Buffer N Bit Byte Swapped Register.

enumerator kFLEXIO\_\_ShifterBufferByteSwapped  
Shifter Buffer N Byte Swapped Register.

enumerator kFLEXIO\_\_ShifterBufferBitByteSwapped  
Shifter Buffer N Bit Swapped Register.

enumerator kFLEXIO\_\_ShifterBufferNibbleByteSwapped  
Shifter Buffer N Nibble Byte Swapped Register.

enumerator kFLEXIO\_\_ShifterBufferHalfWordSwapped  
Shifter Buffer N Half Word Swapped Register.

enumerator kFLEXIO\_\_ShifterBufferNibbleSwapped  
Shifter Buffer N Nibble Swapped Register.

enum \_\_flexio\_gpio\_direction  
FLEXIO gpio direction definition.

*Values:*

enumerator kFLEXIO\_\_DigitalInput  
Set current pin as digital input

enumerator kFLEXIO\_\_DigitalOutput  
Set current pin as digital output

enum \_\_flexio\_pin\_input\_config  
FLEXIO gpio input config.

*Values:*

enumerator kFLEXIO\_\_InputInterruptDisabled  
Interrupt request is disabled.

enumerator kFLEXIO\_\_InputInterruptEnable  
Interrupt request is enable.

enumerator kFLEXIO\_\_FlagRisingEdgeEnable  
Input pin flag on rising edge.

enumerator kFLEXIO\_\_FlagFallingEdgeEnable  
Input pin flag on falling edge.

typedef enum \_\_flexio\_timer\_trigger\_polarity flexio\_timer\_trigger\_polarity\_t  
Define time of timer trigger polarity.

`typedef enum _flexio_timer_trigger_source flexio_timer_trigger_source_t`  
Define type of timer trigger source.

`typedef enum _flexio_pin_config flexio_pin_config_t`  
Define type of timer/shifter pin configuration.

`typedef enum _flexio_pin_polarity flexio_pin_polarity_t`  
Definition of pin polarity.

`typedef enum _flexio_timer_mode flexio_timer_mode_t`  
Define type of timer work mode.

`typedef enum _flexio_timer_output flexio_timer_output_t`  
Define type of timer initial output or timer reset condition.

`typedef enum _flexio_timer_decrement_source flexio_timer_decrement_source_t`  
Define type of timer decrement.

`typedef enum _flexio_timer_reset_condition flexio_timer_reset_condition_t`  
Define type of timer reset condition.

`typedef enum _flexio_timer_disable_condition flexio_timer_disable_condition_t`  
Define type of timer disable condition.

`typedef enum _flexio_timer_enable_condition flexio_timer_enable_condition_t`  
Define type of timer enable condition.

`typedef enum _flexio_timer_stop_bit_condition flexio_timer_stop_bit_condition_t`  
Define type of timer stop bit generate condition.

`typedef enum _flexio_timer_start_bit_condition flexio_timer_start_bit_condition_t`  
Define type of timer start bit generate condition.

`typedef enum _flexio_timer_output_state flexio_timer_output_state_t`  
FlexIO as PWM channel output state.

`typedef enum _flexio_shifter_timer_polarity flexio_shifter_timer_polarity_t`  
Define type of timer polarity for shifter control.

`typedef enum _flexio_shifter_mode flexio_shifter_mode_t`  
Define type of shifter working mode.

`typedef enum _flexio_shifter_input_source flexio_shifter_input_source_t`  
Define type of shifter input source.

`typedef enum _flexio_shifter_stop_bit flexio_shifter_stop_bit_t`  
Define of STOP bit configuration.

`typedef enum _flexio_shifter_start_bit flexio_shifter_start_bit_t`  
Define type of START bit configuration.

`typedef enum _flexio_shifter_buffer_type flexio_shifter_buffer_type_t`  
Define FlexIO shifter buffer type.

`typedef struct _flexio_config flexio_config_t`  
Define FlexIO user configuration structure.

`typedef struct _flexio_timer_config flexio_timer_config_t`  
Define FlexIO timer configuration structure.

`typedef struct _flexio_shifter_config flexio_shifter_config_t`  
Define FlexIO shifter configuration structure.



typedef enum *flexio\_gpio\_direction* flexio\_gpio\_direction\_t  
FLEXIO gpio direction definition.

typedef enum *flexio\_pin\_input\_config* flexio\_pin\_input\_config\_t  
FLEXIO gpio input config.

typedef struct *flexio\_gpio\_config* flexio\_gpio\_config\_t  
The FLEXIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, use inputConfig param. If configured as an output pin, use outputLogic.

typedef void (\*flexio\_isr\_t)(void \*base, void \*handle)  
typedef for FlexIO simulated driver interrupt handler.

FLEXIO\_Type \*const s\_flexioBases[]  
Pointers to flexio bases for each instance.

const clock\_ip\_name\_t s\_flexioClocks[]  
Pointers to flexio clocks for each instance.

void FLEXIO\_SetPinConfig(FLEXIO\_Type \*base, uint32\_t pin, *flexio\_gpio\_config\_t* \*config)  
Configure a FLEXIO pin used by the board.

To Config the FLEXIO PIN, define a pin configuration, as either input or output, in the user file. Then, call the FLEXIO\_SetPinConfig() function.

This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
flexio_gpio_config_t config =
{
    kFLEXIO_DigitalInput,
    0U,
    kFLEXIO_FlagRisingEdgeEnable | kFLEXIO_InputInterruptEnable,
}
Define a digital output pin configuration,
flexio_gpio_config_t config =
{
    kFLEXIO_DigitalOutput,
    0U,
    0U
}
```

### Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.
- config – FLEXIO pin configuration pointer.

FLEXIO\_TIMER\_TRIGGER\_SEL\_PININPUT(x)  
Calculate FlexIO timer trigger.

FLEXIO\_TIMER\_TRIGGER\_SEL\_SHIFThSTAT(x)

FLEXIO\_TIMER\_TRIGGER\_SEL\_TIMn(x)

struct flexio\_config\_  
*#include <fsl\_flexio.h>* Define FlexIO user configuration structure.

### Public Members

`bool enableFlexio`  
Enable/disable FlexIO module

`bool enableInDoze`  
Enable/disable FlexIO operation in doze mode

`bool enableInDebug`  
Enable/disable FlexIO operation in debug mode

`bool enableFastAccess`  
Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

`struct _flexio_timer_config`

*#include <fsl\_flexio.h>* Define FlexIO timer configuration structure.

### Public Members

`uint32_t triggerSelect`  
The internal trigger selection number using MACROs.

`flexio_timer_trigger_polarity_t triggerPolarity`  
Trigger Polarity.

`flexio_timer_trigger_source_t triggerSource`  
Trigger Source, internal (see 'trgsel') or external.

`flexio_pin_config_t pinConfig`  
Timer Pin Configuration.

`uint32_t pinSelect`  
Timer Pin number Select.

`flexio_pin_polarity_t pinPolarity`  
Timer Pin Polarity.

`flexio_timer_mode_t timerMode`  
Timer work Mode.

`flexio_timer_output_t timerOutput`  
Configures the initial state of the Timer Output and whether it is affected by the Timer reset.

`flexio_timer_decrement_source_t timerDecrement`  
Configures the source of the Timer decrement and the source of the Shift clock.

`flexio_timer_reset_condition_t timerReset`  
Configures the condition that causes the timer counter (and optionally the timer output) to be reset.

`flexio_timer_disable_condition_t timerDisable`  
Configures the condition that causes the Timer to be disabled and stop decrementing.

`flexio_timer_enable_condition_t timerEnable`  
Configures the condition that causes the Timer to be enabled and start decrementing.

`flexio_timer_stop_bit_condition_t timerStop`  
Timer STOP Bit generation.

*flexio\_timer\_start\_bit\_condition\_t* timerStart

Timer STRAT Bit generation.

uint32\_t timerCompare

Value for Timer Compare N Register.

struct *\_flexio\_shifter\_config*

*#include <fsl\_flexio.h>* Define FlexIO shifter configuration structure.

### Public Members

uint32\_t timerSelect

Selects which Timer is used for controlling the logic/shift register and generating the Shift clock.

*flexio\_shifter\_timer\_polarity\_t* timerPolarity

Timer Polarity.

*flexio\_pin\_config\_t* pinConfig

Shifter Pin Configuration.

uint32\_t pinSelect

Shifter Pin number Select.

*flexio\_pin\_polarity\_t* pinPolarity

Shifter Pin Polarity.

*flexio\_shifter\_mode\_t* shifterMode

Configures the mode of the Shifter.

uint32\_t parallelWidth

Configures the parallel width when using parallel mode.

*flexio\_shifter\_input\_source\_t* inputSource

Selects the input source for the shifter.

*flexio\_shifter\_stop\_bit\_t* shifterStop

Shifter STOP bit.

*flexio\_shifter\_start\_bit\_t* shifterStart

Shifter START bit.

struct *\_flexio\_gpio\_config*

*#include <fsl\_flexio.h>* The FLEXIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, use inputConfig param. If configured as an output pin, use outputLogic.

### Public Members

*flexio\_gpio\_direction\_t* pinDirection

FLEXIO pin direction, input or output

uint8\_t outputLogic

Set a default output logic, which has no use in input

uint8\_t inputConfig

Set an input config

## 2.13 FlexIO eDMA I2S Driver

```
void FLEXIO_I2S_TransferTxCreateHandleEDMA(FLEXIO_I2S_Type *base,  
                                           flexio_i2s_edma_handle_t *handle,  
                                           flexio_i2s_edma_callback_t callback, void  
                                           *userData, edma_handle_t *dmaHandle)
```

Initializes the FlexIO I2S eDMA handle.

This function initializes the FlexIO I2S master DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

### Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer.
- callback – FlexIO I2S eDMA callback function called while finished a block.
- userData – User parameter for callback.
- dmaHandle – eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

```
void FLEXIO_I2S_TransferRxCreateHandleEDMA(FLEXIO_I2S_Type *base,  
                                           flexio_i2s_edma_handle_t *handle,  
                                           flexio_i2s_edma_callback_t callback, void  
                                           *userData, edma_handle_t *dmaHandle)
```

Initializes the FlexIO I2S Rx eDMA handle.

This function initializes the FlexIO I2S slave DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

### Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer.
- callback – FlexIO I2S eDMA callback function called while finished a block.
- userData – User parameter for callback.
- dmaHandle – eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

```
void FLEXIO_I2S_TransferSetFormatEDMA(FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t  
                                       *handle, flexio_i2s_format_t *format, uint32_t  
                                       srcClock_Hz)
```

Configures the FlexIO I2S Tx audio format.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to format.

### Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer
- format – Pointer to FlexIO I2S audio data format structure.
- srcClock\_Hz – FlexIO I2S clock source frequency in Hz, it should be 0 while in slave mode.

```
status_t FLEXIO_I2S_TransferSendEDMA(FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t
                                     *handle, flexio_i2s_transfer_t *xfer)
```

Performs a non-blocking FlexIO I2S transfer using DMA.

---

**Note:** This interface returned immediately after transfer initiates. Users should call FLEXIO\_I2S\_GetTransferStatus to poll the transfer status and check whether the FlexIO I2S transfer is finished.

---

#### Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- xfer – Pointer to DMA transfer structure.

#### Return values

- kStatus\_Success – Start a FlexIO I2S eDMA send successfully.
- kStatus\_InvalidArgument – The input arguments is invalid.
- kStatus\_TxBusy – FlexIO I2S is busy sending data.

```
status_t FLEXIO_I2S_TransferReceiveEDMA(FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t
                                         *handle, flexio_i2s_transfer_t *xfer)
```

Performs a non-blocking FlexIO I2S receive using eDMA.

---

**Note:** This interface returned immediately after transfer initiates. Users should call FLEXIO\_I2S\_GetReceiveRemainingBytes to poll the transfer status and check whether the FlexIO I2S transfer is finished.

---

#### Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- xfer – Pointer to DMA transfer structure.

#### Return values

- kStatus\_Success – Start a FlexIO I2S eDMA receive successfully.
- kStatus\_InvalidArgument – The input arguments is invalid.
- kStatus\_RxBusy – FlexIO I2S is busy receiving data.

```
void FLEXIO_I2S_TransferAbortSendEDMA(FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t
                                       *handle)
```

Aborts a FlexIO I2S transfer using eDMA.

#### Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.

```
void FLEXIO_I2S_TransferAbortReceiveEDMA(FLEXIO_I2S_Type *base,
                                          flexio_i2s_edma_handle_t *handle)
```

Aborts a FlexIO I2S receive using eDMA.

#### Parameters

- base – FlexIO I2S peripheral base address.

- handle – FlexIO I2S DMA handle pointer.

*status\_t* FLEXIO\_I2S\_TransferGetSendCountEDMA(*FLEXIO\_I2S\_Type* \*base,  
flexio\_i2s\_edma\_handle\_t \*handle, size\_t  
\*count)

Gets the remaining bytes to be sent.

#### Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- count – Bytes sent.

#### Return values

- kStatus\_Success – Succeed get the transfer count.
- kStatus\_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

*status\_t* FLEXIO\_I2S\_TransferGetReceiveCountEDMA(*FLEXIO\_I2S\_Type* \*base,  
flexio\_i2s\_edma\_handle\_t \*handle, size\_t  
\*count)

Get the remaining bytes to be received.

#### Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- count – Bytes received.

#### Return values

- kStatus\_Success – Succeed get the transfer count.
- kStatus\_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

FSL\_FLEXIO\_I2S\_EDMA\_DRIVER\_VERSION

FlexIO I2S EDMA driver version 2.1.9.

typedef struct flexio\_i2s\_edma\_handle flexio\_i2s\_edma\_handle\_t

typedef void (\*flexio\_i2s\_edma\_callback\_t)(*FLEXIO\_I2S\_Type* \*base, flexio\_i2s\_edma\_handle\_t  
\*handle, *status\_t* status, void \*userData)

FlexIO I2S eDMA transfer callback function for finish and error.

struct flexio\_i2s\_edma\_handle

#include <fsl\_flexio\_i2s\_edma.h> FlexIO I2S DMA transfer handle, users should not touch the  
content of the handle.

#### Public Members

edma\_handle\_t \*dmaHandle

DMA handler for FlexIO I2S send

uint8\_t bytesPerFrame

Bytes in a frame

uint8\_t nbytes

eDMA minor byte transfer count initially configured.

`uint32_t state`  
Internal state for FlexIO I2S eDMA transfer

`flexio_i2s_edma_callback_t callback`  
Callback for users while transfer finish or error occurred

`void *userData`  
User callback parameter

`edma_tcd_t tcd[(4U) + 1U]`  
TCD pool for eDMA transfer.

`flexio_i2s_transfer_t queue[(4U)]`  
Transfer queue storing queued transfer.

`size_t transferSize[(4U)]`  
Data bytes need to transfer

`volatile uint8_t queueUser`  
Index for user to queue transfer.

`volatile uint8_t queueDriver`  
Index for driver to get the transfer data and size

## 2.14 FlexIO eDMA SPI Driver

`status_t FLEXIO_SPI_MasterTransferCreateHandleEDMA(FLEXIO_SPI_Type *base,  
flexio_spi_master_edma_handle_t  
*handle,  
flexio_spi_master_edma_transfer_callback_t  
callback, void *userData,  
edma_handle_t *txHandle,  
edma_handle_t *rxHandle)`

Initializes the FlexIO SPI master eDMA handle.

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

### Parameters

- `base` – Pointer to `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to `flexio_spi_master_edma_handle_t` structure to store the transfer state.
- `callback` – SPI callback, NULL means no callback.
- `userData` – callback function parameter.
- `txHandle` – User requested eDMA handle for FlexIO SPI RX eDMA transfer.
- `rxHandle` – User requested eDMA handle for FlexIO SPI TX eDMA transfer.

### Return values

- `kStatus_Success` – Successfully create the handle.
- `kStatus_OutOfRange` – The FlexIO SPI eDMA type/handle table out of range.

```
status_t FLEXIO_SPI_MasterTransferEDMA(FLEXIO_SPI_Type *base,  
                                         flexio_spi_master_edma_handle_t *handle,  
                                         flexio_spi_transfer_t *xfer)
```

Performs a non-blocking FlexIO SPI transfer using eDMA.

---

**Note:** This interface returns immediately after transfer initiates. Call FLEXIO\_SPI\_MasterGetTransferCountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

---

#### Parameters

- base – Pointer to FLEXIO\_SPI\_Type structure.
- handle – Pointer to flexio\_spi\_master\_edma\_handle\_t structure to store the transfer state.
- xfer – Pointer to FlexIO SPI transfer structure.

#### Return values

- kStatus\_Success – Successfully start a transfer.
- kStatus\_InvalidArgument – Input argument is invalid.
- kStatus\_FLEXIO\_SPI\_Busy – FlexIO SPI is not idle, is running another transfer.

```
void FLEXIO_SPI_MasterTransferAbortEDMA(FLEXIO_SPI_Type *base,  
                                         flexio_spi_master_edma_handle_t *handle)
```

Aborts a FlexIO SPI transfer using eDMA.

#### Parameters

- base – Pointer to FLEXIO\_SPI\_Type structure.
- handle – FlexIO SPI eDMA handle pointer.

```
status_t FLEXIO_SPI_MasterTransferGetCountEDMA(FLEXIO_SPI_Type *base,  
                                                flexio_spi_master_edma_handle_t *handle,  
                                                size_t *count)
```

Gets the number of bytes transferred so far using FlexIO SPI master eDMA.

#### Parameters

- base – Pointer to FLEXIO\_SPI\_Type structure.
- handle – FlexIO SPI eDMA handle pointer.
- count – Number of bytes transferred so far by the non-blocking transaction.

```
static inline void FLEXIO_SPI_SlaveTransferCreateHandleEDMA(FLEXIO_SPI_Type *base,  
                                                           flexio_spi_slave_edma_handle_t *handle,  
                                                           flexio_spi_slave_edma_transfer_callback_t callback,  
                                                           void *userData,  
                                                           edma_handle_t *txHandle,  
                                                           edma_handle_t *rxHandle)
```

Initializes the FlexIO SPI slave eDMA handle.

This function initializes the FlexIO SPI slave eDMA handle.

#### Parameters

- base – Pointer to FLEXIO\_SPI\_Type structure.



- `handle` – Pointer to `flexio_spi_slave_edma_handle_t` structure to store the transfer state.
- `callback` – SPI callback, NULL means no callback.
- `userData` – callback function parameter.
- `txHandle` – User requested eDMA handle for FlexIO SPI TX eDMA transfer.
- `rxHandle` – User requested eDMA handle for FlexIO SPI RX eDMA transfer.

```
status_t FLEXIO_SPI_SlaveTransferEDMA(FLEXIO_SPI_Type *base,
                                       flexio_spi_slave_edma_handle_t *handle,
                                       flexio_spi_transfer_t *xfer)
```

Performs a non-blocking FlexIO SPI transfer using eDMA.

---

**Note:** This interface returns immediately after transfer initiates. Call `FLEXIO_SPI_SlaveGetTransferCountEDMA` to poll the transfer status and check whether the FlexIO SPI transfer is finished.

---

#### Parameters

- `base` – Pointer to `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to `flexio_spi_slave_edma_handle_t` structure to store the transfer state.
- `xfer` – Pointer to FlexIO SPI transfer structure.

#### Return values

- `kStatus_Success` – Successfully start a transfer.
- `kStatus_InvalidArgument` – Input argument is invalid.
- `kStatus_FLEXIO_SPI_Busy` – FlexIO SPI is not idle, is running another transfer.

```
static inline void FLEXIO_SPI_SlaveTransferAbortEDMA(FLEXIO_SPI_Type *base,
                                                    flexio_spi_slave_edma_handle_t
                                                    *handle)
```

Aborts a FlexIO SPI transfer using eDMA.

#### Parameters

- `base` – Pointer to `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to `flexio_spi_slave_edma_handle_t` structure to store the transfer state.

```
static inline status_t FLEXIO_SPI_SlaveTransferGetCountEDMA(FLEXIO_SPI_Type *base,
                                                           flexio_spi_slave_edma_handle_t
                                                           *handle, size_t *count)
```

Gets the number of bytes transferred so far using FlexIO SPI slave eDMA.

#### Parameters

- `base` – Pointer to `FLEXIO_SPI_Type` structure.
- `handle` – FlexIO SPI eDMA handle pointer.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

`FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION`

FlexIO SPI EDMA driver version.

```
typedef struct flexio_spi_master_edma_handle flexio_spi_master_edma_handle_t
    typedef for flexio_spi_master_edma_handle_t in advance.
```

```
typedef flexio_spi_master_edma_handle_t flexio_spi_slave_edma_handle_t
    Slave handle is the same with master handle.
```

```
typedef void (*flexio_spi_master_edma_transfer_callback_t)(FLEXIO_SPI_Type *base,
flexio_spi_master_edma_handle_t *handle, status_t status, void *userData)
```

FlexIO SPI master callback for finished transmit.

```
typedef void (*flexio_spi_slave_edma_transfer_callback_t)(FLEXIO_SPI_Type *base,
flexio_spi_slave_edma_handle_t *handle, status_t status, void *userData)
```

FlexIO SPI slave callback for finished transmit.

```
struct flexio_spi_master_edma_handle
```

```
    #include <fsl_flexio_spi_edma.h> FlexIO SPI eDMA transfer handle, users should not touch
    the content of the handle.
```

### Public Members

*size\_t* transferSize

Total bytes to be transferred.

*uint8\_t* nbytes

eDMA minor byte transfer count initially configured.

*bool* txInProgress

Send transfer in progress

*bool* rxInProgress

Receive transfer in progress

*edma\_handle\_t* \*txHandle

DMA handler for SPI send

*edma\_handle\_t* \*rxHandle

DMA handler for SPI receive

*flexio\_spi\_master\_edma\_transfer\_callback\_t* callback

Callback for SPI DMA transfer

*void* \*userData

User Data for SPI DMA callback

## 2.15 FlexIO eDMA UART Driver

```
status_t FLEXIO_UART_TransferCreateHandleEDMA(FLEXIO_UART_Type *base,
    flexio_uart_edma_handle_t *handle,
    flexio_uart_edma_transfer_callback_t
    callback, void *userData, edma_handle_t
    *txEdmaHandle, edma_handle_t
    *rxEdmaHandle)
```

Initializes the UART handle which is used in transactional functions.

### Parameters

- base – Pointer to *FLEXIO\_UART\_Type*.
- handle – Pointer to *flexio\_uart\_edma\_handle\_t* structure.

- callback – The callback function.
- userData – The parameter of the callback function.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.

**Return values**

- kStatus\_\_Success – Successfully create the handle.
- kStatus\_\_OutOfRange – The FlexIO SPI eDMA type/handle table out of range.

```
status_t FLEXIO_UART__TransferSendEDMA(FLEXIO_UART_Type *base,  
                                         flexio_uart_edma_handle_t *handle,  
                                         flexio_uart_transfer_t *xfer)
```

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent out, the send callback function is called.

**Parameters**

- base – Pointer to FLEXIO\_UART\_Type
- handle – UART handle pointer.
- xfer – UART eDMA transfer structure, see flexio\_uart\_transfer\_t.

**Return values**

- kStatus\_\_Success – if succeed, others failed.
- kStatus\_\_FLEXIO\_UART\_\_TxBusy – Previous transfer on going.

```
status_t FLEXIO_UART__TransferReceiveEDMA(FLEXIO_UART_Type *base,  
                                           flexio_uart_edma_handle_t *handle,  
                                           flexio_uart_transfer_t *xfer)
```

Receives data using eDMA.

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

**Parameters**

- base – Pointer to FLEXIO\_UART\_Type
- handle – Pointer to flexio\_uart\_edma\_handle\_t structure
- xfer – UART eDMA transfer structure, see flexio\_uart\_transfer\_t.

**Return values**

- kStatus\_\_Success – if succeed, others failed.
- kStatus\_\_UART\_\_RxBusy – Previous transfer on going.

```
void FLEXIO_UART__TransferAbortSendEDMA(FLEXIO_UART_Type *base,  
                                         flexio_uart_edma_handle_t *handle)
```

Aborts the sent data which using eDMA.

This function aborts sent data which using eDMA.

**Parameters**

- base – Pointer to FLEXIO\_UART\_Type
- handle – Pointer to flexio\_uart\_edma\_handle\_t structure

```
void FLEXIO_UART_TransferAbortReceiveEDMA(FLEXIO_UART_Type *base,  
                                           flexio_uart_edma_handle_t *handle)
```

Aborts the receive data which using eDMA.

This function aborts the receive data which using eDMA.

#### Parameters

- base – Pointer to *FLEXIO\_UART\_Type*
- handle – Pointer to *flexio\_uart\_edma\_handle\_t* structure

```
status_t FLEXIO_UART_TransferGetSendCountEDMA(FLEXIO_UART_Type *base,  
                                                flexio_uart_edma_handle_t *handle,  
                                                size_t *count)
```

Gets the number of bytes sent out.

This function gets the number of bytes sent out.

#### Parameters

- base – Pointer to *FLEXIO\_UART\_Type*
- handle – Pointer to *flexio\_uart\_edma\_handle\_t* structure
- count – Number of bytes sent so far by the non-blocking transaction.

#### Return values

- *kStatus\_NoTransferInProgress* – transfer has finished or no transfer in progress.
- *kStatus\_Success* – Successfully return the count.

```
status_t FLEXIO_UART_TransferGetReceiveCountEDMA(FLEXIO_UART_Type *base,  
                                                  flexio_uart_edma_handle_t *handle,  
                                                  size_t *count)
```

Gets the number of bytes received.

This function gets the number of bytes received.

#### Parameters

- base – Pointer to *FLEXIO\_UART\_Type*
- handle – Pointer to *flexio\_uart\_edma\_handle\_t* structure
- count – Number of bytes received so far by the non-blocking transaction.

#### Return values

- *kStatus\_NoTransferInProgress* – transfer has finished or no transfer in progress.
- *kStatus\_Success* – Successfully return the count.

```
FSL_FLEXIO_UART_EDMA_DRIVER_VERSION
```

FlexIO UART EDMA driver version.

```
typedef struct flexio_uart_edma_handle flexio_uart_edma_handle_t
```

```
typedef void (*flexio_uart_edma_transfer_callback_t)(FLEXIO_UART_Type *base,  
flexio_uart_edma_handle_t *handle, status_t status, void *userData)
```

UART transfer callback function.

```
struct flexio_uart_edma_handle
```

```
#include <fsl_flexio_uart_edma.h> UART eDMA handle.
```

## Public Members

*flexio\_uart\_edma\_transfer\_callback\_t* callback

Callback function.

*void \*userData*

UART callback function parameter.

*size\_t txDataSizeAll*

Total bytes to be sent.

*size\_t rxDataSizeAll*

Total bytes to be received.

*edma\_handle\_t \*txEdmaHandle*

The eDMA TX channel used.

*edma\_handle\_t \*rxEdmaHandle*

The eDMA RX channel used.

*uint8\_t nbytes*

eDMA minor byte transfer count initially configured.

*volatile uint8\_t txState*

TX transfer state.

*volatile uint8\_t rxState*

RX transfer state

## 2.16 FlexIO I2C Master Driver

*status\_t* FLEXIO\_I2C\_CheckForBusyBus(*FLEXIO\_I2C\_Type \*base*)

Make sure the bus isn't already pulled down.

Check the FLEXIO pin status to see whether either of SDA and SCL pin is pulled down.

### Parameters

- *base* – Pointer to *FLEXIO\_I2C\_Type* structure..

### Return values

- *kStatus\_Success* –
- *kStatus\_FLEXIO\_I2C\_Busy* –

*status\_t* FLEXIO\_I2C\_MasterInit(*FLEXIO\_I2C\_Type \*base*, *flexio\_i2c\_master\_config\_t \*masterConfig*, *uint32\_t srcClock\_Hz*)

Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO I2C hardware configuration.

### Example

```

FLEXIO_I2C_Type base = {
    .flexioBase = FLEXIO,
    .SDAPinIndex = 0,
    .SCLPinIndex = 1,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_i2c_master_config_t config = {
    .enableInDoze = false,

```

(continues on next page)

(continued from previous page)

```
.enableInDebug = true,  
.enableFastAccess = false,  
.baudRate_Bps = 100000  
};  
FLEXIO_I2C_MasterInit(base, &config, srcClock_Hz);
```

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure.
- masterConfig – Pointer to flexio\_i2c\_master\_config\_t structure.
- srcClock\_Hz – FlexIO source clock in Hz.

**Return values**

- kStatus\_Success – Initialization successful
- kStatus\_InvalidArgument – The source clock exceed upper range limitation

void FLEXIO\_I2C\_MasterDeinit(*FLEXIO\_I2C\_Type* \*base)

De-initializes the FlexIO I2C master peripheral. Calling this API Resets the FlexIO I2C master shifer and timer config, module can't work unless the FLEXIO\_I2C\_MasterInit is called.

**Parameters**

- base – pointer to FLEXIO\_I2C\_Type structure.

void FLEXIO\_I2C\_MasterGetDefaultConfig(*flexio\_i2c\_master\_config\_t* \*masterConfig)

Gets the default configuration to configure the FlexIO module. The configuration can be used directly for calling the FLEXIO\_I2C\_MasterInit().

Example:

```
flexio_i2c_master_config_t config;  
FLEXIO_I2C_MasterGetDefaultConfig(&config);
```

**Parameters**

- masterConfig – Pointer to flexio\_i2c\_master\_config\_t structure.

static inline void FLEXIO\_I2C\_MasterEnable(*FLEXIO\_I2C\_Type* \*base, bool enable)

Enables/disables the FlexIO module operation.

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure.
- enable – Pass true to enable module, false does not have any effect.

uint32\_t FLEXIO\_I2C\_MasterGetStatusFlags(*FLEXIO\_I2C\_Type* \*base)

Gets the FlexIO I2C master status flags.

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure

**Returns**

Status flag, use status flag to AND `_flexio_i2c_master_status_flags` can get the related status.

void FLEXIO\_I2C\_MasterClearStatusFlags(*FLEXIO\_I2C\_Type* \*base, uint32\_t mask)

Clears the FlexIO I2C master status flags.

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure.

- mask – Status flag. The parameter can be any combination of the following values:
  - kFLEXIO\_I2C\_RxFullFlag
  - kFLEXIO\_I2C\_ReceiveNakFlag

void FLEXIO\_I2C\_MasterEnableInterrupts(*FLEXIO\_I2C\_Type* \*base, uint32\_t mask)

Enables the FlexIO i2c master interrupt requests.

#### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure.
- mask – Interrupt source. Currently only one interrupt request source:
  - kFLEXIO\_I2C\_TransferCompleteInterruptEnable

void FLEXIO\_I2C\_MasterDisableInterrupts(*FLEXIO\_I2C\_Type* \*base, uint32\_t mask)

Disables the FlexIO I2C master interrupt requests.

#### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure.
- mask – Interrupt source.

void FLEXIO\_I2C\_MasterSetBaudRate(*FLEXIO\_I2C\_Type* \*base, uint32\_t baudRate\_Bps,  
uint32\_t srcClock\_Hz)

Sets the FlexIO I2C master transfer baudrate.

#### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure
- baudRate\_Bps – the baud rate value in HZ
- srcClock\_Hz – source clock in HZ

void FLEXIO\_I2C\_MasterStart(*FLEXIO\_I2C\_Type* \*base, uint8\_t address, *flexio\_i2c\_direction\_t*  
direction)

Sends START + 7-bit address to the bus.

---

**Note:** This API should be called when the transfer configuration is ready to send a START signal and 7-bit address to the bus. This is a non-blocking API, which returns directly after the address is put into the data register but the address transfer is not finished on the bus. Ensure that the kFLEXIO\_I2C\_RxFullFlag status is asserted before calling this API.

---

#### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure.
- address – 7-bit address.
- direction – transfer direction. This parameter is one of the values in *flexio\_i2c\_direction\_t*:
  - kFLEXIO\_I2C\_Write: Transmit
  - kFLEXIO\_I2C\_Read: Receive

void FLEXIO\_I2C\_MasterStop(*FLEXIO\_I2C\_Type* \*base)

Sends the stop signal on the bus.

#### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure.

void FLEXIO\_I2C\_MasterRepeatedStart(*FLEXIO\_I2C\_Type* \*base)

Sends the repeated start signal on the bus.

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure.

void FLEXIO\_I2C\_MasterAbortStop(*FLEXIO\_I2C\_Type* \*base)

Sends the stop signal when transfer is still on-going.

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure.

void FLEXIO\_I2C\_MasterEnableAck(*FLEXIO\_I2C\_Type* \*base, bool enable)

Configures the sent ACK/NAK for the following byte.

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure.
- enable – True to configure send ACK, false configure to send NAK.

*status\_t* FLEXIO\_I2C\_MasterSetTransferCount(*FLEXIO\_I2C\_Type* \*base, uint16\_t count)

Sets the number of bytes to be transferred from a start signal to a stop signal.

---

**Note:** Call this API before a transfer begins because the timer generates a number of clocks according to the number of bytes that need to be transferred.

---

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure.
- count – Number of bytes need to be transferred from a start signal to a re-start/stop signal

**Return values**

- kStatus\_Success – Successfully configured the count.
- kStatus\_InvalidArgument – Input argument is invalid.

static inline void FLEXIO\_I2C\_MasterWriteByte(*FLEXIO\_I2C\_Type* \*base, uint32\_t data)

Writes one byte of data to the I2C bus.

---

**Note:** This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

---

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure.
- data – a byte of data.

static inline uint8\_t FLEXIO\_I2C\_MasterReadByte(*FLEXIO\_I2C\_Type* \*base)

Reads one byte of data from the I2C bus.

---

**Note:** This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the data is ready in the register.

---

**Parameters**



- base – Pointer to FLEXIO\_I2C\_Type structure.

**Returns**

data byte read.

*status\_t* FLEXIO\_I2C\_MasterWriteBlocking(*FLEXIO\_I2C\_Type* \*base, const uint8\_t \*txBuff, uint8\_t txSize)

Sends a buffer of data in bytes.

---

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

---

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure.
- txBuff – The data bytes to send.
- txSize – The number of data bytes to send.

**Return values**

- kStatus\_Success – Successfully write data.
- kStatus\_FLEXIO\_I2C\_Nak – Receive NAK during writing data.
- kStatus\_FLEXIO\_I2C\_Timeout – Timeout polling status flags.

*status\_t* FLEXIO\_I2C\_MasterReadBlocking(*FLEXIO\_I2C\_Type* \*base, uint8\_t \*rxBuff, uint8\_t rxSize)

Receives a buffer of bytes.

---

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

---

**Parameters**

- base – Pointer to FLEXIO\_I2C\_Type structure.
- rxBuff – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

**Return values**

- kStatus\_Success – Successfully read data.
- kStatus\_FLEXIO\_I2C\_Timeout – Timeout polling status flags.

*status\_t* FLEXIO\_I2C\_MasterTransferBlocking(*FLEXIO\_I2C\_Type* \*base, *flexio\_i2c\_master\_transfer\_t* \*xfer)

Performs a master polling transfer on the I2C bus.

---

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

---

**Parameters**

- base – pointer to FLEXIO\_I2C\_Type structure.
- xfer – pointer to flexio\_i2c\_master\_transfer\_t structure.

**Returns**

status of status\_t.

```
status_t FLEXIO_I2C_MasterTransferCreateHandle(FLEXIO_I2C_Type *base,  
                                              flexio_i2c_master_handle_t *handle,  
                                              flexio_i2c_master_transfer_callback_t  
                                              callback, void *userData)
```

Initializes the I2C handle which is used in transactional functions.

#### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure.
- handle – Pointer to flexio\_i2c\_master\_handle\_t structure to store the transfer state.
- callback – Pointer to user callback function.
- userData – User param passed to the callback function.

#### Return values

- kStatus\_Success – Successfully create the handle.
- kStatus\_OutOfRange – The FlexIO type/handle/isr table out of range.

```
status_t FLEXIO_I2C_MasterTransferNonBlocking(FLEXIO_I2C_Type *base,  
                                              flexio_i2c_master_handle_t *handle,  
                                              flexio_i2c_master_transfer_t *xfer)
```

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

---

**Note:** The API returns immediately after the transfer initiates. Call FLEXIO\_I2C\_MasterTransferGetCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus\_FLEXIO\_I2C\_Busy, the transfer is finished.

---

#### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure
- handle – Pointer to flexio\_i2c\_master\_handle\_t structure which stores the transfer state
- xfer – pointer to flexio\_i2c\_master\_transfer\_t structure

#### Return values

- kStatus\_Success – Successfully start a transfer.
- kStatus\_FLEXIO\_I2C\_Busy – FlexIO I2C is not idle, is running another transfer.

```
status_t FLEXIO_I2C_MasterTransferGetCount(FLEXIO_I2C_Type *base,  
                                              flexio_i2c_master_handle_t *handle, size_t  
                                              *count)
```

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

#### Parameters

- base – Pointer to FLEXIO\_I2C\_Type structure.
- handle – Pointer to flexio\_i2c\_master\_handle\_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

#### Return values

- kStatus\_InvalidArgument – count is Invalid.

- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.
- `kStatus_Success` – Successfully return the count.

`void FLEXIO_I2C_MasterTransferAbort(FLEXIO_I2C_Type *base, flexio_i2c_master_handle_t *handle)`

Aborts an interrupt non-blocking transfer early.

---

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

---

#### Parameters

- `base` – Pointer to `FLEXIO_I2C_Type` structure
- `handle` – Pointer to `flexio_i2c_master_handle_t` structure which stores the transfer state

`void FLEXIO_I2C_MasterTransferHandleIRQ(void *i2cType, void *i2cHandle)`

Master interrupt handler.

#### Parameters

- `i2cType` – Pointer to `FLEXIO_I2C_Type` structure
- `i2cHandle` – Pointer to `flexio_i2c_master_transfer_t` structure

`FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION`

FlexIO I2C transfer status.

*Values:*

enumerator `kStatus_FLEXIO_I2C_Busy`  
I2C is busy doing transfer.

enumerator `kStatus_FLEXIO_I2C_Idle`  
I2C is busy doing transfer.

enumerator `kStatus_FLEXIO_I2C_Nak`  
NAK received during transfer.

enumerator `kStatus_FLEXIO_I2C_Timeout`  
Timeout polling status flags.

enum `_flexio_i2c_master_interrupt`

Define FlexIO I2C master interrupt mask.

*Values:*

enumerator `kFLEXIO_I2C_TxEmptyInterruptEnable`  
Tx buffer empty interrupt enable.

enumerator `kFLEXIO_I2C_RxFullInterruptEnable`  
Rx buffer full interrupt enable.

enum `_flexio_i2c_master_status_flags`

Define FlexIO I2C master status mask.

*Values:*

enumerator kFLEXIO\_I2C\_TxEmptyFlag  
Tx shifter empty flag.

enumerator kFLEXIO\_I2C\_RxFullFlag  
Rx shifter full/Transfer complete flag.

enumerator kFLEXIO\_I2C\_ReceiveNakFlag  
Receive NAK flag.

enum \_flexio\_i2c\_direction  
Direction of master transfer.  
*Values:*

enumerator kFLEXIO\_I2C\_Write  
Master send to slave.

enumerator kFLEXIO\_I2C\_Read  
Master receive from slave.

typedef enum \_flexio\_i2c\_direction flexio\_i2c\_direction\_t  
Direction of master transfer.

typedef struct \_flexio\_i2c\_type FLEXIO\_I2C\_Type  
Define FlexIO I2C master access structure typedef.

typedef struct \_flexio\_i2c\_master\_config flexio\_i2c\_master\_config\_t  
Define FlexIO I2C master user configuration structure.

typedef struct \_flexio\_i2c\_master\_transfer flexio\_i2c\_master\_transfer\_t  
Define FlexIO I2C master transfer structure.

typedef struct \_flexio\_i2c\_master\_handle flexio\_i2c\_master\_handle\_t  
FlexIO I2C master handle typedef.

typedef void (\*flexio\_i2c\_master\_transfer\_callback\_t)(FLEXIO\_I2C\_Type \*base,  
flexio\_i2c\_master\_handle\_t \*handle, status\_t status, void \*userData)  
FlexIO I2C master transfer callback typedef.

I2C\_RETRY\_TIMES  
Retry times for waiting flag.

struct \_flexio\_i2c\_type  
*#include <fsl\_flexio\_i2c\_master.h>* Define FlexIO I2C master access structure typedef.

### Public Members

FLEXIO\_Type \*flexioBase  
FlexIO base pointer.

uint8\_t SDAPinIndex  
Pin select for I2C SDA.

uint8\_t SCLPinIndex  
Pin select for I2C SCL.

uint8\_t shifterIndex[2]  
Shifter index used in FlexIO I2C.

uint8\_t timerIndex[3]  
Timer index used in FlexIO I2C.

uint32\_t baudrate

Master transfer baudrate, used to calculate delay time.

struct \_flexio\_i2c\_master\_config

*#include <fsl\_flexio\_i2c\_master.h>* Define FlexIO I2C master user configuration structure.

### Public Members

bool enableMaster

Enables the FlexIO I2C peripheral at initialization time.

bool enableInDoze

Enable/disable FlexIO operation in doze mode.

bool enableInDebug

Enable/disable FlexIO operation in debug mode.

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32\_t baudRate\_Bps

Baud rate in Bps.

struct \_flexio\_i2c\_master\_transfer

*#include <fsl\_flexio\_i2c\_master.h>* Define FlexIO I2C master transfer structure.

### Public Members

uint32\_t flags

Transfer flag which controls the transfer, reserved for FlexIO I2C.

uint8\_t slaveAddress

7-bit slave address.

*flexio\_i2c\_direction\_t* direction

Transfer direction, read or write.

uint32\_t subaddress

Sub address. Transferred MSB first.

uint8\_t subaddressSize

Size of sub address.

uint8\_t volatile \*data

Transfer buffer.

volatile size\_t dataSize

Transfer size.

struct \_flexio\_i2c\_master\_handle

*#include <fsl\_flexio\_i2c\_master.h>* Define FlexIO I2C master handle structure.

### Public Members

*flexio\_i2c\_master\_transfer\_t* transfer

FlexIO I2C master transfer copy.

size\_t transferSize

Total bytes to be transferred.

uint8\_t state

Transfer state maintained during transfer.

*flexio\_i2c\_master\_transfer\_callback\_t* completionCallback

Callback function called at transfer event. Callback function called at transfer event.

void \*userData

Callback parameter passed to callback function.

bool needRestart

Whether master needs to send re-start signal.

## 2.17 FlexIO I2S Driver

void FLEXIO\_I2S\_Init(*FLEXIO\_I2S\_Type* \*base, const *flexio\_i2s\_config\_t* \*config)

Initializes the FlexIO I2S.

This API configures FlexIO pins and shifter to I2S and configures the FlexIO I2S with a configuration structure. The configuration structure can be filled by the user, or be set with default values by FLEXIO\_I2S\_GetDefaultConfig().

---

**Note:** This API should be called at the beginning of the application to use the FlexIO I2S driver. Otherwise, any access to the FlexIO I2S module can cause hard fault because the clock is not enabled.

---

### Parameters

- base – FlexIO I2S base pointer
- config – FlexIO I2S configure structure.

void FLEXIO\_I2S\_GetDefaultConfig(*flexio\_i2s\_config\_t* \*config)

Sets the FlexIO I2S configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in FLEXIO\_I2S\_Init(). Users may use the initialized structure unchanged in FLEXIO\_I2S\_Init() or modify some fields of the structure before calling FLEXIO\_I2S\_Init().

### Parameters

- config – pointer to master configuration structure

void FLEXIO\_I2S\_Deinit(*FLEXIO\_I2S\_Type* \*base)

De-initializes the FlexIO I2S.

Calling this API resets the FlexIO I2S shifter and timer config. After calling this API, call the FLEXIO\_I2S\_Init to use the FlexIO I2S module.

### Parameters

- base – FlexIO I2S base pointer

static inline void FLEXIO\_I2S\_Enable(*FLEXIO\_I2S\_Type* \*base, bool enable)

Enables/disables the FlexIO I2S module operation.

### Parameters

- base – Pointer to FLEXIO\_I2S\_Type

- enable – True to enable, false dose not have any effect.

uint32\_t FLEXIO\_I2S\_GetStatusFlags(*FLEXIO\_I2S\_Type* \*base)

Gets the FlexIO I2S status flags.

#### Parameters

- base – Pointer to FLEXIO\_I2S\_Type structure

#### Returns

Status flag, which are ORed by the enumerators in the `_flexio_i2s_status_flags`.

void FLEXIO\_I2S\_EnableInterrupts(*FLEXIO\_I2S\_Type* \*base, uint32\_t mask)

Enables the FlexIO I2S interrupt.

This function enables the FlexIO UART interrupt.

#### Parameters

- base – Pointer to FLEXIO\_I2S\_Type structure
- mask – interrupt source

void FLEXIO\_I2S\_DisableInterrupts(*FLEXIO\_I2S\_Type* \*base, uint32\_t mask)

Disables the FlexIO I2S interrupt.

This function enables the FlexIO UART interrupt.

#### Parameters

- base – pointer to FLEXIO\_I2S\_Type structure
- mask – interrupt source

static inline void FLEXIO\_I2S\_TxEnableDMA(*FLEXIO\_I2S\_Type* \*base, bool enable)

Enables/disables the FlexIO I2S Tx DMA requests.

#### Parameters

- base – FlexIO I2S base pointer
- enable – True means enable DMA, false means disable DMA.

static inline void FLEXIO\_I2S\_RxEnableDMA(*FLEXIO\_I2S\_Type* \*base, bool enable)

Enables/disables the FlexIO I2S Rx DMA requests.

#### Parameters

- base – FlexIO I2S base pointer
- enable – True means enable DMA, false means disable DMA.

static inline uint32\_t FLEXIO\_I2S\_TxGetDataRegisterAddress(*FLEXIO\_I2S\_Type* \*base)

Gets the FlexIO I2S send data register address.

This function returns the I2S data register address, mainly used by DMA/eDMA.

#### Parameters

- base – Pointer to FLEXIO\_I2S\_Type structure

#### Returns

FlexIO i2s send data register address.

static inline uint32\_t FLEXIO\_I2S\_RxGetDataRegisterAddress(*FLEXIO\_I2S\_Type* \*base)

Gets the FlexIO I2S receive data register address.

This function returns the I2S data register address, mainly used by DMA/eDMA.

#### Parameters

- base – Pointer to FLEXIO\_I2S\_Type structure

**Returns**

FlexIO i2s receive data register address.

```
void FLEXIO_I2S_MasterSetFormat(FLEXIO_I2S_Type *base, flexio_i2s_format_t *format,  
                                uint32_t srcClock_Hz)
```

Configures the FlexIO I2S audio format in master mode.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

**Parameters**

- base – Pointer to FLEXIO\_I2S\_Type structure
- format – Pointer to FlexIO I2S audio data format structure.
- srcClock\_Hz – I2S master clock source frequency in Hz.

```
void FLEXIO_I2S_SlaveSetFormat(FLEXIO_I2S_Type *base, flexio_i2s_format_t *format)
```

Configures the FlexIO I2S audio format in slave mode.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

**Parameters**

- base – Pointer to FLEXIO\_I2S\_Type structure
- format – Pointer to FlexIO I2S audio data format structure.

```
status_t FLEXIO_I2S_WriteBlocking(FLEXIO_I2S_Type *base, uint8_t bitWidth, uint8_t *txData,  
                                   size_t size)
```

Sends data using a blocking method.

---

**Note:** This function blocks via polling until data is ready to be sent.

---

**Parameters**

- base – FlexIO I2S base pointer.
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- txData – Pointer to the data to be written.
- size – Bytes to be written.

**Return values**

- kStatus\_Success – Successfully write data.
- kStatus\_FLEXIO\_I2C\_Timeout – Timeout polling status flags.

```
static inline void FLEXIO_I2S_WriteData(FLEXIO_I2S_Type *base, uint8_t bitWidth, uint32_t  
                                         data)
```

Writes data into a data register.

**Parameters**

- base – FlexIO I2S base pointer.
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- data – Data to be written.



*status\_t* FLEXIO\_I2S\_ReadBlocking(*FLEXIO\_I2S\_Type* \*base, uint8\_t bitWidth, uint8\_t \*rxData, size\_t size)

Receives a piece of data using a blocking method.

---

**Note:** This function blocks via polling until data is ready to be sent.

---

#### Parameters

- base – FlexIO I2S base pointer
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- rxData – Pointer to the data to be read.
- size – Bytes to be read.

#### Return values

- kStatus\_Success – Successfully read data.
- kStatus\_FLEXIO\_I2C\_Timeout – Timeout polling status flags.

static inline uint32\_t FLEXIO\_I2S\_ReadData(*FLEXIO\_I2S\_Type* \*base)

Reads a data from the data register.

#### Parameters

- base – FlexIO I2S base pointer

#### Returns

Data read from data register.

void FLEXIO\_I2S\_TransferTxCreateHandle(*FLEXIO\_I2S\_Type* \*base, *flexio\_i2s\_handle\_t* \*handle, *flexio\_i2s\_callback\_t* callback, void \*userData)

Initializes the FlexIO I2S handle.

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

#### Parameters

- base – Pointer to FLEXIO\_I2S\_Type structure
- handle – Pointer to flexio\_i2s\_handle\_t structure to store the transfer state.
- callback – FlexIO I2S callback function, which is called while finished a block.
- userData – User parameter for the FlexIO I2S callback.

void FLEXIO\_I2S\_TransferSetFormat(*FLEXIO\_I2S\_Type* \*base, *flexio\_i2s\_handle\_t* \*handle, *flexio\_i2s\_format\_t* \*format, uint32\_t srcClock\_Hz)

Configures the FlexIO I2S audio format.

Audio format can be changed at run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

#### Parameters

- base – Pointer to FLEXIO\_I2S\_Type structure.
- handle – FlexIO I2S handle pointer.
- format – Pointer to audio data format structure.
- srcClock\_Hz – FlexIO I2S bit clock source frequency in Hz. This parameter should be 0 while in slave mode.

```
void FLEXIO_I2S_TransferRxCreateHandle(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle,  
                                       flexio_i2s_callback_t callback, void *userData)
```

Initializes the FlexIO I2S receive handle.

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

#### Parameters

- base – Pointer to *FLEXIO\_I2S\_Type* structure.
- handle – Pointer to *flexio\_i2s\_handle\_t* structure to store the transfer state.
- callback – FlexIO I2S callback function, which is called while finished a block.
- userData – User parameter for the FlexIO I2S callback.

```
status_t FLEXIO_I2S_TransferSendNonBlocking(FLEXIO_I2S_Type *base, flexio_i2s_handle_t  
                                             *handle, flexio_i2s_transfer_t *xfer)
```

Performs an interrupt non-blocking send transfer on FlexIO I2S.

---

**Note:** The API returns immediately after transfer initiates. Call *FLEXIO\_I2S\_GetRemainingBytes* to poll the transfer status and check whether the transfer is finished. If the return status is 0, the transfer is finished.

---

#### Parameters

- base – Pointer to *FLEXIO\_I2S\_Type* structure.
- handle – Pointer to *flexio\_i2s\_handle\_t* structure which stores the transfer state
- xfer – Pointer to *flexio\_i2s\_transfer\_t* structure

#### Return values

- *kStatus\_Success* – Successfully start the data transmission.
- *kStatus\_FLEXIO\_I2S\_TxBusy* – Previous transmission still not finished, data not all written to TX register yet.
- *kStatus\_InvalidArgument* – The input parameter is invalid.

```
status_t FLEXIO_I2S_TransferReceiveNonBlocking(FLEXIO_I2S_Type *base, flexio_i2s_handle_t  
                                                *handle, flexio_i2s_transfer_t *xfer)
```

Performs an interrupt non-blocking receive transfer on FlexIO I2S.

---

**Note:** The API returns immediately after transfer initiates. Call *FLEXIO\_I2S\_GetRemainingBytes* to poll the transfer status to check whether the transfer is finished. If the return status is 0, the transfer is finished.

---

#### Parameters

- base – Pointer to *FLEXIO\_I2S\_Type* structure.
- handle – Pointer to *flexio\_i2s\_handle\_t* structure which stores the transfer state
- xfer – Pointer to *flexio\_i2s\_transfer\_t* structure

#### Return values

- *kStatus\_Success* – Successfully start the data receive.

- `kStatus_FLEXIO_I2S_RxBusy` – Previous receive still not finished.
- `kStatus_InvalidArgument` – The input parameter is invalid.

`void FLEXIO_I2S_TransferAbortSend(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle)`  
Aborts the current send.

---

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

---

#### Parameters

- `base` – Pointer to `FLEXIO_I2S_Type` structure.
- `handle` – Pointer to `flexio_i2s_handle_t` structure which stores the transfer state

`void FLEXIO_I2S_TransferAbortReceive(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle)`  
Aborts the current receive.

---

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

---

#### Parameters

- `base` – Pointer to `FLEXIO_I2S_Type` structure.
- `handle` – Pointer to `flexio_i2s_handle_t` structure which stores the transfer state

`status_t FLEXIO_I2S_TransferGetSendCount(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, size_t *count)`

Gets the remaining bytes to be sent.

#### Parameters

- `base` – Pointer to `FLEXIO_I2S_Type` structure.
- `handle` – Pointer to `flexio_i2s_handle_t` structure which stores the transfer state
- `count` – Bytes sent.

#### Return values

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

`status_t FLEXIO_I2S_TransferGetReceiveCount(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, size_t *count)`

Gets the remaining bytes to be received.

#### Parameters

- `base` – Pointer to `FLEXIO_I2S_Type` structure.
- `handle` – Pointer to `flexio_i2s_handle_t` structure which stores the transfer state
- `count` – Bytes recieved.

#### Return values

- kStatus\_Success – Succeed get the transfer count.
- kStatus\_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

**Returns**

count Bytes received.

void FLEXIO\_I2S\_TransferTxHandleIRQ(void \*i2sBase, void \*i2sHandle)

Tx interrupt handler.

**Parameters**

- i2sBase – Pointer to FLEXIO\_I2S\_Type structure.
- i2sHandle – Pointer to flexio\_i2s\_handle\_t structure

void FLEXIO\_I2S\_TransferRxHandleIRQ(void \*i2sBase, void \*i2sHandle)

Rx interrupt handler.

**Parameters**

- i2sBase – Pointer to FLEXIO\_I2S\_Type structure.
- i2sHandle – Pointer to flexio\_i2s\_handle\_t structure.

FSL\_FLEXIO\_I2S\_DRIVER\_VERSION

FlexIO I2S driver version 2.2.2.

FlexIO I2S transfer status.

*Values:*

enumerator kStatus\_FLEXIO\_I2S\_Idle

FlexIO I2S is in idle state

enumerator kStatus\_FLEXIO\_I2S\_TxBusy

FlexIO I2S Tx is busy

enumerator kStatus\_FLEXIO\_I2S\_RxBusy

FlexIO I2S Rx is busy

enumerator kStatus\_FLEXIO\_I2S\_Error

FlexIO I2S error occurred

enumerator kStatus\_FLEXIO\_I2S\_QueueFull

FlexIO I2S transfer queue is full.

enumerator kStatus\_FLEXIO\_I2S\_Timeout

FlexIO I2S timeout polling status flags.

enum \_flexio\_i2s\_master\_slave

Master or slave mode.

*Values:*

enumerator kFLEXIO\_I2S\_Master

Master mode

enumerator kFLEXIO\_I2S\_Slave

Slave mode

\_flexio\_i2s\_interrupt\_enable Define FlexIO FlexIO I2S interrupt mask.

*Values:*

enumerator kFLEXIO\_I2S\_TxDataRegEmptyInterruptEnable  
Transmit buffer empty interrupt enable.

enumerator kFLEXIO\_I2S\_RxDataRegFullInterruptEnable  
Receive buffer full interrupt enable.

\_flexio\_i2s\_status\_flags Define FlexIO FlexIO I2S status mask.

*Values:*

enumerator kFLEXIO\_I2S\_TxDataRegEmptyFlag  
Transmit buffer empty flag.

enumerator kFLEXIO\_I2S\_RxDataRegFullFlag  
Receive buffer full flag.

enum \_flexio\_i2s\_sample\_rate  
Audio sample rate.

*Values:*

enumerator kFLEXIO\_I2S\_SampleRate8KHz  
Sample rate 8000Hz

enumerator kFLEXIO\_I2S\_SampleRate11025Hz  
Sample rate 11025Hz

enumerator kFLEXIO\_I2S\_SampleRate12KHz  
Sample rate 12000Hz

enumerator kFLEXIO\_I2S\_SampleRate16KHz  
Sample rate 16000Hz

enumerator kFLEXIO\_I2S\_SampleRate22050Hz  
Sample rate 22050Hz

enumerator kFLEXIO\_I2S\_SampleRate24KHz  
Sample rate 24000Hz

enumerator kFLEXIO\_I2S\_SampleRate32KHz  
Sample rate 32000Hz

enumerator kFLEXIO\_I2S\_SampleRate44100Hz  
Sample rate 44100Hz

enumerator kFLEXIO\_I2S\_SampleRate48KHz  
Sample rate 48000Hz

enumerator kFLEXIO\_I2S\_SampleRate96KHz  
Sample rate 96000Hz

enum \_flexio\_i2s\_word\_width  
Audio word width.

*Values:*

enumerator kFLEXIO\_I2S\_WordWidth8bits  
Audio data width 8 bits

enumerator kFLEXIO\_I2S\_WordWidth16bits  
Audio data width 16 bits

enumerator `kFLEXIO_I2S_WordWidth24bits`  
Audio data width 24 bits

enumerator `kFLEXIO_I2S_WordWidth32bits`  
Audio data width 32 bits

typedef struct *flexio\_i2s\_type* FLEXIO\_I2S\_Type  
Define FlexIO I2S access structure typedef.

typedef enum *flexio\_i2s\_master\_slave* flexio\_i2s\_master\_slave\_t  
Master or slave mode.

typedef struct *flexio\_i2s\_config* flexio\_i2s\_config\_t  
FlexIO I2S configure structure.

typedef struct *flexio\_i2s\_format* flexio\_i2s\_format\_t  
FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx.

typedef enum *flexio\_i2s\_sample\_rate* flexio\_i2s\_sample\_rate\_t  
Audio sample rate.

typedef enum *flexio\_i2s\_word\_width* flexio\_i2s\_word\_width\_t  
Audio word width.

typedef struct *flexio\_i2s\_transfer* flexio\_i2s\_transfer\_t  
Define FlexIO I2S transfer structure.

typedef struct *flexio\_i2s\_handle* flexio\_i2s\_handle\_t

typedef void (\*flexio\_i2s\_callback\_t)(FLEXIO\_I2S\_Type \*base, flexio\_i2s\_handle\_t \*handle, status\_t status, void \*userData)  
FlexIO I2S xfer callback prototype.

I2S\_RETRY\_TIMES  
Retry times for waiting flag.

FLEXIO\_I2S\_XFER\_QUEUE\_SIZE  
FlexIO I2S transfer queue size, user can refine it according to use case.

struct *flexio\_i2s\_type*  
*#include <fsl\_flexio\_i2s.h>* Define FlexIO I2S access structure typedef.

### Public Members

FLEXIO\_Type \*flexioBase  
FlexIO base pointer

uint8\_t txPinIndex  
Tx data pin index in FlexIO pins

uint8\_t rxPinIndex  
Rx data pin index

uint8\_t bclkPinIndex  
Bit clock pin index

uint8\_t fsPinIndex  
Frame sync pin index

uint8\_t txShifterIndex  
Tx data shifter index

```
uint8_t rxShifterIndex
    Rx data shifter index
uint8_t bclkTimerIndex
    Bit clock timer index
uint8_t fsTimerIndex
    Frame sync timer index
struct _flexio_i2s_config
    #include <fsl_flexio_i2s.h> FlexIO I2S configure structure.
```

### Public Members

```
bool enableI2S
    Enable FlexIO I2S
flexio_i2s_master_slave_t masterSlave
    Master or slave
flexio_pin_polarity_t txPinPolarity
    Tx data pin polarity, active high or low
flexio_pin_polarity_t rxPinPolarity
    Rx data pin polarity
flexio_pin_polarity_t bclkPinPolarity
    Bit clock pin polarity
flexio_pin_polarity_t fsPinPolarity
    Frame sync pin polarity
flexio_shifter_timer_polarity_t txTimerPolarity
    Tx data valid on bclk rising or falling edge
flexio_shifter_timer_polarity_t rxTimerPolarity
    Rx data valid on bclk rising or falling edge
struct _flexio_i2s_format
    #include <fsl_flexio_i2s.h> FlexIO I2S audio format, FlexIO I2S only support the same format
    in Tx and Rx.
```

### Public Members

```
uint8_t bitWidth
    Bit width of audio data, always 8/16/24/32 bits
uint32_t sampleRate_Hz
    Sample rate of the audio data
struct _flexio_i2s_transfer
    #include <fsl_flexio_i2s.h> Define FlexIO I2S transfer structure.
```

### Public Members

```
uint8_t *data
    Data buffer start pointer
```

size\_t dataSize  
Bytes to be transferred.

struct \_flexio\_i2s\_handle  
*#include <fsl\_flexio\_i2s.h>* Define FlexIO I2S handle structure.

### Public Members

uint32\_t state  
Internal state

*flexio\_i2s\_callback\_t* callback  
Callback function called at transfer event

void \*userData  
Callback parameter passed to callback function

uint8\_t bitWidth  
Bit width for transfer, 8/16/24/32bits

*flexio\_i2s\_transfer\_t* queue[(4U)]  
Transfer queue storing queued transfer

size\_t transferSize[(4U)]  
Data bytes need to transfer

volatile uint8\_t queueUser  
Index for user to queue transfer

volatile uint8\_t queueDriver  
Index for driver to get the transfer data and size

## 2.18 FlexIO SPI Driver

void FLEXIO\_SPI\_MasterInit(*FLEXIO\_SPI\_Type* \*base, *flexio\_spi\_master\_config\_t* \*masterConfig, uint32\_t srcClock\_Hz)

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI master hardware, and configures the FlexIO SPI with FlexIO SPI master configuration. The configuration structure can be filled by the user, or be set with default values by the FLEXIO\_SPI\_MasterGetDefaultConfig().

### Example

```
FLEXIO_SPI_Type spiDev = {  
    .flexioBase = FLEXIO,  
    .SDOPinIndex = 0,  
    .SDIPinIndex = 1,  
    .SCKPinIndex = 2,  
    .CSnPinIndex = 3,  
    .shifterIndex = {0,1},  
    .timerIndex = {0,1}  
};  
flexio_spi_master_config_t config = {  
    .enableMaster = true,  
    .enableInDoze = false,  
    .enableInDebug = true,
```

(continues on next page)



(continued from previous page)

```
.enableFastAccess = false,
.baudRate_Bps = 500000,
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
.direction = kFLEXIO_SPI_MsbFirst,
.dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_MasterInit(&spiDev, &config, srcClock_Hz);
```

**Note:** 1.FlexIO SPI master only support CPOL = 0, which means clock inactive low. 2.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI master communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by  $2*2=4$ . If FlexIO SPI master communicates with FlexIO SPI slave, the maximum baud rate is FlexIO clock frequency divided by  $(1.5+2.5)*2=8$ .

### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- masterConfig – Pointer to the flexio\_spi\_master\_config\_t structure.
- srcClock\_Hz – FlexIO source clock in Hz.

void FLEXIO\_SPI\_MasterDeinit(*FLEXIO\_SPI\_Type* \*base)

Resets the FlexIO SPI timer and shifter config.

### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type.

void FLEXIO\_SPI\_MasterGetDefaultConfig(*flexio\_spi\_master\_config\_t* \*masterConfig)

Gets the default configuration to configure the FlexIO SPI master. The configuration can be used directly by calling the FLEXIO\_SPI\_MasterConfigure(). Example:

```
flexio_spi_master_config_t masterConfig;
FLEXIO_SPI_MasterGetDefaultConfig(&masterConfig);
```

### Parameters

- masterConfig – Pointer to the flexio\_spi\_master\_config\_t structure.

void FLEXIO\_SPI\_SlaveInit(*FLEXIO\_SPI\_Type* \*base, *flexio\_spi\_slave\_config\_t* \*slaveConfig)

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI slave hardware configuration, and configures the FlexIO SPI with FlexIO SPI slave configuration. The configuration structure can be filled by the user, or be set with default values by the FLEXIO\_SPI\_SlaveGetDefaultConfig().

**Note:** 1.Only one timer is needed in the FlexIO SPI slave. As a result, the second timer index is ignored. 2.FlexIO SPI slave only support CPOL = 0, which means clock inactive low. 3.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI slave communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by  $3*2=6$ . If FlexIO SPI slave communicates with FlexIO SPI master, the maximum baud rate is FlexIO clock frequency divided by  $(1.5+2.5)*2=8$ . Example

```
FLEXIO_SPI_Type spiDev = {
.flexioBase = FLEXIO,
.SDOPinIndex = 0,
```

(continues on next page)

(continued from previous page)

```
.SDIPinIndex = 1,  
.SCKPinIndex = 2,  
.CSnPinIndex = 3,  
.shifterIndex = {0,1},  
.timerIndex = {0}  
};  
flexio_spi_slave_config_t config = {  
.enableSlave = true,  
.enableInDoze = false,  
.enableInDebug = true,  
.enableFastAccess = false,  
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,  
.direction = kFLEXIO_SPI_MsbFirst,  
.dataMode = kFLEXIO_SPI_8BitMode  
};  
FLEXIO_SPI_SlaveInit(&spiDev, &config);
```

---

### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- slaveConfig – Pointer to the flexio\_spi\_slave\_config\_t structure.

void FLEXIO\_SPI\_SlaveDeinit(*FLEXIO\_SPI\_Type* \*base)

Gates the FlexIO clock.

### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type.

void FLEXIO\_SPI\_SlaveGetDefaultConfig(*flexio\_spi\_slave\_config\_t* \*slaveConfig)

Gets the default configuration to configure the FlexIO SPI slave. The configuration can be used directly for calling the FLEXIO\_SPI\_SlaveConfigure(). Example:

```
flexio_spi_slave_config_t slaveConfig;  
FLEXIO_SPI_SlaveGetDefaultConfig(&slaveConfig);
```

### Parameters

- slaveConfig – Pointer to the flexio\_spi\_slave\_config\_t structure.

uint32\_t FLEXIO\_SPI\_GetStatusFlags(*FLEXIO\_SPI\_Type* \*base)

Gets FlexIO SPI status flags.

### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.

### Returns

status flag; Use the status flag to AND the following flag mask and get the status.

- kFLEXIO\_SPI\_TxEmptyFlag
- kFLEXIO\_SPI\_RxEmptyFlag

void FLEXIO\_SPI\_ClearStatusFlags(*FLEXIO\_SPI\_Type* \*base, uint32\_t mask)

Clears FlexIO SPI status flags.

### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.

- **mask** – status flag The parameter can be any combination of the following values:
  - kFLEXIO\_SPI\_TxEmptyFlag
  - kFLEXIO\_SPI\_RxEmptyFlag

void FLEXIO\_SPI\_EnableInterrupts(*FLEXIO\_SPI\_Type* \*base, uint32\_t mask)

Enables the FlexIO SPI interrupt.

This function enables the FlexIO SPI interrupt.

#### Parameters

- **base** – Pointer to the FLEXIO\_SPI\_Type structure.
- **mask** – interrupt source. The parameter can be any combination of the following values:
  - kFLEXIO\_SPI\_RxFullInterruptEnable
  - kFLEXIO\_SPI\_TxEmptyInterruptEnable

void FLEXIO\_SPI\_DisableInterrupts(*FLEXIO\_SPI\_Type* \*base, uint32\_t mask)

Disables the FlexIO SPI interrupt.

This function disables the FlexIO SPI interrupt.

#### Parameters

- **base** – Pointer to the FLEXIO\_SPI\_Type structure.
- **mask** – interrupt source The parameter can be any combination of the following values:
  - kFLEXIO\_SPI\_RxFullInterruptEnable
  - kFLEXIO\_SPI\_TxEmptyInterruptEnable

void FLEXIO\_SPI\_EnableDMA(*FLEXIO\_SPI\_Type* \*base, uint32\_t mask, bool enable)

Enables/disables the FlexIO SPI transmit DMA. This function enables/disables the FlexIO SPI Tx DMA, which means that asserting the kFLEXIO\_SPI\_TxEmptyFlag does/doesn't trigger the DMA request.

#### Parameters

- **base** – Pointer to the FLEXIO\_SPI\_Type structure.
- **mask** – SPI DMA source.
- **enable** – True means enable DMA, false means disable DMA.

static inline uint32\_t FLEXIO\_SPI\_GetTxDataRegisterAddress(*FLEXIO\_SPI\_Type* \*base,  
flexio\_spi\_shift\_direction\_t  
direction)

Gets the FlexIO SPI transmit data register address for MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

#### Parameters

- **base** – Pointer to the FLEXIO\_SPI\_Type structure.
- **direction** – Shift direction of MSB first or LSB first.

#### Returns

FlexIO SPI transmit data register address.

```
static inline uint32_t FLEXIO_SPI_GetRxDataRegisterAddress(FLEXIO_SPI_Type *base,  
                                                         flexio_spi_shift_direction_t  
                                                         direction)
```

Gets the FlexIO SPI receive data register address for the MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

#### Parameters

- base – Pointer to the *FLEXIO\_SPI\_Type* structure.
- direction – Shift direction of MSB first or LSB first.

#### Returns

FlexIO SPI receive data register address.

```
static inline void FLEXIO_SPI_Enable(FLEXIO_SPI_Type *base, bool enable)
```

Enables/disables the FlexIO SPI module operation.

#### Parameters

- base – Pointer to the *FLEXIO\_SPI\_Type*.
- enable – True to enable, false does not have any effect.

```
void FLEXIO_SPI_MasterSetBaudRate(FLEXIO_SPI_Type *base, uint32_t baudRate_Bps,  
                                  uint32_t srcClockHz)
```

Sets baud rate for the FlexIO SPI transfer, which is only used for the master.

#### Parameters

- base – Pointer to the *FLEXIO\_SPI\_Type* structure.
- baudRate\_Bps – Baud Rate needed in Hz.
- srcClockHz – SPI source clock frequency in Hz.

```
static inline void FLEXIO_SPI_WriteData(FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t  
                                       direction, uint32_t data)
```

Writes one byte of data, which is sent using the MSB method.

---

**Note:** This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

---

#### Parameters

- base – Pointer to the *FLEXIO\_SPI\_Type* structure.
- direction – Shift direction of MSB first or LSB first.
- data – 8/16/32 bit data.

```
static inline uint32_t FLEXIO_SPI_ReadData(FLEXIO_SPI_Type *base,  
                                           flexio_spi_shift_direction_t direction)
```

Reads 8 bit/16 bit data.

---

**Note:** This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

---

#### Parameters

- base – Pointer to the *FLEXIO\_SPI\_Type* structure.

- direction – Shift direction of MSB first or LSB first.

**Returns**

8 bit/16 bit data received.

*status\_t* FLEXIO\_SPI\_WriteBlocking(*FLEXIO\_SPI\_Type* \*base, *flexio\_spi\_shift\_direction\_t* direction, const uint8\_t \*buffer, size\_t size)

Sends a buffer of data bytes.

---

**Note:** This function blocks using the polling method until all bytes have been sent.

---

**Parameters**

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The data bytes to send.
- size – The number of data bytes to send.

**Return values**

- kStatus\_Success – Successfully create the handle.
- kStatus\_FLEXIO\_SPI\_Timeout – The transfer timed out and was aborted.

*status\_t* FLEXIO\_SPI\_ReadBlocking(*FLEXIO\_SPI\_Type* \*base, *flexio\_spi\_shift\_direction\_t* direction, uint8\_t \*buffer, size\_t size)

Receives a buffer of bytes.

---

**Note:** This function blocks using the polling method until all bytes have been received.

---

**Parameters**

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The buffer to store the received bytes.
- size – The number of data bytes to be received.

**Return values**

- kStatus\_Success – Successfully create the handle.
- kStatus\_FLEXIO\_SPI\_Timeout – The transfer timed out and was aborted.

*status\_t* FLEXIO\_SPI\_MasterTransferBlocking(*FLEXIO\_SPI\_Type* \*base, *flexio\_spi\_transfer\_t* \*xfer)

Receives a buffer of bytes.

---

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

---

**Parameters**

- base – pointer to FLEXIO\_SPI\_Type structure
- xfer – FlexIO SPI transfer structure, see flexio\_spi\_transfer\_t.

**Return values**

- kStatus\_Success – Successfully create the handle.

- kStatus\_FLEXIO\_SPI\_Timeout – The transfer timed out and was aborted.

void FLEXIO\_SPI\_FlushShifters(*FLEXIO\_SPI\_Type* \*base)

Flush tx/rx shifters.

#### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.

*status\_t* FLEXIO\_SPI\_MasterTransferCreateHandle(*FLEXIO\_SPI\_Type* \*base,  
flexio\_spi\_master\_handle\_t \*handle,  
flexio\_spi\_master\_transfer\_callback\_t  
callback, void \*userData)

Initializes the FlexIO SPI Master handle, which is used in transactional functions.

#### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- handle – Pointer to the flexio\_spi\_master\_handle\_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

#### Return values

- kStatus\_Success – Successfully create the handle.
- kStatus\_OutOfRange – The FlexIO type/handle/ISR table out of range.

*status\_t* FLEXIO\_SPI\_MasterTransferNonBlocking(*FLEXIO\_SPI\_Type* \*base,  
flexio\_spi\_master\_handle\_t \*handle,  
flexio\_spi\_transfer\_t \*xfer)

Master transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

#### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- handle – Pointer to the flexio\_spi\_master\_handle\_t structure to store the transfer state.
- xfer – FlexIO SPI transfer structure. See flexio\_spi\_transfer\_t.

#### Return values

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

void FLEXIO\_SPI\_MasterTransferAbort(*FLEXIO\_SPI\_Type* \*base, flexio\_spi\_master\_handle\_t  
\*handle)

Aborts the master data transfer, which used IRQ.

#### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- handle – Pointer to the flexio\_spi\_master\_handle\_t structure to store the transfer state.

```
status_t FLEXIO_SPI_MasterTransferGetCount(FLEXIO_SPI_Type *base,  
                                            flexio_spi_master_handle_t *handle, size_t  
                                            *count)
```

Gets the data transfer status which used IRQ.

#### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- handle – Pointer to the flexio\_spi\_master\_handle\_t structure to store the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

#### Return values

- kStatus\_InvalidArgument – count is Invalid.
- kStatus\_Success – Successfully return the count.

```
void FLEXIO_SPI_MasterTransferHandleIRQ(void *spiType, void *spiHandle)
```

FlexIO SPI master IRQ handler function.

#### Parameters

- spiType – Pointer to the FLEXIO\_SPI\_Type structure.
- spiHandle – Pointer to the flexio\_spi\_master\_handle\_t structure to store the transfer state.

```
status_t FLEXIO_SPI_SlaveTransferCreateHandle(FLEXIO_SPI_Type *base,  
                                              flexio_spi_slave_handle_t *handle,  
                                              flexio_spi_slave_transfer_callback_t callback,  
                                              void *userData)
```

Initializes the FlexIO SPI Slave handle, which is used in transactional functions.

#### Parameters

- base – Pointer to the FLEXIO\_SPI\_Type structure.
- handle – Pointer to the flexio\_spi\_slave\_handle\_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

#### Return values

- kStatus\_Success – Successfully create the handle.
- kStatus\_OutOfRange – The FlexIO type/handle/ISR table out of range.

```
status_t FLEXIO_SPI_SlaveTransferNonBlocking(FLEXIO_SPI_Type *base,  
                                              flexio_spi_slave_handle_t *handle,  
                                              flexio_spi_transfer_t *xfer)
```

Slave transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

#### Parameters

- handle – Pointer to the flexio\_spi\_slave\_handle\_t structure to store the transfer state.
- base – Pointer to the FLEXIO\_SPI\_Type structure.
- xfer – FlexIO SPI transfer structure. See flexio\_spi\_transfer\_t.

**Return values**

- `kStatus__Success` – Successfully start a transfer.
- `kStatus__InvalidArgument` – Input argument is invalid.
- `kStatus__FLEXIO_SPI_Busy` – SPI is not idle; it is running another transfer.

```
static inline void FLEXIO_SPI_SlaveTransferAbort(FLEXIO_SPI_Type *base,  
                                                flexio_spi_slave_handle_t *handle)
```

Aborts the slave data transfer which used IRQ, share same API with master.

**Parameters**

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.

```
static inline status_t FLEXIO_SPI_SlaveTransferGetCount(FLEXIO_SPI_Type *base,  
                                                       flexio_spi_slave_handle_t *handle,  
                                                       size_t *count)
```

Gets the data transfer status which used IRQ, share same API with master.

**Parameters**

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

**Return values**

- `kStatus__InvalidArgument` – count is Invalid.
- `kStatus__Success` – Successfully return the count.

```
void FLEXIO_SPI_SlaveTransferHandleIRQ(void *spiType, void *spiHandle)
```

FlexIO SPI slave IRQ handler function.

**Parameters**

- `spiType` – Pointer to the `FLEXIO_SPI_Type` structure.
- `spiHandle` – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.

```
FSL_FLEXIO_SPI_DRIVER_VERSION
```

FlexIO SPI driver version.

Error codes for the FlexIO SPI driver.

*Values:*

```
enumerator kStatus_FLEXIO_SPI_Busy
```

FlexIO SPI is busy.

```
enumerator kStatus_FLEXIO_SPI_Idle
```

SPI is idle

```
enumerator kStatus_FLEXIO_SPI_Error
```

FlexIO SPI error.

```
enumerator kStatus_FLEXIO_SPI_Timeout
```

FlexIO SPI timeout polling status flags.



enum \_flexio\_spi\_clock\_phase

FlexIO SPI clock phase configuration.

*Values:*

enumerator kFLEXIO\_SPI\_ClockPhaseFirstEdge

First edge on SPCK occurs at the middle of the first cycle of a data transfer.

enumerator kFLEXIO\_SPI\_ClockPhaseSecondEdge

First edge on SPCK occurs at the start of the first cycle of a data transfer.

enum \_flexio\_spi\_shift\_direction

FlexIO SPI data shifter direction options.

*Values:*

enumerator kFLEXIO\_SPI\_MsbFirst

Data transfers start with most significant bit.

enumerator kFLEXIO\_SPI\_LsbFirst

Data transfers start with least significant bit.

enum \_flexio\_spi\_data\_bitcount\_mode

FlexIO SPI data length mode options.

*Values:*

enumerator kFLEXIO\_SPI\_8BitMode

8-bit data transmission mode.

enumerator kFLEXIO\_SPI\_16BitMode

16-bit data transmission mode.

enumerator kFLEXIO\_SPI\_32BitMode

32-bit data transmission mode.

enum \_flexio\_spi\_interrupt\_enable

Define FlexIO SPI interrupt mask.

*Values:*

enumerator kFLEXIO\_SPI\_TxEmptyInterruptEnable

Transmit buffer empty interrupt enable.

enumerator kFLEXIO\_SPI\_RxFullInterruptEnable

Receive buffer full interrupt enable.

enum \_flexio\_spi\_status\_flags

Define FlexIO SPI status mask.

*Values:*

enumerator kFLEXIO\_SPI\_TxBufferEmptyFlag

Transmit buffer empty flag.

enumerator kFLEXIO\_SPI\_RxBufferFullFlag

Receive buffer full flag.

enum \_flexio\_spi\_dma\_enable

Define FlexIO SPI DMA mask.

*Values:*

enumerator kFLEXIO\_SPI\_TxDmaEnable

Tx DMA request source

enumerator kFLEXIO\_SPI\_RxDmaEnable

Rx DMA request source

enumerator kFLEXIO\_SPI\_DmaAllEnable

All DMA request source

enum \_flexio\_spi\_transfer\_flags

Define FlexIO SPI transfer flags.

---

**Note:** Use kFLEXIO\_SPI\_csContinuous and one of the other flags to OR together to form the transfer flag.

---

*Values:*

enumerator kFLEXIO\_SPI\_8bitMsb

FlexIO SPI 8-bit MSB first

enumerator kFLEXIO\_SPI\_8bitLsb

FlexIO SPI 8-bit LSB first

enumerator kFLEXIO\_SPI\_16bitMsb

FlexIO SPI 16-bit MSB first

enumerator kFLEXIO\_SPI\_16bitLsb

FlexIO SPI 16-bit LSB first

enumerator kFLEXIO\_SPI\_32bitMsb

FlexIO SPI 32-bit MSB first

enumerator kFLEXIO\_SPI\_32bitLsb

FlexIO SPI 32-bit LSB first

enumerator kFLEXIO\_SPI\_csContinuous

Enable the CS signal continuous mode

typedef enum \_flexio\_spi\_clock\_phase flexio\_spi\_clock\_phase\_t

FlexIO SPI clock phase configuration.

typedef enum \_flexio\_spi\_shift\_direction flexio\_spi\_shift\_direction\_t

FlexIO SPI data shifter direction options.

typedef enum \_flexio\_spi\_data\_bitcount\_mode flexio\_spi\_data\_bitcount\_mode\_t

FlexIO SPI data length mode options.

typedef struct \_flexio\_spi\_type FLEXIO\_SPI\_Type

Define FlexIO SPI access structure typedef.

typedef struct \_flexio\_spi\_master\_config flexio\_spi\_master\_config\_t

Define FlexIO SPI master configuration structure.

typedef struct \_flexio\_spi\_slave\_config flexio\_spi\_slave\_config\_t

Define FlexIO SPI slave configuration structure.

typedef struct \_flexio\_spi\_transfer flexio\_spi\_transfer\_t

Define FlexIO SPI transfer structure.

typedef struct \_flexio\_spi\_master\_handle flexio\_spi\_master\_handle\_t

typedef for flexio\_spi\_master\_handle\_t in advance.

typedef flexio\_spi\_master\_handle\_t flexio\_spi\_slave\_handle\_t

Slave handle is the same with master handle.

```
typedef void (*flexio_spi_master_transfer_callback_t)(FLEXIO_SPI_Type *base,  
flexio_spi_master_handle_t *handle, status_t status, void *userData)
```

FlexIO SPI master callback for finished transmit.

```
typedef void (*flexio_spi_slave_transfer_callback_t)(FLEXIO_SPI_Type *base,  
flexio_spi_slave_handle_t *handle, status_t status, void *userData)
```

FlexIO SPI slave callback for finished transmit.

```
FLEXIO_SPI_DUMMYDATA
```

FlexIO SPI dummy transfer data, the data is sent while txData is NULL.

```
SPI_RETRY_TIMES
```

Retry times for waiting flag.

```
FLEXIO_SPI_XFER_DATA_FORMAT(flag)
```

Get the transfer data format of width and bit order.

```
struct _flexio_spi_type
```

*#include <fsl\_flexio\_spi.h>* Define FlexIO SPI access structure typedef.

### Public Members

```
FLEXIO_Type *flexioBase
```

FlexIO base pointer.

```
uint8_t SDOPinIndex
```

Pin select for data output. To set SDO pin in Hi-Z state, user needs to mux the pin as GPIO input and disable all pull up/down in application.

```
uint8_t SDIPinIndex
```

Pin select for data input.

```
uint8_t SCKPinIndex
```

Pin select for clock.

```
uint8_t CSnPinIndex
```

Pin select for enable.

```
uint8_t shifterIndex[2]
```

Shifter index used in FlexIO SPI.

```
uint8_t timerIndex[2]
```

Timer index used in FlexIO SPI.

```
struct _flexio_spi_master_config
```

*#include <fsl\_flexio\_spi.h>* Define FlexIO SPI master configuration structure.

### Public Members

```
bool enableMaster
```

Enable/disable FlexIO SPI master after configuration.

```
bool enableInDoze
```

Enable/disable FlexIO operation in doze mode.

```
bool enableInDebug
```

Enable/disable FlexIO operation in debug mode.

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32\_t baudRate\_Bps

Baud rate in Bps.

*flexio\_spi\_clock\_phase\_t* phase

Clock phase.

*flexio\_spi\_data\_bitcount\_mode\_t* dataMode

8bit or 16bit mode.

struct *\_flexio\_spi\_slave\_config*

*#include <fsl\_flexio\_spi.h>* Define FlexIO SPI slave configuration structure.

### Public Members

bool enableSlave

Enable/disable FlexIO SPI slave after configuration.

bool enableInDoze

Enable/disable FlexIO operation in doze mode.

bool enableInDebug

Enable/disable FlexIO operation in debug mode.

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

*flexio\_spi\_clock\_phase\_t* phase

Clock phase.

*flexio\_spi\_data\_bitcount\_mode\_t* dataMode

8bit or 16bit mode.

struct *\_flexio\_spi\_transfer*

*#include <fsl\_flexio\_spi.h>* Define FlexIO SPI transfer structure.

### Public Members

const uint8\_t \*txData

Send buffer.

uint8\_t \*rxData

Receive buffer.

size\_t dataSize

Transfer bytes.

uint8\_t flags

FlexIO SPI control flag, MSB first or LSB first.

struct *\_flexio\_spi\_master\_handle*

*#include <fsl\_flexio\_spi.h>* Define FlexIO SPI handle structure.

## Public Members

`const uint8_t *txData`  
Transfer buffer.

`uint8_t *rxData`  
Receive buffer.

`size_t transferSize`  
Total bytes to be transferred.

`volatile size_t txRemainingBytes`  
Send data remaining in bytes.

`volatile size_t rxRemainingBytes`  
Receive data remaining in bytes.

`volatile uint32_t state`  
FlexIO SPI internal state.

`uint8_t bytePerFrame`  
SPI mode, 2bytes or 1byte in a frame

`flexio_spi_shift_direction_t direction`  
Shift direction.

`flexio_spi_master_transfer_callback_t callback`  
FlexIO SPI callback.

`void *userData`  
Callback parameter.

`bool isCsContinuous`  
Is current transfer using CS continuous mode.

`uint32_t timer1Cfg`  
TIMER1 TIMCFG regiser value backup.

## 2.19 FlexIO UART Driver

`status_t FLEXIO_UART_Init(FLEXIO_UART_Type *base, const flexio_uart_config_t *userConfig, uint32_t srcClock_Hz)`

Ungates the FlexIO clock, resets the FlexIO module, configures FlexIO UART hardware, and configures the FlexIO UART with FlexIO UART configuration. The configuration structure can be filled by the user or be set with default values by `FLEXIO_UART_GetDefaultConfig()`.

### Example

```
FLEXIO_UART_Type base = {
    .flexioBase = FLEXIO,
    .TxPinIndex = 0,
    .RxPinIndex = 1,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_uart_config_t config = {
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .baudRate_Bps = 115200U,
```

(continues on next page)

(continued from previous page)

```
.bitCountPerChar = 8  
};  
FLEXIO_UART_Init(base, &config, srcClock_Hz);
```

**Parameters**

- base – Pointer to the FLEXIO\_UART\_Type structure.
- userConfig – Pointer to the flexio\_uart\_config\_t structure.
- srcClock\_Hz – FlexIO source clock in Hz.

**Return values**

- kStatus\_Success – Configuration success.
- kStatus\_FLEXIO\_UART\_BaudrateNotSupport – Baudrate is not supported for current clock source frequency.

```
void FLEXIO_UART_Deinit(FLEXIO_UART_Type *base)
```

Resets the FlexIO UART shifter and timer config.

---

**Note:** After calling this API, call the FLEXIO\_UART\_Init to use the FlexIO UART module.

---

**Parameters**

- base – Pointer to FLEXIO\_UART\_Type structure

```
void FLEXIO_UART_GetDefaultConfig(flexio_uart_config_t *userConfig)
```

Gets the default configuration to configure the FlexIO UART. The configuration can be used directly for calling the FLEXIO\_UART\_Init(). Example:

```
flexio_uart_config_t config;  
FLEXIO_UART_GetDefaultConfig(&userConfig);
```

**Parameters**

- userConfig – Pointer to the flexio\_uart\_config\_t structure.

```
uint32_t FLEXIO_UART_GetStatusFlags(FLEXIO_UART_Type *base)
```

Gets the FlexIO UART status flags.

**Parameters**

- base – Pointer to the FLEXIO\_UART\_Type structure.

**Returns**

FlexIO UART status flags.

```
void FLEXIO_UART_ClearStatusFlags(FLEXIO_UART_Type *base, uint32_t mask)
```

Gets the FlexIO UART status flags.

**Parameters**

- base – Pointer to the FLEXIO\_UART\_Type structure.
- mask – Status flag. The parameter can be any combination of the following values:
  - kFLEXIO\_UART\_TxDataRegEmptyFlag
  - kFLEXIO\_UART\_RxEmptyFlag

– kFLEXIO\_UART\_RxOverRunFlag

void FLEXIO\_UART\_EnableInterrupts(*FLEXIO\_UART\_Type* \*base, uint32\_t mask)

Enables the FlexIO UART interrupt.

This function enables the FlexIO UART interrupt.

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- mask – Interrupt source.

void FLEXIO\_UART\_DisableInterrupts(*FLEXIO\_UART\_Type* \*base, uint32\_t mask)

Disables the FlexIO UART interrupt.

This function disables the FlexIO UART interrupt.

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- mask – Interrupt source.

static inline uint32\_t FLEXIO\_UART\_GetTxDataRegisterAddress(*FLEXIO\_UART\_Type* \*base)

Gets the FlexIO UART transmit data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.

#### Returns

FlexIO UART transmit data register address.

static inline uint32\_t FLEXIO\_UART\_GetRxDataRegisterAddress(*FLEXIO\_UART\_Type* \*base)

Gets the FlexIO UART receive data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.

#### Returns

FlexIO UART receive data register address.

static inline void FLEXIO\_UART\_EnableTxDMA(*FLEXIO\_UART\_Type* \*base, bool enable)

Enables/disables the FlexIO UART transmit DMA. This function enables/disables the FlexIO UART Tx DMA, which means asserting the kFLEXIO\_UART\_TxDataRegEmptyFlag does/doesn't trigger the DMA request.

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- enable – True to enable, false to disable.

static inline void FLEXIO\_UART\_EnableRxDMA(*FLEXIO\_UART\_Type* \*base, bool enable)

Enables/disables the FlexIO UART receive DMA. This function enables/disables the FlexIO UART Rx DMA, which means asserting kFLEXIO\_UART\_RxDataRegFullFlag does/doesn't trigger the DMA request.

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- enable – True to enable, false to disable.

```
static inline void FLEXIO_UART_Enable(FLEXIO_UART_Type *base, bool enable)
```

Enables/disables the FlexIO UART module operation.

#### Parameters

- base – Pointer to the *FLEXIO\_UART\_Type*.
- enable – True to enable, false does not have any effect.

```
static inline void FLEXIO_UART_WriteByte(FLEXIO_UART_Type *base, const uint8_t *buffer)
```

Writes one byte of data.

---

**Note:** This is a non-blocking API, which returns directly after the data is put into the data register. Ensure that the TxEmptyFlag is asserted before calling this API.

---

#### Parameters

- base – Pointer to the *FLEXIO\_UART\_Type* structure.
- buffer – The data bytes to send.

```
static inline void FLEXIO_UART_ReadByte(FLEXIO_UART_Type *base, uint8_t *buffer)
```

Reads one byte of data.

---

**Note:** This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

---

#### Parameters

- base – Pointer to the *FLEXIO\_UART\_Type* structure.
- buffer – The buffer to store the received bytes.

```
status_t FLEXIO_UART_WriteBlocking(FLEXIO_UART_Type *base, const uint8_t *txData, size_t txSize)
```

Sends a buffer of data bytes.

---

**Note:** This function blocks using the polling method until all bytes have been sent.

---

#### Parameters

- base – Pointer to the *FLEXIO\_UART\_Type* structure.
- txData – The data bytes to send.
- txSize – The number of data bytes to send.

#### Return values

- kStatus\_FLEXIO\_UART\_Timeout – Transmission timed out and was aborted.
- kStatus\_Success – Successfully wrote all data.

```
status_t FLEXIO_UART_ReadBlocking(FLEXIO_UART_Type *base, uint8_t *rxData, size_t rxSize)
```

Receives a buffer of bytes.

---

**Note:** This function blocks using the polling method until all bytes have been received.

---



### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- rxData – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

### Return values

- kStatus\_FLEXIO\_UART\_Timeout – Transmission timed out and was aborted.
- kStatus\_Success – Successfully received all data.

```
status_t FLEXIO_UART_TransferCreateHandle(FLEXIO_UART_Type *base, flexio_uart_handle_t
                                         *handle, flexio_uart_transfer_callback_t callback,
                                         void *userData)
```

Initializes the UART handle.

This function initializes the FlexIO UART handle, which can be used for other FlexIO UART transactional APIs. Call this API once to get the initialized handle.

The UART driver supports the “background” receiving, which means that users can set up a RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn’t call the FLEXIO\_UART\_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as ringBuffer.

### Parameters

- base – to FLEXIO\_UART\_Type structure.
- handle – Pointer to the flexio\_uart\_handle\_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

### Return values

- kStatus\_Success – Successfully create the handle.
- kStatus\_OutOfRange – The FlexIO type/handle/ISR table out of range.

```
void FLEXIO_UART_TransferStartRingBuffer(FLEXIO_UART_Type *base, flexio_uart_handle_t
                                         *handle, uint8_t *ringBuffer, size_t
                                         ringBufferSize)
```

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn’t call the UART\_ReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly.

---

**Note:** When using the RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, only 31 bytes are used for saving data.

---

### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- handle – Pointer to the flexio\_uart\_handle\_t structure to store the transfer state.

- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – Size of the ring buffer.

void FLEXIO\_UART\_TransferStopRingBuffer(*FLEXIO\_UART\_Type* \*base, *flexio\_uart\_handle\_t* \*handle)

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- handle – Pointer to the flexio\_uart\_handle\_t structure to store the transfer state.

*status\_t* FLEXIO\_UART\_TransferSendNonBlocking(*FLEXIO\_UART\_Type* \*base, *flexio\_uart\_handle\_t* \*handle, *flexio\_uart\_transfer\_t* \*xfer)

Transmits a buffer of data using the interrupt method.

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in ISR, the FlexIO UART driver calls the callback function and passes the kStatus\_FLEXIO\_UART\_TxIdle as status parameter.

---

**Note:** The kStatus\_FLEXIO\_UART\_TxIdle is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out.

---

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- handle – Pointer to the flexio\_uart\_handle\_t structure to store the transfer state.
- xfer – FlexIO UART transfer structure. See flexio\_uart\_transfer\_t.

#### Return values

- kStatus\_Success – Successfully starts the data transmission.
- kStatus\_UART\_TxBusy – Previous transmission still not finished, data not written to the TX register.

void FLEXIO\_UART\_TransferAbortSend(*FLEXIO\_UART\_Type* \*base, *flexio\_uart\_handle\_t* \*handle)

Aborts the interrupt-driven data transmit.

This function aborts the interrupt-driven data sending. Get the remainBytes to find out how many bytes are still not sent out.

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- handle – Pointer to the flexio\_uart\_handle\_t structure to store the transfer state.

*status\_t* FLEXIO\_UART\_TransferGetSendCount(*FLEXIO\_UART\_Type* \*base, *flexio\_uart\_handle\_t* \*handle, *size\_t* \*count)

Gets the number of bytes sent.

This function gets the number of bytes sent driven by interrupt.

### Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.
- `count` – Number of bytes sent so far by the non-blocking transaction.

### Return values

- `kStatus_NoTransferInProgress` – transfer has finished or no transfer in progress.
- `kStatus_Success` – Successfully return the count.

```
status_t FLEXIO_UART_TransferReceiveNonBlocking(FLEXIO_UART_Type *base,  
                                                flexio_uart_handle_t *handle,  
                                                flexio_uart_transfer_t *xfer, size_t  
                                                *receivedBytes)
```

Receives a buffer of data using the interrupt method.

This function receives data using the interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in ring buffer is not enough to read, the receive request is saved by the UART driver. When new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter `kStatus_UART_RxIdle`. For example, if the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer, the 5 bytes are copied to `xfer->data`. This function returns with the parameter `receivedBytes` set to 5. For the last 5 bytes, newly arrived data is saved from the `xfer->data[5]`. When 5 bytes are received, the UART driver notifies upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to `xfer->data`. When all data is received, the upper layer is notified.

### Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.
- `xfer` – UART transfer structure. See `flexio_uart_transfer_t`.
- `receivedBytes` – Bytes received from the ring buffer directly.

### Return values

- `kStatus_Success` – Successfully queue the transfer into the transmit queue.
- `kStatus_FLEXIO_UART_RxBusy` – Previous receive request is not finished.

```
void FLEXIO_UART_TransferAbortReceive(FLEXIO_UART_Type *base, flexio_uart_handle_t  
                                     *handle)
```

Aborts the receive data which was using IRQ.

This function aborts the receive data which was using IRQ.

### Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.

*status\_t* FLEXIO\_UART\_TransferGetReceiveCount(*FLEXIO\_UART\_Type* \*base,  
flexio\_uart\_handle\_t \*handle, size\_t \*count)

Gets the number of bytes received.

This function gets the number of bytes received driven by interrupt.

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.
- handle – Pointer to the flexio\_uart\_handle\_t structure to store the transfer state.
- count – Number of bytes received so far by the non-blocking transaction.

#### Return values

- kStatus\_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus\_Success – Successfully return the count.

void FLEXIO\_UART\_TransferHandleIRQ(void \*uartType, void \*uartHandle)

FlexIO UART IRQ handler function.

This function processes the FlexIO UART transmit and receives the IRQ request.

#### Parameters

- uartType – Pointer to the FLEXIO\_UART\_Type structure.
- uartHandle – Pointer to the flexio\_uart\_handle\_t structure to store the transfer state.

void FLEXIO\_UART\_FlushShifters(*FLEXIO\_UART\_Type* \*base)

Flush tx/rx shifters.

#### Parameters

- base – Pointer to the FLEXIO\_UART\_Type structure.

FSL\_FLEXIO\_UART\_DRIVER\_VERSION

FlexIO UART driver version.

Error codes for the UART driver.

*Values:*

enumerator kStatus\_FLEXIO\_UART\_TxBusy

Transmitter is busy.

enumerator kStatus\_FLEXIO\_UART\_RxBusy

Receiver is busy.

enumerator kStatus\_FLEXIO\_UART\_TxIdle

UART transmitter is idle.

enumerator kStatus\_FLEXIO\_UART\_RxIdle

UART receiver is idle.

enumerator kStatus\_FLEXIO\_UART\_ERROR

ERROR happens on UART.

enumerator kStatus\_FLEXIO\_UART\_RxRingBufferOverrun

UART RX software ring buffer overrun.

enumerator kStatus\_FLEXIO\_UART\_RxHardwareOverrun  
UART RX receiver overrun.

enumerator kStatus\_FLEXIO\_UART\_Timeout  
UART times out.

enumerator kStatus\_FLEXIO\_UART\_BaudrateNotSupport  
Baudrate is not supported in current clock source

enum \_flexio\_uart\_bit\_count\_per\_char  
FlexIO UART bit count per char.

*Values:*

enumerator kFLEXIO\_UART\_7BitsPerChar  
7-bit data characters

enumerator kFLEXIO\_UART\_8BitsPerChar  
8-bit data characters

enumerator kFLEXIO\_UART\_9BitsPerChar  
9-bit data characters

enum \_flexio\_uart\_interrupt\_enable  
Define FlexIO UART interrupt mask.

*Values:*

enumerator kFLEXIO\_UART\_TxDataRegEmptyInterruptEnable  
Transmit buffer empty interrupt enable.

enumerator kFLEXIO\_UART\_RxDataRegFullInterruptEnable  
Receive buffer full interrupt enable.

enum \_flexio\_uart\_status\_flags  
Define FlexIO UART status mask.

*Values:*

enumerator kFLEXIO\_UART\_TxDataRegEmptyFlag  
Transmit buffer empty flag.

enumerator kFLEXIO\_UART\_RxDataRegFullFlag  
Receive buffer full flag.

enumerator kFLEXIO\_UART\_RxOverRunFlag  
Receive buffer over run flag.

typedef enum \_flexio\_uart\_bit\_count\_per\_char flexio\_uart\_bit\_count\_per\_char\_t  
FlexIO UART bit count per char.

typedef struct \_flexio\_uart\_type FLEXIO\_UART\_Type  
Define FlexIO UART access structure typedef.

typedef struct \_flexio\_uart\_config flexio\_uart\_config\_t  
Define FlexIO UART user configuration structure.

typedef struct \_flexio\_uart\_transfer flexio\_uart\_transfer\_t  
Define FlexIO UART transfer structure.

typedef struct \_flexio\_uart\_handle flexio\_uart\_handle\_t

```
typedef void (*flexio_uart_transfer_callback_t)(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, status_t status, void *userData)
```

FlexIO UART transfer callback function.

```
UART_RETRY_TIMES
```

Retry times for waiting flag.

```
struct _flexio_uart_type
```

*#include <fsl\_flexio\_uart.h>* Define FlexIO UART access structure typedef.

### Public Members

```
FLEXIO_Type *flexioBase
```

FlexIO base pointer.

```
uint8_t TxPinIndex
```

Pin select for UART\_Tx.

```
uint8_t RxPinIndex
```

Pin select for UART\_Rx.

```
uint8_t shifterIndex[2]
```

Shifter index used in FlexIO UART.

```
uint8_t timerIndex[2]
```

Timer index used in FlexIO UART.

```
struct _flexio_uart_config
```

*#include <fsl\_flexio\_uart.h>* Define FlexIO UART user configuration structure.

### Public Members

```
bool enableUart
```

Enable/disable FlexIO UART TX & RX.

```
bool enableInDoze
```

Enable/disable FlexIO operation in doze mode

```
bool enableInDebug
```

Enable/disable FlexIO operation in debug mode

```
bool enableFastAccess
```

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

```
uint32_t baudRate_Bps
```

Baud rate in Bps.

```
flexio_uart_bit_count_per_char_t bitCountPerChar
```

number of bits, 7/8/9 -bit

```
struct _flexio_uart_transfer
```

*#include <fsl\_flexio\_uart.h>* Define FlexIO UART transfer structure.

### Public Members

```
size_t dataSize
```

Transfer size

struct `_flexio_uart_handle`

*#include <fsl\_flexio\_uart.h>* Define FLEXIO UART handle structure.

### Public Members

const uint8\_t \*volatile txData

Address of remaining data to send.

volatile size\_t txDataSize

Size of the remaining data to send.

uint8\_t \*volatile rxData

Address of remaining data to receive.

volatile size\_t rxDataSize

Size of the remaining data to receive.

size\_t txDataSizeAll

Total bytes to be sent.

size\_t rxDataSizeAll

Total bytes to be received.

uint8\_t \*rxRingBuffer

Start address of the receiver ring buffer.

size\_t rxRingBufferSize

Size of the ring buffer.

volatile uint16\_t rxRingBufferHead

Index for the driver to store received data into ring buffer.

volatile uint16\_t rxRingBufferTail

Index for the user to get data from the ring buffer.

*flexio\_uart\_transfer\_callback\_t* callback

Callback function.

void \*userData

UART callback function parameter.

volatile uint8\_t txState

TX transfer state.

volatile uint8\_t rxState

RX transfer state

union `__unnamed105__`

### 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.

## 2.20 FLEXSPI: Flexible Serial Peripheral Interface Driver

uint32\_t FLEXSPI\_GetInstance(FLEXSPI\_Type \*base)

Get the instance number for FLEXSPI.

### Parameters

- base – FLEXSPI base pointer.

status\_t FLEXSPI\_CheckAndClearError(FLEXSPI\_Type \*base, uint32\_t status)

Check and clear IP command execution errors.

### Parameters

- base – FLEXSPI base pointer.
- status – interrupt status.

void FLEXSPI\_Init(FLEXSPI\_Type \*base, const flexspi\_config\_t \*config)

Initializes the FLEXSPI module and internal state.

This function enables the clock for FLEXSPI and also configures the FLEXSPI with the input configure parameters. Users should call this function before any FLEXSPI operations.

### Parameters

- base – FLEXSPI peripheral base address.
- config – FLEXSPI configure structure.

void FLEXSPI\_GetDefaultConfig(flexspi\_config\_t \*config)

Gets default settings for FLEXSPI.

### Parameters

- config – FLEXSPI configuration structure.

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

Deinitializes the FLEXSPI module.

Clears the FLEXSPI state and FLEXSPI module registers.

### Parameters

- base – FLEXSPI peripheral base address.

void FLEXSPI\_UpdateDllValue(FLEXSPI\_Type \*base, flexspi\_device\_config\_t \*config, flexspi\_port\_t port)

Update FLEXSPI DLL value depending on currently flexspi root clock.

### Parameters

- base – FLEXSPI peripheral base address.
- config – Flash configuration parameters.
- port – FLEXSPI Operation port.

void FLEXSPI\_SetFlashConfig(FLEXSPI\_Type \*base, flexspi\_device\_config\_t \*config, flexspi\_port\_t port)

Configures the connected device parameter.

This function configures the connected device relevant parameters, such as the size, command, and so on. The flash configuration value cannot have a default value. The user needs to configure it according to the connected device.

### Parameters

- base – FLEXSPI peripheral base address.



- config – Flash configuration parameters.
- port – FLEXSPI Operation port.

void FLEXSPI\_SoftwareReset(FLEXSPI\_Type \*base)

Software reset for the FLEXSPI logic.

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

#### Parameters

- base – FLEXSPI peripheral base address.

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

Enables or disables the FLEXSPI module.

#### Parameters

- base – FLEXSPI peripheral base address.
- enable – True means enable FLEXSPI, false means disable.

void FLEXSPI\_UpdateAhbBuffersSettings(FLEXSPI\_Type \*base, flexspi\_ahbBuffers\_ctrl\_t \*ptrAhbBufferCtrl)

Update all AHB buffers' settings, including buffer size, master ID.

#### Parameters

- base – FLEXSPI peripheral base address.
- ptrAhbBufferCtrl – Pointer to structure flexspi\_ahbBuffers\_ctrl\_t which store all AHB buffers' settings.

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

Enables the FLEXSPI interrupts.

#### Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

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

Disable the FLEXSPI interrupts.

#### Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

static inline void FLEXSPI\_EnableTxDMA(FLEXSPI\_Type \*base, bool enable)

Enables or disables FLEXSPI IP Tx FIFO DMA requests.

#### Parameters

- base – FLEXSPI peripheral base address.
- enable – Enable flag for transmit DMA request. Pass true for enable, false for disable.

static inline void FLEXSPI\_EnableRxDMA(FLEXSPI\_Type \*base, bool enable)

Enables or disables FLEXSPI IP Rx FIFO DMA requests.

#### Parameters

- base – FLEXSPI peripheral base address.
- enable – Enable flag for receive DMA request. Pass true for enable, false for disable.

```
static inline uint32_t FLEXSPI_GetTxFifoAddress(FLEXSPI_Type *base)
```

Gets FLEXSPI IP tx fifo address for DMA transfer.

**Parameters**

- base – FLEXSPI peripheral base address.

**Return values**

The – tx fifo address.

```
static inline uint32_t FLEXSPI_GetRxFifoAddress(FLEXSPI_Type *base)
```

Gets FLEXSPI IP rx fifo address for DMA transfer.

**Parameters**

- base – FLEXSPI peripheral base address.

**Return values**

The – rx fifo address.

```
static inline void FLEXSPI_ResetFifos(FLEXSPI_Type *base, bool txFifo, bool rxFifo)
```

Clears the FLEXSPI IP FIFO logic.

**Parameters**

- base – FLEXSPI peripheral base address.
- txFifo – Pass true to reset TX FIFO.
- rxFifo – Pass true to reset RX FIFO.

```
static inline void FLEXSPI_GetFifoCounts(FLEXSPI_Type *base, size_t *txCount, size_t  
                                         *rxCount)
```

Gets the valid data entries in the FLEXSPI FIFOs.

**Parameters**

- base – FLEXSPI peripheral base address.
- txCount – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

```
static inline uint32_t FLEXSPI_GetInterruptStatusFlags(FLEXSPI_Type *base)
```

Get the FLEXSPI interrupt status flags.

**Parameters**

- base – FLEXSPI peripheral base address.

**Return values**

interrupt – status flag, use status flag to AND flexspi\_flags\_t could get the related status.

```
static inline void FLEXSPI_ClearInterruptStatusFlags(FLEXSPI_Type *base, uint32_t mask)
```

Get the FLEXSPI interrupt status flags.

**Parameters**

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

```
static inline flexspi_arb_command_source_t FLEXSPI_GetArbitratorCommandSource(FLEXSPI_Type  
                                                                              *base)
```

Gets the trigger source of current command sequence granted by arbitrator.

**Parameters**

- base – FLEXSPI peripheral base address.

**Return values**

trigger – source of current command sequence.

```
static inline flexspi_ip_error_code_t FLEXSPI_GetIPCommandErrorCode(FLEXSPI_Type *base,  
                                                                    uint8_t *index)
```

Gets the error code when IP command error detected.

**Parameters**

- base – FLEXSPI peripheral base address.
- index – Pointer to a uint8\_t type variable to receive the sequence index when error detected.

**Return values**

error – code when IP command error detected.

```
static inline flexspi_ahb_error_code_t FLEXSPI_GetAHBCommandErrorCode(FLEXSPI_Type  
                                                                    *base, uint8_t  
                                                                    *index)
```

Gets the error code when AHB command error detected.

**Parameters**

- base – FLEXSPI peripheral base address.
- index – Pointer to a uint8\_t type variable to receive the sequence index when error detected.

**Return values**

error – code when AHB command error detected.

```
static inline bool FLEXSPI_GetBusIdleStatus(FLEXSPI_Type *base)
```

Returns whether the bus is idle.

**Parameters**

- base – FLEXSPI peripheral base address.

**Return values**

- true – Bus is idle.
- false – Bus is busy.

```
void FLEXSPI_UpdateRxSampleClock(FLEXSPI_Type *base, flexspi_read_sample_clock_t  
                                clockSource)
```

Update read sample clock source.

**Parameters**

- base – FLEXSPI peripheral base address.
- clockSource – clockSource of type flexspi\_read\_sample\_clock\_t

```
void FLEXSPI_UpdateLUT(FLEXSPI_Type *base, uint32_t index, const uint32_t *cmd, uint32_t  
                      count)
```

Updates the LUT table.

**Parameters**

- base – FLEXSPI peripheral base address.
- index – From which index start to update. It could be any index of the LUT table, which also allows user to update command content inside a command. Each command consists of up to 8 instructions and occupy 4\*32-bit memory.

- `cmd` – Command sequence array.
- `count` – Number of sequences.

`static inline void FLEXSPI_WriteData(FLEXSPI_Type *base, uint32_t data, uint8_t fifoIndex)`

Writes data into FIFO.

#### Parameters

- `base` – FLEXSPI peripheral base address
- `data` – The data bytes to send
- `fifoIndex` – Destination fifo index.

`static inline uint32_t FLEXSPI_ReadData(FLEXSPI_Type *base, uint8_t fifoIndex)`

Receives data from data FIFO.

#### Parameters

- `base` – FLEXSPI peripheral base address
- `fifoIndex` – Source fifo index.

#### Returns

The data in the FIFO.

`status_t FLEXSPI_WriteBlocking(FLEXSPI_Type *base, uint8_t *buffer, size_t size)`

Sends a buffer of data bytes using blocking method.

---

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

---

#### Parameters

- `base` – FLEXSPI peripheral base address
- `buffer` – The data bytes to send
- `size` – The number of data bytes to send

#### Return values

- `kStatus_Success` – write success without error
- `kStatus_FLEXSPI_SequenceExecutionTimeout` – sequence execution timeout
- `kStatus_FLEXSPI_IpCommandSequenceError` – IP command sequence error detected
- `kStatus_FLEXSPI_IpCommandGrantTimeout` – IP command grant timeout detected

`status_t FLEXSPI_ReadBlocking(FLEXSPI_Type *base, uint8_t *buffer, size_t size)`

Receives a buffer of data bytes using a blocking method.

---

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

---

#### Parameters

- `base` – FLEXSPI peripheral base address
- `buffer` – The data bytes to send
- `size` – The number of data bytes to receive

#### Return values

- `kStatus_Success` – read success without error
- `kStatus_FLEXSPI_SequenceExecutionTimeout` – sequence execution timeout
- `kStatus_FLEXSPI_IpCommandSequenceError` – IP command sequence error detected
- `kStatus_FLEXSPI_IpCommandGrantTimeout` – IP command grant timeout detected

`status_t FLEXSPI_TransferBlocking(FLEXSPI_Type *base, flexspi_transfer_t *xfer)`

Execute command to transfer a buffer data bytes using a blocking method.

#### Parameters

- `base` – FLEXSPI peripheral base address
- `xfer` – pointer to the transfer structure.

#### Return values

- `kStatus_Success` – command transfer success without error
- `kStatus_FLEXSPI_SequenceExecutionTimeout` – sequence execution timeout
- `kStatus_FLEXSPI_IpCommandSequenceError` – IP command sequence error detected
- `kStatus_FLEXSPI_IpCommandGrantTimeout` – IP command grant timeout detected

`void FLEXSPI_TransferCreateHandle(FLEXSPI_Type *base, flexspi_handle_t *handle, flexspi_transfer_callback_t callback, void *userData)`

Initializes the FLEXSPI handle which is used in transactional functions.

#### Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – pointer to `flexspi_handle_t` structure to store the transfer state.
- `callback` – pointer to user callback function.
- `userData` – user parameter passed to the callback function.

`status_t FLEXSPI_TransferNonBlocking(FLEXSPI_Type *base, flexspi_handle_t *handle, flexspi_transfer_t *xfer)`

Performs a interrupt non-blocking transfer on the FLEXSPI bus.

---

**Note:** Calling the API returns immediately after transfer initiates. The user needs to call `FLEXSPI_GetTransferCount` to poll the transfer status to check whether the transfer is finished. If the return status is not `kStatus_FLEXSPI_Busy`, the transfer is finished. For `FLEXSPI_Read`, the `dataSize` should be multiple of rx watermark level, or FLEXSPI could not read data properly.

---

#### Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – pointer to `flexspi_handle_t` structure which stores the transfer state.
- `xfer` – pointer to `flexspi_transfer_t` structure.

#### Return values

- `kStatus_Success` – Successfully start the data transmission.
- `kStatus_FLEXSPI_Busy` – Previous transmission still not finished.

`status_t FLEXSPI_TransferGetCount(FLEXSPI_Type *base, flexspi_handle_t *handle, size_t *count)`

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

#### Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – pointer to `flexspi_handle_t` structure which stores the transfer state.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

#### Return values

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

`void FLEXSPI_TransferAbort(FLEXSPI_Type *base, flexspi_handle_t *handle)`

Aborts an interrupt non-blocking transfer early.

---

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

---

#### Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – pointer to `flexspi_handle_t` structure which stores the transfer state

`void FLEXSPI_TransferHandleIRQ(FLEXSPI_Type *base, flexspi_handle_t *handle)`

Master interrupt handler.

#### Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – pointer to `flexspi_handle_t` structure.

`FSL_FLEXSPI_DRIVER_VERSION`

FLEXSPI driver version.

Status structure of FLEXSPI.

*Values:*

enumerator `kStatus_FLEXSPI_Busy`  
FLEXSPI is busy

enumerator `kStatus_FLEXSPI_SequenceExecutionTimeout`  
Sequence execution timeout error occurred during FLEXSPI transfer.

enumerator `kStatus_FLEXSPI_IpCommandSequenceError`  
IP command Sequence execution timeout error occurred during FLEXSPI transfer.

enumerator `kStatus_FLEXSPI_IpCommandGrantTimeout`  
IP command grant timeout error occurred during FLEXSPI transfer.

CMD definition of FLEXSPI, use to form LUT instruction, `_flexspi_command`.

*Values:*

enumerator `kFLEXSPI_Command_STOP`

Stop execution, deassert CS.

enumerator `kFLEXSPI_Command_SDR`

Transmit Command code to Flash, using SDR mode.

enumerator `kFLEXSPI_Command_RADDR_SDR`

Transmit Row Address to Flash, using SDR mode.

enumerator `kFLEXSPI_Command_CADDR_SDR`

Transmit Column Address to Flash, using SDR mode.

enumerator `kFLEXSPI_Command_MODE1_SDR`

Transmit 1-bit Mode bits to Flash, using SDR mode.

enumerator `kFLEXSPI_Command_MODE2_SDR`

Transmit 2-bit Mode bits to Flash, using SDR mode.

enumerator `kFLEXSPI_Command_MODE4_SDR`

Transmit 4-bit Mode bits to Flash, using SDR mode.

enumerator `kFLEXSPI_Command_MODE8_SDR`

Transmit 8-bit Mode bits to Flash, using SDR mode.

enumerator `kFLEXSPI_Command_WRITE_SDR`

Transmit Programming Data to Flash, using SDR mode.

enumerator `kFLEXSPI_Command_READ_SDR`

Receive Read Data from Flash, using SDR mode.

enumerator `kFLEXSPI_Command_LEARN_SDR`

Receive Read Data or Preamble bit from Flash, SDR mode.

enumerator `kFLEXSPI_Command_DATSZ_SDR`

Transmit Read/Program Data size (byte) to Flash, SDR mode.

enumerator `kFLEXSPI_Command_DUMMY_SDR`

Leave data lines undriven by FlexSPI controller.

enumerator `kFLEXSPI_Command_DUMMY_RWDS_SDR`

Leave data lines undriven by FlexSPI controller; dummy cycles decided by RWDS.

enumerator `kFLEXSPI_Command_DDR`

Transmit Command code to Flash, using DDR mode.

enumerator `kFLEXSPI_Command_RADDR_DDR`

Transmit Row Address to Flash, using DDR mode.

enumerator `kFLEXSPI_Command_CADDR_DDR`

Transmit Column Address to Flash, using DDR mode.

enumerator `kFLEXSPI_Command_MODE1_DDR`

Transmit 1-bit Mode bits to Flash, using DDR mode.

enumerator `kFLEXSPI_Command_MODE2_DDR`

Transmit 2-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI\_Command\_MODE4\_DDR

Transmit 4-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI\_Command\_MODE8\_DDR

Transmit 8-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI\_Command\_WRITE\_DDR

Transmit Programming Data to Flash, using DDR mode.

enumerator kFLEXSPI\_Command\_READ\_DDR

Receive Read Data from Flash, using DDR mode.

enumerator kFLEXSPI\_Command\_LEARN\_DDR

Receive Read Data or Preamble bit from Flash, DDR mode.

enumerator kFLEXSPI\_Command\_DATSZ\_DDR

Transmit Read/Program Data size (byte) to Flash, DDR mode.

enumerator kFLEXSPI\_Command\_DUMMY\_DDR

Leave data lines undriven by FlexSPI controller.

enumerator kFLEXSPI\_Command\_DUMMY\_RWDS\_DDR

Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

enumerator kFLEXSPI\_Command\_JUMP\_ON\_CS

Stop execution, deassert CS and save operand[7:0] as the instruction start pointer for next sequence

enum \_flexspi\_pad

pad definition of FLEXSPI, use to form LUT instruction.

*Values:*

enumerator kFLEXSPI\_1PAD

Transmit command/address and transmit/receive data only through DATA0/DATA1.

enumerator kFLEXSPI\_2PAD

Transmit command/address and transmit/receive data only through DATA[1:0].

enumerator kFLEXSPI\_4PAD

Transmit command/address and transmit/receive data only through DATA[3:0].

enumerator kFLEXSPI\_8PAD

Transmit command/address and transmit/receive data only through DATA[7:0].

enum \_flexspi\_flags

FLEXSPI interrupt status flags.

*Values:*

enumerator kFLEXSPI\_SequenceExecutionTimeoutFlag

Sequence execution timeout.

enumerator kFLEXSPI\_AhbBusErrorFlag

AHB Bus error flag.

enumerator kFLEXSPI\_SckStoppedBecauseTxEmptyFlag

SCK is stopped during command sequence because Async TX FIFO empty.

enumerator kFLEXSPI\_SckStoppedBecauseRxFullFlag

SCK is stopped during command sequence because Async RX FIFO full.

enumerator kFLEXSPI\_IpTxFifoWatermarkEmptyFlag

IP TX FIFO WaterMark empty.



enumerator kFLEXSPI\_IpRxFifoWatermarkAvailableFlag  
IP RX FIFO WaterMark available.

enumerator kFLEXSPI\_AhbCommandSequenceErrorFlag  
AHB triggered Command Sequences Error.

enumerator kFLEXSPI\_IpCommandSequenceErrorFlag  
IP triggered Command Sequences Error.

enumerator kFLEXSPI\_AhbCommandGrantTimeoutFlag  
AHB triggered Command Sequences Grant Timeout.

enumerator kFLEXSPI\_IpCommandGrantTimeoutFlag  
IP triggered Command Sequences Grant Timeout.

enumerator kFLEXSPI\_IpCommandExecutionDoneFlag  
IP triggered Command Sequences Execution finished.

enumerator kFLEXSPI\_AllInterruptFlags  
All flags.

enum \_flexspi\_read\_sample\_clock  
FLEXSPI sample clock source selection for Flash Reading.

*Values:*

enumerator kFLEXSPI\_ReadSampleClkLoopbackInternally  
Dummy Read strobe generated by FlexSPI Controller and loopback internally.

enumerator kFLEXSPI\_ReadSampleClkLoopbackFromDqsPad  
Dummy Read strobe generated by FlexSPI Controller and loopback from DQS pad.

enumerator kFLEXSPI\_ReadSampleClkLoopbackFromSckPad  
SCK output clock and loopback from SCK pad.

enumerator kFLEXSPI\_ReadSampleClkExternalInputFromDqsPad  
Flash provided Read strobe and input from DQS pad.

enum \_flexspi\_cs\_interval\_cycle\_unit  
FLEXSPI interval unit for flash device select.

*Values:*

enumerator kFLEXSPI\_CsIntervalUnit1SckCycle  
Chip selection interval: CSINTERVAL \* 1 serial clock cycle.

enumerator kFLEXSPI\_CsIntervalUnit256SckCycle  
Chip selection interval: CSINTERVAL \* 256 serial clock cycle.

enum \_flexspi\_ahb\_write\_wait\_unit  
FLEXSPI AHB wait interval unit for writing.

*Values:*

enumerator kFLEXSPI\_AhbWriteWaitUnit2AhbCycle  
AWRWAIT unit is 2 ahb clock cycle.

enumerator kFLEXSPI\_AhbWriteWaitUnit8AhbCycle  
AWRWAIT unit is 8 ahb clock cycle.

enumerator kFLEXSPI\_AhbWriteWaitUnit32AhbCycle  
AWRWAIT unit is 32 ahb clock cycle.

enumerator kFLEXSPI\_AhbWriteWaitUnit128AhbCycle

AWRWAIT unit is 128 ahb clock cycle.

enumerator kFLEXSPI\_AhbWriteWaitUnit512AhbCycle

AWRWAIT unit is 512 ahb clock cycle.

enumerator kFLEXSPI\_AhbWriteWaitUnit2048AhbCycle

AWRWAIT unit is 2048 ahb clock cycle.

enumerator kFLEXSPI\_AhbWriteWaitUnit8192AhbCycle

AWRWAIT unit is 8192 ahb clock cycle.

enumerator kFLEXSPI\_AhbWriteWaitUnit32768AhbCycle

AWRWAIT unit is 32768 ahb clock cycle.

enum \_flexspi\_ip\_error\_code

Error Code when IP command Error detected.

*Values:*

enumerator kFLEXSPI\_IpCmdErrorNoError

No error.

enumerator kFLEXSPI\_IpCmdErrorJumpOnCsInIpCmd

IP command with JMP\_ON\_CS instruction used.

enumerator kFLEXSPI\_IpCmdErrorUnknownOpCode

Unknown instruction opcode in the sequence.

enumerator kFLEXSPI\_IpCmdErrorSdrDummyInDdrSequence

Instruction DUMMY\_SDR/DUMMY\_RWDS\_SDR used in DDR sequence.

enumerator kFLEXSPI\_IpCmdErrorDdrDummyInSdrSequence

Instruction DUMMY\_DDR/DUMMY\_RWDS\_DDR used in SDR sequence.

enumerator kFLEXSPI\_IpCmdErrorInvalidAddress

Flash access start address exceed the whole flash address range (A1/A2/B1/B2).

enumerator kFLEXSPI\_IpCmdErrorSequenceExecutionTimeout

Sequence execution timeout.

enumerator kFLEXSPI\_IpCmdErrorFlashBoundaryAcrosss

Flash boundary crossed.

enum \_flexspi\_ahb\_error\_code

Error Code when AHB command Error detected.

*Values:*

enumerator kFLEXSPI\_AhbCmdErrorNoError

No error.

enumerator kFLEXSPI\_AhbCmdErrorJumpOnCsInWriteCmd

AHB Write command with JMP\_ON\_CS instruction used in the sequence.

enumerator kFLEXSPI\_AhbCmdErrorUnknownOpCode

Unknown instruction opcode in the sequence.

enumerator kFLEXSPI\_AhbCmdErrorSdrDummyInDdrSequence

Instruction DUMMY\_SDR/DUMMY\_RWDS\_SDR used in DDR sequence.

enumerator kFLEXSPI\_AhbCmdErrorDdrDummyInSdrSequence

Instruction DUMMY\_DDR/DUMMY\_RWDS\_DDR used in SDR sequence.

enumerator kFLEXSPI\_AhbCmdSequenceExecutionTimeout  
Sequence execution timeout.

enum \_flexspi\_port

FLEXSPI operation port select.

*Values:*

enumerator kFLEXSPI\_PortA1  
Access flash on A1 port.

enumerator kFLEXSPI\_PortA2  
Access flash on A2 port.

enumerator kFLEXSPI\_PortCount

enum \_flexspi\_arb\_command\_source

Trigger source of current command sequence granted by arbitrator.

*Values:*

enumerator kFLEXSPI\_AhbReadCommand

enumerator kFLEXSPI\_AhbWriteCommand

enumerator kFLEXSPI\_IpCommand

enumerator kFLEXSPI\_SuspendedCommand

enum \_flexspi\_command\_type

Command type.

*Values:*

enumerator kFLEXSPI\_Command  
FlexSPI operation: Only command, both TX and Rx buffer are ignored.

enumerator kFLEXSPI\_Config  
FlexSPI operation: Configure device mode, the TX fifo size is fixed in LUT.

enumerator kFLEXSPI\_Read

enumerator kFLEXSPI\_Write

typedef enum \_flexspi\_pad flexspi\_pad\_t

pad definition of FLEXSPI, use to form LUT instruction.

typedef enum \_flexspi\_flags flexspi\_flags\_t

FLEXSPI interrupt status flags.

typedef enum \_flexspi\_read\_sample\_clock flexspi\_read\_sample\_clock\_t

FLEXSPI sample clock source selection for Flash Reading.

typedef enum \_flexspi\_cs\_interval\_cycle\_unit flexspi\_cs\_interval\_cycle\_unit\_t

FLEXSPI interval unit for flash device select.

typedef enum \_flexspi\_ahb\_write\_wait\_unit flexspi\_ahb\_write\_wait\_unit\_t

FLEXSPI AHB wait interval unit for writing.

typedef enum \_flexspi\_ip\_error\_code flexspi\_ip\_error\_code\_t

Error Code when IP command Error detected.

typedef enum \_flexspi\_ahb\_error\_code flexspi\_ahb\_error\_code\_t

Error Code when AHB command Error detected.

```
typedef enum _flexspi_port flexspi_port_t
    FLEXSPI operation port select.

typedef enum _flexspi_arb_command_source flexspi_arb_command_source_t
    Trigger source of current command sequence granted by arbitrator.

typedef enum _flexspi_command_type flexspi_command_type_t
    Command type.

typedef struct _flexspi_ahbBuffer_config flexspi_ahbBuffer_config_t

typedef struct _flexspi_ahbBuffers_ctrl flexspi_ahbBuffers_ctrl_t
    Structure to control all AHB buffers.

typedef struct _flexspi_config flexspi_config_t
    FLEXSPI configuration structure.

typedef struct _flexspi_device_config flexspi_device_config_t
    External device configuration items.

typedef struct _flexspi_transfer flexspi_transfer_t
    Transfer structure for FLEXSPI.

typedef struct _flexspi_handle flexspi_handle_t

typedef void (*flexspi_transfer_callback_t)(FLEXSPI_Type *base, flexspi_handle_t *handle,
status_t status, void *userData)
    FLEXSPI transfer callback function.

typedef struct _flexspi_addr_map_config flexspi_addr_map_config_t
    Address mapping configuration structure.

FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNT

FLEXSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)
    Formula to form FLEXSPI instructions in LUT table.

struct flexspi_ahbBuffer_config
    #include <fsl_flexspi.h>
```

### Public Members

```
uint8_t priority
    This priority for AHB Master Read which this AHB RX Buffer is assigned.

uint8_t masterIndex
    AHB Master ID the AHB RX Buffer is assigned.

uint16_t bufferSize
    AHB buffer size in byte.

bool enablePrefetch
    AHB Read Prefetch Enable for current AHB RX Buffer corresponding Master, allows
    prefetch disable/enable separately for each master.

struct flexspi_ahbBuffers_ctrl
    #include <fsl_flexspi.h> Structure to control all AHB buffers.
```

### Public Members

*flexspi\_ahbBuffer\_config\_t* buffer[FSL\_FEATURE\_FLEXSPI\_AHB\_BUFFER\_COUNTn(0)]  
Configurations of all AHB buffers.

struct *\_flexspi\_config*  
*#include <fsl\_flexspi.h>* FLEXSPI configuration structure.

### Public Members

*flexspi\_read\_sample\_clock\_t* rxSampleClock  
Sample Clock source selection for Flash Reading.

bool enableSckFreeRunning  
Enable/disable SCK output free-running.

bool enableDoze  
Enable/disable doze mode support.

bool enableHalfSpeedAccess  
Enable/disable divide by 2 of the clock for half speed commands.

*flexspi\_read\_sample\_clock\_t* rxSampleClockPortB  
Sample Clock source\_b selection for Flash Reading.

bool rxSampleClockDiff  
Sample Clock source or source\_b selection for Flash Reading.

bool enableSameConfigForAll  
Enable/disable same configuration for all connected devices when enabled, same configuration in FLASHA1CRx is applied to all.

uint16\_t seqTimeoutCycle  
Timeout wait cycle for command sequence execution, timeout after ahbGrantTimeoutCycle\*1024 serial root clock cycles.

uint8\_t ipGrantTimeoutCycle  
Timeout wait cycle for IP command grant, timeout after ipGrantTimeoutCycle\*1024 AHB clock cycles.

uint8\_t txWatermark  
FLEXSPI IP transmit watermark value.

uint8\_t rxWatermark  
FLEXSPI receive watermark value.

struct *\_flexspi\_device\_config*  
*#include <fsl\_flexspi.h>* External device configuration items.

### Public Members

uint32\_t flexspiRootClk  
FLEXSPI serial root clock.

bool isSck2Enabled  
FLEXSPI use SCK2.

uint32\_t flashSize  
Flash size in KByte.

bool addressShift

Address shift.

*flexspi\_cs\_interval\_cycle\_unit\_t* CSIntervalUnit

CS interval unit, 1 or 256 cycle.

uint16\_t CSInterval

CS line assert interval, multiply CS interval unit to get the CS line assert interval cycles.

uint8\_t CSHoldTime

CS line hold time.

uint8\_t CSSetupTime

CS line setup time.

uint8\_t dataValidTime

Data valid time for external device.

uint8\_t columnSpace

Column space size.

bool enableWordAddress

If enable word address.

uint8\_t AWRSeqIndex

Sequence ID for AHB write command.

uint8\_t AWRSeqNumber

Sequence number for AHB write command.

uint8\_t ARDSeqIndex

Sequence ID for AHB read command.

uint8\_t ARDSeqNumber

Sequence number for AHB read command.

*flexspi\_ahb\_write\_wait\_unit\_t* AHBWriteWaitUnit

AHB write wait unit.

uint16\_t AHBWriteWaitInterval

AHB write wait interval, multiply AHB write interval unit to get the AHB write wait cycles.

bool enableWriteMask

Enable/Disable FLEXSPI drive DQS pin as write mask when writing to external device.

bool isFroClockSource

Is FRO clock source or not.

struct \_flexspi\_transfer

*#include <fsl\_flexspi.h>* Transfer structure for FLEXSPI.

### Public Members

uint32\_t deviceAddress

Operation device address.

*flexspi\_port\_t* port

Operation port.

*flexspi\_command\_type\_t* cmdType

Execution command type.

uint8\_t seqIndex

Sequence ID for command.

uint8\_t SeqNumber

Sequence number for command.

uint32\_t \*data

Data buffer.

size\_t dataSize

Data size in bytes.

struct \_flexspi\_handle

*#include <fsl\_flexspi.h>* Transfer handle structure for FLEXSPI.

### Public Members

uint32\_t state

Internal state for FLEXSPI transfer

uint8\_t \*data

Data buffer.

size\_t dataSize

Remaining Data size in bytes.

size\_t transferTotalSize

Total Data size in bytes.

*flexspi\_transfer\_callback\_t* completionCallback

Callback for users while transfer finish or error occurred

void \*userData

FLEXSPI callback function parameter.

struct \_flexspi\_addr\_map\_config

*#include <fsl\_flexspi.h>* Address mapping configuration structure.

### Public Members

uint32\_t addrStart

Remapping start address.

uint32\_t addrEnd

Remapping end address.

uint32\_t addrOffset

Address offset.

bool remapEnable

Enable address remapping.

struct ahbConfig

## Public Members

uint8\_t ahbGrantTimeoutCycle

Timeout wait cycle for AHB command grant, timeout after ahbGrantTimeoutCycle\*1024 AHB clock cycles.

uint16\_t ahbBusTimeoutCycle

Timeout wait cycle for AHB read/write access, timeout after ahbBusTimeoutCycle\*1024 AHB clock cycles.

uint8\_t resumeWaitCycle

Wait cycle for idle state before suspended command sequence resume, timeout after ahbBusTimeoutCycle AHB clock cycles.

bool disableAhbReadResume

**True:** Suspended AHB read prefetch does not resume once aborted; **False:** Suspended AHB read prefetch resumes when AHB is IDLE.

*flexspi\_ahbBuffer\_config\_t* buffer[FSL\_FEATURE\_FLEXSPI\_AHB\_BUFFER\_COUNTn(0)]

AHB buffer size.

bool enableClearAHBBufferOpt

Enable/disable automatically clean AHB RX Buffer and TX Buffer when FLEXSPI returns STOP mode ACK.

bool enableReadAddressOpt

Enable/disable remove AHB read burst start address alignment limitation. when enable, there is no AHB read burst start address alignment limitation.

bool enableAHBPrefetch

Enable/disable AHB read prefetch feature, when enabled, FLEXSPI will fetch more data than current AHB burst.

bool enableAHBBufferable

Enable/disable AHB bufferable write access support, when enabled, FLEXSPI return before waiting for command execution finished.

bool enableAHBCachable

Enable AHB bus cachable read access support.

## 2.21 FLEXSPI eDMA Driver

```
void FLEXSPI_TransferCreateHandleEDMA(FLEXSPI_Type *base, flexspi_edma_handle_t
                                     *handle, flexspi_edma_callback_t callback, void
                                     *userData, edma_handle_t *txDmaHandle,
                                     edma_handle_t *rxDmaHandle)
```

Initializes the FLEXSPI handle for transfer which is used in transactional functions and set the callback.

### Parameters

- base – FLEXSPI peripheral base address
- handle – Pointer to flexspi\_edma\_handle\_t structure
- callback – FLEXSPI callback, NULL means no callback.
- userData – User callback function data.
- txDmaHandle – User requested DMA handle for TX DMA transfer.
- rxDmaHandle – User requested DMA handle for RX DMA transfer.



```
void FLEXSPI_TransferUpdateSizeEDMA(FLEXSPI_Type *base, flexspi_edma_handle_t *handle,
                                   flexspi_edma_transfer_nsize_t nsize)
```

Update FLEXSPI EDMA transfer source data transfer size(SSIZE) and destination data transfer size(DSIZE).

**See also:**

flexspi\_edma\_transfer\_nsize\_t.

**Parameters**

- base – FLEXSPI peripheral base address
- handle – Pointer to flexspi\_edma\_handle\_t structure
- nsize – FLEXSPI DMA transfer data transfer size(SSIZE/DSIZE), by default the size is kFLEXSPI\_EDMASize1Bytes(one byte).

```
status_t FLEXSPI_TransferEDMA(FLEXSPI_Type *base, flexspi_edma_handle_t *handle,
                              flexspi_transfer_t *xfer)
```

Transfers FLEXSPI data using an eDMA non-blocking method.

This function writes/receives data to/from the FLEXSPI transmit/receive FIFO. This function is non-blocking.

**Parameters**

- base – FLEXSPI peripheral base address.
- handle – Pointer to flexspi\_edma\_handle\_t structure
- xfer – FLEXSPI transfer structure.

**Return values**

- kStatus\_FLEXSPI\_Busy – FLEXSPI is busy transfer.
- kStatus\_InvalidArgument – The watermark configuration is invalid, the watermark should be power of 2 to do successfully EDMA transfer.
- kStatus\_Success – FLEXSPI successfully start edma transfer.

```
void FLEXSPI_TransferAbortEDMA(FLEXSPI_Type *base, flexspi_edma_handle_t *handle)
```

Aborts the transfer data using eDMA.

This function aborts the transfer data using eDMA.

**Parameters**

- base – FLEXSPI peripheral base address.
- handle – Pointer to flexspi\_edma\_handle\_t structure

```
status_t FLEXSPI_TransferGetTransferCountEDMA(FLEXSPI_Type *base, flexspi_edma_handle_t
                                              *handle, size_t *count)
```

Gets the transferred counts of transfer.

**Parameters**

- base – FLEXSPI peripheral base address.
- handle – Pointer to flexspi\_edma\_handle\_t structure.
- count – Bytes transfer.

**Return values**

- kStatus\_Success – Succeed get the transfer count.

- kStatus\_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

FSL\_FLEXSPI\_EDMA\_DRIVER\_VERSION

FLEXSPI EDMA driver version.

enum \_flexspi\_edma\_ntransfer\_size

eDMA transfer configuration

*Values:*

enumerator kFLEXPSI\_EDMAAnSize1Bytes

Source/Destination data transfer size is 1 byte every time

enumerator kFLEXPSI\_EDMAAnSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kFLEXPSI\_EDMAAnSize4Bytes

Source/Destination data transfer size is 4 bytes every time

enumerator kFLEXPSI\_EDMAAnSize8Bytes

Source/Destination data transfer size is 8 bytes every time

enumerator kFLEXPSI\_EDMAAnSize32Bytes

Source/Destination data transfer size is 32 bytes every time

typedef struct *flexspi\_edma\_handle* flexspi\_edma\_handle\_t

typedef void (\*flexspi\_edma\_callback\_t)(FLEXSPI\_Type \*base, *flexspi\_edma\_handle\_t* \*handle, *status\_t* status, void \*userData)

FLEXSPI eDMA transfer callback function for finish and error.

typedef enum *flexspi\_edma\_ntransfer\_size* flexspi\_edma\_transfer\_nsize\_t

eDMA transfer configuration

struct \_flexspi\_edma\_handle

*#include <fsl\_flexspi\_edma.h>* FLEXSPI DMA transfer handle, users should not touch the content of the handle.

## Public Members

*edma\_handle\_t* \*txDmaHandle

eDMA handler for FLEXSPI Tx.

*edma\_handle\_t* \*rxDmaHandle

eDMA handler for FLEXSPI Rx.

size\_t transferSize

Bytes need to transfer.

*flexspi\_edma\_transfer\_nsize\_t* nsize

eDMA SSIZE/DSIZE in each transfer.

uint8\_t nbytes

eDMA minor byte transfer count initially configured.

uint8\_t count

The transfer data count in a DMA request.

uint32\_t state

Internal state for FLEXSPI eDMA transfer.

*flexspi\_edma\_callback\_t* completionCallback

A callback function called after the eDMA transfer is finished.

void \*userData

User callback parameter

## 2.22 I3C: I3C Driver

FSL\_I3C\_DRIVER\_VERSION

I3C driver version.

I3C status return codes.

*Values:*

enumerator kStatus\_I3C\_Busy

The master is already performing a transfer.

enumerator kStatus\_I3C\_Idle

The slave driver is idle.

enumerator kStatus\_I3C\_Nak

The slave device sent a NAK in response to an address.

enumerator kStatus\_I3C\_WriteAbort

The slave device sent a NAK in response to a write.

enumerator kStatus\_I3C\_Term

The master terminates slave read.

enumerator kStatus\_I3C\_HdrParityError

Parity error from DDR read.

enumerator kStatus\_I3C\_CrcError

CRC error from DDR read.

enumerator kStatus\_I3C\_ReadFifoError

Read from M/SRDATAB register when FIFO empty.

enumerator kStatus\_I3C\_WriteFifoError

Write to M/SWDATAB register when FIFO full.

enumerator kStatus\_I3C\_MsgError

Message SDR/DDR mismatch or read/write message in wrong state

enumerator kStatus\_I3C\_InvalidReq

Invalid use of request.

enumerator kStatus\_I3C\_Timeout

The module has stalled too long in a frame.

enumerator kStatus\_I3C\_SlaveCountExceed

The I3C slave count has exceed the definition in I3C\_MAX\_DEVCNT.

enumerator kStatus\_I3C\_IBIWon

The I3C slave event IBI or MR or HJ won the arbitration on a header address.

enumerator kStatus\_I3C\_OverrunError

Slave internal from-bus buffer/FIFO overrun.

enumerator kStatus\_I3C\_UnderrunError

Slave internal to-bus buffer/FIFO underrun

enumerator kStatus\_I3C\_UnderrunNak

Slave internal from-bus buffer/FIFO underrun and NACK error

enumerator kStatus\_I3C\_InvalidStart

Slave invalid start flag

enumerator kStatus\_I3C\_SdrParityError

SDR parity error

enumerator kStatus\_I3C\_S0S1Error

S0 or S1 error

enum \_i3c\_hdr\_mode

I3C HDR modes.

*Values:*

enumerator kI3C\_HDRModeNone

enumerator kI3C\_HDRModeDDR

enumerator kI3C\_HDRModeTSP

enumerator kI3C\_HDRModeTSL

typedef enum \_i3c\_hdr\_mode i3c\_hdr\_mode\_t

I3C HDR modes.

typedef struct \_i3c\_device\_info i3c\_device\_info\_t

I3C device information.

I3C\_RETRY\_TIMES

Max loops to wait for I3C operation status complete.

This is the maximum number of loops to wait for I3C operation status complete. If set to 0, it will wait indefinitely.

I3C\_MAX\_DEVCNT

I3C\_IBI\_BUFF\_SIZE

struct \_i3c\_device\_info

*#include <fsl\_i3c.h>* I3C device information.

### Public Members

uint8\_t dynamicAddr

Device dynamic address.

uint8\_t staticAddr

Static address.

uint8\_t dcr

Device characteristics register information.

uint8\_t bcr

Bus characteristics register information.

uint16\_t vendorID

Device vendor ID(manufacture ID).

uint32\_t partNumber

Device part number info

uint16\_t maxReadLength

Maximum read length.

uint16\_t maxWriteLength

Maximum write length.

uint8\_t hdrMode

Support hdr mode, could be OR logic in i3c\_hdr\_mode.

## 2.23 I3C Common Driver

typedef struct *i3c\_config* i3c\_config\_t

Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C\_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

uint32\_t I3C\_GetInstance(I3C\_Type \*base)

Get which instance current I3C is used.

### Parameters

- base – The I3C peripheral base address.

void I3C\_GetDefaultConfig(*i3c\_config\_t* \*config)

Provides a default configuration for the I3C peripheral, the configuration covers both master functionality and slave functionality.

This function provides the following default configuration for I3C:

```
config->enableMaster      = kI3C_MasterCapable;
config->disableTimeout    = false;
config->hKeep             = kI3C_MasterHighKeeperNone;
config->enableOpenDrainStop = true;
config->enableOpenDrainHigh = true;
config->baudRate_Hz.i2cBaud = 400000U;
config->baudRate_Hz.i3cPushPullBaud = 12500000U;
config->baudRate_Hz.i3cOpenDrainBaud = 2500000U;
config->masterDynamicAddress = 0x0AU;
config->slowClock_Hz       = 1000000U;
config->enableSlave        = true;
config->vendorID           = 0x11BU;
config->enableRandomPart   = false;
config->partNumber         = 0;
config->dcr                = 0;
config->bcr = 0;
config->hdrMode            = (uint8_t)kI3C_HDRModeDDR;
config->nakAllRequest       = false;
config->ignoreS0S1Error    = false;
config->offline            = false;
config->matchSlaveStartStop = false;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the common I3C driver with I3C\_Init().

### Parameters

- `config` – **[out]** User provided configuration structure for default values. Refer to `i3c_config_t`.

`void I3C_Init(I3C_Type *base, const i3c_config_t *config, uint32_t sourceClock_Hz)`

Initializes the I3C peripheral. This function enables the peripheral clock and initializes the I3C peripheral as described by the user provided configuration. This will initialize both the master peripheral and slave peripheral so that I3C module could work as pure master, pure slave or secondary master, etc. A software reset is performed prior to configuration.

### Parameters

- `base` – The I3C peripheral base address.
- `config` – User provided peripheral configuration. Use `I3C_GetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

`struct _i3c_config`

*#include <fsl\_i3c.h>* Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_GetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

### Public Members

*i3c\_master\_enable\_t* enableMaster

Enable master mode.

`bool` disableTimeout

Whether to disable timeout to prevent the ERRWARN.

*i3c\_master\_hkeep\_t* hKeep

High keeper mode setting.

`bool` enableOpenDrainStop

Whether to emit open-drain speed STOP.

`bool` enableOpenDrainHigh

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

*i3c\_baudrate\_hz\_t* baudRate\_Hz

Desired baud rate settings.

*i3c\_start\_scl\_delay\_t* startSclDelay

I3C SCL delay after START.

*i3c\_start\_scl\_delay\_t* restartSclDelay

I3C SCL delay after Repeated START.

`uint8_t` masterDynamicAddress

Main master dynamic address configuration.

`uint32_t` maxWriteLength

Maximum write length.

`uint32_t` `maxReadLength`  
Maximum read length.

`bool` `enableSlave`  
Whether to enable slave.

`uint8_t` `staticAddr`  
Static address.

`uint16_t` `vendorID`  
Device vendor ID(manufacture ID).

`uint32_t` `partNumber`  
Device part number info

`uint8_t` `dcr`  
Device characteristics register information.

`uint8_t` `bcr`  
Bus characteristics register information.

`uint8_t` `hdrMode`  
Support hdr mode, could be OR logic in enumeration:`i3c_hdr_mode_t`.

`bool` `nakAllRequest`  
Whether to reply NAK to all requests except broadcast CCC.

`bool` `ignoreS0S1Error`  
Whether to ignore S0/S1 error in SDR mode.

`bool` `offline`  
Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

`bool` `matchSlaveStartStop`  
Whether to assert start/stop status only the time slave is addressed.

## 2.24 I3C Master Driver

`void` `I3C_MasterGetDefaultConfig(i3c_master_config_t *masterConfig)`

Provides a default configuration for the I3C master peripheral.

This function provides the following default configuration for the I3C master peripheral:

```
masterConfig->enableMaster      = kI3C_MasterOn;
masterConfig->disableTimeout    = false;
masterConfig->hKeep             = kI3C_MasterHighKeeperNone;
masterConfig->enableOpenDrainStop = true;
masterConfig->enableOpenDrainHigh = true;
masterConfig->baudRate_Hz       = 100000U;
masterConfig->busType           = kI3C_TypeI2C;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with `I3C_MasterInit()`.

### Parameters

- `masterConfig` – **[out]** User provided configuration structure for default values. Refer to `i3c_master_config_t`.

```
void I3C_MasterInit(I3C_Type *base, const i3c_master_config_t *masterConfig, uint32_t
                    sourceClock_Hz)
```

Initializes the I3C master peripheral.

This function enables the peripheral clock and initializes the I3C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

#### Parameters

- `base` – The I3C peripheral base address.
- `masterConfig` – User provided peripheral configuration. Use `I3C_MasterGetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

```
void I3C_MasterDeinit(I3C_Type *base)
```

Deinitializes the I3C master peripheral.

This function disables the I3C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

#### Parameters

- `base` – The I3C peripheral base address.

```
status_t I3C_MasterCheckAndClearError(I3C_Type *base, uint32_t status)
```

```
status_t I3C_MasterWaitForCtrlDone(I3C_Type *base, bool waitIdle)
```

```
status_t I3C_CheckForBusyBus(I3C_Type *base)
```

```
static inline void I3C_MasterEnable(I3C_Type *base, i3c_master_enable_t enable)
```

Set I3C module master mode.

#### Parameters

- `base` – The I3C peripheral base address.
- `enable` – Enable master mode.

```
void I3C_SlaveGetDefaultConfig(i3c_slave_config_t *slaveConfig)
```

Provides a default configuration for the I3C slave peripheral.

This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableslave = true;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with `I3C_SlaveInit()`.

#### Parameters

- `slaveConfig` – **[out]** User provided configuration structure for default values. Refer to `i3c_slave_config_t`.

```
void I3C_SlaveInit(I3C_Type *base, const i3c_slave_config_t *slaveConfig, uint32_t
                  slowClock_Hz)
```

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

#### Parameters



- `base` – The I3C peripheral base address.
- `slaveConfig` – User provided peripheral configuration. Use `I3C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- `slowClock_Hz` – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If `FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH` defines as 1, this parameter is useless.

`void I3C_SlaveDeinit(I3C_Type *base)`

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

#### Parameters

- `base` – The I3C peripheral base address.

`static inline void I3C_SlaveEnable(I3C_Type *base, bool isEnabled)`

Enable/Disable Slave.

#### Parameters

- `base` – The I3C peripheral base address.
- `isEnabled` – Enable or disable.

`static inline uint32_t I3C_MasterGetStatusFlags(I3C_Type *base)`

Gets the I3C master status flags.

A bit mask with the state of all I3C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

#### See also:

`_i3c_master_flags`

#### Parameters

- `base` – The I3C peripheral base address.

#### Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

`static inline void I3C_MasterClearStatusFlags(I3C_Type *base, uint32_t statusMask)`

Clears the I3C master status flag state.

The following status register flags can be cleared:

- `kI3C_MasterSlaveStartFlag`
- `kI3C_MasterControlDoneFlag`
- `kI3C_MasterCompleteFlag`
- `kI3C_MasterArbitrationWonFlag`
- `kI3C_MasterSlave2MasterFlag`

Attempts to clear other flags has no effect.

**See also:**

`_i3c_master_flags`.

**Parameters**

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_master_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_MasterGetStatusFlags()`.

```
static inline uint32_t I3C_MasterGetErrorStatusFlags(I3C_Type *base)
```

Gets the I3C master error status flags.

A bit mask with the state of all I3C master error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

**See also:**

`_i3c_master_error_flags`

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_MasterClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C master error status flag state.

**See also:**

`_i3c_master_error_flags`.

**Parameters**

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_master_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_MasterGetStatusFlags()`.

```
i3c_master_state_t I3C_MasterGetState(I3C_Type *base)
```

Gets the I3C master state.

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

I3C master state.

```
static inline uint32_t I3C_SlaveGetStatusFlags(I3C_Type *base)
```

Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

**See also:**

`_i3c_slave_flags`

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_SlaveClearStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C slave status flag state.

The following status register flags can be cleared:

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`

Attempts to clear other flags has no effect.

**See also:**

`_i3c_slave_flags`.

**Parameters**

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetStatusFlags()`.

```
static inline uint32_t I3C_SlaveGetErrorStatusFlags(I3C_Type *base)
```

Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

**See also:**

`_i3c_slave_error_flags`

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_SlaveClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C slave error status flag state.

**See also:**

`_i3c_slave_error_flags`.

**Parameters**

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_slave_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetErrorStatusFlags()`.

`i3c_slave_activity_state_t` `I3C_SlaveGetActivityState(I3C_Type *base)`

Gets the I3C slave state.

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

I3C slave activity state, refer `i3c_slave_activity_state_t`.

`status_t` `I3C_SlaveCheckAndClearError(I3C_Type *base, uint32_t status)`

`static inline void` `I3C_MasterEnableInterrupts(I3C_Type *base, uint32_t interruptMask)`

Enables the I3C master interrupt requests.

All flags except `kI3C_MasterBetweenFlag` and `kI3C_MasterNackDetectFlag` can be enabled as interrupts.

**Parameters**

- `base` – The I3C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_i3c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline void` `I3C_MasterDisableInterrupts(I3C_Type *base, uint32_t interruptMask)`

Disables the I3C master interrupt requests.

All flags except `kI3C_MasterBetweenFlag` and `kI3C_MasterNackDetectFlag` can be enabled as interrupts.

**Parameters**

- `base` – The I3C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_i3c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline uint32_t` `I3C_MasterGetEnabledInterrupts(I3C_Type *base)`

Returns the set of currently enabled I3C master interrupt requests.

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

A bitmask composed of `_i3c_master_flags` enumerators OR'd together to indicate the set of enabled interrupts.

`static inline uint32_t` `I3C_MasterGetPendingInterrupts(I3C_Type *base)`

Returns the set of pending I3C master interrupt requests.

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

A bitmask composed of `_i3c_master_flags` enumerators OR'd together to indicate the set of pending interrupts.

static inline void I3C\_SlaveEnableInterrupts(I3C\_Type \*base, uint32\_t interruptMask)

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`
- `kI3C_SlaveErrorFlag`
- `kI3C_SlaveHDRCommandMatchFlag`
- `kI3C_SlaveCCCHandledFlag`
- `kI3C_SlaveEventSentFlag`

**Parameters**

- `base` – The I3C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline void I3C\_SlaveDisableInterrupts(I3C\_Type \*base, uint32\_t interruptMask)

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`
- `kI3C_SlaveErrorFlag`
- `kI3C_SlaveHDRCommandMatchFlag`
- `kI3C_SlaveCCCHandledFlag`
- `kI3C_SlaveEventSentFlag`

**Parameters**

- `base` – The I3C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline uint32_t I3C_SlaveGetEnabledInterrupts(I3C_Type *base)
```

Returns the set of currently enabled I3C slave interrupt requests.

**Parameters**

- base – The I3C peripheral base address.

**Returns**

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

```
static inline uint32_t I3C_SlaveGetPendingInterrupts(I3C_Type *base)
```

Returns the set of pending I3C slave interrupt requests.

**Parameters**

- base – The I3C peripheral base address.

**Returns**

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of pending interrupts.

```
static inline void I3C_MasterEnableDMA(I3C_Type *base, bool enableTx, bool enableRx,  
                                       uint32_t width)
```

Enables or disables I3C master DMA requests.

**Parameters**

- base – The I3C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.
- width – DMA read/write unit in bytes.

```
static inline uint32_t I3C_MasterGetTxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C master transmit data register address for DMA transfer.

**Parameters**

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

**Returns**

The I3C Master Transmit Data Register address.

```
static inline uint32_t I3C_MasterGetRxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C master receive data register address for DMA transfer.

**Parameters**

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

**Returns**

The I3C Master Receive Data Register address.

```
static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t  
                                       width)
```

Enables or disables I3C slave DMA requests.

**Parameters**

- base – The I3C peripheral base address.

- `enableTx` – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- `enableRx` – Enable flag for receive DMA request. Pass true for enable, false for disable.
- `width` – DMA read/write unit in bytes.

static inline uint32\_t I3C\_SlaveGetTxFifoAddress(I3C\_Type \*base, uint32\_t width)

Gets I3C slave transmit data register address for DMA transfer.

#### Parameters

- `base` – The I3C peripheral base address.
- `width` – DMA read/write unit in bytes.

#### Returns

The I3C Slave Transmit Data Register address.

static inline uint32\_t I3C\_SlaveGetRxFifoAddress(I3C\_Type \*base, uint32\_t width)

Gets I3C slave receive data register address for DMA transfer.

#### Parameters

- `base` – The I3C peripheral base address.
- `width` – DMA read/write unit in bytes.

#### Returns

The I3C Slave Receive Data Register address.

static inline void I3C\_MasterSetWatermarks(I3C\_Type \*base, *i3c\_tx\_trigger\_level\_t* txLvl,  
*i3c\_rx\_trigger\_level\_t* rxLvl, bool flushTx, bool flushRx)

Sets the watermarks for I3C master FIFOs.

#### Parameters

- `base` – The I3C peripheral base address.
- `txLvl` – Transmit FIFO watermark level. The `kI3C_MasterTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches `txLvl`.
- `rxLvl` – Receive FIFO watermark level. The `kI3C_MasterRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches `rxLvl`.
- `flushTx` – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- `flushRx` – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

static inline void I3C\_MasterGetFifoCounts(I3C\_Type \*base, size\_t \*rxCount, size\_t \*txCount)

Gets the current number of bytes in the I3C master FIFOs.

#### Parameters

- `base` – The I3C peripheral base address.
- `txCount` – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

static inline void I3C\_SlaveSetWatermarks(I3C\_Type \*base, *i3c\_tx\_trigger\_level\_t* txLvl,  
*i3c\_rx\_trigger\_level\_t* rxLvl, bool flushTx, bool flushRx)

Sets the watermarks for I3C slave FIFOs.

#### Parameters

- `base` – The I3C peripheral base address.
- `txLvl` – Transmit FIFO watermark level. The `kI3C_SlaveTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches `txLvl`.
- `rxLvl` – Receive FIFO watermark level. The `kI3C_SlaveRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches `rxLvl`.
- `flushTx` – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- `flushRx` – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

```
static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of bytes in the I3C slave FIFOs.

#### Parameters

- `base` – The I3C peripheral base address.
- `txCount` – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

```
void I3C_MasterSetBaudRate(I3C_Type *base, const i3c_baudrate_hz_t *baudRate_Hz, uint32_t  
sourceClock_Hz)
```

Sets the I3C bus frequency for master transactions.

The I3C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

#### Parameters

- `base` – The I3C peripheral base address.
- `baudRate_Hz` – Pointer to structure of requested bus frequency in Hertz.
- `sourceClock_Hz` – I3C functional clock frequency in Hertz.

```
static inline bool I3C_MasterGetBusIdleState(I3C_Type *base)
```

Returns whether the bus is idle.

Requires the master mode to be enabled.

#### Parameters

- `base` – The I3C peripheral base address.

#### Return values

- `true` – Bus is busy.
- `false` – Bus is idle.

```
status_t I3C_MasterStartWithRxSize(I3C_Type *base, i3c_bus_type_t type, uint8_t address,  
i3c_direction_t dir, uint8_t rxSize)
```

Sends a START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the `address` parameter. Note that this function



does not actually wait until the START and address are successfully sent on the bus before returning.

#### Parameters

- `base` – The I3C peripheral base address.
- `type` – The bus type to use in this transaction.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- `rxSize` – Read terminate size for the followed read transfer, limit to 255 bytes.

#### Return values

- `kStatus_Success` – START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.

*status\_t* I3C\_MasterStart(I3C\_Type \*base, i3c\_bus\_type\_t type, uint8\_t address, i3c\_direction\_t dir)

Sends a START signal and slave address on the I2C/I3C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

#### Parameters

- `base` – The I3C peripheral base address.
- `type` – The bus type to use in this transaction.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

#### Return values

- `kStatus_Success` – START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.

*status\_t* I3C\_MasterRepeatedStartWithRxSize(I3C\_Type \*base, i3c\_bus\_type\_t type, uint8\_t address, i3c\_direction\_t dir, uint8\_t rxSize)

Sends a repeated START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to send a Repeated START signal when a transfer is already in progress. Like `I3C_MasterStart()`, it also sends the specified 7-bit address. Call this API also configures the read terminate size for the following read transfer. For example, set the `rxSize` = 2, the following read transfer will be terminated after two bytes of data received. Write transfer will not be affected by the `rxSize` configuration.

---

**Note:** This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

---

**Parameters**

- `base` – The I3C peripheral base address.
- `type` – The bus type to use in this transaction.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- `rxSize` – Read terminate size for the followed read transfer, limit to 255 bytes.

**Return values**

`kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.

```
static inline status_t I3C_MasterRepeatedStart(I3C_Type *base, i3c_bus_type_t type, uint8_t address, i3c_direction_t dir)
```

Sends a repeated START signal and slave address on the I2C/I3C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like `I3C_MasterStart()`, it also sends the specified 7-bit address.

---

**Note:** This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

---

**Parameters**

- `base` – The I3C peripheral base address.
- `type` – The bus type to use in this transaction.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

**Return values**

`kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.

```
status_t I3C_MasterSend(I3C_Type *base, const void *txBuff, size_t txSize, uint32_t flags)
```

Performs a polling send transfer on the I2C/I3C bus.

Sends up to `txSize` number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns `kStatus_I3C_Nak`.

**Parameters**

- `base` – The I3C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.
- `flags` – Bit mask of options for the transfer. See enumeration `_i3c_master_transfer_flags` for available options.

**Return values**

- `kStatus_Success` – Data was sent successfully.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.

- `kStatus_I3C_Nak` – The slave device sent a NAK in response to an address.
- `kStatus_I3C_WriteAbort` – The slave device sent a NAK in response to a write.
- `kStatus_I3C_MsgError` – Message SDR/DDR mismatch or read/write message in wrong state.
- `kStatus_I3C_WriteFifoError` – Write to M/SWDATAB register when FIFO full.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

*status\_t* I3C\_MasterReceive(I3C\_Type \*base, void \*rxBuff, size\_t rxSize, uint32\_t flags)

Performs a polling receive transfer on the I2C/I3C bus.

#### Parameters

- `base` – The I3C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.
- `flags` – Bit mask of options for the transfer. See enumeration `_i3c_master_transfer_flags` for available options.

#### Return values

- `kStatus_Success` – Data was received successfully.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.
- `kStatus_I3C_Term` – The master terminates slave read.
- `kStatus_I3C_HdrParityError` – Parity error from DDR read.
- `kStatus_I3C_CrcError` – CRC error from DDR read.
- `kStatus_I3C_MsgError` – Message SDR/DDR mismatch or read/write message in wrong state.
- `kStatus_I3C_ReadFifoError` – Read from M/SRDATAB register when FIFO empty.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

*status\_t* I3C\_MasterStop(I3C\_Type \*base)

Sends a STOP signal on the I2C/I3C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

#### Parameters

- `base` – The I3C peripheral base address.

#### Return values

- `kStatus_Success` – The STOP signal was successfully sent on the bus and the transaction terminated.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

void I3C\_MasterEmitRequest(I3C\_Type \*base, *i3c\_bus\_request\_t* masterReq)

I3C master emit request.

**Parameters**

- base – The I3C peripheral base address.
- masterReq – I3C master request of type *i3c\_bus\_request\_t*

static inline void I3C\_MasterEmitIBIResponse(I3C\_Type \*base, *i3c\_ibi\_response\_t* ibiResponse)

I3C master emit request.

**Parameters**

- base – The I3C peripheral base address.
- ibiResponse – I3C master emit IBI response of type *i3c\_ibi\_response\_t*

void I3C\_MasterRegisterIBI(I3C\_Type \*base, *i3c\_register\_ibi\_addr\_t* \*ibiRule)

I3C master register IBI rule.

**Parameters**

- base – The I3C peripheral base address.
- ibiRule – Pointer to ibi rule description of type *i3c\_register\_ibi\_addr\_t*

void I3C\_MasterGetIBIRules(I3C\_Type \*base, *i3c\_register\_ibi\_addr\_t* \*ibiRule)

I3C master get IBI rule.

**Parameters**

- base – The I3C peripheral base address.
- ibiRule – Pointer to store the read out ibi rule description.

*i3c\_ibi\_type\_t* I3C\_GetIBIType(I3C\_Type \*base)

I3C master get IBI Type.

**Parameters**

- base – The I3C peripheral base address.

**Return values**

*i3c\_ibi\_type\_t* – Type of *i3c\_ibi\_type\_t*.

static inline uint8\_t I3C\_GetIBIAddress(I3C\_Type \*base)

I3C master get IBI Address.

**Parameters**

- base – The I3C peripheral base address.

**Return values**

The – 8-bit IBI address.

*status\_t* I3C\_MasterProcessDAASpecifiedBaudrate(I3C\_Type \*base, uint8\_t \*addressList, uint32\_t count, *i3c\_master\_daa\_baudrate\_t* \*daaBaudRate)

Performs a DAA in the i3c bus with specified temporary baud rate.

**Parameters**

- base – The I3C peripheral base address.
- addressList – The pointer for address list which is used to do DAA.
- count – The address count in the address list.

- `daaBaudRate` – The temporary baud rate in DAA process, NULL for using initial setting. The initial setting is set back between the completion of the DAA and the return of this function.

#### Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- `kStatus_I3C_SlaveCountExceed` – The I3C slave count has exceed the definition in `I3C_MAX_DEVCNT`.

```
static inline status_t I3C_MasterProcessDAA(I3C_Type *base, uint8_t *addressList, uint32_t count)
```

Performs a DAA in the i3c bus.

#### Parameters

- `base` – The I3C peripheral base address.
- `addressList` – The pointer for address list which is used to do DAA.
- `count` – The address count in the address list. The initial setting is set back between the completion of the DAA and the return of this function.

#### Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- `kStatus_I3C_SlaveCountExceed` – The I3C slave count has exceed the definition in `I3C_MAX_DEVCNT`.

```
i3c_device_info_t *I3C_MasterGetDeviceListAfterDAA(I3C_Type *base, uint8_t *count)
```

Get device information list after DAA process is done.

#### Parameters

- `base` – The I3C peripheral base address.
- `count` – **[out]** The pointer to store the available device count.

#### Returns

Pointer to the `i3c_device_info_t` array.

```
void I3C_MasterClearDeviceCount(I3C_Type *base)
```

Clear the global device count which represents current devices number on the bus. When user resets all dynamic addresses on the bus, should call this API.

#### Parameters

- `base` – The I3C peripheral base address.

```
status_t I3C_MasterTransferBlocking(I3C_Type *base, i3c_master_transfer_t *transfer)
```

Performs a master polling transfer on the I2C/I3C bus.

---

**Note:** The API does not return until the transfer succeeds or fails due to error happens during transfer.

---

#### Parameters

- `base` – The I3C peripheral base address.
- `transfer` – Pointer to the transfer structure.

**Return values**

- `kStatus_I3C_Success` – Data was received successfully.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_IBIWon` – The I3C slave event IBI or MR or HJ won the arbitration on a header address.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.
- `kStatus_I3C_Nak` – The slave device sent a NAK in response to an address.
- `kStatus_I3C_WriteAbort` – The slave device sent a NAK in response to a write.
- `kStatus_I3C_Term` – The master terminates slave read.
- `kStatus_I3C_HdrParityError` – Parity error from DDR read.
- `kStatus_I3C_CrcError` – CRC error from DDR read.
- `kStatus_I3C_MsgError` – Message SDR/DDR mismatch or read/write message in wrong state.
- `kStatus_I3C_ReadFifoError` – Read from M/SRDATAB register when FIFO empty.
- `kStatus_I3C_WriteFifoError` – Write to M/SWDATAB register when FIFO full.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

*status\_t* I3C\_SlaveSend(I3C\_Type \*base, const void \*txBuff, size\_t txSize)

Performs a polling send transfer on the I3C bus.

**Parameters**

- `base` – The I3C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.

**Returns**

Error or success status returned by API.

*status\_t* I3C\_SlaveReceive(I3C\_Type \*base, void \*rxBuff, size\_t rxSize)

Performs a polling receive transfer on the I3C bus.

**Parameters**

- `base` – The I3C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.

**Returns**

Error or success status returned by API.

`void` I3C\_MasterTransferCreateHandle(I3C\_Type \*base, *i3c\_master\_handle\_t* \*handle, const *i3c\_master\_transfer\_callback\_t* \*callback, void \*userData)

Creates a new handle for the I3C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_MasterTransferAbort()` API shall be called.

---

**Note:** The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

---

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – **[out]** Pointer to the I3C master driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

*status\_t* I3C\_MasterTransferNonBlocking(I3C\_Type \*base, i3c\_master\_handle\_t \*handle, i3c\_master\_transfer\_t \*transfer)

Performs a non-blocking transaction on the I2C/I3C bus.

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.
- `transfer` – The pointer to the transfer descriptor.

#### Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

*status\_t* I3C\_MasterTransferGetCount(I3C\_Type \*base, i3c\_master\_handle\_t \*handle, size\_t \*count)

Returns number of bytes transferred so far.

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.
- `count` – **[out]** Number of bytes transferred so far by the non-blocking transaction.

#### Return values

- `kStatus_Success` –
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

*void* I3C\_MasterTransferAbort(I3C\_Type \*base, i3c\_master\_handle\_t \*handle)

Terminates a non-blocking I3C master transmission early.

---

**Note:** It is not safe to call this function from an IRQ handler that has a higher priority than the I3C peripheral's IRQ priority.

---

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.

`void I3C_MasterTransferHandleIRQ(I3C_Type *base, void *intHandle)`

Reusable routine to handle master interrupts.

---

**Note:** This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

---

#### Parameters

- `base` – The I3C peripheral base address.
- `intHandle` – Pointer to the I3C master driver handle.

`enum _i3c_master_flags`

I3C master peripheral flags.

The following status register flags can be cleared:

- `kI3C_MasterSlaveStartFlag`
- `kI3C_MasterControlDoneFlag`
- `kI3C_MasterCompleteFlag`
- `kI3C_MasterArbitrationWonFlag`
- `kI3C_MasterSlave2MasterFlag`

All flags except `kI3C_MasterBetweenFlag` and `kI3C_MasterNackDetectFlag` can be enabled as interrupts.

---

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

---

#### Values:

enumerator `kI3C_MasterBetweenFlag`

Between messages/DAA's flag

enumerator `kI3C_MasterNackDetectFlag`

NACK detected flag

enumerator `kI3C_MasterSlaveStartFlag`

Slave request start flag

enumerator `kI3C_MasterControlDoneFlag`

Master request complete flag

enumerator `kI3C_MasterCompleteFlag`

Transfer complete flag

enumerator `kI3C_MasterRxReadyFlag`

Rx data ready in Rx buffer flag

enumerator `kI3C_MasterTxReadyFlag`

Tx buffer ready for Tx data flag

enumerator `kI3C_MasterArbitrationWonFlag`

Header address won arbitration flag

enumerator `kI3C_MasterErrorFlag`

Error occurred flag



enumerator kI3C\_MasterSlave2MasterFlag

Switch from slave to master flag

enumerator kI3C\_MasterClearFlags

enum \_i3c\_master\_error\_flags

I3C master error flags to indicate the causes.

---

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

---

*Values:*

enumerator kI3C\_MasterErrorNackFlag

Slave NACKed the last address

enumerator kI3C\_MasterErrorWriteAbortFlag

Slave NACKed the write data

enumerator kI3C\_MasterErrorParityFlag

Parity error from DDR read

enumerator kI3C\_MasterErrorCrcFlag

CRC error from DDR read

enumerator kI3C\_MasterErrorReadFlag

Read from MRDATAB register when FIFO empty

enumerator kI3C\_MasterErrorWriteFlag

Write to MWDATAB register when FIFO full

enumerator kI3C\_MasterErrorMsgFlag

Message SDR/DDR mismatch or read/write message in wrong state

enumerator kI3C\_MasterErrorInvalidReqFlag

Invalid use of request

enumerator kI3C\_MasterErrorTimeoutFlag

The module has stalled too long in a frame

enumerator kI3C\_MasterAllErrorFlags

All error flags

enum \_i3c\_master\_state

I3C working master state.

*Values:*

enumerator kI3C\_MasterStateIdle

Bus stopped.

enumerator kI3C\_MasterStateSlvReq

Bus stopped but slave holding SDA low.

enumerator kI3C\_MasterStateMsgSdr

In SDR Message mode from using MWMSG\_SDR.

enumerator kI3C\_MasterStateNormAct

In normal active SDR mode.

enumerator kI3C\_MasterStateDdr

In DDR Message mode.

enumerator kI3C\_MasterStateDaa

In ENTDAAs mode.

enumerator kI3C\_MasterStateIbiAck

Waiting on IBI ACK/NACK decision.

enumerator kI3C\_MasterStateIbiRcv

Receiving IBI.

enum \_i3c\_master\_enable

I3C master enable configuration.

*Values:*

enumerator kI3C\_MasterOff

Master off.

enumerator kI3C\_MasterOn

Master on.

enumerator kI3C\_MasterCapable

Master capable.

enum \_i3c\_master\_hkeep

I3C high keeper configuration.

*Values:*

enumerator kI3C\_MasterHighKeeperNone

Use PUR to hold SCL high.

enumerator kI3C\_MasterHighKeeperWiredIn

Use pin\_HK controls.

enumerator kI3C\_MasterPassiveSDA

Hi-Z for Bus Free and hold SDA.

enumerator kI3C\_MasterPassiveSDASCL

Hi-Z both for Bus Free, and can Hi-Z SDA for hold.

enum \_i3c\_bus\_request

Emits the requested operation when doing in pieces vs. by message.

*Values:*

enumerator kI3C\_RequestNone

No request.

enumerator kI3C\_RequestEmitStartAddr

Request to emit start and address on bus.

enumerator kI3C\_RequestEmitStop

Request to emit stop on bus.

enumerator kI3C\_RequestIbiAckNack

Manual IBI ACK or NACK.

enumerator kI3C\_RequestProcessDAA

Process DAA.

enumerator kI3C\_RequestForceExit

Request to force exit.

enumerator kI3C\_RequestAutoIbi  
Hold in stopped state, but Auto-emit START,7E.

enum \_i3c\_bus\_type

Bus type with EmitStartAddr.

*Values:*

enumerator kI3C\_TypeI3CSdr  
SDR mode of I3C.

enumerator kI3C\_TypeI2C  
Standard i2c protocol.

enumerator kI3C\_TypeI3CDdr  
HDR-DDR mode of I3C.

enum \_i3c\_ibi\_response

IBI response.

*Values:*

enumerator kI3C\_IbiRespAck  
ACK with no mandatory byte.

enumerator kI3C\_IbiRespNack  
NACK.

enumerator kI3C\_IbiRespAckMandatory  
ACK with mandatory byte.

enumerator kI3C\_IbiRespManual  
Reserved.

enum \_i3c\_ibi\_type

IBI type.

*Values:*

enumerator kI3C\_IbiNormal  
In-band interrupt.

enumerator kI3C\_IbiHotJoin  
slave hot join.

enumerator kI3C\_IbiMasterRequest  
slave master ship request.

enum \_i3c\_ibi\_state

IBI state.

*Values:*

enumerator kI3C\_IbiReady  
In-band interrupt ready state, ready for user to handle.

enumerator kI3C\_IbiDataBuffNeed  
In-band interrupt need data buffer for data receive.

enumerator kI3C\_IbiAckNackPending  
In-band interrupt Ack/Nack pending for decision.

enum \_i3c\_direction

Direction of master and slave transfers.

*Values:*

enumerator kI3C\_Write

Master transmit.

enumerator kI3C\_Read

Master receive.

enum \_i3c\_tx\_trigger\_level

Watermark of TX int/dma trigger level.

*Values:*

enumerator kI3C\_TxTriggerOnEmpty

Trigger on empty.

enumerator kI3C\_TxTriggerUntilOneQuarterOrLess

Trigger on 1/4 full or less.

enumerator kI3C\_TxTriggerUntilOneHalfOrLess

Trigger on 1/2 full or less.

enumerator kI3C\_TxTriggerUntilOneLessThanFull

Trigger on 1 less than full or less.

enum \_i3c\_rx\_trigger\_level

Watermark of RX int/dma trigger level.

*Values:*

enumerator kI3C\_RxTriggerOnNotEmpty

Trigger on not empty.

enumerator kI3C\_RxTriggerUntilOneQuarterOrMore

Trigger on 1/4 full or more.

enumerator kI3C\_RxTriggerUntilOneHalfOrMore

Trigger on 1/2 full or more.

enumerator kI3C\_RxTriggerUntilThreeQuarterOrMore

Trigger on 3/4 full or more.

enum \_i3c\_rx\_term\_ops

I3C master read termination operations.

*Values:*

enumerator kI3C\_RxTermDisable

Master doesn't terminate read, used for CCC transfer.

enumerator kI3C\_RxAutoTerm

Master auto terminate read after receiving specified bytes(<=255).

enumerator kI3C\_RxTermLastByte

Master terminates read at any time after START, no length limitation.

enum \_i3c\_start\_scl\_delay

I3C start SCL delay options.

*Values:*

enumerator kI3C\_NoDelay

No delay.

enumerator kI3C\_IncreaseSclHalfPeriod

Increases SCL clock period by 1/2.

enumerator kI3C\_IncreaseSclOnePeriod

Increases SCL clock period by 1.

enumerator kI3C\_IncreaseSclOneAndHalfPeriod

Increases SCL clock period by 1 1/2

enum \_i3c\_master\_transfer\_flags

Transfer option flags.

---

**Note:** These enumerations are intended to be OR'd together to form a bit mask of options for the `_i3c_master_transfer::flags` field.

---

*Values:*

enumerator kI3C\_TransferDefaultFlag

Transfer starts with a start signal, stops with a stop signal.

enumerator kI3C\_TransferNoStartFlag

Don't send a start condition, address, and sub address

enumerator kI3C\_TransferRepeatedStartFlag

Send a repeated start condition

enumerator kI3C\_TransferNoStopFlag

Don't send a stop condition.

enumerator kI3C\_TransferWordsFlag

Transfer in words, else transfer in bytes.

enumerator kI3C\_TransferDisableRxTermFlag

Disable Rx termination. Note: It's for I3C CCC transfer.

enumerator kI3C\_TransferRxAutoTermFlag

Set Rx auto-termination. Note: It's adaptive based on Rx size(<=255 bytes) except in I3C\_MasterReceive.

enumerator kI3C\_TransferStartWithBroadcastAddr

Start transfer with 0x7E, then read/write data with device address.

typedef enum \_i3c\_master\_state i3c\_master\_state\_t

I3C working master state.

typedef enum \_i3c\_master\_enable i3c\_master\_enable\_t

I3C master enable configuration.

typedef enum \_i3c\_master\_hkeep i3c\_master\_hkeep\_t

I3C high keeper configuration.

typedef enum \_i3c\_bus\_request i3c\_bus\_request\_t

Emits the requested operation when doing in pieces vs. by message.

typedef enum \_i3c\_bus\_type i3c\_bus\_type\_t

Bus type with EmitStartAddr.

```
typedef enum _i3c_ibi_response i3c_ibi_response_t
    IBI response.
```

```
typedef enum _i3c_ibi_type i3c_ibi_type_t
    IBI type.
```

```
typedef enum _i3c_ibi_state i3c_ibi_state_t
    IBI state.
```

```
typedef enum _i3c_direction i3c_direction_t
    Direction of master and slave transfers.
```

```
typedef enum _i3c_tx_trigger_level i3c_tx_trigger_level_t
    Watermark of TX int/dma trigger level.
```

```
typedef enum _i3c_rx_trigger_level i3c_rx_trigger_level_t
    Watermark of RX int/dma trigger level.
```

```
typedef enum _i3c_rx_term_ops i3c_rx_term_ops_t
    I3C master read termination operations.
```

```
typedef enum _i3c_start_scl_delay i3c_start_scl_delay_t
    I3C start SCL delay options.
```

```
typedef struct _i3c_register_ibi_addr i3c_register_ibi_addr_t
    Structure with setting master IBI rules and slave registry.
```

```
typedef struct _i3c_baudrate i3c_baudrate_hz_t
    Structure with I3C baudrate settings.
```

```
typedef struct _i3c_master_daa_baudrate i3c_master_daa_baudrate_t
    I3C DAA baud rate configuration.
```

```
typedef struct _i3c_master_config i3c_master_config_t
    Structure with settings to initialize the I3C master module.
```

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_MasterGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

```
typedef struct _i3c_master_transfer i3c_master_transfer_t
```

```
typedef struct _i3c_master_handle i3c_master_handle_t
```

```
typedef struct _i3c_master_transfer_callback i3c_master_transfer_callback_t
    i3c master callback functions.
```

```
typedef void (*i3c_master_isr_t)(I3C_Type *base, void *handle)
    Typedef for master interrupt handler.
```

```
struct _i3c_register_ibi_addr
    #include <fsl_i3c.h> Structure with setting master IBI rules and slave registry.
```

### Public Members

```
uint8_t address[5]
    Address array for registry.
```

```
bool i3cFastStart
    Allow the START header to run as push-pull speed if all dynamic addresses take MSB 0.
```

bool ibiHasPayload

Whether the address array has mandatory IBI byte.

struct \_\_i3c\_\_baudrate

*#include <fsl\_i3c.h>* Structure with I3C baudrate settings.

### Public Members

uint32\_t i2cBaud

Desired I2C baud rate in Hertz.

uint32\_t i3cPushPullBaud

Desired I3C push-pull baud rate in Hertz.

uint32\_t i3cOpenDrainBaud

Desired I3C open-drain baud rate in Hertz.

struct \_\_i3c\_\_master\_\_daa\_\_baudrate

*#include <fsl\_i3c.h>* I3C DAA baud rate configuration.

### Public Members

uint32\_t sourceClock\_Hz

FCLK, function clock in Hertz.

uint32\_t i3cPushPullBaud

Desired I3C push-pull baud rate in Hertz.

uint32\_t i3cOpenDrainBaud

Desired I3C open-drain baud rate in Hertz.

struct \_\_i3c\_\_master\_\_config

*#include <fsl\_i3c.h>* Structure with settings to initialize the I3C master module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C\_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

### Public Members

i3c\_master\_enable\_t enableMaster

Enable master mode.

bool disableTimeout

Whether to disable timeout to prevent the ERRWARN.

i3c\_master\_hkeep\_t hKeep

High keeper mode setting.

bool enableOpenDrainStop

Whether to emit open-drain speed STOP.

bool enableOpenDrainHigh

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

i3c\_baudrate\_hz\_t baudRate\_Hz

Desired baud rate settings.

*i3c\_start\_scl\_delay\_t* startSclDelay

I3C SCL delay after START.

*i3c\_start\_scl\_delay\_t* restartSclDelay

I3C SCL delay after Repeated START.

struct *\_i3c\_master\_transfer\_callback*

#include <fsl\_i3c.h> i3c master callback functions.

### Public Members

void (\*slave2Master)(I3C\_Type \*base, void \*userData)

Transfer complete callback

void (\*ibiCallback)(I3C\_Type \*base, *i3c\_master\_handle\_t* \*handle, *i3c\_ibi\_type\_t* ibiType, *i3c\_ibi\_state\_t* ibiState)

IBI event callback

void (\*transferComplete)(I3C\_Type \*base, *i3c\_master\_handle\_t* \*handle, *status\_t* completionStatus, void \*userData)

Transfer complete callback

struct *\_i3c\_master\_transfer*

#include <fsl\_i3c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the I3C\_MasterTransferNonBlocking() API.

### Public Members

uint32\_t flags

Bit mask of options for the transfer. See enumeration *\_i3c\_master\_transfer\_flags* for available options. Set to 0 or *kI3C\_TransferDefaultFlag* for normal transfers.

uint8\_t slaveAddress

The 7-bit slave address.

*i3c\_direction\_t* direction

Either *kI3C\_Read* or *kI3C\_Write*.

uint32\_t subaddress

Sub address. Transferred MSB first.

size\_t subaddressSize

Length of sub address to send in bytes. Maximum size is 4 bytes.

void \*data

Pointer to data to transfer.

size\_t dataSize

Number of bytes to transfer.

*i3c\_bus\_type\_t* busType

bus type.

*i3c\_ibi\_response\_t* ibiResponse

ibi response during transfer.



```
struct _i3c_master_handle
    #include <fsl_i3c.h> Driver handle for master non-blocking APIs.
```

---

**Note:** The contents of this structure are private and subject to change.

---

### Public Members

```
uint8_t state
    Transfer state machine current state.

uint32_t remainingBytes
    Remaining byte count in current state.

i3c_rx_term_ops_t rxTermOps
    Read termination operation.

i3c_master_transfer_t transfer
    Copy of the current transfer info.

uint8_t ibiAddress
    Slave address which request IBI.

uint8_t *ibiBuff
    Pointer to IBI buffer to keep ibi bytes.

size_t ibiPayloadSize
    IBI payload size.

i3c_ibi_type_t ibiType
    IBI type.

i3c_master_transfer_callback_t callback
    Callback functions pointer.

void *userData
    Application data passed to callback.
```

## 2.25 I3C Master DMA Driver

```
void I3C_MasterTransferCreateHandleEDMA(I3C_Type *base, i3c_master_edma_handle_t
    *handle, const i3c_master_edma_callback_t
    *callback, void *userData, edma_handle_t
    *rxDmaHandle, edma_handle_t *txDmaHandle)
```

Create a new handle for the I3C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_MasterTransferAbortDMA()` API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

### Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C master driver handle.
- *callback* – User provided pointer to the asynchronous callback function.

- `userData` – User provided pointer to the application callback data.
- `rxDmaHandle` – Handle for the DMA receive channel. Created by the user prior to calling this function.
- `txDmaHandle` – Handle for the DMA transmit channel. Created by the user prior to calling this function.

`status_t I3C_MasterTransferEDMA(I3C_Type *base, i3c_master_edma_handle_t *handle, i3c_master_transfer_t *transfer)`

Performs a non-blocking DMA-based transaction on the I3C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.
- `transfer` – The pointer to the transfer descriptor.

#### Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

`status_t I3C_MasterTransferGetCountEDMA(I3C_Type *base, i3c_master_edma_handle_t *handle, size_t *count)`

Returns number of bytes transferred so far.

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.
- `count` – **[out]** Number of bytes transferred so far by the non-blocking transaction.

#### Return values

- `kStatus_Success` –
- `kStatus_NoTransferInProgress` – There is not a DMA transaction currently in progress.

`void I3C_MasterTransferAbortEDMA(I3C_Type *base, i3c_master_edma_handle_t *handle)`

Terminates a non-blocking I3C master transmission early.

---

**Note:** It is not safe to call this function from an IRQ handler that has a higher priority than the DMA peripheral's IRQ priority.

---

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.

`void I3C_MasterTransferEDMAHandleIRQ(I3C_Type *base, void *i3cHandle)`

Reusable routine to handle master interrupts.

---

**Note:** This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

---

### Parameters

- `base` – The I3C peripheral base address.
- `i3cHandle` – Pointer to the I3C master DMA driver handle.

```
typedef struct _i3c_master_edma_handle i3c_master_edma_handle_t
```

```
typedef struct _i3c_master_edma_callback i3c_master_edma_callback_t
    i3c master callback functions.
```

```
struct _i3c_master_edma_callback
    #include <fsl_i3c_edma.h> i3c master callback functions.
```

### Public Members

```
void (*slave2Master)(I3C_Type *base, void *userData)
    Target asks for controller request.
```

```
void (*ibiCallback)(I3C_Type *base, i3c_master_edma_handle_t *handle, i3c_ibi_type_t
ibiType, i3c_ibi_state_t ibiState)
    IBI event callback.
```

```
void (*transferComplete)(I3C_Type *base, i3c_master_edma_handle_t *handle, status_t
status, void *userData)
    Transfer complete callback.
```

```
struct _i3c_master_edma_handle
    #include <fsl_i3c_edma.h> Driver handle for master EDMA APIs.
```

---

**Note:** The contents of this structure are private and subject to change.

---

### Public Members

```
I3C_Type *base
    I3C base pointer.
```

```
uint8_t state
    Transfer state machine current state.
```

```
uint32_t transferCount
    Indicates progress of the transfer
```

```
uint8_t subaddressBuffer[4]
    Saving subaddress command.
```

```
uint8_t subaddressCount
    Saving command count.
```

```
i3c_master_transfer_t transfer
    Copy of the current transfer info.
```

```
i3c_master_edma_callback_t callback
    Callback function pointer.
```

`void *userData`  
Application data passed to callback.

`edma_handle_t *rxDmaHandle`  
Handle for receive DMA channel.

`edma_handle_t *txDmaHandle`  
Handle for transmit DMA channel.

`bool ibiFlag`  
IBIWON flag.

`uint8_t ibiAddress`  
Slave address which request IBI.

`uint8_t *ibiBuff`  
Pointer to IBI buffer to keep ibi bytes.

`size_t ibiPayloadSize`  
IBI payload size.

`i3c_ibi_type_t ibiType`  
IBI type.

`status_t result`  
Transfer result.

## 2.26 I3C Slave Driver

`void I3C_SlaveGetDefaultConfig(i3c_slave_config_t *slaveConfig)`

Provides a default configuration for the I3C slave peripheral.

This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableslave = true;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with `I3C_SlaveInit()`.

### Parameters

- `slaveConfig` – **[out]** User provided configuration structure for default values. Refer to `i3c_slave_config_t`.

`void I3C_SlaveInit(I3C_Type *base, const i3c_slave_config_t *slaveConfig, uint32_t slowClock_Hz)`

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

### Parameters

- `base` – The I3C peripheral base address.
- `slaveConfig` – User provided peripheral configuration. Use `I3C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- `slowClock_Hz` – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If `FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH` defines as 1, this parameter is useless.

`void I3C_SlaveDeinit(I3C_Type *base)`

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

**Parameters**

- `base` – The I3C peripheral base address.

`static inline void I3C_SlaveEnable(I3C_Type *base, bool isEnabled)`

Enable/Disable Slave.

**Parameters**

- `base` – The I3C peripheral base address.
- `isEnabled` – Enable or disable.

`static inline uint32_t I3C_SlaveGetStatusFlags(I3C_Type *base)`

Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

**See also:**

`_i3c_slave_flags`

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

`static inline void I3C_SlaveClearStatusFlags(I3C_Type *base, uint32_t statusMask)`

Clears the I3C slave status flag state.

The following status register flags can be cleared:

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`

Attempts to clear other flags has no effect.

**See also:**

`_i3c_slave_flags`.

**Parameters**

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetStatusFlags()`.

`static inline uint32_t I3C_SlaveGetErrorStatusFlags(I3C_Type *base)`

Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

**See also:**

`_i3c_slave_error_flags`

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

`static inline void I3C_SlaveClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)`

Clears the I3C slave error status flag state.

**See also:**

`_i3c_slave_error_flags`.

**Parameters**

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_slave_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetErrorStatusFlags()`.

`i3c_slave_activity_state_t I3C_SlaveGetActivityState(I3C_Type *base)`

Gets the I3C slave state.

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

I3C slave activity state, refer `i3c_slave_activity_state_t`.

`status_t I3C_SlaveCheckAndClearError(I3C_Type *base, uint32_t status)`

`static inline void I3C_SlaveEnableInterrupts(I3C_Type *base, uint32_t interruptMask)`

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`

- `kI3C_SlaveErrorFlag`
- `kI3C_SlaveHDRCommandMatchFlag`
- `kI3C_SlaveCCCHandledFlag`
- `kI3C_SlaveEventSentFlag`

**Parameters**

- `base` – The I3C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline void I3C_SlaveDisableInterrupts(I3C_Type *base, uint32_t interruptMask)`

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`
- `kI3C_SlaveErrorFlag`
- `kI3C_SlaveHDRCommandMatchFlag`
- `kI3C_SlaveCCCHandledFlag`
- `kI3C_SlaveEventSentFlag`

**Parameters**

- `base` – The I3C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline uint32_t I3C_SlaveGetEnabledInterrupts(I3C_Type *base)`

Returns the set of currently enabled I3C slave interrupt requests.

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

`static inline uint32_t I3C_SlaveGetPendingInterrupts(I3C_Type *base)`

Returns the set of pending I3C slave interrupt requests.

**Parameters**

- `base` – The I3C peripheral base address.

**Returns**

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of pending interrupts.

```
static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)
```

Enables or disables I3C slave DMA requests.

#### Parameters

- *base* – The I3C peripheral base address.
- *enableTx* – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- *enableRx* – Enable flag for receive DMA request. Pass true for enable, false for disable.
- *width* – DMA read/write unit in bytes.

```
static inline uint32_t I3C_SlaveGetTxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave transmit data register address for DMA transfer.

#### Parameters

- *base* – The I3C peripheral base address.
- *width* – DMA read/write unit in bytes.

#### Returns

The I3C Slave Transmit Data Register address.

```
static inline uint32_t I3C_SlaveGetRxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave receive data register address for DMA transfer.

#### Parameters

- *base* – The I3C peripheral base address.
- *width* – DMA read/write unit in bytes.

#### Returns

The I3C Slave Receive Data Register address.

```
static inline void I3C_SlaveSetWatermarks(I3C_Type *base, i3c_tx_trigger_level_t txLvl, i3c_rx_trigger_level_t rxLvl, bool flushTx, bool flushRx)
```

Sets the watermarks for I3C slave FIFOs.

#### Parameters

- *base* – The I3C peripheral base address.
- *txLvl* – Transmit FIFO watermark level. The `kI3C_SlaveTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
- *rxLvl* – Receive FIFO watermark level. The `kI3C_SlaveRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
- *flushTx* – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- *flushRx* – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

```
static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of bytes in the I3C slave FIFOs.

#### Parameters

- *base* – The I3C peripheral base address.
- *txCount* – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.



- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

`status_t I3C_SlaveSend(I3C_Type *base, const void *txBuff, size_t txSize)`

Performs a polling send transfer on the I3C bus.

#### Parameters

- `base` – The I3C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.

#### Returns

Error or success status returned by API.

`status_t I3C_SlaveReceive(I3C_Type *base, void *rxBuff, size_t rxSize)`

Performs a polling receive transfer on the I3C bus.

#### Parameters

- `base` – The I3C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.

#### Returns

Error or success status returned by API.

`void I3C_SlaveTransferCreateHandle(I3C_Type *base, i3c_slave_handle_t *handle, i3c_slave_transfer_callback_t callback, void *userData)`

Creates a new handle for the I3C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_SlaveTransferAbort()` API shall be called.

---

**Note:** The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

---

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – **[out]** Pointer to the I3C slave driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

`status_t I3C_SlaveTransferNonBlocking(I3C_Type *base, i3c_slave_handle_t *handle, uint32_t eventMask)`

Starts accepting slave transfers.

Call this API after calling `I2C_SlaveInit()` and `I3C_SlaveTransferCreateHandle()` to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to `I3C_SlaveTransferCreateHandle()`. The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the `eventMask` parameter to the OR'd combination of `i3c_slave_transfer_event_t` enumerators for the events you wish to receive. The `kI3C_SlaveTransmitEvent` and `kI3C_SlaveReceiveEvent` events are always enabled and do not need to be included in the mask. Alternatively, you can pass

0 to get a default set of only the transmit and receive events that are always enabled. In addition, the `kI3C_SlaveAllEvents` constant is provided as a convenient way to enable all events.

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to struct: `_i3c_slave_handle` structure which stores the transfer state.
- `eventMask` – Bit mask formed by OR'ing together `i3c_slave_transfer_event_t` enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and `kI3C_SlaveAllEvents` to enable all events.

#### Return values

- `kStatus__Success` – Slave transfers were successfully started.
- `kStatus__I3C__Busy` – Slave transfers have already been started on this handle.

`status_t I3C_SlaveTransferGetCount(I3C_Type *base, i3c_slave_handle_t *handle, size_t *count)`

Gets the slave transfer status during a non-blocking transfer.

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to `i2c_slave_handle_t` structure.
- `count` – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

#### Return values

- `kStatus__Success` –
- `kStatus__NoTransferInProgress` –

`void I3C_SlaveTransferAbort(I3C_Type *base, i3c_slave_handle_t *handle)`

Aborts the slave non-blocking transfers.

---

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

---

#### Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to struct: `_i3c_slave_handle` structure which stores the transfer state.

`void I3C_SlaveTransferHandleIRQ(I3C_Type *base, void *intHandle)`

Reusable routine to handle slave interrupts.

---

**Note:** This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

---

#### Parameters

- `base` – The I3C peripheral base address.
- `intHandle` – Pointer to struct: `_i3c_slave_handle` structure which stores the transfer state.

enum \_i3c\_slave\_flags

I3C slave peripheral flags.

The following status register flags can be cleared:

- kI3C\_SlaveBusStartFlag
- kI3C\_SlaveMatchedFlag
- kI3C\_SlaveBusStopFlag

Only below flags can be enabled as interrupts.

- kI3C\_SlaveBusStartFlag
- kI3C\_SlaveMatchedFlag
- kI3C\_SlaveBusStopFlag
- kI3C\_SlaveRxReadyFlag
- kI3C\_SlaveTxReadyFlag
- kI3C\_SlaveDynamicAddrChangedFlag
- kI3C\_SlaveReceivedCCCFlag
- kI3C\_SlaveErrorFlag
- kI3C\_SlaveHDRCommandMatchFlag
- kI3C\_SlaveCCCHandledFlag
- kI3C\_SlaveEventSentFlag

---

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

---

*Values:*

enumerator kI3C\_SlaveNotStopFlag

Slave status not stop flag

enumerator kI3C\_SlaveMessageFlag

Slave status message, indicating slave is listening to the bus traffic or responding

enumerator kI3C\_SlaveRequiredReadFlag

Slave status required, either is master doing SDR read from slave, or is IBI pushing out.

enumerator kI3C\_SlaveRequiredWriteFlag

Slave status request write, master is doing SDR write to slave, except slave in ENTDAAMode

enumerator kI3C\_SlaveBusDAAModeFlag

I3C bus is in ENTDAAMode

enumerator kI3C\_SlaveBusHDRModeFlag

I3C bus is in HDR mode

enumerator kI3C\_SlaveBusStartFlag

Start/Re-start event is seen since the bus was last cleared

enumerator kI3C\_SlaveMatchedFlag

Slave address(dynamic/static) matched since last cleared

enumerator kI3C\_SlaveBusStopFlag

Stop event is seen since the bus was last cleared

enumerator kI3C\_SlaveRxReadyFlag  
Rx data ready in rx buffer flag

enumerator kI3C\_SlaveTxReadyFlag  
Tx buffer ready for Tx data flag

enumerator kI3C\_SlaveDynamicAddrChangedFlag  
Slave dynamic address has been assigned, re-assigned, or lost

enumerator kI3C\_SlaveReceivedCCCFlag  
Slave received Common command code

enumerator kI3C\_SlaveErrorFlag  
Error occurred flag

enumerator kI3C\_SlaveHDRCommandMatchFlag  
High data rate command match

enumerator kI3C\_SlaveCCCHandledFlag  
Slave received Common command code is handled by I3C module

enumerator kI3C\_SlaveEventSentFlag  
Slave IBI/P2P/MR/HJ event has been sent

enumerator kI3C\_SlaveIbiDisableFlag  
Slave in band interrupt is disabled.

enumerator kI3C\_SlaveMasterRequestDisabledFlag  
Slave master request is disabled.

enumerator kI3C\_SlaveHotJoinDisabledFlag  
Slave Hot-Join is disabled.

enumerator kI3C\_SlaveClearFlags  
All flags which are cleared by the driver upon starting a transfer.

enumerator kI3C\_SlaveAllIrqFlags

enum \_i3c\_slave\_error\_flags  
I3C slave error flags to indicate the causes.

---

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

---

*Values:*

enumerator kI3C\_SlaveErrorOverrunFlag  
Slave internal from-bus buffer/FIFO overrun.

enumerator kI3C\_SlaveErrorUnderrunFlag  
Slave internal to-bus buffer/FIFO underrun

enumerator kI3C\_SlaveErrorUnderrunNakFlag  
Slave internal from-bus buffer/FIFO underrun and NACK error

enumerator kI3C\_SlaveErrorTermFlag  
Terminate error from master

enumerator kI3C\_SlaveErrorInvalidStartFlag  
Slave invalid start flag

enumerator kI3C\_SlaveErrorSdrParityFlag  
SDR parity error

enumerator kI3C\_SlaveErrorHdrParityFlag

HDR parity error

enumerator kI3C\_SlaveErrorHdrCRCFlag

HDR-DDR CRC error

enumerator kI3C\_SlaveErrorS0S1Flag

S0 or S1 error

enumerator kI3C\_SlaveErrorOverreadFlag

Over-read error

enumerator kI3C\_SlaveErrorOverwriteFlag

Over-write error

enum \_i3c\_slave\_event

I3C slave.event.

*Values:*

enumerator kI3C\_SlaveEventNormal

Normal mode.

enumerator kI3C\_SlaveEventIBI

In band interrupt event.

enumerator kI3C\_SlaveEventMasterReq

Master request event.

enumerator kI3C\_SlaveEventHotJoinReq

Hot-join event.

enum \_i3c\_slave\_activity\_state

I3C slave.activity state.

*Values:*

enumerator kI3C\_SlaveNoLatency

Normal bus operation

enumerator kI3C\_SlaveLatency1Ms

1ms of latency.

enumerator kI3C\_SlaveLatency100Ms

100ms of latency.

enumerator kI3C\_SlaveLatency10S

10s latency.

enum \_i3c\_slave\_transfer\_event

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to `I3C_SlaveTransferNonBlocking()` in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

---

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

---

*Values:*

enumerator `kI3C_SlaveAddressMatchEvent`

Received the slave address after a start or repeated start.

enumerator `kI3C_SlaveTransmitEvent`

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator `kI3C_SlaveReceiveEvent`

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

enumerator `kI3C_SlaveRequiredTransmitEvent`

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

enumerator `kI3C_SlaveStartEvent`

A start/repeated start was detected.

enumerator `kI3C_SlaveHDRCommandMatchEvent`

Slave Match HDR Command.

enumerator `kI3C_SlaveCompletionEvent`

A stop was detected, completing the transfer.

enumerator `kI3C_SlaveRequestSentEvent`

Slave request event sent.

enumerator `kI3C_SlaveReceivedCCCEvent`

Slave received CCC event, need to handle by application.

enumerator `kI3C_SlaveAllEvents`

Bit mask of all available events.

typedef enum `_i3c_slave_event` `i3c_slave_event_t`

I3C slave.event.

typedef enum `_i3c_slave_activity_state` `i3c_slave_activity_state_t`

I3C slave.activity state.

typedef struct `_i3c_slave_config` `i3c_slave_config_t`

Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_SlaveGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum `_i3c_slave_transfer_event` `i3c_slave_transfer_event_t`

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to `I3C_SlaveTransferNonBlocking()` in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

---

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

---

typedef struct `_i3c_slave_transfer` `i3c_slave_transfer_t`

I3C slave transfer structure.

typedef struct `_i3c_slave_handle` `i3c_slave_handle_t`

```
typedef void (*i3c_slave_transfer_callback_t)(I3C_Type *base, i3c_slave_transfer_t *transfer, void *userData)
```

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the I3C\_SlaveSetCallback() function after you have created a handle.

**Param base**

Base address for the I3C instance on which the event occurred.

**Param transfer**

Pointer to transfer descriptor containing values passed to and/or from the callback.

**Param userData**

Arbitrary pointer-sized value passed from the application.

```
typedef void (*i3c_slave_isr_t)(I3C_Type *base, void *handle)
```

Typedef for slave interrupt handler.

```
struct _i3c_slave_config
```

*#include <fsl\_i3c.h>* Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C\_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

## Public Members

bool enableSlave

Whether to enable slave.

uint8\_t staticAddr

Static address.

uint16\_t vendorID

Device vendor ID(manufacture ID).

uint32\_t partNumber

Device part number info

uint8\_t dcr

Device characteristics register information.

uint8\_t bcr

Bus characteristics register information.

uint8\_t hdrMode

Support hdr mode, could be OR logic in enumeration:i3c\_hdr\_mode\_t.

bool nakAllRequest

Whether to reply NAK to all requests except broadcast CCC.

bool ignoreS0S1Error

Whether to ignore S0/S1 error in SDR mode.

bool offline

Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

bool matchSlaveStartStop

Whether to assert start/stop status only the time slave is addressed.

uint32\_t maxWriteLength

Maximum write length.

uint32\_t maxReadLength

Maximum read length.

struct \_\_i3c\_slave\_transfer

*#include <fsl\_i3c.h>* I3C slave transfer structure.

### Public Members

uint32\_t event

Reason the callback is being invoked.

uint8\_t \*txData

Transfer buffer

size\_t txDataSize

Transfer size

uint8\_t \*rxData

Transfer buffer

size\_t rxDataSize

Transfer size

status\_t completionStatus

Success or error code describing how the transfer completed. Only applies for kI3C\_SlaveCompletionEvent.

size\_t transferredCount

Number of bytes actually transferred since start or last repeated start.

struct \_\_i3c\_slave\_handle

*#include <fsl\_i3c.h>* I3C slave handle structure.

---

**Note:** The contents of this structure are private and subject to change.

---

### Public Members

i3c\_slave\_transfer\_t transfer

I3C slave transfer copy.

bool isBusy

Whether transfer is busy.

bool wasTransmit

Whether the last transfer was a transmit.

uint32\_t eventMask

Mask of enabled events.

uint32\_t transferredCount

Count of bytes transferred.



*i3c\_slave\_transfer\_callback\_t* callback

Callback function called at transfer event.

void \*userData

Callback parameter passed to callback.

size\_t txFifoSize

Tx Fifo size

## 2.27 I3C Slave DMA Driver

```
void I3C_SlaveTransferCreateHandleEDMA(I3C_Type *base, i3c_slave_edma_handle_t *handle,
                                       i3c_slave_edma_callback_t callback, void *userData,
                                       edma_handle_t *rxDmaHandle, edma_handle_t
                                       *txDmaHandle)
```

Create a new handle for the I3C slave DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_SlaveTransferAbortDMA()` API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

### Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C slave driver handle.
- *callback* – User provided pointer to the asynchronous callback function.
- *userData* – User provided pointer to the application callback data.
- *rxDmaHandle* – Handle for the DMA receive channel. Created by the user prior to calling this function.
- *txDmaHandle* – Handle for the DMA transmit channel. Created by the user prior to calling this function.

```
status_t I3C_SlaveTransferEDMA(I3C_Type *base, i3c_slave_edma_handle_t *handle,
                               i3c_slave_edma_transfer_t *transfer, uint32_t eventMask)
```

Prepares for a non-blocking DMA-based transaction on the I3C bus.

The API will do DMA configuration according to the input transfer descriptor, and the data will be transferred when there's bus master requesting transfer from/to this slave. So the timing of call to this API need be aligned with master application to ensure the transfer is executed as expected. Callback specified when the *handle* was created is invoked when the transaction has completed.

### Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C slave driver handle.
- *transfer* – The pointer to the transfer descriptor.
- *eventMask* – Bit mask formed by OR'ing together `i3c_slave_transfer_event_t` enumerators to specify which events to send to the callback. The transmit and receive events is not allowed to be enabled.

### Return values

- `kStatus_Success` – The transaction was started successfully.

- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.
- `kStatus_Fail` – The transaction can't be set.

`void I3C_SlaveTransferAbortEDMA(I3C_Type *base, i3c_slave_edma_handle_t *handle)`

Abort a slave edma non-blocking transfer in a early time.

#### Parameters

- `base` – I3C peripheral base address
- `handle` – pointer to `i3c_slave_edma_handle_t` structure

`void I3C_SlaveTransferEDMAHandleIRQ(I3C_Type *base, void *i3CHandle)`

Reusable routine to handle slave interrupts.

---

**Note:** This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

---

#### Parameters

- `base` – The I3C peripheral base address.
- `i3CHandle` – Pointer to the I3C slave DMA driver handle.

`typedef struct _i3c_slave_edma_handle i3c_slave_edma_handle_t`

`typedef struct _i3c_slave_edma_transfer i3c_slave_edma_transfer_t`

I3C slave transfer structure.

`typedef void (*i3c_slave_edma_callback_t)(I3C_Type *base, i3c_slave_edma_transfer_t *transfer, void *userData)`

Slave event callback function pointer type.

This callback is used only for the slave DMA transfer API.

#### Param base

Base address for the I3C instance on which the event occurred.

#### Param handle

Pointer to slave DMA transfer handle.

#### Param transfer

Pointer to transfer descriptor containing values passed to and/or from the callback.

#### Param userData

Arbitrary pointer-sized value passed from the application.

`struct _i3c_slave_edma_transfer`

`#include <fsl_i3c_edma.h>` I3C slave transfer structure.

#### Public Members

`uint32_t event`

Reason the callback is being invoked.

`uint8_t *txData`

Transfer buffer

`size_t txDataSize`

Transfer size

`uint8_t *rxData`  
Transfer buffer

`size_t rxDataSize`  
Transfer size

`status_t completionStatus`  
Success or error code describing how the transfer completed. Only applies for `kI3C_SlaveCompletionEvent`.

`struct _i3c_slave_edma_handle`  
`#include <fsl_i3c_edma.h>` I3C slave edma handle structure.

---

**Note:** The contents of this structure are private and subject to change.

---

### Public Members

`I3C_Type *base`  
I3C base pointer.

`i3c_slave_edma_transfer_t transfer`  
I3C slave transfer copy.

`bool isBusy`  
Whether transfer is busy.

`bool wasTransmit`  
Whether the last transfer was a transmit.

`bool isDdrMode`  
Whether this is HDR-DDR transfer.

`uint32_t eventMask`  
Mask of enabled events.

`i3c_slave_edma_callback_t callback`  
Callback function called at transfer event.

`edma_handle_t *rxDmaHandle`  
Handle for receive DMA channel.

`edma_handle_t *txDmaHandle`  
Handle for transmit DMA channel.

`void *userData`  
Callback parameter passed to callback.

## 2.28 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.

---

**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
- inputOn – The software input on

```
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.

---

**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 1.0.0.

IOMUXC\_PAD\_DAP\_TDI\_\_JTAG\_MUX\_TDI

IOMUXC\_PAD\_DAP\_TDI\_\_MQS2\_LEFT

IOMUXC\_PAD\_DAP\_TDI\_\_CAN2\_TX

IOMUXC\_PAD\_DAP\_TDI\_\_FLEXIO2\_FLEXIO30

IOMUXC\_PAD\_DAP\_TDI\_\_GPIO3\_IO28

IOMUXC\_PAD\_DAP\_TDI\_\_LPUART5\_RX

IOMUXC\_PAD\_DAP\_TMS\_SWDIO\_\_JTAG\_MUX\_TMS

IOMUXC\_PAD\_DAP\_TMS\_SWDIO\_\_FLEXIO2\_FLEXIO31

IOMUXC\_PAD\_DAP\_TMS\_SWDIO\_\_GPIO3\_IO29

IOMUXC\_PAD\_DAP\_TMS\_SWDIO\_\_LPUART5\_RTS\_B

IOMUXC\_PAD\_DAP\_TCLK\_SWCLK\_\_JTAG\_MUX\_TCK

IOMUXC\_PAD\_DAP\_TCLK\_SWCLK\_\_FLEXIO1\_FLEXIO30

IOMUXC\_PAD\_DAP\_TCLK\_SWCLK\_\_GPIO3\_IO30

IOMUXC\_PAD\_DAP\_TCLK\_SWCLK\_\_LPUART5\_CTS\_B

IOMUXC\_PAD\_DAP\_TDO\_TRACESWO\_\_JTAG\_MUX\_TDO  
IOMUXC\_PAD\_DAP\_TDO\_TRACESWO\_\_MQS2\_RIGHT  
IOMUXC\_PAD\_DAP\_TDO\_TRACESWO\_\_CAN2\_RX  
IOMUXC\_PAD\_DAP\_TDO\_TRACESWO\_\_FLEXIO1\_FLEXIO31  
IOMUXC\_PAD\_DAP\_TDO\_TRACESWO\_\_GPIO3\_IO31  
IOMUXC\_PAD\_DAP\_TDO\_TRACESWO\_\_LPUART5\_TX  
IOMUXC\_PAD\_GPIO\_IO00\_\_GPIO2\_IO00  
IOMUXC\_PAD\_GPIO\_IO00\_\_LPI2C3\_SDA  
IOMUXC\_PAD\_GPIO\_IO00\_\_MEDIAMIX\_CAM\_CLK  
IOMUXC\_PAD\_GPIO\_IO00\_\_MEDIAMIX\_DISP\_CLK  
IOMUXC\_PAD\_GPIO\_IO00\_\_LPSPI6\_PCS0  
IOMUXC\_PAD\_GPIO\_IO00\_\_LPUART5\_TX  
IOMUXC\_PAD\_GPIO\_IO00\_\_LPI2C5\_SDA  
IOMUXC\_PAD\_GPIO\_IO00\_\_FLEXIO1\_FLEXIO00  
IOMUXC\_PAD\_GPIO\_IO01\_\_GPIO2\_IO01  
IOMUXC\_PAD\_GPIO\_IO01\_\_LPI2C3\_SCL  
IOMUXC\_PAD\_GPIO\_IO01\_\_MEDIAMIX\_CAM\_DATA00  
IOMUXC\_PAD\_GPIO\_IO01\_\_MEDIAMIX\_DISP\_DE  
IOMUXC\_PAD\_GPIO\_IO01\_\_LPSPI6\_SIN  
IOMUXC\_PAD\_GPIO\_IO01\_\_LPUART5\_RX  
IOMUXC\_PAD\_GPIO\_IO01\_\_LPI2C5\_SCL  
IOMUXC\_PAD\_GPIO\_IO01\_\_FLEXIO1\_FLEXIO01  
IOMUXC\_PAD\_GPIO\_IO02\_\_GPIO2\_IO02  
IOMUXC\_PAD\_GPIO\_IO02\_\_LPI2C4\_SDA  
IOMUXC\_PAD\_GPIO\_IO02\_\_MEDIAMIX\_CAM\_VSYNC  
IOMUXC\_PAD\_GPIO\_IO02\_\_MEDIAMIX\_DISP\_VSYNC  
IOMUXC\_PAD\_GPIO\_IO02\_\_LPSPI6\_SOUT  
IOMUXC\_PAD\_GPIO\_IO02\_\_LPUART5\_CTS\_B  
IOMUXC\_PAD\_GPIO\_IO02\_\_LPI2C6\_SDA  
IOMUXC\_PAD\_GPIO\_IO02\_\_FLEXIO1\_FLEXIO02  
IOMUXC\_PAD\_GPIO\_IO03\_\_GPIO2\_IO03  
IOMUXC\_PAD\_GPIO\_IO03\_\_LPI2C4\_SCL  
IOMUXC\_PAD\_GPIO\_IO03\_\_MEDIAMIX\_CAM\_HSYNC

IOMUXC\_PAD\_GPIO\_IO03\_\_\_MEDIAMIX\_DISP\_HSYNC  
IOMUXC\_PAD\_GPIO\_IO03\_\_\_LPSPI6\_SCK  
IOMUXC\_PAD\_GPIO\_IO03\_\_\_LPUART5\_RTS\_B  
IOMUXC\_PAD\_GPIO\_IO03\_\_\_LPI2C6\_SCL  
IOMUXC\_PAD\_GPIO\_IO03\_\_\_FLEXIO1\_FLEXIO03  
IOMUXC\_PAD\_GPIO\_IO04\_\_\_GPIO2\_IO04  
IOMUXC\_PAD\_GPIO\_IO04\_\_\_TPM3\_CH0  
IOMUXC\_PAD\_GPIO\_IO04\_\_\_PDM\_CLK  
IOMUXC\_PAD\_GPIO\_IO04\_\_\_MEDIAMIX\_DISP\_DATA00  
IOMUXC\_PAD\_GPIO\_IO04\_\_\_LPSPI7\_PCS0  
IOMUXC\_PAD\_GPIO\_IO04\_\_\_LPUART6\_TX  
IOMUXC\_PAD\_GPIO\_IO04\_\_\_LPI2C6\_SDA  
IOMUXC\_PAD\_GPIO\_IO04\_\_\_FLEXIO1\_FLEXIO04  
IOMUXC\_PAD\_GPIO\_IO05\_\_\_GPIO2\_IO05  
IOMUXC\_PAD\_GPIO\_IO05\_\_\_TPM4\_CH0  
IOMUXC\_PAD\_GPIO\_IO05\_\_\_PDM\_BIT\_STREAM00  
IOMUXC\_PAD\_GPIO\_IO05\_\_\_MEDIAMIX\_DISP\_DATA01  
IOMUXC\_PAD\_GPIO\_IO05\_\_\_LPSPI7\_SIN  
IOMUXC\_PAD\_GPIO\_IO05\_\_\_LPUART6\_RX  
IOMUXC\_PAD\_GPIO\_IO05\_\_\_LPI2C6\_SCL  
IOMUXC\_PAD\_GPIO\_IO05\_\_\_FLEXIO1\_FLEXIO05  
IOMUXC\_PAD\_GPIO\_IO06\_\_\_GPIO2\_IO06  
IOMUXC\_PAD\_GPIO\_IO06\_\_\_TPM5\_CH0  
IOMUXC\_PAD\_GPIO\_IO06\_\_\_PDM\_BIT\_STREAM01  
IOMUXC\_PAD\_GPIO\_IO06\_\_\_MEDIAMIX\_DISP\_DATA02  
IOMUXC\_PAD\_GPIO\_IO06\_\_\_LPSPI7\_SOUT  
IOMUXC\_PAD\_GPIO\_IO06\_\_\_LPUART6\_CTS\_B  
IOMUXC\_PAD\_GPIO\_IO06\_\_\_LPI2C7\_SDA  
IOMUXC\_PAD\_GPIO\_IO06\_\_\_FLEXIO1\_FLEXIO06  
IOMUXC\_PAD\_GPIO\_IO07\_\_\_GPIO2\_IO07  
IOMUXC\_PAD\_GPIO\_IO07\_\_\_LPSPI3\_PCS1  
IOMUXC\_PAD\_GPIO\_IO07\_\_\_MEDIAMIX\_CAM\_DATA01  
IOMUXC\_PAD\_GPIO\_IO07\_\_\_MEDIAMIX\_DISP\_DATA03

IOMUXC\_PAD\_GPIO\_IO07\_\_LPSPI7\_SCK  
IOMUXC\_PAD\_GPIO\_IO07\_\_LPUART6\_RTS\_B  
IOMUXC\_PAD\_GPIO\_IO07\_\_LPI2C7\_SCL  
IOMUXC\_PAD\_GPIO\_IO07\_\_FLEXIO1\_FLEXIO07  
IOMUXC\_PAD\_GPIO\_IO08\_\_GPIO2\_IO08  
IOMUXC\_PAD\_GPIO\_IO08\_\_LPSPI3\_PCS0  
IOMUXC\_PAD\_GPIO\_IO08\_\_MEDIAMIX\_CAM\_DATA02  
IOMUXC\_PAD\_GPIO\_IO08\_\_MEDIAMIX\_DISP\_DATA04  
IOMUXC\_PAD\_GPIO\_IO08\_\_TPM6\_CH0  
IOMUXC\_PAD\_GPIO\_IO08\_\_LPUART7\_TX  
IOMUXC\_PAD\_GPIO\_IO08\_\_LPI2C7\_SDA  
IOMUXC\_PAD\_GPIO\_IO08\_\_FLEXIO1\_FLEXIO08  
IOMUXC\_PAD\_GPIO\_IO09\_\_GPIO2\_IO09  
IOMUXC\_PAD\_GPIO\_IO09\_\_LPSPI3\_SIN  
IOMUXC\_PAD\_GPIO\_IO09\_\_MEDIAMIX\_CAM\_DATA03  
IOMUXC\_PAD\_GPIO\_IO09\_\_MEDIAMIX\_DISP\_DATA05  
IOMUXC\_PAD\_GPIO\_IO09\_\_TPM3\_EXTCLK  
IOMUXC\_PAD\_GPIO\_IO09\_\_LPUART7\_RX  
IOMUXC\_PAD\_GPIO\_IO09\_\_LPI2C7\_SCL  
IOMUXC\_PAD\_GPIO\_IO09\_\_FLEXIO1\_FLEXIO09  
IOMUXC\_PAD\_GPIO\_IO10\_\_GPIO2\_IO10  
IOMUXC\_PAD\_GPIO\_IO10\_\_LPSPI3\_SOUT  
IOMUXC\_PAD\_GPIO\_IO10\_\_MEDIAMIX\_CAM\_DATA04  
IOMUXC\_PAD\_GPIO\_IO10\_\_MEDIAMIX\_DISP\_DATA06  
IOMUXC\_PAD\_GPIO\_IO10\_\_TPM4\_EXTCLK  
IOMUXC\_PAD\_GPIO\_IO10\_\_LPUART7\_CTS\_B  
IOMUXC\_PAD\_GPIO\_IO10\_\_LPI2C8\_SDA  
IOMUXC\_PAD\_GPIO\_IO10\_\_FLEXIO1\_FLEXIO10  
IOMUXC\_PAD\_GPIO\_IO11\_\_GPIO2\_IO11  
IOMUXC\_PAD\_GPIO\_IO11\_\_LPSPI3\_SCK  
IOMUXC\_PAD\_GPIO\_IO11\_\_MEDIAMIX\_CAM\_DATA05  
IOMUXC\_PAD\_GPIO\_IO11\_\_MEDIAMIX\_DISP\_DATA07  
IOMUXC\_PAD\_GPIO\_IO11\_\_TPM5\_EXTCLK

IOMUXC\_PAD\_GPIO\_IO11\_\_LPUART7\_RTS\_B  
IOMUXC\_PAD\_GPIO\_IO11\_\_LPI2C8\_SCL  
IOMUXC\_PAD\_GPIO\_IO11\_\_FLEXIO1\_FLEXIO11  
IOMUXC\_PAD\_GPIO\_IO12\_\_GPIO2\_IO12  
IOMUXC\_PAD\_GPIO\_IO12\_\_TPM3\_CH2  
IOMUXC\_PAD\_GPIO\_IO12\_\_PDM\_BIT\_STREAM02  
IOMUXC\_PAD\_GPIO\_IO12\_\_MEDIAMIX\_DISP\_DATA08  
IOMUXC\_PAD\_GPIO\_IO12\_\_LPSPI8\_PCS0  
IOMUXC\_PAD\_GPIO\_IO12\_\_LPUART8\_TX  
IOMUXC\_PAD\_GPIO\_IO12\_\_LPI2C8\_SDA  
IOMUXC\_PAD\_GPIO\_IO12\_\_SAI3\_RX\_SYNC  
IOMUXC\_PAD\_GPIO\_IO13\_\_GPIO2\_IO13  
IOMUXC\_PAD\_GPIO\_IO13\_\_TPM4\_CH2  
IOMUXC\_PAD\_GPIO\_IO13\_\_PDM\_BIT\_STREAM03  
IOMUXC\_PAD\_GPIO\_IO13\_\_MEDIAMIX\_DISP\_DATA09  
IOMUXC\_PAD\_GPIO\_IO13\_\_LPSPI8\_SIN  
IOMUXC\_PAD\_GPIO\_IO13\_\_LPUART8\_RX  
IOMUXC\_PAD\_GPIO\_IO13\_\_LPI2C8\_SCL  
IOMUXC\_PAD\_GPIO\_IO13\_\_FLEXIO1\_FLEXIO13  
IOMUXC\_PAD\_GPIO\_IO14\_\_GPIO2\_IO14  
IOMUXC\_PAD\_GPIO\_IO14\_\_LPUART3\_TX  
IOMUXC\_PAD\_GPIO\_IO14\_\_MEDIAMIX\_CAM\_DATA06  
IOMUXC\_PAD\_GPIO\_IO14\_\_MEDIAMIX\_DISP\_DATA10  
IOMUXC\_PAD\_GPIO\_IO14\_\_LPSPI8\_SOUT  
IOMUXC\_PAD\_GPIO\_IO14\_\_LPUART8\_CTS\_B  
IOMUXC\_PAD\_GPIO\_IO14\_\_LPUART4\_TX  
IOMUXC\_PAD\_GPIO\_IO14\_\_FLEXIO1\_FLEXIO14  
IOMUXC\_PAD\_GPIO\_IO15\_\_GPIO2\_IO15  
IOMUXC\_PAD\_GPIO\_IO15\_\_LPUART3\_RX  
IOMUXC\_PAD\_GPIO\_IO15\_\_MEDIAMIX\_CAM\_DATA07  
IOMUXC\_PAD\_GPIO\_IO15\_\_MEDIAMIX\_DISP\_DATA11  
IOMUXC\_PAD\_GPIO\_IO15\_\_LPSPI8\_SCK  
IOMUXC\_PAD\_GPIO\_IO15\_\_LPUART8\_RTS\_B



IOMUXC\_PAD\_GPIO\_IO15\_\_LPUART4\_RX  
IOMUXC\_PAD\_GPIO\_IO15\_\_FLEXIO1\_FLEXIO15  
IOMUXC\_PAD\_GPIO\_IO16\_\_GPIO2\_IO16  
IOMUXC\_PAD\_GPIO\_IO16\_\_SAI3\_TX\_BCLK  
IOMUXC\_PAD\_GPIO\_IO16\_\_PDM\_BIT\_STREAM02  
IOMUXC\_PAD\_GPIO\_IO16\_\_MEDIAMIX\_DISP\_DATA12  
IOMUXC\_PAD\_GPIO\_IO16\_\_LPUART3\_CTS\_B  
IOMUXC\_PAD\_GPIO\_IO16\_\_LPSPI4\_PCS2  
IOMUXC\_PAD\_GPIO\_IO16\_\_LPUART4\_CTS\_B  
IOMUXC\_PAD\_GPIO\_IO16\_\_FLEXIO1\_FLEXIO16  
IOMUXC\_PAD\_GPIO\_IO17\_\_GPIO2\_IO17  
IOMUXC\_PAD\_GPIO\_IO17\_\_SAI3\_MCLK  
IOMUXC\_PAD\_GPIO\_IO17\_\_MEDIAMIX\_CAM\_DATA08  
IOMUXC\_PAD\_GPIO\_IO17\_\_MEDIAMIX\_DISP\_DATA13  
IOMUXC\_PAD\_GPIO\_IO17\_\_LPUART3\_RTS\_B  
IOMUXC\_PAD\_GPIO\_IO17\_\_LPSPI4\_PCS1  
IOMUXC\_PAD\_GPIO\_IO17\_\_LPUART4\_RTS\_B  
IOMUXC\_PAD\_GPIO\_IO17\_\_FLEXIO1\_FLEXIO17  
IOMUXC\_PAD\_GPIO\_IO18\_\_GPIO2\_IO18  
IOMUXC\_PAD\_GPIO\_IO18\_\_SAI3\_RX\_BCLK  
IOMUXC\_PAD\_GPIO\_IO18\_\_MEDIAMIX\_CAM\_DATA09  
IOMUXC\_PAD\_GPIO\_IO18\_\_MEDIAMIX\_DISP\_DATA14  
IOMUXC\_PAD\_GPIO\_IO18\_\_LPSPI5\_PCS0  
IOMUXC\_PAD\_GPIO\_IO18\_\_LPSPI4\_PCS0  
IOMUXC\_PAD\_GPIO\_IO18\_\_TPM5\_CH2  
IOMUXC\_PAD\_GPIO\_IO18\_\_FLEXIO1\_FLEXIO18  
IOMUXC\_PAD\_GPIO\_IO19\_\_GPIO2\_IO19  
IOMUXC\_PAD\_GPIO\_IO19\_\_SAI3\_RX\_SYNC  
IOMUXC\_PAD\_GPIO\_IO19\_\_PDM\_BIT\_STREAM03  
IOMUXC\_PAD\_GPIO\_IO19\_\_MEDIAMIX\_DISP\_DATA15  
IOMUXC\_PAD\_GPIO\_IO19\_\_LPSPI5\_SIN  
IOMUXC\_PAD\_GPIO\_IO19\_\_LPSPI4\_SIN  
IOMUXC\_PAD\_GPIO\_IO19\_\_TPM6\_CH2

IOMUXC\_PAD\_GPIO\_IO19\_\_SAI3\_TX\_DATA00  
IOMUXC\_PAD\_GPIO\_IO20\_\_GPIO2\_IO20  
IOMUXC\_PAD\_GPIO\_IO20\_\_SAI3\_RX\_DATA00  
IOMUXC\_PAD\_GPIO\_IO20\_\_PDM\_BIT\_STREAM00  
IOMUXC\_PAD\_GPIO\_IO20\_\_MEDIAMIX\_DISP\_DATA16  
IOMUXC\_PAD\_GPIO\_IO20\_\_LPSPI5\_SOUT  
IOMUXC\_PAD\_GPIO\_IO20\_\_LPSPI4\_SOUT  
IOMUXC\_PAD\_GPIO\_IO20\_\_TPM3\_CH1  
IOMUXC\_PAD\_GPIO\_IO20\_\_FLEXIO1\_FLEXIO20  
IOMUXC\_PAD\_GPIO\_IO21\_\_GPIO2\_IO21  
IOMUXC\_PAD\_GPIO\_IO21\_\_SAI3\_TX\_DATA00  
IOMUXC\_PAD\_GPIO\_IO21\_\_PDM\_CLK  
IOMUXC\_PAD\_GPIO\_IO21\_\_MEDIAMIX\_DISP\_DATA17  
IOMUXC\_PAD\_GPIO\_IO21\_\_LPSPI5\_SCK  
IOMUXC\_PAD\_GPIO\_IO21\_\_LPSPI4\_SCK  
IOMUXC\_PAD\_GPIO\_IO21\_\_TPM4\_CH1  
IOMUXC\_PAD\_GPIO\_IO21\_\_SAI3\_RX\_BCLK  
IOMUXC\_PAD\_GPIO\_IO22\_\_GPIO2\_IO22  
IOMUXC\_PAD\_GPIO\_IO22\_\_USDHC3\_CLK  
IOMUXC\_PAD\_GPIO\_IO22\_\_SPDIF\_IN  
IOMUXC\_PAD\_GPIO\_IO22\_\_MEDIAMIX\_DISP\_DATA18  
IOMUXC\_PAD\_GPIO\_IO22\_\_TPM5\_CH1  
IOMUXC\_PAD\_GPIO\_IO22\_\_TPM6\_EXTCLK  
IOMUXC\_PAD\_GPIO\_IO22\_\_LPI2C5\_SDA  
IOMUXC\_PAD\_GPIO\_IO22\_\_FLEXIO1\_FLEXIO22  
IOMUXC\_PAD\_GPIO\_IO23\_\_GPIO2\_IO23  
IOMUXC\_PAD\_GPIO\_IO23\_\_USDHC3\_CMD  
IOMUXC\_PAD\_GPIO\_IO23\_\_SPDIF\_OUT  
IOMUXC\_PAD\_GPIO\_IO23\_\_MEDIAMIX\_DISP\_DATA19  
IOMUXC\_PAD\_GPIO\_IO23\_\_TPM6\_CH1  
IOMUXC\_PAD\_GPIO\_IO23\_\_LPI2C5\_SCL  
IOMUXC\_PAD\_GPIO\_IO23\_\_FLEXIO1\_FLEXIO23  
IOMUXC\_PAD\_GPIO\_IO24\_\_GPIO2\_IO24

IOMUXC\_PAD\_GPIO\_IO24\_\_USDHC3\_DATA0  
IOMUXC\_PAD\_GPIO\_IO24\_\_MEDIAMIX\_DISP\_DATA20  
IOMUXC\_PAD\_GPIO\_IO24\_\_TPM3\_CH3  
IOMUXC\_PAD\_GPIO\_IO24\_\_JTAG\_MUX\_TDO  
IOMUXC\_PAD\_GPIO\_IO24\_\_LPSPI6\_PCS1  
IOMUXC\_PAD\_GPIO\_IO24\_\_FLEXIO1\_FLEXIO24  
IOMUXC\_PAD\_GPIO\_IO25\_\_GPIO2\_IO25  
IOMUXC\_PAD\_GPIO\_IO25\_\_USDHC3\_DATA1  
IOMUXC\_PAD\_GPIO\_IO25\_\_CAN2\_TX  
IOMUXC\_PAD\_GPIO\_IO25\_\_MEDIAMIX\_DISP\_DATA21  
IOMUXC\_PAD\_GPIO\_IO25\_\_TPM4\_CH3  
IOMUXC\_PAD\_GPIO\_IO25\_\_JTAG\_MUX\_TCK  
IOMUXC\_PAD\_GPIO\_IO25\_\_LPSPI7\_PCS1  
IOMUXC\_PAD\_GPIO\_IO25\_\_FLEXIO1\_FLEXIO25  
IOMUXC\_PAD\_GPIO\_IO26\_\_GPIO2\_IO26  
IOMUXC\_PAD\_GPIO\_IO26\_\_USDHC3\_DATA2  
IOMUXC\_PAD\_GPIO\_IO26\_\_PDM\_BIT\_STREAM01  
IOMUXC\_PAD\_GPIO\_IO26\_\_MEDIAMIX\_DISP\_DATA22  
IOMUXC\_PAD\_GPIO\_IO26\_\_TPM5\_CH3  
IOMUXC\_PAD\_GPIO\_IO26\_\_JTAG\_MUX\_TDI  
IOMUXC\_PAD\_GPIO\_IO26\_\_LPSPI8\_PCS1  
IOMUXC\_PAD\_GPIO\_IO26\_\_SAI3\_TX\_SYNC  
IOMUXC\_PAD\_GPIO\_IO27\_\_GPIO2\_IO27  
IOMUXC\_PAD\_GPIO\_IO27\_\_USDHC3\_DATA3  
IOMUXC\_PAD\_GPIO\_IO27\_\_CAN2\_RX  
IOMUXC\_PAD\_GPIO\_IO27\_\_MEDIAMIX\_DISP\_DATA23  
IOMUXC\_PAD\_GPIO\_IO27\_\_TPM6\_CH3  
IOMUXC\_PAD\_GPIO\_IO27\_\_JTAG\_MUX\_TMS  
IOMUXC\_PAD\_GPIO\_IO27\_\_LPSPI5\_PCS1  
IOMUXC\_PAD\_GPIO\_IO27\_\_FLEXIO1\_FLEXIO27  
IOMUXC\_PAD\_GPIO\_IO28\_\_GPIO2\_IO28  
IOMUXC\_PAD\_GPIO\_IO28\_\_LPI2C3\_SDA  
IOMUXC\_PAD\_GPIO\_IO28\_\_FLEXIO1\_FLEXIO28

IOMUXC\_PAD\_GPIO\_IO29\_\_GPIO2\_IO29  
IOMUXC\_PAD\_GPIO\_IO29\_\_LPI2C3\_SCL  
IOMUXC\_PAD\_GPIO\_IO29\_\_FLEXIO1\_FLEXIO29  
IOMUXC\_PAD\_CCM\_CLKO1\_\_CCMSRCGPCMIX\_CLKO1  
IOMUXC\_PAD\_CCM\_CLKO1\_\_FLEXIO1\_FLEXIO26  
IOMUXC\_PAD\_CCM\_CLKO1\_\_GPIO3\_IO26  
IOMUXC\_PAD\_CCM\_CLKO2\_\_GPIO3\_IO27  
IOMUXC\_PAD\_CCM\_CLKO2\_\_CCMSRCGPCMIX\_CLKO2  
IOMUXC\_PAD\_CCM\_CLKO2\_\_FLEXIO1\_FLEXIO27  
IOMUXC\_PAD\_CCM\_CLKO3\_\_CCMSRCGPCMIX\_CLKO3  
IOMUXC\_PAD\_CCM\_CLKO3\_\_FLEXIO2\_FLEXIO28  
IOMUXC\_PAD\_CCM\_CLKO3\_\_GPIO4\_IO28  
IOMUXC\_PAD\_CCM\_CLKO4\_\_CCMSRCGPCMIX\_CLKO4  
IOMUXC\_PAD\_CCM\_CLKO4\_\_FLEXIO2\_FLEXIO29  
IOMUXC\_PAD\_CCM\_CLKO4\_\_GPIO4\_IO29  
IOMUXC\_PAD\_ENET1\_MDC\_\_ENET\_QOS\_MDC  
IOMUXC\_PAD\_ENET1\_MDC\_\_LPUART3\_DCB\_B  
IOMUXC\_PAD\_ENET1\_MDC\_\_I3C2\_SCL  
IOMUXC\_PAD\_ENET1\_MDC\_\_HSIOMIX\_OTG\_ID1  
IOMUXC\_PAD\_ENET1\_MDC\_\_FLEXIO2\_FLEXIO00  
IOMUXC\_PAD\_ENET1\_MDC\_\_GPIO4\_IO00  
IOMUXC\_PAD\_ENET1\_MDIO\_\_ENET\_QOS\_MDIO  
IOMUXC\_PAD\_ENET1\_MDIO\_\_LPUART3\_RIN\_B  
IOMUXC\_PAD\_ENET1\_MDIO\_\_I3C2\_SDA  
IOMUXC\_PAD\_ENET1\_MDIO\_\_HSIOMIX\_OTG\_PWR1  
IOMUXC\_PAD\_ENET1\_MDIO\_\_FLEXIO2\_FLEXIO01  
IOMUXC\_PAD\_ENET1\_MDIO\_\_GPIO4\_IO01  
IOMUXC\_PAD\_ENET1\_TD3\_\_ENET\_QOS\_RGMII\_TD3  
IOMUXC\_PAD\_ENET1\_TD3\_\_CAN2\_TX  
IOMUXC\_PAD\_ENET1\_TD3\_\_HSIOMIX\_OTG\_ID2  
IOMUXC\_PAD\_ENET1\_TD3\_\_FLEXIO2\_FLEXIO02  
IOMUXC\_PAD\_ENET1\_TD3\_\_GPIO4\_IO02  
IOMUXC\_PAD\_ENET1\_TD2\_\_ENET\_QOS\_RGMII\_TD2

IOMUXC\_PAD\_ENET1\_TD2\_\_CCM\_ENET\_QOS\_CLOCK\_GENERATE\_REF\_CLK  
IOMUXC\_PAD\_ENET1\_TD2\_\_CAN2\_RX  
IOMUXC\_PAD\_ENET1\_TD2\_\_HSIOMUX\_OTG\_OC2  
IOMUXC\_PAD\_ENET1\_TD2\_\_FLEXIO2\_FLEXIO03  
IOMUXC\_PAD\_ENET1\_TD2\_\_GPIO4\_IO03  
IOMUXC\_PAD\_ENET1\_TD1\_\_ENET\_QOS\_RGMII\_TD1  
IOMUXC\_PAD\_ENET1\_TD1\_\_LPUART3\_RTS\_B  
IOMUXC\_PAD\_ENET1\_TD1\_\_I3C2\_PUR  
IOMUXC\_PAD\_ENET1\_TD1\_\_HSIOMUX\_OTG\_OC1  
IOMUXC\_PAD\_ENET1\_TD1\_\_FLEXIO2\_FLEXIO04  
IOMUXC\_PAD\_ENET1\_TD1\_\_GPIO4\_IO04  
IOMUXC\_PAD\_ENET1\_TD1\_\_I3C2\_PUR\_B  
IOMUXC\_PAD\_ENET1\_TD0\_\_ENET\_QOS\_RGMII\_TD0  
IOMUXC\_PAD\_ENET1\_TD0\_\_LPUART3\_TX  
IOMUXC\_PAD\_ENET1\_TD0\_\_FLEXIO2\_FLEXIO05  
IOMUXC\_PAD\_ENET1\_TD0\_\_GPIO4\_IO05  
IOMUXC\_PAD\_ENET1\_TX\_CTL\_\_ENET\_QOS\_RGMII\_TX\_CTL  
IOMUXC\_PAD\_ENET1\_TX\_CTL\_\_LPUART3\_DTR\_B  
IOMUXC\_PAD\_ENET1\_TX\_CTL\_\_FLEXIO2\_FLEXIO06  
IOMUXC\_PAD\_ENET1\_TX\_CTL\_\_GPIO4\_IO06  
IOMUXC\_PAD\_ENET1\_TXC\_\_CCM\_ENET\_QOS\_CLOCK\_GENERATE\_TX\_CLK  
IOMUXC\_PAD\_ENET1\_TXC\_\_ENET\_QOS\_TX\_ER  
IOMUXC\_PAD\_ENET1\_TXC\_\_FLEXIO2\_FLEXIO07  
IOMUXC\_PAD\_ENET1\_TXC\_\_GPIO4\_IO07  
IOMUXC\_PAD\_ENET1\_RX\_CTL\_\_ENET\_QOS\_RGMII\_RX\_CTL  
IOMUXC\_PAD\_ENET1\_RX\_CTL\_\_LPUART3\_DSR\_B  
IOMUXC\_PAD\_ENET1\_RX\_CTL\_\_HSIOMUX\_OTG\_PWR2  
IOMUXC\_PAD\_ENET1\_RX\_CTL\_\_FLEXIO2\_FLEXIO08  
IOMUXC\_PAD\_ENET1\_RX\_CTL\_\_GPIO4\_IO08  
IOMUXC\_PAD\_ENET1\_RXC\_\_CCM\_ENET\_QOS\_CLOCK\_GENERATE\_RX\_CLK  
IOMUXC\_PAD\_ENET1\_RXC\_\_ENET\_QOS\_RX\_ER  
IOMUXC\_PAD\_ENET1\_RXC\_\_FLEXIO2\_FLEXIO09  
IOMUXC\_PAD\_ENET1\_RXC\_\_GPIO4\_IO09

IOMUXC\_PAD\_ENET1\_RD0\_\_ENET\_QOS\_RGMII\_RD0  
IOMUXC\_PAD\_ENET1\_RD0\_\_LPUART3\_RX  
IOMUXC\_PAD\_ENET1\_RD0\_\_FLEXIO2\_FLEXIO10  
IOMUXC\_PAD\_ENET1\_RD0\_\_GPIO4\_IO10  
IOMUXC\_PAD\_ENET1\_RD1\_\_ENET\_QOS\_RGMII\_RD1  
IOMUXC\_PAD\_ENET1\_RD1\_\_LPUART3\_CTS\_B  
IOMUXC\_PAD\_ENET1\_RD1\_\_LPTMR2\_ALT1  
IOMUXC\_PAD\_ENET1\_RD1\_\_FLEXIO2\_FLEXIO11  
IOMUXC\_PAD\_ENET1\_RD1\_\_GPIO4\_IO11  
IOMUXC\_PAD\_ENET1\_RD2\_\_ENET\_QOS\_RGMII\_RD2  
IOMUXC\_PAD\_ENET1\_RD2\_\_LPTMR2\_ALT2  
IOMUXC\_PAD\_ENET1\_RD2\_\_FLEXIO2\_FLEXIO12  
IOMUXC\_PAD\_ENET1\_RD2\_\_GPIO4\_IO12  
IOMUXC\_PAD\_ENET1\_RD3\_\_ENET\_QOS\_RGMII\_RD3  
IOMUXC\_PAD\_ENET1\_RD3\_\_LPTMR2\_ALT3  
IOMUXC\_PAD\_ENET1\_RD3\_\_FLEXIO2\_FLEXIO13  
IOMUXC\_PAD\_ENET1\_RD3\_\_GPIO4\_IO13  
IOMUXC\_PAD\_ENET2\_MDC\_\_ENET1\_MDC  
IOMUXC\_PAD\_ENET2\_MDC\_\_LPUART4\_DCB\_B  
IOMUXC\_PAD\_ENET2\_MDC\_\_SAI2\_RX\_SYNC  
IOMUXC\_PAD\_ENET2\_MDC\_\_FLEXIO2\_FLEXIO14  
IOMUXC\_PAD\_ENET2\_MDC\_\_GPIO4\_IO14  
IOMUXC\_PAD\_ENET2\_MDIO\_\_ENET1\_MDIO  
IOMUXC\_PAD\_ENET2\_MDIO\_\_LPUART4\_RIN\_B  
IOMUXC\_PAD\_ENET2\_MDIO\_\_SAI2\_RX\_BCLK  
IOMUXC\_PAD\_ENET2\_MDIO\_\_FLEXIO2\_FLEXIO15  
IOMUXC\_PAD\_ENET2\_MDIO\_\_GPIO4\_IO15  
IOMUXC\_PAD\_ENET2\_TD3\_\_SAI2\_RX\_DATA00  
IOMUXC\_PAD\_ENET2\_TD3\_\_FLEXIO2\_FLEXIO16  
IOMUXC\_PAD\_ENET2\_TD3\_\_GPIO4\_IO16  
IOMUXC\_PAD\_ENET2\_TD3\_\_ENET1\_RGMII\_TD3  
IOMUXC\_PAD\_ENET2\_TD2\_\_ENET1\_RGMII\_TD2  
IOMUXC\_PAD\_ENET2\_TD2\_\_ENET1\_TX\_CLK

IOMUXC\_PAD\_ENET2\_TD2\_\_SAI2\_RX\_DATA01  
IOMUXC\_PAD\_ENET2\_TD2\_\_FLEXIO2\_FLEXIO17  
IOMUXC\_PAD\_ENET2\_TD2\_\_GPIO4\_IO17  
IOMUXC\_PAD\_ENET2\_TD1\_\_ENET1\_RGMII\_TD1  
IOMUXC\_PAD\_ENET2\_TD1\_\_LPUART4\_RTS\_B  
IOMUXC\_PAD\_ENET2\_TD1\_\_SAI2\_RX\_DATA02  
IOMUXC\_PAD\_ENET2\_TD1\_\_FLEXIO2\_FLEXIO18  
IOMUXC\_PAD\_ENET2\_TD1\_\_GPIO4\_IO18  
IOMUXC\_PAD\_ENET2\_TD0\_\_ENET1\_RGMII\_TD0  
IOMUXC\_PAD\_ENET2\_TD0\_\_LPUART4\_TX  
IOMUXC\_PAD\_ENET2\_TD0\_\_SAI2\_RX\_DATA03  
IOMUXC\_PAD\_ENET2\_TD0\_\_FLEXIO2\_FLEXIO19  
IOMUXC\_PAD\_ENET2\_TD0\_\_GPIO4\_IO19  
IOMUXC\_PAD\_ENET2\_TX\_CTL\_\_ENET1\_RGMII\_TX\_CTL  
IOMUXC\_PAD\_ENET2\_TX\_CTL\_\_LPUART4\_DTR\_B  
IOMUXC\_PAD\_ENET2\_TX\_CTL\_\_SAI2\_TX\_SYNC  
IOMUXC\_PAD\_ENET2\_TX\_CTL\_\_FLEXIO2\_FLEXIO20  
IOMUXC\_PAD\_ENET2\_TX\_CTL\_\_GPIO4\_IO20  
IOMUXC\_PAD\_ENET2\_TXC\_\_ENET1\_RGMII\_TXC  
IOMUXC\_PAD\_ENET2\_TXC\_\_ENET1\_TX\_ER  
IOMUXC\_PAD\_ENET2\_TXC\_\_SAI2\_TX\_BCLK  
IOMUXC\_PAD\_ENET2\_TXC\_\_FLEXIO2\_FLEXIO21  
IOMUXC\_PAD\_ENET2\_TXC\_\_GPIO4\_IO21  
IOMUXC\_PAD\_ENET2\_RX\_CTL\_\_ENET1\_RGMII\_RX\_CTL  
IOMUXC\_PAD\_ENET2\_RX\_CTL\_\_LPUART4\_DSR\_B  
IOMUXC\_PAD\_ENET2\_RX\_CTL\_\_SAI2\_TX\_DATA00  
IOMUXC\_PAD\_ENET2\_RX\_CTL\_\_FLEXIO2\_FLEXIO22  
IOMUXC\_PAD\_ENET2\_RX\_CTL\_\_GPIO4\_IO22  
IOMUXC\_PAD\_ENET2\_RXC\_\_ENET1\_RGMII\_RXC  
IOMUXC\_PAD\_ENET2\_RXC\_\_ENET1\_RX\_ER  
IOMUXC\_PAD\_ENET2\_RXC\_\_SAI2\_TX\_DATA01  
IOMUXC\_PAD\_ENET2\_RXC\_\_FLEXIO2\_FLEXIO23  
IOMUXC\_PAD\_ENET2\_RXC\_\_GPIO4\_IO23

IOMUXC\_PAD\_ENET2\_RD0\_\_ENET1\_RGMII\_RD0  
IOMUXC\_PAD\_ENET2\_RD0\_\_LPUART4\_RX  
IOMUXC\_PAD\_ENET2\_RD0\_\_SAI2\_TX\_DATA02  
IOMUXC\_PAD\_ENET2\_RD0\_\_FLEXIO2\_FLEXIO24  
IOMUXC\_PAD\_ENET2\_RD0\_\_GPIO4\_IO24  
IOMUXC\_PAD\_ENET2\_RD1\_\_ENET1\_RGMII\_RD1  
IOMUXC\_PAD\_ENET2\_RD1\_\_SPDIF\_IN  
IOMUXC\_PAD\_ENET2\_RD1\_\_SAI2\_TX\_DATA03  
IOMUXC\_PAD\_ENET2\_RD1\_\_FLEXIO2\_FLEXIO25  
IOMUXC\_PAD\_ENET2\_RD1\_\_GPIO4\_IO25  
IOMUXC\_PAD\_ENET2\_RD2\_\_ENET1\_RGMII\_RD2  
IOMUXC\_PAD\_ENET2\_RD2\_\_LPUART4\_CTS\_B  
IOMUXC\_PAD\_ENET2\_RD2\_\_SAI2\_MCLK  
IOMUXC\_PAD\_ENET2\_RD2\_\_MQS2\_RIGHT  
IOMUXC\_PAD\_ENET2\_RD2\_\_FLEXIO2\_FLEXIO26  
IOMUXC\_PAD\_ENET2\_RD2\_\_GPIO4\_IO26  
IOMUXC\_PAD\_ENET2\_RD3\_\_ENET1\_RGMII\_RD3  
IOMUXC\_PAD\_ENET2\_RD3\_\_SPDIF\_OUT  
IOMUXC\_PAD\_ENET2\_RD3\_\_SPDIF\_IN  
IOMUXC\_PAD\_ENET2\_RD3\_\_MQS2\_LEFT  
IOMUXC\_PAD\_ENET2\_RD3\_\_FLEXIO2\_FLEXIO27  
IOMUXC\_PAD\_ENET2\_RD3\_\_GPIO4\_IO27  
IOMUXC\_PAD\_SD1\_CLK\_\_FLEXIO1\_FLEXIO08  
IOMUXC\_PAD\_SD1\_CLK\_\_GPIO3\_IO08  
IOMUXC\_PAD\_SD1\_CLK\_\_USDHC1\_CLK  
IOMUXC\_PAD\_SD1\_CMD\_\_USDHC1\_CMD  
IOMUXC\_PAD\_SD1\_CMD\_\_FLEXIO1\_FLEXIO09  
IOMUXC\_PAD\_SD1\_CMD\_\_GPIO3\_IO09  
IOMUXC\_PAD\_SD1\_DATA0\_\_USDHC1\_DATA0  
IOMUXC\_PAD\_SD1\_DATA0\_\_FLEXIO1\_FLEXIO10  
IOMUXC\_PAD\_SD1\_DATA0\_\_GPIO3\_IO10  
IOMUXC\_PAD\_SD1\_DATA1\_\_USDHC1\_DATA1  
IOMUXC\_PAD\_SD1\_DATA1\_\_FLEXIO1\_FLEXIO11



IOMUXC\_PAD\_SD1\_DATA1\_\_\_GPIO3\_IO11  
IOMUXC\_PAD\_SD1\_DATA2\_\_\_USDHC1\_DATA2  
IOMUXC\_PAD\_SD1\_DATA2\_\_\_FLEXIO1\_FLEXIO12  
IOMUXC\_PAD\_SD1\_DATA2\_\_\_GPIO3\_IO12  
IOMUXC\_PAD\_SD1\_DATA3\_\_\_USDHC1\_DATA3  
IOMUXC\_PAD\_SD1\_DATA3\_\_\_FLEXSPI1\_A\_SS1\_B  
IOMUXC\_PAD\_SD1\_DATA3\_\_\_FLEXIO1\_FLEXIO13  
IOMUXC\_PAD\_SD1\_DATA3\_\_\_GPIO3\_IO13  
IOMUXC\_PAD\_SD1\_DATA4\_\_\_USDHC1\_DATA4  
IOMUXC\_PAD\_SD1\_DATA4\_\_\_FLEXSPI1\_A\_DATA04  
IOMUXC\_PAD\_SD1\_DATA4\_\_\_FLEXIO1\_FLEXIO14  
IOMUXC\_PAD\_SD1\_DATA4\_\_\_GPIO3\_IO14  
IOMUXC\_PAD\_SD1\_DATA5\_\_\_USDHC1\_DATA5  
IOMUXC\_PAD\_SD1\_DATA5\_\_\_FLEXSPI1\_A\_DATA05  
IOMUXC\_PAD\_SD1\_DATA5\_\_\_USDHC1\_RESET\_B  
IOMUXC\_PAD\_SD1\_DATA5\_\_\_FLEXIO1\_FLEXIO15  
IOMUXC\_PAD\_SD1\_DATA5\_\_\_GPIO3\_IO15  
IOMUXC\_PAD\_SD1\_DATA6\_\_\_USDHC1\_DATA6  
IOMUXC\_PAD\_SD1\_DATA6\_\_\_FLEXSPI1\_A\_DATA06  
IOMUXC\_PAD\_SD1\_DATA6\_\_\_USDHC1\_CD\_B  
IOMUXC\_PAD\_SD1\_DATA6\_\_\_FLEXIO1\_FLEXIO16  
IOMUXC\_PAD\_SD1\_DATA6\_\_\_GPIO3\_IO16  
IOMUXC\_PAD\_SD1\_DATA7\_\_\_USDHC1\_DATA7  
IOMUXC\_PAD\_SD1\_DATA7\_\_\_FLEXSPI1\_A\_DATA07  
IOMUXC\_PAD\_SD1\_DATA7\_\_\_USDHC1\_WP  
IOMUXC\_PAD\_SD1\_DATA7\_\_\_FLEXIO1\_FLEXIO17  
IOMUXC\_PAD\_SD1\_DATA7\_\_\_GPIO3\_IO17  
IOMUXC\_PAD\_SD1\_STROBE\_\_\_USDHC1\_STROBE  
IOMUXC\_PAD\_SD1\_STROBE\_\_\_FLEXSPI1\_A\_DQS  
IOMUXC\_PAD\_SD1\_STROBE\_\_\_FLEXIO1\_FLEXIO18  
IOMUXC\_PAD\_SD1\_STROBE\_\_\_GPIO3\_IO18  
IOMUXC\_PAD\_SD2\_VSELECT\_\_\_USDHC2\_VSELECT  
IOMUXC\_PAD\_SD2\_VSELECT\_\_\_USDHC2\_WP

IOMUXC\_PAD\_SD2\_VSELECT\_\_LPTMR2\_ALT3  
IOMUXC\_PAD\_SD2\_VSELECT\_\_FLEXIO1\_FLEXIO19  
IOMUXC\_PAD\_SD2\_VSELECT\_\_GPIO3\_IO19  
IOMUXC\_PAD\_SD2\_VSELECT\_\_CCMSRCGPCMIX\_EXT\_CLK1  
IOMUXC\_PAD\_SD3\_CLK\_\_USDHC3\_CLK  
IOMUXC\_PAD\_SD3\_CLK\_\_FLEXSPI1\_A\_SCLK  
IOMUXC\_PAD\_SD3\_CLK\_\_FLEXIO1\_FLEXIO20  
IOMUXC\_PAD\_SD3\_CLK\_\_GPIO3\_IO20  
IOMUXC\_PAD\_SD3\_CMD\_\_USDHC3\_CMD  
IOMUXC\_PAD\_SD3\_CMD\_\_FLEXSPI1\_A\_SS0\_B  
IOMUXC\_PAD\_SD3\_CMD\_\_FLEXIO1\_FLEXIO21  
IOMUXC\_PAD\_SD3\_CMD\_\_GPIO3\_IO21  
IOMUXC\_PAD\_SD3\_DATA0\_\_USDHC3\_DATA0  
IOMUXC\_PAD\_SD3\_DATA0\_\_FLEXSPI1\_A\_DATA00  
IOMUXC\_PAD\_SD3\_DATA0\_\_FLEXIO1\_FLEXIO22  
IOMUXC\_PAD\_SD3\_DATA0\_\_GPIO3\_IO22  
IOMUXC\_PAD\_SD3\_DATA1\_\_USDHC3\_DATA1  
IOMUXC\_PAD\_SD3\_DATA1\_\_FLEXSPI1\_A\_DATA01  
IOMUXC\_PAD\_SD3\_DATA1\_\_FLEXIO1\_FLEXIO23  
IOMUXC\_PAD\_SD3\_DATA1\_\_GPIO3\_IO23  
IOMUXC\_PAD\_SD3\_DATA2\_\_USDHC3\_DATA2  
IOMUXC\_PAD\_SD3\_DATA2\_\_FLEXSPI1\_A\_DATA02  
IOMUXC\_PAD\_SD3\_DATA2\_\_FLEXIO1\_FLEXIO24  
IOMUXC\_PAD\_SD3\_DATA2\_\_GPIO3\_IO24  
IOMUXC\_PAD\_SD3\_DATA3\_\_USDHC3\_DATA3  
IOMUXC\_PAD\_SD3\_DATA3\_\_FLEXSPI1\_A\_DATA03  
IOMUXC\_PAD\_SD3\_DATA3\_\_FLEXIO1\_FLEXIO25  
IOMUXC\_PAD\_SD3\_DATA3\_\_GPIO3\_IO25  
IOMUXC\_PAD\_SD2\_CD\_B\_\_USDHC2\_CD\_B  
IOMUXC\_PAD\_SD2\_CD\_B\_\_ENET\_QOS\_1588\_EVENT0\_IN  
IOMUXC\_PAD\_SD2\_CD\_B\_\_I3C2\_SCL  
IOMUXC\_PAD\_SD2\_CD\_B\_\_FLEXIO1\_FLEXIO00  
IOMUXC\_PAD\_SD2\_CD\_B\_\_GPIO3\_IO00

IOMUXC\_PAD\_SD2\_CLK\_\_USDHC2\_CLK  
IOMUXC\_PAD\_SD2\_CLK\_\_ENET\_QOS\_1588\_EVENT0\_OUT  
IOMUXC\_PAD\_SD2\_CLK\_\_I3C2\_SDA  
IOMUXC\_PAD\_SD2\_CLK\_\_FLEXIO1\_FLEXIO01  
IOMUXC\_PAD\_SD2\_CLK\_\_GPIO3\_IO01  
IOMUXC\_PAD\_SD2\_CLK\_\_CCMSRCGPCMIX\_OBSERVE0  
IOMUXC\_PAD\_SD2\_CMD\_\_USDHC2\_CMD  
IOMUXC\_PAD\_SD2\_CMD\_\_ENET1\_1588\_EVENT0\_IN  
IOMUXC\_PAD\_SD2\_CMD\_\_I3C2\_PUR  
IOMUXC\_PAD\_SD2\_CMD\_\_I3C2\_PUR\_B  
IOMUXC\_PAD\_SD2\_CMD\_\_FLEXIO1\_FLEXIO02  
IOMUXC\_PAD\_SD2\_CMD\_\_GPIO3\_IO02  
IOMUXC\_PAD\_SD2\_CMD\_\_CCMSRCGPCMIX\_OBSERVE1  
IOMUXC\_PAD\_SD2\_DATA0\_\_USDHC2\_DATA0  
IOMUXC\_PAD\_SD2\_DATA0\_\_ENET1\_1588\_EVENT0\_OUT  
IOMUXC\_PAD\_SD2\_DATA0\_\_CAN2\_TX  
IOMUXC\_PAD\_SD2\_DATA0\_\_FLEXIO1\_FLEXIO03  
IOMUXC\_PAD\_SD2\_DATA0\_\_GPIO3\_IO03  
IOMUXC\_PAD\_SD2\_DATA0\_\_CCMSRCGPCMIX\_OBSERVE2  
IOMUXC\_PAD\_SD2\_DATA1\_\_USDHC2\_DATA1  
IOMUXC\_PAD\_SD2\_DATA1\_\_ENET1\_1588\_EVENT1\_IN  
IOMUXC\_PAD\_SD2\_DATA1\_\_CAN2\_RX  
IOMUXC\_PAD\_SD2\_DATA1\_\_FLEXIO1\_FLEXIO04  
IOMUXC\_PAD\_SD2\_DATA1\_\_GPIO3\_IO04  
IOMUXC\_PAD\_SD2\_DATA2\_\_USDHC2\_DATA2  
IOMUXC\_PAD\_SD2\_DATA2\_\_ENET1\_1588\_EVENT1\_OUT  
IOMUXC\_PAD\_SD2\_DATA2\_\_MQS2\_RIGHT  
IOMUXC\_PAD\_SD2\_DATA2\_\_FLEXIO1\_FLEXIO05  
IOMUXC\_PAD\_SD2\_DATA2\_\_GPIO3\_IO05  
IOMUXC\_PAD\_SD2\_DATA3\_\_USDHC2\_DATA3  
IOMUXC\_PAD\_SD2\_DATA3\_\_LPTMR2\_ALT1  
IOMUXC\_PAD\_SD2\_DATA3\_\_MQS2\_LEFT  
IOMUXC\_PAD\_SD2\_DATA3\_\_FLEXIO1\_FLEXIO06

IOMUXC\_PAD\_SD2\_DATA3\_\_\_GPIO3\_IO06  
IOMUXC\_PAD\_SD2\_RESET\_B\_\_\_USDHC2\_RESET\_B  
IOMUXC\_PAD\_SD2\_RESET\_B\_\_\_LPTMR2\_ALT2  
IOMUXC\_PAD\_SD2\_RESET\_B\_\_\_FLEXIO1\_FLEXIO07  
IOMUXC\_PAD\_SD2\_RESET\_B\_\_\_GPIO3\_IO07  
IOMUXC\_PAD\_SD2\_RESET\_B\_\_\_CCMSRCGPCMIX\_SYSTEM\_RESET  
IOMUXC\_PAD\_I2C1\_SCL\_\_\_LPI2C1\_SCL  
IOMUXC\_PAD\_I2C1\_SCL\_\_\_I3C1\_SCL  
IOMUXC\_PAD\_I2C1\_SCL\_\_\_LPUART1\_DCB\_B  
IOMUXC\_PAD\_I2C1\_SCL\_\_\_TPM2\_CH0  
IOMUXC\_PAD\_I2C1\_SCL\_\_\_GPIO1\_IO00  
IOMUXC\_PAD\_I2C1\_SDA\_\_\_LPI2C1\_SDA  
IOMUXC\_PAD\_I2C1\_SDA\_\_\_I3C1\_SDA  
IOMUXC\_PAD\_I2C1\_SDA\_\_\_LPUART1\_RIN\_B  
IOMUXC\_PAD\_I2C1\_SDA\_\_\_TPM2\_CH1  
IOMUXC\_PAD\_I2C1\_SDA\_\_\_GPIO1\_IO01  
IOMUXC\_PAD\_I2C2\_SCL\_\_\_LPI2C2\_SCL  
IOMUXC\_PAD\_I2C2\_SCL\_\_\_I3C1\_PUR  
IOMUXC\_PAD\_I2C2\_SCL\_\_\_LPUART2\_DCB\_B  
IOMUXC\_PAD\_I2C2\_SCL\_\_\_TPM2\_CH2  
IOMUXC\_PAD\_I2C2\_SCL\_\_\_SAI1\_RX\_SYNC  
IOMUXC\_PAD\_I2C2\_SCL\_\_\_GPIO1\_IO02  
IOMUXC\_PAD\_I2C2\_SCL\_\_\_I3C1\_PUR\_B  
IOMUXC\_PAD\_I2C2\_SDA\_\_\_LPI2C2\_SDA  
IOMUXC\_PAD\_I2C2\_SDA\_\_\_LPUART2\_RIN\_B  
IOMUXC\_PAD\_I2C2\_SDA\_\_\_TPM2\_CH3  
IOMUXC\_PAD\_I2C2\_SDA\_\_\_SAI1\_RX\_BCLK  
IOMUXC\_PAD\_I2C2\_SDA\_\_\_GPIO1\_IO03  
IOMUXC\_PAD\_UART1\_RXD\_\_\_LPUART1\_RX  
IOMUXC\_PAD\_UART1\_RXD\_\_\_S400\_UART\_RX  
IOMUXC\_PAD\_UART1\_RXD\_\_\_LPSPI2\_SIN  
IOMUXC\_PAD\_UART1\_RXD\_\_\_TPM1\_CH0  
IOMUXC\_PAD\_UART1\_RXD\_\_\_GPIO1\_IO04

IOMUXC\_PAD\_UART1\_TXD\_\_LPUART1\_TX  
IOMUXC\_PAD\_UART1\_TXD\_\_S400\_UART\_TX  
IOMUXC\_PAD\_UART1\_TXD\_\_LPSPI2\_PCS0  
IOMUXC\_PAD\_UART1\_TXD\_\_TPM1\_CH1  
IOMUXC\_PAD\_UART1\_TXD\_\_GPIO1\_IO05  
IOMUXC\_PAD\_UART2\_RXD\_\_LPUART2\_RX  
IOMUXC\_PAD\_UART2\_RXD\_\_LPUART1\_CTS\_B  
IOMUXC\_PAD\_UART2\_RXD\_\_LPSPI2\_SOUT  
IOMUXC\_PAD\_UART2\_RXD\_\_TPM1\_CH2  
IOMUXC\_PAD\_UART2\_RXD\_\_SAI1\_MCLK  
IOMUXC\_PAD\_UART2\_RXD\_\_GPIO1\_IO06  
IOMUXC\_PAD\_UART2\_TXD\_\_LPUART2\_TX  
IOMUXC\_PAD\_UART2\_TXD\_\_LPUART1\_RTS\_B  
IOMUXC\_PAD\_UART2\_TXD\_\_LPSPI2\_SCK  
IOMUXC\_PAD\_UART2\_TXD\_\_TPM1\_CH3  
IOMUXC\_PAD\_UART2\_TXD\_\_GPIO1\_IO07  
IOMUXC\_PAD\_PDM\_CLK\_\_PDM\_CLK  
IOMUXC\_PAD\_PDM\_CLK\_\_MQS1\_LEFT  
IOMUXC\_PAD\_PDM\_CLK\_\_LPTMR1\_ALT1  
IOMUXC\_PAD\_PDM\_CLK\_\_GPIO1\_IO08  
IOMUXC\_PAD\_PDM\_CLK\_\_CAN1\_TX  
IOMUXC\_PAD\_PDM\_BIT\_STREAM0\_\_PDM\_BIT\_STREAM00  
IOMUXC\_PAD\_PDM\_BIT\_STREAM0\_\_MQS1\_RIGHT  
IOMUXC\_PAD\_PDM\_BIT\_STREAM0\_\_LPSPI1\_PCS1  
IOMUXC\_PAD\_PDM\_BIT\_STREAM0\_\_TPM1\_EXTCLK  
IOMUXC\_PAD\_PDM\_BIT\_STREAM0\_\_LPTMR1\_ALT2  
IOMUXC\_PAD\_PDM\_BIT\_STREAM0\_\_GPIO1\_IO09  
IOMUXC\_PAD\_PDM\_BIT\_STREAM0\_\_CAN1\_RX  
IOMUXC\_PAD\_PDM\_BIT\_STREAM1\_\_PDM\_BIT\_STREAM01  
IOMUXC\_PAD\_PDM\_BIT\_STREAM1\_\_NMI\_GLUE\_NMI  
IOMUXC\_PAD\_PDM\_BIT\_STREAM1\_\_LPSPI2\_PCS1  
IOMUXC\_PAD\_PDM\_BIT\_STREAM1\_\_TPM2\_EXTCLK  
IOMUXC\_PAD\_PDM\_BIT\_STREAM1\_\_LPTMR1\_ALT3

IOMUXC\_PAD\_PDM\_BIT\_STREAM1\_\_GPIO1\_IO10  
IOMUXC\_PAD\_PDM\_BIT\_STREAM1\_\_CCMSRCGPCMIX\_EXT\_CLK1  
IOMUXC\_PAD\_SAI1\_TXFS\_\_SAI1\_TX\_SYNC  
IOMUXC\_PAD\_SAI1\_TXFS\_\_SAI1\_TX\_DATA01  
IOMUXC\_PAD\_SAI1\_TXFS\_\_LPSPI1\_PCS0  
IOMUXC\_PAD\_SAI1\_TXFS\_\_LPUART2\_DTR\_B  
IOMUXC\_PAD\_SAI1\_TXFS\_\_MQS1\_LEFT  
IOMUXC\_PAD\_SAI1\_TXFS\_\_GPIO1\_IO11  
IOMUXC\_PAD\_SAI1\_TXC\_\_SAI1\_TX\_BCLK  
IOMUXC\_PAD\_SAI1\_TXC\_\_LPUART2\_CTS\_B  
IOMUXC\_PAD\_SAI1\_TXC\_\_LPSPI1\_SIN  
IOMUXC\_PAD\_SAI1\_TXC\_\_LPUART1\_DSR\_B  
IOMUXC\_PAD\_SAI1\_TXC\_\_CAN1\_RX  
IOMUXC\_PAD\_SAI1\_TXC\_\_GPIO1\_IO12  
IOMUXC\_PAD\_SAI1\_TXD0\_\_SAI1\_TX\_DATA00  
IOMUXC\_PAD\_SAI1\_TXD0\_\_LPUART2\_RTS\_B  
IOMUXC\_PAD\_SAI1\_TXD0\_\_LPSPI1\_SCK  
IOMUXC\_PAD\_SAI1\_TXD0\_\_LPUART1\_DTR\_B  
IOMUXC\_PAD\_SAI1\_TXD0\_\_CAN1\_TX  
IOMUXC\_PAD\_SAI1\_TXD0\_\_GPIO1\_IO13  
IOMUXC\_PAD\_SAI1\_RXD0\_\_SAI1\_RX\_DATA00  
IOMUXC\_PAD\_SAI1\_RXD0\_\_SAI1\_MCLK  
IOMUXC\_PAD\_SAI1\_RXD0\_\_LPSPI1\_SOUT  
IOMUXC\_PAD\_SAI1\_RXD0\_\_LPUART2\_DSR\_B  
IOMUXC\_PAD\_SAI1\_RXD0\_\_MQS1\_RIGHT  
IOMUXC\_PAD\_SAI1\_RXD0\_\_GPIO1\_IO14  
IOMUXC\_PAD\_WDOG\_ANY\_\_WDOG1\_WDOG\_ANY  
IOMUXC\_PAD\_WDOG\_ANY\_\_GPIO1\_IO15  
IOMUXC\_PAD\_MUX\_MODE\_MASK  
IOMUXC\_PAD\_MUX\_MODE\_SHIFT  
IOMUXC\_PAD\_MUX\_MODE(x)  
IOMUXC\_PAD\_SION\_MASK  
IOMUXC\_PAD\_SION\_SHIFT

IOMUXC\_PAD\_SION(x)  
IOMUXC\_PAD\_DSE\_MASK  
IOMUXC\_PAD\_DSE\_SHIFT  
IOMUXC\_PAD\_DSE(x)  
IOMUXC\_PAD\_FSEL1\_MASK  
IOMUXC\_PAD\_FSEL1\_SHIFT  
IOMUXC\_PAD\_FSEL1(x)  
IOMUXC\_PAD\_PU\_MASK  
IOMUXC\_PAD\_PU\_SHIFT  
IOMUXC\_PAD\_PU(x)  
IOMUXC\_PAD\_PD\_MASK  
IOMUXC\_PAD\_PD\_SHIFT  
IOMUXC\_PAD\_PD(x)  
IOMUXC\_PAD\_OD\_MASK  
IOMUXC\_PAD\_OD\_SHIFT  
IOMUXC\_PAD\_OD(x)  
IOMUXC\_PAD\_HYS\_MASK  
IOMUXC\_PAD\_HYS\_SHIFT  
IOMUXC\_PAD\_HYS(x)  
IOMUXC\_PAD\_APC\_MASK  
IOMUXC\_PAD\_APC\_SHIFT  
IOMUXC\_PAD\_APC(x)  
FSL\_COMPONENT\_ID

## 2.29 Common Driver

FSL\_COMMON\_DRIVER\_VERSION  
common driver version.  
DEBUG\_CONSOLE\_DEVICE\_TYPE\_NONE  
No debug console.  
DEBUG\_CONSOLE\_DEVICE\_TYPE\_UART  
Debug console based on UART.  
DEBUG\_CONSOLE\_DEVICE\_TYPE\_LPUART  
Debug console based on LPUART.  
DEBUG\_CONSOLE\_DEVICE\_TYPE\_LPSCI  
Debug console based on LPSCI.

DEBUG\_CONSOLE\_DEVICE\_TYPE\_USBCDC

Debug console based on USBCDC.

DEBUG\_CONSOLE\_DEVICE\_TYPE\_FLEXCOMM

Debug console based on FLEXCOMM.

DEBUG\_CONSOLE\_DEVICE\_TYPE\_IUART

Debug console based on i.MX UART.

DEBUG\_CONSOLE\_DEVICE\_TYPE\_VUSART

Debug console based on LPC\_VUSART.

DEBUG\_CONSOLE\_DEVICE\_TYPE\_MINI\_USART

Debug console based on LPC\_USART.

DEBUG\_CONSOLE\_DEVICE\_TYPE\_SWO

Debug console based on SWO.

DEBUG\_CONSOLE\_DEVICE\_TYPE\_QSCI

Debug console based on QSCI.

MIN(*a*, *b*)

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

MAX(*a*, *b*)

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

UINT16\_MAX

Max value of uint16\_t type.

UINT32\_MAX

Max value of uint32\_t type.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

USEC\_TO\_COUNT(*us*, *clockFreqInHz*)

Macro to convert a microsecond period to raw count value



COUNT\_TO\_USEC(count, clockFreqInHz)

Macro to convert a raw count value to microsecond

MSEC\_TO\_COUNT(ms, clockFreqInHz)

Macro to convert a millisecond period to raw count value

COUNT\_TO\_MSEC(count, clockFreqInHz)

Macro to convert a raw count value to millisecond

SDK\_ISR\_EXIT\_BARRIER

SDK\_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 (rounded up)

SDK\_SIZEALIGN\_UP(var, alignbytes)

Macro to change a value to a given size aligned value (rounded up), the wrapper of SDK\_SIZEALIGN

SDK\_SIZEALIGN\_DOWN(var, alignbytes)

Macro to change a value to a given size aligned value (rounded down)

SDK\_IS\_ALIGNED(var, alignbytes)

Macro to check if a value is aligned to a given size

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*.

**MCUX\_RAMFUNC**

Function attribute to place function in RAM. For example, to place function `my_func` in ram, use like:

```
MCUX_RAMFUNC my_func
```

**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\_SDIO SLV  
Group number for SDIO SLV status codes.

enumerator kStatusGroup\_MECC  
Group number for MECC status codes.

enumerator kStatusGroup\_ENET\_QOS  
Group number for ENET\_QOS status codes.

enumerator kStatusGroup\_LOG  
Group number for LOG status codes.

enumerator kStatusGroup\_I3CBUS  
Group number for I3CBUS status codes.

enumerator kStatusGroup\_QSCI  
Group number for QSCI status codes.

enumerator kStatusGroup\_ELEMU  
Group number for ELEMU status codes.

enumerator kStatusGroup\_QUEUE DSPI  
Group number for QSPI status codes.

enumerator kStatusGroup\_POWER\_MANAGER  
Group number for POWER\_MANAGER status codes.

enumerator kStatusGroup\_IPED  
Group number for IPED status codes.

enumerator kStatusGroup\_ELS\_PKC  
Group number for ELS PKC status codes.

enumerator kStatusGroup\_CSS\_PKC  
Group number for CSS PKC status codes.

enumerator kStatusGroup\_HOSTIF  
Group number for HOSTIF status codes.

enumerator kStatusGroup\_CLIF

Group number for CLIF status codes.

enumerator kStatusGroup\_BMA

Group number for BMA status codes.

enumerator kStatusGroup\_NETC

Group number for NETC status codes.

enumerator kStatusGroup\_ELE

Group number for ELE status codes.

enumerator kStatusGroup\_GLIKEY

Group number for GLIKEY status codes.

enumerator kStatusGroup\_AON\_POWER

Group number for AON\_POWER status codes.

enumerator kStatusGroup\_AON\_COMMON

Group number for AON\_COMMON status codes.

enumerator kStatusGroup\_ENDAT3

Group number for ENDAT3 status codes.

enumerator kStatusGroup\_HIPERFACE

Group number for HIPERFACE status codes.

enumerator kStatusGroup\_NPX

Group number for NPX status codes.

enumerator kStatusGroup\_ELA\_CSEC

Group number for ELA\_CSEC status codes.

enumerator kStatusGroup\_FLEXIO\_T\_FORMAT

Group number for T-format status codes.

enumerator kStatusGroup\_FLEXIO\_A\_FORMAT

Group number for A-format status codes.

enumerator kStatusGroup\_LPC\_QSPI

Group number for LPC QSPI 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.

## 2.30 LPI2C: Low Power Inter-Integrated Circuit Driver

void LPI2C\_DriverIRQHandler(uint32\_t instance)

LPI2C driver IRQ handler common entry.

This function provides the common IRQ request entry for LPI2C.

### Parameters

- instance – LPI2C instance.

FSL\_LPI2C\_DRIVER\_VERSION

LPI2C driver version.

LPI2C status return codes.

*Values:*

enumerator kStatus\_LPI2C\_Busy

The master is already performing a transfer.

enumerator kStatus\_LPI2C\_Idle

The slave driver is idle.

enumerator kStatus\_LPI2C\_Nak

The slave device sent a NAK in response to a byte.

enumerator kStatus\_LPI2C\_FifoError

FIFO under run or overrun.

enumerator kStatus\_LPI2C\_BitError

Transferred bit was not seen on the bus.

enumerator kStatus\_LPI2C\_ArbitrationLost

Arbitration lost error.

enumerator kStatus\_LPI2C\_PinLowTimeout

SCL or SDA were held low longer than the timeout.

enumerator kStatus\_LPI2C\_NoTransferInProgress

Attempt to abort a transfer when one is not in progress.

enumerator kStatus\_LPI2C\_DmaRequestFail

DMA request failed.

enumerator kStatus\_LPI2C\_Timeout

Timeout polling status flags.

IRQn\_Type const kLpi2cMasterIrqs[]

Array to map LPI2C instance number to IRQ number, used internally for LPI2C master interrupt and EDMA transactional APIs.

IRQn\_Type const kLpi2cSlaveIrqs[]

*lpi2c\_master\_isr\_t* s\_lpi2cMasterIsr

Pointer to master IRQ handler for each instance, used internally for LPI2C master interrupt and EDMA transactional APIs.

void \*s\_lpi2cMasterHandle[]

Pointers to master handles for each instance, used internally for LPI2C master interrupt and EDMA transactional APIs.

uint32\_t LPI2C\_GetInstance(LPI2C\_Type \*base)

Returns an instance number given a base address.

If an invalid base address is passed, debug builds will assert. Release builds will just return instance number 0.

#### Parameters

- base – The LPI2C peripheral base address.

#### Returns

LPI2C instance number starting from 0.

I2C\_RETRY\_TIMES

Retry times for waiting flag.

## 2.31 LPI2C Master Driver

void LPI2C\_MasterGetDefaultConfig(*lpi2c\_master\_config\_t* \*masterConfig)

Provides a default configuration for the LPI2C master peripheral.

This function provides the following default configuration for the LPI2C master peripheral:

```
masterConfig->enableMaster      = true;
masterConfig->debugEnable       = false;
masterConfig->ignoreAck         = false;
masterConfig->pinConfig         = kLPI2C_2PinOpenDrain;
masterConfig->baudRate_Hz       = 100000U;
masterConfig->busIdleTimeout_ns = 0;
masterConfig->pinLowTimeout_ns  = 0;
masterConfig->sdaGlitchFilterWidth_ns = 0;
masterConfig->sclGlitchFilterWidth_ns = 0;
masterConfig->hostRequest.enable = false;
masterConfig->hostRequest.source  = kLPI2C_HostRequestExternalPin;
masterConfig->hostRequest.polarity = kLPI2C_HostRequestPinActiveHigh;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with `LPI2C_MasterInit()`.

#### Parameters

- `masterConfig` – **[out]** User provided configuration structure for default values. Refer to `lpi2c_master_config_t`.

```
void LPI2C_MasterInit(LPI2C_Type *base, const lpi2c_master_config_t *masterConfig, uint32_t sourceClock_Hz)
```

Initializes the LPI2C master peripheral.

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

#### Parameters

- `base` – The LPI2C peripheral base address.
- `masterConfig` – User provided peripheral configuration. Use `LPI2C_MasterGetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

```
void LPI2C_MasterDeinit(LPI2C_Type *base)
```

Deinitializes the LPI2C master peripheral.

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

#### Parameters

- `base` – The LPI2C peripheral base address.

```
void LPI2C_MasterConfigureDataMatch(LPI2C_Type *base, const lpi2c_data_match_config_t *matchConfig)
```

Configures LPI2C master data match feature.

#### Parameters

- `base` – The LPI2C peripheral base address.
- `matchConfig` – Settings for the data match feature.

```
status_t LPI2C_MasterCheckAndClearError(LPI2C_Type *base, uint32_t status)
```

Convert provided flags to status code, and clear any errors if present.

#### Parameters

- `base` – The LPI2C peripheral base address.
- `status` – Current status flags value that will be checked.

#### Return values

- `kStatus_Success` –
- `kStatus_LPI2C_PinLowTimeout` –
- `kStatus_LPI2C_ArbitrationLost` –
- `kStatus_LPI2C_Nak` –
- `kStatus_LPI2C_FifoError` –

*status\_t* LPI2C\_CheckForBusyBus(LPI2C\_Type \*base)

Make sure the bus isn't already busy.

A busy bus is allowed if we are the one driving it.

#### Parameters

- base – The LPI2C peripheral base address.

#### Return values

- kStatus\_Success –
- kStatus\_LPI2C\_Busy –

static inline void LPI2C\_MasterReset(LPI2C\_Type \*base)

Performs a software reset.

Restores the LPI2C master peripheral to reset conditions.

#### Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C\_MasterEnable(LPI2C\_Type \*base, bool enable)

Enables or disables the LPI2C module as master.

#### Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as master.

static inline uint32\_t LPI2C\_MasterGetStatusFlags(LPI2C\_Type \*base)

Gets the LPI2C master status flags.

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

#### See also:

`_lpi2c_master_flags`

#### Parameters

- base – The LPI2C peripheral base address.

#### Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C\_MasterClearStatusFlags(LPI2C\_Type \*base, uint32\_t statusMask)

Clears the LPI2C master status flag state.

The following status register flags can be cleared:

- kLPI2C\_MasterEndOfPacketFlag
- kLPI2C\_MasterStopDetectFlag
- kLPI2C\_MasterNackDetectFlag
- kLPI2C\_MasterArbitrationLostFlag
- kLPI2C\_MasterFifoErrFlag
- kLPI2C\_MasterPinLowTimeoutFlag

- kLPI2C\_MasterDataMatchFlag

Attempts to clear other flags has no effect.

**See also:**

`_lpi2c_master_flags`.

**Parameters**

- `base` – The LPI2C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_lpi2c_master_flags` enumerators OR'd together. You may pass the result of a previous call to `LPI2C_MasterGetStatusFlags()`.

`static inline void LPI2C_MasterEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)`

Enables the LPI2C master interrupt requests.

All flags except `kLPI2C_MasterBusyFlag` and `kLPI2C_MasterBusBusyFlag` can be enabled as interrupts.

**Parameters**

- `base` – The LPI2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_lpi2c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline void LPI2C_MasterDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)`

Disables the LPI2C master interrupt requests.

All flags except `kLPI2C_MasterBusyFlag` and `kLPI2C_MasterBusBusyFlag` can be enabled as interrupts.

**Parameters**

- `base` – The LPI2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_lpi2c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline uint32_t LPI2C_MasterGetEnabledInterrupts(LPI2C_Type *base)`

Returns the set of currently enabled LPI2C master interrupt requests.

**Parameters**

- `base` – The LPI2C peripheral base address.

**Returns**

A bitmask composed of `_lpi2c_master_flags` enumerators OR'd together to indicate the set of enabled interrupts.

`static inline void LPI2C_MasterEnableDMA(LPI2C_Type *base, bool enableTx, bool enableRx)`

Enables or disables LPI2C master DMA requests.

**Parameters**

- `base` – The LPI2C peripheral base address.
- `enableTx` – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- `enableRx` – Enable flag for receive DMA request. Pass true for enable, false for disable.



```
static inline uint32_t LPI2C_MasterGetTxFifoAddress(LPI2C_Type *base)
```

Gets LPI2C master transmit data register address for DMA transfer.

**Parameters**

- `base` – The LPI2C peripheral base address.

**Returns**

The LPI2C Master Transmit Data Register address.

```
static inline uint32_t LPI2C_MasterGetRxFifoAddress(LPI2C_Type *base)
```

Gets LPI2C master receive data register address for DMA transfer.

**Parameters**

- `base` – The LPI2C peripheral base address.

**Returns**

The LPI2C Master Receive Data Register address.

```
static inline void LPI2C_MasterSetWatermarks(LPI2C_Type *base, size_t txWords, size_t rxWords)
```

Sets the watermarks for LPI2C master FIFOs.

**Parameters**

- `base` – The LPI2C peripheral base address.
- `txWords` – Transmit FIFO watermark value in words. The `kLPI2C_MasterTxReadyFlag` flag is set whenever the number of words in the transmit FIFO is equal or less than `txWords`. Writing a value equal or greater than the FIFO size is truncated.
- `rxWords` – Receive FIFO watermark value in words. The `kLPI2C_MasterRxReadyFlag` flag is set whenever the number of words in the receive FIFO is greater than `rxWords`. Writing a value equal or greater than the FIFO size is truncated.

```
static inline void LPI2C_MasterGetFifoCounts(LPI2C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of words in the LPI2C master FIFOs.

**Parameters**

- `base` – The LPI2C peripheral base address.
- `txCount` – **[out]** Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.

```
void LPI2C_MasterSetBaudRate(LPI2C_Type *base, uint32_t sourceClock_Hz, uint32_t baudRate_Hz)
```

Sets the I2C bus frequency for master transactions.

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

---

**Note:** Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

---

**Parameters**

- `base` – The LPI2C peripheral base address.
- `sourceClock_Hz` – LPI2C functional clock frequency in Hertz.
- `baudRate_Hz` – Requested bus frequency in Hertz.

`static inline bool LPI2C_MasterGetBusIdleState(LPI2C_Type *base)`

Returns whether the bus is idle.

Requires the master mode to be enabled.

#### Parameters

- `base` – The LPI2C peripheral base address.

#### Return values

- `true` – Bus is busy.
- `false` – Bus is idle.

`status_t LPI2C_MasterStart(LPI2C_Type *base, uint8_t address, lpi2c_direction_t dir)`

Sends a START signal and slave address on the I2C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

#### Parameters

- `base` – The LPI2C peripheral base address.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kLPI2C_Read` or `kLPI2C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

#### Return values

- `kStatus_Success` – START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.

`static inline status_t LPI2C_MasterRepeatedStart(LPI2C_Type *base, uint8_t address, lpi2c_direction_t dir)`

Sends a repeated START signal and slave address on the I2C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like `LPI2C_MasterStart()`, it also sends the specified 7-bit address.

---

**Note:** This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

---

#### Parameters

- `base` – The LPI2C peripheral base address.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kLPI2C_Read` or `kLPI2C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

#### Return values

- `kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.

*status\_t* LPI2C\_MasterSend(LPI2C\_Type \*base, void \*txBuff, size\_t txSize)

Performs a polling send transfer on the I2C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns `kStatus_LPI2C_Nak`.

#### Parameters

- *base* – The LPI2C peripheral base address.
- *txBuff* – The pointer to the data to be transferred.
- *txSize* – The length in bytes of the data to be transferred.

#### Return values

- `kStatus_Success` – Data was sent successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or over run.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

*status\_t* LPI2C\_MasterReceive(LPI2C\_Type \*base, void \*rxBuff, size\_t rxSize)

Performs a polling receive transfer on the I2C bus.

#### Parameters

- *base* – The LPI2C peripheral base address.
- *rxBuff* – The pointer to the data to be transferred.
- *rxSize* – The length in bytes of the data to be transferred.

#### Return values

- `kStatus_Success` – Data was received successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

*status\_t* LPI2C\_MasterStop(LPI2C\_Type \*base)

Sends a STOP signal on the I2C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

#### Parameters

- *base* – The LPI2C peripheral base address.

#### Return values

- `kStatus_Success` – The STOP signal was successfully sent on the bus and the transaction terminated.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

`status_t LPI2C_MasterTransferBlocking(LPI2C_Type *base, lpi2c_master_transfer_t *transfer)`  
Performs a master polling transfer on the I2C bus.

---

**Note:** The API does not return until the transfer succeeds or fails due to error happens during transfer.

---

#### Parameters

- `base` – The LPI2C peripheral base address.
- `transfer` – Pointer to the transfer structure.

#### Return values

- `kStatus_Success` – Data was received successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

`void LPI2C_MasterTransferCreateHandle(LPI2C_Type *base, lpi2c_master_handle_t *handle, lpi2c_master_transfer_callback_t callback, void *userData)`

Creates a new handle for the LPI2C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `LPI2C_MasterTransferAbort()` API shall be called.

---

**Note:** The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

---

#### Parameters

- `base` – The LPI2C peripheral base address.
- `handle` – **[out]** Pointer to the LPI2C master driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

```
status_t LPI2C_MasterTransferNonBlocking(LPI2C_Type *base, lpi2c_master_handle_t *handle,  
                                         lpi2c_master_transfer_t *transfer)
```

Performs a non-blocking transaction on the I2C bus.

#### Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

#### Return values

- kStatus\_Success – The transaction was started successfully.
- kStatus\_LPI2C\_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

```
status_t LPI2C_MasterTransferGetCount(LPI2C_Type *base, lpi2c_master_handle_t *handle,  
                                       size_t *count)
```

Returns number of bytes transferred so far.

#### Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

#### Return values

- kStatus\_Success –
- kStatus\_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

```
void LPI2C_MasterTransferAbort(LPI2C_Type *base, lpi2c_master_handle_t *handle)
```

Terminates a non-blocking LPI2C master transmission early.

---

**Note:** It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

---

#### Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

```
void LPI2C_MasterTransferHandleIRQ(LPI2C_Type *base, void *lpi2cMasterHandle)
```

Reusable routine to handle master interrupts.

---

**Note:** This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

---

#### Parameters

- base – The LPI2C peripheral base address.
- lpi2cMasterHandle – Pointer to the LPI2C master driver handle.

enum \_lpi2c\_master\_flags

LPI2C master peripheral flags.

The following status register flags can be cleared:

- kLPI2C\_MasterEndOfPacketFlag
- kLPI2C\_MasterStopDetectFlag
- kLPI2C\_MasterNackDetectFlag
- kLPI2C\_MasterArbitrationLostFlag
- kLPI2C\_MasterFifoErrFlag
- kLPI2C\_MasterPinLowTimeoutFlag
- kLPI2C\_MasterDataMatchFlag

All flags except kLPI2C\_MasterBusyFlag and kLPI2C\_MasterBusBusyFlag can be enabled as interrupts.

---

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

---

*Values:*

enumerator kLPI2C\_MasterTxReadyFlag

Transmit data flag

enumerator kLPI2C\_MasterRxReadyFlag

Receive data flag

enumerator kLPI2C\_MasterEndOfPacketFlag

End Packet flag

enumerator kLPI2C\_MasterStopDetectFlag

Stop detect flag

enumerator kLPI2C\_MasterNackDetectFlag

NACK detect flag

enumerator kLPI2C\_MasterArbitrationLostFlag

Arbitration lost flag

enumerator kLPI2C\_MasterFifoErrFlag

FIFO error flag

enumerator kLPI2C\_MasterPinLowTimeoutFlag

Pin low timeout flag

enumerator kLPI2C\_MasterDataMatchFlag

Data match flag

enumerator kLPI2C\_MasterBusyFlag

Master busy flag

enumerator kLPI2C\_MasterBusBusyFlag

Bus busy flag

enumerator kLPI2C\_MasterClearFlags

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

enumerator kLPI2C\_MasterIrqFlags

IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C\_MasterErrorFlags

Errors to check for.

enum \_lpi2c\_direction

Direction of master and slave transfers.

*Values:*

enumerator kLPI2C\_Write

Master transmit.

enumerator kLPI2C\_Read

Master receive.

enum \_lpi2c\_master\_pin\_config

LPI2C pin configuration.

*Values:*

enumerator kLPI2C\_2PinOpenDrain

LPI2C Configured for 2-pin open drain mode

enumerator kLPI2C\_2PinOutputOnly

LPI2C Configured for 2-pin output only mode (ultra-fast mode)

enumerator kLPI2C\_2PinPushPull

LPI2C Configured for 2-pin push-pull mode

enumerator kLPI2C\_4PinPushPull

LPI2C Configured for 4-pin push-pull mode

enumerator kLPI2C\_2PinOpenDrainWithSeparateSlave

LPI2C Configured for 2-pin open drain mode with separate LPI2C slave

enumerator kLPI2C\_2PinOutputOnlyWithSeparateSlave

LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave

enumerator kLPI2C\_2PinPushPullWithSeparateSlave

LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave

enumerator kLPI2C\_4PinPushPullWithInvertedOutput

LPI2C Configured for 4-pin push-pull mode(inverted outputs)

enum \_lpi2c\_host\_request\_source

LPI2C master host request selection.

*Values:*

enumerator kLPI2C\_HostRequestExternalPin

Select the LPI2C\_HREQ pin as the host request input

enumerator kLPI2C\_HostRequestInputTrigger

Select the input trigger as the host request input

enum \_lpi2c\_host\_request\_polarity

LPI2C master host request pin polarity configuration.

*Values:*

enumerator kLPI2C\_HostRequestPinActiveLow

Configure the LPI2C\_HREQ pin active low

enumerator kLPI2C\_HostRequestPinActiveHigh  
Configure the LPI2C\_HREQ pin active high

enum \_lpi2c\_data\_match\_config\_mode  
LPI2C master data match configuration modes.

*Values:*

enumerator kLPI2C\_MatchDisabled  
LPI2C Match Disabled

enumerator kLPI2C\_1stWordEqualsM0OrM1  
LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1

enumerator kLPI2C\_AnyWordEqualsM0OrM1  
LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1

enumerator kLPI2C\_1stWordEqualsM0And2ndWordEqualsM1  
LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1

enumerator kLPI2C\_AnyWordEqualsM0AndNextWordEqualsM1  
LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1

enumerator kLPI2C\_1stWordAndM1EqualsM0AndM1  
LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1

enumerator kLPI2C\_AnyWordAndM1EqualsM0AndM1  
LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1

enum \_lpi2c\_master\_transfer\_flags  
Transfer option flags.

---

**Note:** These enumerations are intended to be OR'd together to form a bit mask of options for the `_lpi2c_master_transfer::flags` field.

---

*Values:*

enumerator kLPI2C\_TransferDefaultFlag  
Transfer starts with a start signal, stops with a stop signal.

enumerator kLPI2C\_TransferNoStartFlag  
Don't send a start condition, address, and sub address

enumerator kLPI2C\_TransferRepeatedStartFlag  
Send a repeated start condition

enumerator kLPI2C\_TransferNoStopFlag  
Don't send a stop condition.

typedef enum \_lpi2c\_direction lpi2c\_direction\_t  
Direction of master and slave transfers.

typedef enum \_lpi2c\_master\_pin\_config lpi2c\_master\_pin\_config\_t  
LPI2C pin configuration.

typedef enum \_lpi2c\_host\_request\_source lpi2c\_host\_request\_source\_t  
LPI2C master host request selection.

typedef enum \_lpi2c\_host\_request\_polarity lpi2c\_host\_request\_polarity\_t  
LPI2C master host request pin polarity configuration.



```
typedef struct _lpi2c_master_config lpi2c_master_config_t
```

Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C\_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

```
typedef enum _lpi2c_data_match_config_mode lpi2c_data_match_config_mode_t
```

LPI2C master data match configuration modes.

```
typedef struct _lpi2c_match_config lpi2c_data_match_config_t
```

LPI2C master data match configuration structure.

```
typedef struct _lpi2c_master_transfer lpi2c_master_transfer_t
```

LPI2C master descriptor of the transfer.

```
typedef struct _lpi2c_master_handle lpi2c_master_handle_t
```

LPI2C master handle of the transfer.

```
typedef void (*lpi2c_master_transfer_callback_t)(LPI2C_Type *base, lpi2c_master_handle_t
*handle, status_t completionStatus, void *userData)
```

Master completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C\_MasterTransferCreateHandle().

#### **Param base**

The LPI2C peripheral base address.

#### **Param handle**

Pointer to the LPI2C master driver handle.

#### **Param completionStatus**

Either kStatus\_Success or an error code describing how the transfer completed.

#### **Param userData**

Arbitrary pointer-sized value passed from the application.

```
typedef void (*lpi2c_master_isr_t)(LPI2C_Type *base, void *handle)
```

Typedef for master interrupt handler, used internally for LPI2C master interrupt and EDMA transactional APIs.

```
struct _lpi2c_master_config
```

*#include <fsl\_lpi2c.h>* Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C\_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

## **Public Members**

```
bool enableMaster
```

Whether to enable master mode.

```
bool enableDoze
```

Whether master is enabled in doze mode.

bool debugEnable

Enable transfers to continue when halted in debug mode.

bool ignoreAck

Whether to ignore ACK/NACK.

*lpi2c\_master\_pin\_config\_t* pinConfig

The pin configuration option.

uint32\_t baudRate\_Hz

Desired baud rate in Hertz.

uint32\_t busIdleTimeout\_ns

Bus idle timeout in nanoseconds. Set to 0 to disable.

uint32\_t pinLowTimeout\_ns

Pin low timeout in nanoseconds. Set to 0 to disable.

uint8\_t sdaGlitchFilterWidth\_ns

Width in nanoseconds of glitch filter on SDA pin. Set to 0 to disable.

uint8\_t sclGlitchFilterWidth\_ns

Width in nanoseconds of glitch filter on SCL pin. Set to 0 to disable.

struct *\_lpi2c\_master\_config* hostRequest

Host request options.

struct *\_lpi2c\_match\_config*

*#include <fsl\_lpi2c.h>* LPI2C master data match configuration structure.

### Public Members

*lpi2c\_data\_match\_config\_mode\_t* matchMode

Data match configuration setting.

bool rxDataMatchOnly

When set to true, received data is ignored until a successful match.

uint32\_t match0

Match value 0.

uint32\_t match1

Match value 1.

struct *\_lpi2c\_master\_transfer*

*#include <fsl\_lpi2c.h>* Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the LPI2C\_MasterTransferNonBlocking() API.

### Public Members

uint32\_t flags

Bit mask of options for the transfer. See enumeration *\_lpi2c\_master\_transfer\_flags* for available options. Set to 0 or *kLPI2C\_TransferDefaultFlag* for normal transfers.

uint16\_t slaveAddress

The 7-bit slave address.

*lpi2c\_direction\_t* direction

Either kLPI2C\_Read or kLPI2C\_Write.

uint32\_t subaddress

Sub address. Transferred MSB first.

size\_t subaddressSize

Length of sub address to send in bytes. Maximum size is 4 bytes.

void \*data

Pointer to data to transfer.

size\_t dataSize

Number of bytes to transfer.

struct \_lpi2c\_master\_handle

*#include <fsl\_lpi2c.h>* Driver handle for master non-blocking APIs.

---

**Note:** The contents of this structure are private and subject to change.

---

### Public Members

uint8\_t state

Transfer state machine current state.

uint16\_t remainingBytes

Remaining byte count in current state.

uint8\_t \*buf

Buffer pointer for current state.

uint16\_t commandBuffer[6]

LPI2C command sequence. When all 6 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word]

*lpi2c\_master\_transfer\_t* transfer

Copy of the current transfer info.

*lpi2c\_master\_transfer\_callback\_t* completionCallback

Callback function pointer.

void \*userData

Application data passed to callback.

struct hostRequest

### Public Members

bool enable

Enable host request.

*lpi2c\_host\_request\_source\_t* source

Host request source.

*lpi2c\_host\_request\_polarity\_t* polarity

Host request pin polarity.

## 2.32 LPI2C Master DMA Driver

```
void LPI2C_MasterCreateEDMAHandle(LPI2C_Type *base, lpi2c_master_edma_handle_t *handle,
                                   edma_handle_t *rxDmaHandle, edma_handle_t
                                   *txDmaHandle, lpi2c_master_edma_transfer_callback_t
                                   callback, void *userData)
```

Create a new handle for the LPI2C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C\_MasterTransferAbortEDMA() API shall be called.

For devices where the LPI2C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

### Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – **[out]** Pointer to the LPI2C master driver handle.
- *rxDmaHandle* – Handle for the eDMA receive channel. Created by the user prior to calling this function.
- *txDmaHandle* – Handle for the eDMA transmit channel. Created by the user prior to calling this function.
- *callback* – User provided pointer to the asynchronous callback function.
- *userData* – User provided pointer to the application callback data.

```
status_t LPI2C_MasterTransferEDMA(LPI2C_Type *base, lpi2c_master_edma_handle_t *handle,
                                   lpi2c_master_transfer_t *transfer)
```

Performs a non-blocking DMA-based transaction on the I2C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

### Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – Pointer to the LPI2C master driver handle.
- *transfer* – The pointer to the transfer descriptor.

### Return values

- *kStatus\_Success* – The transaction was started successfully.
- *kStatus\_LPI2C\_Busy* – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

```
status_t LPI2C_MasterTransferGetCountEDMA(LPI2C_Type *base, lpi2c_master_edma_handle_t
                                           *handle, size_t *count)
```

Returns number of bytes transferred so far.

### Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – Pointer to the LPI2C master driver handle.
- *count* – **[out]** Number of bytes transferred so far by the non-blocking transaction.

### Return values

- *kStatus\_Success* –

- `kStatus_NoTransferInProgress` – There is not a DMA transaction currently in progress.

`status_t` LPI2C\_MasterTransferAbortEDMA(LPI2C\_Type \*base, *lpi2c\_master\_edma\_handle\_t* \*handle)

Terminates a non-blocking LPI2C master transmission early.

---

**Note:** It is not safe to call this function from an IRQ handler that has a higher priority than the eDMA peripheral's IRQ priority.

---

### Parameters

- `base` – The LPI2C peripheral base address.
- `handle` – Pointer to the LPI2C master driver handle.

### Return values

- `kStatus_Success` – A transaction was successfully aborted.
- `kStatus_LPI2C_Idle` – There is not a DMA transaction currently in progress.

`typedef struct lpi2c_master_edma_handle lpi2c_master_edma_handle_t`  
LPI2C master EDMA handle of the transfer.

`typedef void (*lpi2c_master_edma_transfer_callback_t)(LPI2C_Type *base, lpi2c_master_edma_handle_t *handle, status_t completionStatus, void *userData)`

Master DMA completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to `LPI2C_MasterCreateEDMAHandle()`.

### Param base

The LPI2C peripheral base address.

### Param handle

Handle associated with the completed transfer.

### Param completionStatus

Either `kStatus_Success` or an error code describing how the transfer completed.

### Param userData

Arbitrary pointer-sized value passed from the application.

`struct _lpi2c_master_edma_handle`  
`#include <fsl_lpi2c_edma.h>` Driver handle for master DMA APIs.

---

**Note:** The contents of this structure are private and subject to change.

---

### Public Members

`LPI2C_Type *base`  
LPI2C base pointer.

`bool isBusy`  
Transfer state machine current state.

uint8\_t nbytes

eDMA minor byte transfer count initially configured.

uint16\_t commandBuffer[20]

LPI2C command sequence. When all 10 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word] + receive&Size[4 words]

*lpi2c\_master\_transfer\_t* transfer

Copy of the current transfer info.

*lpi2c\_master\_edma\_transfer\_callback\_t* completionCallback

Callback function pointer.

void \*userData

Application data passed to callback.

*edma\_handle\_t* \*rx

Handle for receive DMA channel.

*edma\_handle\_t* \*tx

Handle for transmit DMA channel.

*edma\_tcd\_t* tcds[3]

Software TCD. Three are allocated to provide enough room to align to 32-bytes.

## 2.33 LPI2C Slave Driver

void LPI2C\_SlaveGetDefaultConfig(*lpi2c\_slave\_config\_t* \*slaveConfig)

Provides a default configuration for the LPI2C slave peripheral.

This function provides the following default configuration for the LPI2C slave peripheral:

```
slaveConfig->enableSlave      = true;
slaveConfig->address0         = 0U;
slaveConfig->address1         = 0U;
slaveConfig->addressMatchMode = kLPI2C_MatchAddress0;
slaveConfig->filterDozeEnable  = true;
slaveConfig->filterEnable     = true;
slaveConfig->enableGeneralCall = false;
slaveConfig->sclStall.enableAck = false;
slaveConfig->sclStall.enableTx  = true;
slaveConfig->sclStall.enableRx  = true;
slaveConfig->sclStall.enableAddress = true;
slaveConfig->ignoreAck         = false;
slaveConfig->enableReceivedAddressRead = false;
slaveConfig->sdaGlitchFilterWidth_ns = 0;
slaveConfig->sclGlitchFilterWidth_ns = 0;
slaveConfig->dataValidDelay_ns  = 0;
slaveConfig->clockHoldTime_ns   = 0;
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with LPI2C\_SlaveInit(). Be sure to override at least the *address0* member of the configuration structure with the desired slave address.

### Parameters

- slaveConfig – **[out]** User provided configuration structure that is set to default values. Refer to *lpi2c\_slave\_config\_t*.

```
void LPI2C_SlaveInit(LPI2C_Type *base, const lpi2c_slave_config_t *slaveConfig, uint32_t  
                    sourceClock_Hz)
```

Initializes the LPI2C slave peripheral.

This function enables the peripheral clock and initializes the LPI2C slave peripheral as described by the user provided configuration.

#### Parameters

- `base` – The LPI2C peripheral base address.
- `slaveConfig` – User provided peripheral configuration. Use `LPI2C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the LPI2C functional clock. Used to calculate the filter widths, data valid delay, and clock hold time.

```
void LPI2C_SlaveDeinit(LPI2C_Type *base)
```

Deinitializes the LPI2C slave peripheral.

This function disables the LPI2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

#### Parameters

- `base` – The LPI2C peripheral base address.

```
static inline void LPI2C_SlaveReset(LPI2C_Type *base)
```

Performs a software reset of the LPI2C slave peripheral.

#### Parameters

- `base` – The LPI2C peripheral base address.

```
static inline void LPI2C_SlaveEnable(LPI2C_Type *base, bool enable)
```

Enables or disables the LPI2C module as slave.

#### Parameters

- `base` – The LPI2C peripheral base address.
- `enable` – Pass true to enable or false to disable the specified LPI2C as slave.

```
static inline uint32_t LPI2C_SlaveGetStatusFlags(LPI2C_Type *base)
```

Gets the LPI2C slave status flags.

A bit mask with the state of all LPI2C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

#### See also:

`_lpi2c_slave_flags`

#### Parameters

- `base` – The LPI2C peripheral base address.

#### Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void LPI2C_SlaveClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)
```

Clears the LPI2C status flag state.

The following status register flags can be cleared:

- kLPI2C\_SlaveRepeatedStartDetectFlag
- kLPI2C\_SlaveStopDetectFlag
- kLPI2C\_SlaveBitErrFlag
- kLPI2C\_SlaveFifoErrFlag

Attempts to clear other flags has no effect.

**See also:**

`_lpi2c_slave_flags`.

#### Parameters

- `base` – The LPI2C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_lpi2c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `LPI2C_SlaveGetStatusFlags()`.

```
static inline void LPI2C_SlaveEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)
```

Enables the LPI2C slave interrupt requests.

All flags except `kLPI2C_SlaveBusyFlag` and `kLPI2C_SlaveBusBusyFlag` can be enabled as interrupts.

#### Parameters

- `base` – The LPI2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_lpi2c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline void LPI2C_SlaveDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)
```

Disables the LPI2C slave interrupt requests.

All flags except `kLPI2C_SlaveBusyFlag` and `kLPI2C_SlaveBusBusyFlag` can be enabled as interrupts.

#### Parameters

- `base` – The LPI2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_lpi2c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline uint32_t LPI2C_SlaveGetEnabledInterrupts(LPI2C_Type *base)
```

Returns the set of currently enabled LPI2C slave interrupt requests.

#### Parameters

- `base` – The LPI2C peripheral base address.

#### Returns

A bitmask composed of `_lpi2c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

```
static inline void LPI2C_SlaveEnableDMA(LPI2C_Type *base, bool enableAddressValid, bool enableRx, bool enableTx)
```

Enables or disables the LPI2C slave peripheral DMA requests.



**Parameters**

- `base` – The LPI2C peripheral base address.
- `enableAddressValid` – Enable flag for the address valid DMA request. Pass true for enable, false for disable. The address valid DMA request is shared with the receive data DMA request.
- `enableRx` – Enable flag for the receive data DMA request. Pass true for enable, false for disable.
- `enableTx` – Enable flag for the transmit data DMA request. Pass true for enable, false for disable.

`static inline bool LPI2C_SlaveGetBusIdleState(LPI2C_Type *base)`

Returns whether the bus is idle.

Requires the slave mode to be enabled.

**Parameters**

- `base` – The LPI2C peripheral base address.

**Return values**

- `true` – Bus is busy.
- `false` – Bus is idle.

`static inline void LPI2C_SlaveTransmitAck(LPI2C_Type *base, bool ackOrNack)`

Transmits either an ACK or NAK on the I2C bus in response to a byte from the master.

Use this function to send an ACK or NAK when the `kLPI2C_SlaveTransmitAckFlag` is asserted. This only happens if you enable the `sclStall.enableAck` field of the `lpi2c_slave_config_t` configuration structure used to initialize the slave peripheral.

**Parameters**

- `base` – The LPI2C peripheral base address.
- `ackOrNack` – Pass true for an ACK or false for a NAK.

`static inline void LPI2C_SlaveEnableAckStall(LPI2C_Type *base, bool enable)`

Enables or disables ACKSTALL.

When enables ACKSTALL, software can transmit either an ACK or NAK on the I2C bus in response to a byte from the master.

**Parameters**

- `base` – The LPI2C peripheral base address.
- `enable` – True will enable ACKSTALL, false will disable ACKSTALL.

`static inline uint32_t LPI2C_SlaveGetReceivedAddress(LPI2C_Type *base)`

Returns the slave address sent by the I2C master.

This function should only be called if the `kLPI2C_SlaveAddressValidFlag` is asserted.

**Parameters**

- `base` – The LPI2C peripheral base address.

**Returns**

The 8-bit address matched by the LPI2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

`status_t LPI2C_SlaveSend(LPI2C_Type *base, void *txBuff, size_t txSize, size_t *actualTxSize)`

Performs a polling send transfer on the I2C bus.

**Parameters**

- `base` – The LPI2C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.
- `actualTxSize` – **[out]**

**Returns**

Error or success status returned by API.

`status_t` LPI2C\_SlaveReceive(LPI2C\_Type \*base, void \*rxBuff, size\_t rxSize, size\_t \*actualRxSize)

Performs a polling receive transfer on the I2C bus.

**Parameters**

- `base` – The LPI2C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.
- `actualRxSize` – **[out]**

**Returns**

Error or success status returned by API.

`void` LPI2C\_SlaveTransferCreateHandle(LPI2C\_Type \*base, *lpi2c\_slave\_handle\_t* \*handle, *lpi2c\_slave\_transfer\_callback\_t* callback, void \*userData)

Creates a new handle for the LPI2C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C\_SlaveTransferAbort() API shall be called.

---

**Note:** The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

---

**Parameters**

- `base` – The LPI2C peripheral base address.
- `handle` – **[out]** Pointer to the LPI2C slave driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

`status_t` LPI2C\_SlaveTransferNonBlocking(LPI2C\_Type \*base, *lpi2c\_slave\_handle\_t* \*handle, uint32\_t eventMask)

Starts accepting slave transfers.

Call this API after calling I2C\_SlaveInit() and LPI2C\_SlaveTransferCreateHandle() to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to LPI2C\_SlaveTransferCreateHandle(). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of *lpi2c\_slave\_transfer\_event\_t* enumerators for the events you wish to receive. The *kLPI2C\_SlaveTransmitEvent* and *kLPI2C\_SlaveReceiveEvent* events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the *kLPI2C\_SlaveAllEvents* constant is provided as a convenient way to enable all events.

**Parameters**

- base – The LPI2C peripheral base address.
- handle – Pointer to `lpi2c_slave_handle_t` structure which stores the transfer state.
- eventMask – Bit mask formed by OR'ing together `lpi2c_slave_transfer_event_t` enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and `kLPI2C_SlaveAllEvents` to enable all events.

**Return values**

- `kStatus__Success` – Slave transfers were successfully started.
- `kStatus__LPI2C_Busy` – Slave transfers have already been started on this handle.

`status_t LPI2C_SlaveTransferGetCount(LPI2C_Type *base, lpi2c_slave_handle_t *handle, size_t *count)`

Gets the slave transfer status during a non-blocking transfer.

**Parameters**

- base – The LPI2C peripheral base address.
- handle – Pointer to `i2c_slave_handle_t` structure.
- count – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

**Return values**

- `kStatus__Success` –
- `kStatus__NoTransferInProgress` –

`void LPI2C_SlaveTransferAbort(LPI2C_Type *base, lpi2c_slave_handle_t *handle)`

Aborts the slave non-blocking transfers.

---

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

---

**Parameters**

- base – The LPI2C peripheral base address.
- handle – Pointer to `lpi2c_slave_handle_t` structure which stores the transfer state.

`void LPI2C_SlaveTransferHandleIRQ(LPI2C_Type *base, lpi2c_slave_handle_t *handle)`

Reusable routine to handle slave interrupts.

---

**Note:** This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

---

**Parameters**

- base – The LPI2C peripheral base address.
- handle – Pointer to `lpi2c_slave_handle_t` structure which stores the transfer state.

enum \_lpi2c\_slave\_flags

LPI2C slave peripheral flags.

The following status register flags can be cleared:

- kLPI2C\_SlaveRepeatedStartDetectFlag
- kLPI2C\_SlaveStopDetectFlag
- kLPI2C\_SlaveBitErrFlag
- kLPI2C\_SlaveFifoErrFlag

All flags except kLPI2C\_SlaveBusyFlag and kLPI2C\_SlaveBusBusyFlag can be enabled as interrupts.

---

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

---

*Values:*

enumerator kLPI2C\_SlaveTxReadyFlag

Transmit data flag

enumerator kLPI2C\_SlaveRxReadyFlag

Receive data flag

enumerator kLPI2C\_SlaveAddressValidFlag

Address valid flag

enumerator kLPI2C\_SlaveTransmitAckFlag

Transmit ACK flag

enumerator kLPI2C\_SlaveRepeatedStartDetectFlag

Repeated start detect flag

enumerator kLPI2C\_SlaveStopDetectFlag

Stop detect flag

enumerator kLPI2C\_SlaveBitErrFlag

Bit error flag

enumerator kLPI2C\_SlaveFifoErrFlag

FIFO error flag

enumerator kLPI2C\_SlaveAddressMatch0Flag

Address match 0 flag

enumerator kLPI2C\_SlaveAddressMatch1Flag

Address match 1 flag

enumerator kLPI2C\_SlaveGeneralCallFlag

General call flag

enumerator kLPI2C\_SlaveBusyFlag

Master busy flag

enumerator kLPI2C\_SlaveBusBusyFlag

Bus busy flag

enumerator kLPI2C\_SlaveClearFlags

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

enumerator kLPI2C\_SlaveIrqFlags

IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C\_SlaveErrorFlags

Errors to check for.

enum \_\_lpi2c\_slave\_address\_match

LPI2C slave address match options.

*Values:*

enumerator kLPI2C\_MatchAddress0

Match only address 0.

enumerator kLPI2C\_MatchAddress0OrAddress1

Match either address 0 or address 1.

enumerator kLPI2C\_MatchAddress0ThroughAddress1

Match a range of slave addresses from address 0 through address 1.

enum \_\_lpi2c\_slave\_transfer\_event

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C\_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

---

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

---

*Values:*

enumerator kLPI2C\_SlaveAddressMatchEvent

Received the slave address after a start or repeated start.

enumerator kLPI2C\_SlaveTransmitEvent

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator kLPI2C\_SlaveReceiveEvent

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

enumerator kLPI2C\_SlaveTransmitAckEvent

Callback needs to either transmit an ACK or NACK.

enumerator kLPI2C\_SlaveRepeatedStartEvent

A repeated start was detected.

enumerator kLPI2C\_SlaveCompletionEvent

A stop was detected, completing the transfer.

enumerator kLPI2C\_SlaveAllEvents

Bit mask of all available events.

typedef enum \_\_lpi2c\_slave\_address\_match lpi2c\_slave\_address\_match\_t

LPI2C slave address match options.

typedef struct \_\_lpi2c\_slave\_config lpi2c\_slave\_config\_t

Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the LPI2C\_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

```
typedef enum _lpi2c_slave_transfer_event lpi2c_slave_transfer_event_t
```

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C\_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

---

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

---

```
typedef struct _lpi2c_slave_transfer lpi2c_slave_transfer_t
```

LPI2C slave transfer structure.

```
typedef struct _lpi2c_slave_handle lpi2c_slave_handle_t
```

LPI2C slave handle structure.

```
typedef void (*lpi2c_slave_transfer_callback_t)(LPI2C_Type *base, lpi2c_slave_transfer_t  
*transfer, void *userData)
```

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the LPI2C\_SlaveSetCallback() function after you have created a handle.

**Param base**

Base address for the LPI2C instance on which the event occurred.

**Param transfer**

Pointer to transfer descriptor containing values passed to and/or from the callback.

**Param userData**

Arbitrary pointer-sized value passed from the application.

```
struct _lpi2c_slave_config
```

*#include <fsl\_lpi2c.h>* Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the LPI2C\_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

**Public Members**

bool enableSlave

Enable slave mode.

uint8\_t address0

Slave's 7-bit address.

uint8\_t address1

Alternate slave 7-bit address.

*lpi2c\_slave\_address\_match\_t* addressMatchMode

Address matching options.

bool filterDozeEnable

Enable digital glitch filter in doze mode.

`bool filterEnable`  
Enable digital glitch filter.

`bool enableGeneralCall`  
Enable general call address matching.

`struct _lpi2c_slave_config sclStall`  
SCL stall enable options.

`bool ignoreAck`  
Continue transfers after a NACK is detected.

`bool enableReceivedAddressRead`  
Enable reading the address received address as the first byte of data.

`uint32_t sdaGlitchFilterWidth_ns`  
Width in nanoseconds of the digital filter on the SDA signal. Set to 0 to disable.

`uint32_t sclGlitchFilterWidth_ns`  
Width in nanoseconds of the digital filter on the SCL signal. Set to 0 to disable.

`uint32_t dataValidDelay_ns`  
Width in nanoseconds of the data valid delay.

`uint32_t clockHoldTime_ns`  
Width in nanoseconds of the clock hold time.

`struct _lpi2c_slave_transfer`  
*#include <fsl\_lpi2c.h>* LPI2C slave transfer structure.

**Public Members**

`lpi2c_slave_transfer_event_t event`  
Reason the callback is being invoked.

`uint8_t receivedAddress`  
Matching address send by master.

`uint8_t *data`  
Transfer buffer

`size_t dataSize`  
Transfer size

`status_t completionStatus`  
Success or error code describing how the transfer completed. Only applies for `kLPI2C_SlaveCompletionEvent`.

`size_t transferredCount`  
Number of bytes actually transferred since start or last repeated start.

`struct _lpi2c_slave_handle`  
*#include <fsl\_lpi2c.h>* LPI2C slave handle structure.

---

**Note:** The contents of this structure are private and subject to change.

---

**Public Members**

*lpi2c\_slave\_transfer\_t* transfer  
LPI2C slave transfer copy.

bool isBusy  
Whether transfer is busy.

bool wasTransmit  
Whether the last transfer was a transmit.

uint32\_t eventMask  
Mask of enabled events.

uint32\_t transferredCount  
Count of bytes transferred.

*lpi2c\_slave\_transfer\_callback\_t* callback  
Callback function called at transfer event.

void \*userData  
Callback parameter passed to callback.

struct sclStall

**Public Members**

bool enableAck  
Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted. Clock stretching occurs when transmitting the 9th bit. When enableAckSCLStall is enabled, there is no need to set either enableRxDatSCLStall or enableAddressSCLStall.

bool enableTx  
Enables SCL clock stretching when the transmit data flag is set during a slave-transmit transfer.

bool enableRx  
Enables SCL clock stretching when receive data flag is set during a slave-receive transfer.

bool enableAddress  
Enables SCL clock stretching when the address valid flag is asserted.

## 2.34 LPIT: Low-Power Interrupt Timer

void LPIT\_Init(LPIT\_Type \*base, const *lpit\_config\_t* \*config)  
Ungates the LPIT clock and configures the peripheral for a basic operation.  
This function issues a software reset to reset all channels and registers except the Module Control register.

---

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

---

**Parameters**



- `base` – LPIT peripheral base address.
- `config` – Pointer to the user configuration structure.

`void LPIT_Deinit(LPIT_Type *base)`

Disables the module and gates the LPIT clock.

#### Parameters

- `base` – LPIT peripheral base address.

`void LPIT_GetDefaultConfig(lpit_config_t *config)`

Fills in the LPIT configuration structure with default settings.

The default values are:

```
config->enableRunInDebug = false;
config->enableRunInDoze = false;
```

#### Parameters

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

`status_t LPIT_SetupChannel(LPIT_Type *base, lpit_chnl_t channel, const lpit_chnl_params_t *chnlSetup)`

Sets up an LPIT channel based on the user's preference.

This function sets up the operation mode to one of the options available in the enumeration `lpit_timer_modes_t`. It sets the trigger source as either internal or external, trigger selection and the timers behaviour when a timeout occurs. It also chains the timer if a prior timer if requested by the user.

#### Parameters

- `base` – LPIT peripheral base address.
- `channel` – Channel that is being configured.
- `chnlSetup` – Configuration parameters.

`static inline void LPIT_EnableInterrupts(LPIT_Type *base, uint32_t mask)`

Enables the selected PIT interrupts.

#### Parameters

- `base` – LPIT peripheral base address.
- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `lpit_interrupt_enable_t`

`static inline void LPIT_DisableInterrupts(LPIT_Type *base, uint32_t mask)`

Disables the selected PIT interrupts.

#### Parameters

- `base` – LPIT peripheral base address.
- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `lpit_interrupt_enable_t`

`static inline uint32_t LPIT_GetEnabledInterrupts(LPIT_Type *base)`

Gets the enabled LPIT interrupts.

#### Parameters

- `base` – LPIT peripheral base address.

**Returns**

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

```
static inline uint32_t LPIT_GetStatusFlags(LPIT_Type *base)
```

Gets the LPIT status flags.

**Parameters**

- `base` – LPIT peripheral base address.

**Returns**

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

```
static inline void LPIT_ClearStatusFlags(LPIT_Type *base, uint32_t mask)
```

Clears the LPIT status flags.

**Parameters**

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

```
static inline void LPIT_SetTimerPeriod(LPIT_Type *base, lpit_chnl_t channel, uint32_t ticks)
```

Sets the timer period in units of count.

Timers begin counting down from the value set by this function until it reaches 0, at which point it generates an interrupt and loads this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

---

**Note:** User can call the utility macros provided in `fsl_common.h` to convert to ticks.

---

**Parameters**

- `base` – LPIT peripheral base address.
- `channel` – Timer channel number.
- `ticks` – Timer period in units of ticks.

```
static inline void LPIT_SetTimerValue(LPIT_Type *base, lpit_chnl_t channel, uint32_t ticks)
```

Sets the timer period in units of count.

In the Dual 16-bit Periodic Counter mode, the counter will load and then the lower 16-bits will decrement down to zero, which will assert the output pre-trigger. The upper 16-bits will then decrement down to zero, which will negate the output pre-trigger and set the timer interrupt flag.

---

**Note:** Set TVAL register to 0 or 1 is invalid in compare mode.

---

**Parameters**

- `base` – LPIT peripheral base address.
- `channel` – Timer channel number.
- `ticks` – Timer period in units of ticks.

static inline uint32\_t LPIT\_GetCurrentTimerCount(LPIT\_Type \*base, *lpit\_chnl\_t* channel)

Reads the current timer counting value.

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

---

**Note:** User can call the utility macros provided in `fsl_common.h` to convert ticks to microseconds or milliseconds.

---

#### Parameters

- `base` – LPIT peripheral base address.
- `channel` – Timer channel number.

#### Returns

Current timer counting value in ticks.

static inline void LPIT\_StartTimer(LPIT\_Type \*base, *lpit\_chnl\_t* channel)

Starts the timer counting.

After calling this function, timers load the period value and count down to 0. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

#### Parameters

- `base` – LPIT peripheral base address.
- `channel` – Timer channel number.

static inline void LPIT\_StopTimer(LPIT\_Type \*base, *lpit\_chnl\_t* channel)

Stops the timer counting.

#### Parameters

- `base` – LPIT peripheral base address.
- `channel` – Timer channel number.

FSL\_LPIT\_DRIVER\_VERSION

Version 2.1.3

enum *\_lpit\_chnl*

List of LPIT channels.

---

**Note:** Actual number of available channels is SoC-dependent

---

#### Values:

enumerator `kLPIT_Chnl_0`  
LPIT channel number 0

enumerator `kLPIT_Chnl_1`  
LPIT channel number 1

enumerator `kLPIT_Chnl_2`  
LPIT channel number 2

enumerator `kLPIT_Chnl_3`  
LPIT channel number 3

enum *\_lpit\_timer\_modes*

Mode options available for the LPIT timer.

#### Values:

enumerator kLPIT\_PeriodicCounter

Use the all 32-bits, counter loads and decrements to zero

enumerator kLPIT\_DualPeriodicCounter

Counter loads, lower 16-bits decrement to zero, then upper 16-bits decrement

enumerator kLPIT\_TriggerAccumulator

Counter loads on first trigger and decrements on each trigger

enumerator kLPIT\_InputCapture

Counter loads with 0xFFFFFFFF, decrements to zero. It stores the inverse of the current value when a input trigger is detected

enum \_lpit\_trigger\_select

Trigger options available.

This is used for both internal and external trigger sources. The actual trigger options available is SoC-specific, user should refer to the reference manual.

*Values:*

enumerator kLPIT\_Trigger\_TimerChn0

Channel 0 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn1

Channel 1 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn2

Channel 2 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn3

Channel 3 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn4

Channel 4 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn5

Channel 5 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn6

Channel 6 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn7

Channel 7 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn8

Channel 8 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn9

Channel 9 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn10

Channel 10 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn11

Channel 11 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn12

Channel 12 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn13

Channel 13 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn14  
Channel 14 is selected as a trigger source

enumerator kLPIT\_Trigger\_TimerChn15  
Channel 15 is selected as a trigger source

enum \_lpit\_trigger\_source  
Trigger source options available.

*Values:*

enumerator kLPIT\_TriggerSource\_External  
Use external trigger input

enumerator kLPIT\_TriggerSource\_Internal  
Use internal trigger

enum \_lpit\_interrupt\_enable  
List of LPIT interrupts.

---

**Note:** Number of timer channels are SoC-specific. See the SoC Reference Manual.

---

*Values:*

enumerator kLPIT\_Channel0TimerInterruptEnable  
Channel 0 Timer interrupt

enumerator kLPIT\_Channel1TimerInterruptEnable  
Channel 1 Timer interrupt

enumerator kLPIT\_Channel2TimerInterruptEnable  
Channel 2 Timer interrupt

enumerator kLPIT\_Channel3TimerInterruptEnable  
Channel 3 Timer interrupt

enum \_lpit\_status\_flags  
List of LPIT status flags.

---

**Note:** Number of timer channels are SoC-specific. See the SoC Reference Manual.

---

*Values:*

enumerator kLPIT\_Channel0TimerFlag  
Channel 0 Timer interrupt flag

enumerator kLPIT\_Channel1TimerFlag  
Channel 1 Timer interrupt flag

enumerator kLPIT\_Channel2TimerFlag  
Channel 2 Timer interrupt flag

enumerator kLPIT\_Channel3TimerFlag  
Channel 3 Timer interrupt flag

typedef enum \_lpit\_chnl lpit\_chnl\_t  
List of LPIT channels.

---

**Note:** Actual number of available channels is SoC-dependent

---

typedef enum *\_lpit\_timer\_modes* lpit\_timer\_modes\_t

Mode options available for the LPIT timer.

typedef enum *\_lpit\_trigger\_select* lpit\_trigger\_select\_t

Trigger options available.

This is used for both internal and external trigger sources. The actual trigger options available is SoC-specific, user should refer to the reference manual.

typedef enum *\_lpit\_trigger\_source* lpit\_trigger\_source\_t

Trigger source options available.

typedef enum *\_lpit\_interrupt\_enable* lpit\_interrupt\_enable\_t

List of LPIT interrupts.

---

**Note:** Number of timer channels are SoC-specific. See the SoC Reference Manual.

---

typedef enum *\_lpit\_status\_flags* lpit\_status\_flags\_t

List of LPIT status flags.

---

**Note:** Number of timer channels are SoC-specific. See the SoC Reference Manual.

---

typedef struct *\_lpit\_chnl\_params* lpit\_chnl\_params\_t

Structure to configure the channel timer.

typedef struct *\_lpit\_config* lpit\_config\_t

LPIT configuration structure.

This structure holds the configuration settings for the LPIT peripheral. To initialize this structure to reasonable defaults, call the LPIT\_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

static void LPIT\_ResetStateDelay(void)

Short wait for LPIT state reset.

After clear or set LPIT\_EN, there should be delay longer than 4 LPIT functional clock.

static inline void LPIT\_Reset(LPIT\_Type \*base)

Performs a software reset on the LPIT module.

This resets all channels and registers except the Module Control Register.

#### Parameters

- base – LPIT peripheral base address.

LPIT\_RESET\_STATE\_DELAY

Delay used in LPIT\_Reset.

The macro value should be larger than 4 \* core clock / LPIT peripheral clock.

struct *\_lpit\_chnl\_params*

*#include <fsl\_lpit.h>* Structure to configure the channel timer.

#### Public Members

bool chainChannel

true: Timer chained to previous timer; false: Timer not chained

*lpit\_timer\_modes\_t* timerMode

Timers mode of operation.

*lpit\_trigger\_select\_t* triggerSelect

Trigger selection for the timer

*lpit\_trigger\_source\_t* triggerSource

Decides if we use external or internal trigger.

bool enableReloadOnTrigger

true: Timer reloads when a trigger is detected; false: No effect

bool enableStopOnTimeout

true: Timer will stop after timeout; false: does not stop after timeout

bool enableStartOnTrigger

true: Timer starts when a trigger is detected; false: decrement immediately

struct *\_lpit\_config*

*#include <fsl\_lpit.h>* LPIT configuration structure.

This structure holds the configuration settings for the LPIT peripheral. To initialize this structure to reasonable defaults, call the `LPIT_GetDefaultConfig()` function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

### Public Members

bool enableRunInDebug

true: Timers run in debug mode; false: Timers stop in debug mode

bool enableRunInDoze

true: Timers run in doze mode; false: Timers stop in doze mode

## 2.35 LPSPI: Low Power Serial Peripheral Interface

### 2.36 LPSPI Peripheral driver

void LPSPI\_MasterInit(LPSPI\_Type \*base, const *lpspi\_master\_config\_t* \*masterConfig, uint32\_t srcClock\_Hz)

Initializes the LPSPI master.

#### Parameters

- base – LPSPI peripheral address.
- masterConfig – Pointer to structure *lpspi\_master\_config\_t*.
- srcClock\_Hz – Module source input clock in Hertz

void LPSPI\_MasterGetDefaultConfig(*lpspi\_master\_config\_t* \*masterConfig)

Sets the *lpspi\_master\_config\_t* structure to default values.

This API initializes the configuration structure for `LPSPI_MasterInit()`. The initialized structure can remain unchanged in `LPSPI_MasterInit()`, or can be modified before calling the `LPSPI_MasterInit()`. Example:

```
lpspi_master_config_t masterConfig;  
LPSPI_MasterGetDefaultConfig(&masterConfig);
```

### Parameters

- masterConfig – pointer to lpspi\_master\_config\_t structure

void LPSPI\_SlaveInit(LPSPI\_Type \*base, const *lpspi\_slave\_config\_t* \*slaveConfig)  
LPSPI slave configuration.

### Parameters

- base – LPSPI peripheral address.
- slaveConfig – Pointer to a structure lpspi\_slave\_config\_t.

void LPSPI\_SlaveGetDefaultConfig(*lpspi\_slave\_config\_t* \*slaveConfig)  
Sets the lpspi\_slave\_config\_t structure to default values.

This API initializes the configuration structure for LPSPI\_SlaveInit(). The initialized structure can remain unchanged in LPSPI\_SlaveInit() or can be modified before calling the LPSPI\_SlaveInit(). Example:

```
lpspi_slave_config_t slaveConfig;  
LPSPI_SlaveGetDefaultConfig(&slaveConfig);
```

### Parameters

- slaveConfig – pointer to lpspi\_slave\_config\_t structure.

void LPSPI\_Deinit(LPSPI\_Type \*base)  
De-initializes the LPSPI peripheral. Call this API to disable the LPSPI clock.

### Parameters

- base – LPSPI peripheral address.

void LPSPI\_Reset(LPSPI\_Type \*base)  
Restores the LPSPI peripheral to reset state. Note that this function sets all registers to reset state. As a result, the LPSPI module can't work after calling this API.

### Parameters

- base – LPSPI peripheral address.

uint32\_t LPSPI\_GetInstance(LPSPI\_Type \*base)  
Get the LPSPI instance from peripheral base address.

### Parameters

- base – LPSPI peripheral base address.

### Returns

LPSPI instance.

static inline void LPSPI\_Enable(LPSPI\_Type \*base, bool enable)  
Enables the LPSPI peripheral and sets the MCR MDIS to 0.

### Parameters

- base – LPSPI peripheral address.
- enable – Pass true to enable module, false to disable module.



```
static inline uint32_t LPSPI_GetStatusFlags(LPSPI_Type *base)
```

Gets the LPSPI status flag state.

#### Parameters

- base – LPSPI peripheral address.

#### Returns

The LPSPI status(in SR register).

```
static inline uint8_t LPSPI_GetTxFifoSize(LPSPI_Type *base)
```

Gets the LPSPI Tx FIFO size.

#### Parameters

- base – LPSPI peripheral address.

#### Returns

The LPSPI Tx FIFO size.

```
static inline uint8_t LPSPI_GetRxFifoSize(LPSPI_Type *base)
```

Gets the LPSPI Rx FIFO size.

#### Parameters

- base – LPSPI peripheral address.

#### Returns

The LPSPI Rx FIFO size.

```
static inline uint32_t LPSPI_GetTxFifoCount(LPSPI_Type *base)
```

Gets the LPSPI Tx FIFO count.

#### Parameters

- base – LPSPI peripheral address.

#### Returns

The number of words in the transmit FIFO.

```
static inline uint32_t LPSPI_GetRxFifoCount(LPSPI_Type *base)
```

Gets the LPSPI Rx FIFO count.

#### Parameters

- base – LPSPI peripheral address.

#### Returns

The number of words in the receive FIFO.

```
static inline void LPSPI_ClearStatusFlags(LPSPI_Type *base, uint32_t statusFlags)
```

Clears the LPSPI status flag.

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status flag bit to clear. The list of status flags is defined in the `_lpspi_flags`. Example usage:

```
LPSPI_ClearStatusFlags(base, kLPSPI_TxDataRequestFlag|kLPSPI_RxDataReadyFlag);
```

#### Parameters

- base – LPSPI peripheral address.
- statusFlags – The status flag used from type `_lpspi_flags`.

```
static inline uint32_t LPSPI_GetTcr(LPSPI_Type *base)
```

```
static inline void LPSPI_EnableInterrupts(LPSPI_Type *base, uint32_t mask)
```

Enables the LPSPI interrupts.

This function configures the various interrupt masks of the LPSPI. The parameters are base and an interrupt mask. Note that, for Tx fill and Rx FIFO drain requests, enabling the interrupt request disables the DMA request.

```
LPSPI_EnableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

#### Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_interrupt_enable`.

```
static inline void LPSPI_DisableInterrupts(LPSPI_Type *base, uint32_t mask)
```

Disables the LPSPI interrupts.

```
LPSPI_DisableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

#### Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_interrupt_enable`.

```
static inline void LPSPI_EnableDMA(LPSPI_Type *base, uint32_t mask)
```

Enables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
LPSPI_EnableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

#### Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_dma_enable`.

```
static inline void LPSPI_DisableDMA(LPSPI_Type *base, uint32_t mask)
```

Disables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
SPI_DisableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

#### Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_dma_enable`.

```
static inline uint32_t LPSPI_GetTxRegisterAddress(LPSPI_Type *base)
```

Gets the LPSPI Transmit Data Register address for a DMA operation.

This function gets the LPSPI Transmit Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

#### Parameters

- base – LPSPI peripheral address.

#### Returns

The LPSPI Transmit Data Register address.

```
static inline uint32_t LPSPI_GetRxRegisterAddress(LPSPI_Type *base)
```

Gets the LPSPI Receive Data Register address for a DMA operation.

This function gets the LPSPI Receive Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

**Parameters**

- base – LPSPI peripheral address.

**Returns**

The LPSPI Receive Data Register address.

```
bool LPSPI_CheckTransferArgument(LPSPI_Type *base, lpspi_transfer_t *transfer, bool isEdma)
```

Check the argument for transfer .

**Parameters**

- base – LPSPI peripheral address.
- transfer – the transfer struct to be used.
- isEdma – True to check for EDMA transfer, false to check interrupt non-blocking transfer

**Returns**

Return true for right and false for wrong.

```
static inline void LPSPI_SetMasterSlaveMode(LPSPI_Type *base, lpspi_master_slave_mode_t mode)
```

Configures the LPSPI for either master or slave.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx\_CR\_MEN = 0).

**Parameters**

- base – LPSPI peripheral address.
- mode – Mode setting (master or slave) of type lpspi\_master\_slave\_mode\_t.

```
static inline void LPSPI_SelectTransferPCS(LPSPI_Type *base, lpspi_which_pcs_t select)
```

Configures the peripheral chip select used for the transfer.

**Parameters**

- base – LPSPI peripheral address.
- select – LPSPI Peripheral Chip Select (PCS) configuration.

```
static inline void LPSPI_SetPCSContinuous(LPSPI_Type *base, bool IsContinuous)
```

Set the PCS signal to continuous or uncontinuous mode.

---

**Note:** In master mode, continuous transfer will keep the PCS asserted at the end of the frame size, until a command word is received that starts a new frame. So PCS must be set back to uncontinuous when transfer finishes. In slave mode, when continuous transfer is enabled, the LPSPI will only transmit the first frame size bits, after that the LPSPI will transmit received data back (assuming a 32-bit shift register).

---

**Parameters**

- base – LPSPI peripheral address.
- IsContinuous – True to set the transfer PCS to continuous mode, false to set to uncontinuous mode.

```
static inline bool LPSPI_IsMaster(LPSPI_Type *base)
```

Returns whether the LPSPI module is in master mode.

#### Parameters

- base – LPSPI peripheral address.

#### Returns

Returns true if the module is in master mode or false if the module is in slave mode.

```
static inline void LPSPI_FlushFifo(LPSPI_Type *base, bool flushTxFifo, bool flushRxFifo)
```

Flushes the LPSPI FIFOs.

#### Parameters

- base – LPSPI peripheral address.
- flushTxFifo – Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO.
- flushRxFifo – Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO.

```
static inline void LPSPI_SetFifoWatermarks(LPSPI_Type *base, uint32_t txWater, uint32_t rxWater)
```

Sets the transmit and receive FIFO watermark values.

This function allows the user to set the receive and transmit FIFO watermarks. The function does not compare the watermark settings to the FIFO size. The FIFO watermark should not be equal to or greater than the FIFO size. It is up to the higher level driver to make this check.

#### Parameters

- base – LPSPI peripheral address.
- txWater – The TX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.
- rxWater – The RX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.

```
static inline void LPSPI_SetAllPcsPolarity(LPSPI_Type *base, uint32_t mask)
```

Configures all LPSPI peripheral chip select polarities simultaneously.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx\_CR\_MEN = 0).

This is an example: PCS0 and PCS1 set to active low and other PCSs set to active high. Note that the number of PCS is device-specific.

```
LPSPI_SetAllPcsPolarity(base, kLPSPI_Pcs0ActiveLow | kLPSPI_Pcs1ActiveLow);
```

#### Parameters

- base – LPSPI peripheral address.
- mask – The PCS polarity mask; Use the enum `_lpspi_pcs_polarity`.

```
static inline void LPSPI_SetFrameSize(LPSPI_Type *base, uint32_t frameSize)
```

Configures the frame size.

The minimum frame size is 8-bits and the maximum frame size is 4096-bits. If the frame size is less than or equal to 32-bits, the word size and frame size are identical. If the frame size is greater than 32-bits, the word size is 32-bits for each word except the last (the last word contains the remainder bits if the frame size is not divisible by 32). The minimum word size is 2-bits. A frame size of 33-bits (or similar) is not supported.

Note 1: The transmit command register should be initialized before enabling the LPSPI in slave mode, although the command register does not update until after the LPSPI is enabled. After it is enabled, the transmit command register should only be changed if the LPSPI is idle.

Note 2: The transmit and command FIFO is a combined FIFO that includes both transmit data and command words. That means the TCR register should be written to when the Tx FIFO is not full.

### Parameters

- `base` – LPSPI peripheral address.
- `frameSize` – The frame size in number of bits.

```
uint32_t LPSPI_MasterSetBaudRate(LPSPI_Type *base, uint32_t baudRate_Bps, uint32_t
                                srcClock_Hz, uint32_t *tcrPrescaleValue)
```

Sets the LPSPI baud rate in bits per second.

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate and returns the calculated baud rate in bits-per-second. It requires the caller to provide the frequency of the module source clock (in Hertz). Note that the baud rate does not go into effect until the Transmit Control Register (TCR) is programmed with the prescale value. Hence, this function returns the prescale `tcrPrescaleValue` parameter for later programming in the TCR. The higher level peripheral driver should alert the user of an out of range baud rate input.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

### Parameters

- `base` – LPSPI peripheral address.
- `baudRate_Bps` – The desired baud rate in bits per second.
- `srcClock_Hz` – Module source input clock in Hertz.
- `tcrPrescaleValue` – The TCR prescale value needed to program the TCR.

### Returns

The actual calculated baud rate. This function may also return a “0” if the LPSPI is not configured for master mode or if the LPSPI module is not disabled.

```
void LPSPI_MasterSetDelayScaler(LPSPI_Type *base, uint32_t scaler, lpspi_delay_type_t
                                whichDelay)
```

Manually configures a specific LPSPI delay parameter (module must be disabled to change the delay values).

This function configures the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type `lpspi_delay_type_t`.

The user passes the desired delay along with the delay value. This allows the user to directly set the delay values if they have pre-calculated them or if they simply wish to manually increment the value.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

### Parameters

- `base` – LPSPI peripheral address.
- `scaler` – The 8-bit delay value 0x00 to 0xFF (255).

- `whichDelay` – The desired delay to configure, must be of type `lpspi_delay_type_t`.

`uint32_t LPSPI_MasterSetDelayTimes(LPSPI_Type *base, uint32_t delayTimeInNanoSec, lpspi_delay_type_t whichDelay, uint32_t srcClock_Hz)`

Calculates the delay based on the desired delay input in nanoseconds (module must be disabled to change the delay values).

This function calculates the values for the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type `lpspi_delay_type_t`.

The user passes the desired delay and the desired delay value in nano-seconds. The function calculates the value needed for the desired delay parameter and returns the actual calculated delay because an exact delay match may not be possible. In this case, the closest match is calculated without going below the desired delay value input. It is possible to input a very large delay value that exceeds the capability of the part, in which case the maximum supported delay is returned. It is up to the higher level peripheral driver to alert the user of an out of range delay input.

Note that the LPSPI module must be configured for master mode before configuring this. And note that the `delayTime = LPSPI_clockSource / (PRESCALE * Delay_scaler)`.

#### Parameters

- `base` – LPSPI peripheral address.
- `delayTimeInNanoSec` – The desired delay value in nano-seconds.
- `whichDelay` – The desired delay to configuration, which must be of type `lpspi_delay_type_t`.
- `srcClock_Hz` – Module source input clock in Hertz.

#### Returns

actual Calculated delay value in nano-seconds.

`static inline void LPSPI_WriteData(LPSPI_Type *base, uint32_t data)`

Writes data into the transmit data buffer.

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user has to manage sending the data one 32-bit word at a time. Any writes to the TDR result in an immediate push to the transmit FIFO. This function can be used for either master or slave modes.

#### Parameters

- `base` – LPSPI peripheral address.
- `data` – The data word to be sent.

`static inline uint32_t LPSPI_ReadData(LPSPI_Type *base)`

Reads data from the data buffer.

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

#### Parameters

- `base` – LPSPI peripheral address.

#### Returns

The data read from the data buffer.

```
void LPSPI_SetDummyData(LPSPI_Type *base, uint8_t dummyData)
```

Set up the dummy data.

#### Parameters

- *base* – LPSPI peripheral address.
- *dummyData* – Data to be transferred when tx buffer is NULL. Note: This API has no effect when LPSPI in slave interrupt mode, because driver will set the TXMSK bit to 1 if txData is NULL, no data is loaded from transmit FIFO and output pin is tristated.

```
void LPSPI_MasterTransferCreateHandle(LPSPI_Type *base, lpspi_master_handle_t *handle,
                                     lpspi_master_transfer_callback_t callback, void
                                     *userData)
```

Initializes the LPSPI master handle.

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

#### Parameters

- *base* – LPSPI peripheral address.
- *handle* – LPSPI handle pointer to *lpspi\_master\_handle\_t*.
- *callback* – DSPI callback.
- *userData* – callback function parameter.

```
status_t LPSPI_MasterTransferBlocking(LPSPI_Type *base, lpspi_transfer_t *transfer)
```

LPSPI master transfer data using a polling method.

This function transfers data using a polling method. This is a blocking function, which does not return until all transfers have been completed.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

#### Parameters

- *base* – LPSPI peripheral address.
- *transfer* – pointer to *lpspi\_transfer\_t* structure.

#### Returns

status of *status\_t*.

```
status_t LPSPI_MasterTransferNonBlocking(LPSPI_Type *base, lpspi_master_handle_t *handle,
                                         lpspi_transfer_t *transfer)
```

LPSPI master transfer data using an interrupt method.

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

#### Parameters

- *base* – LPSPI peripheral address.
- *handle* – pointer to *lpspi\_master\_handle\_t* structure which stores the transfer state.

- transfer – pointer to `lpspi_transfer_t` structure.

**Returns**

status of `status_t`.

`status_t` LPSPI\_MasterTransferGetCount(LPSPI\_Type \*base, *lpspi\_master\_handle\_t* \*handle, size\_t \*count)

Gets the master transfer remaining bytes.

This function gets the master transfer remaining bytes.

**Parameters**

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_master_handle_t` structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

**Returns**

status of `status_t`.

void LPSPI\_MasterTransferAbort(LPSPI\_Type \*base, *lpspi\_master\_handle\_t* \*handle)

LPSPI master abort transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

**Parameters**

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_master_handle_t` structure which stores the transfer state.

void LPSPI\_MasterTransferHandleIRQ(LPSPI\_Type \*base, *lpspi\_master\_handle\_t* \*handle)

LPSPI Master IRQ handler function.

This function processes the LPSPI transmit and receive IRQ.

**Parameters**

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_master_handle_t` structure which stores the transfer state.

void LPSPI\_SlaveTransferCreateHandle(LPSPI\_Type \*base, *lpspi\_slave\_handle\_t* \*handle, *lpspi\_slave\_transfer\_callback\_t* callback, void \*userData)

Initializes the LPSPI slave handle.

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

**Parameters**

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to `lpspi_slave_handle_t`.
- callback – DSPI callback.
- userData – callback function parameter.

`status_t` LPSPI\_SlaveTransferNonBlocking(LPSPI\_Type \*base, *lpspi\_slave\_handle\_t* \*handle, *lpspi\_transfer\_t* \*transfer)

LPSPI slave transfer data using an interrupt method.

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.



Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

#### Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.
- transfer – pointer to `lpspi_transfer_t` structure.

#### Returns

status of `status_t`.

`status_t` LPSPI\_SlaveTransferGetCount(LPSPI\_Type \*base, *lpspi\_slave\_handle\_t* \*handle, size\_t \*count)

Gets the slave transfer remaining bytes.

This function gets the slave transfer remaining bytes.

#### Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

#### Returns

status of `status_t`.

void LPSPI\_SlaveTransferAbort(LPSPI\_Type \*base, *lpspi\_slave\_handle\_t* \*handle)

LPSPI slave aborts a transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

#### Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.

void LPSPI\_SlaveTransferHandleIRQ(LPSPI\_Type \*base, *lpspi\_slave\_handle\_t* \*handle)

LPSPI Slave IRQ handler function.

This function processes the LPSPI transmit and receives an IRQ.

#### Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.

bool LPSPI\_WaitTxFifoEmpty(LPSPI\_Type \*base)

Wait for tx FIFO to be empty.

This function wait the tx fifo empty

#### Parameters

- base – LPSPI peripheral address.

#### Returns

true for the tx FIFO is ready, false is not.

void LPSPI\_DriverIRQHandler(uint32\_t instance)

LPSPI driver IRQ handler common entry.

This function provides the common IRQ request entry for LPSPI.

#### Parameters

- instance – LPSPI instance.

FSL\_LPSPI\_DRIVER\_VERSION

LPSPI driver version.

Status for the LPSPI driver.

*Values:*

enumerator kStatus\_LPSPI\_Busy

LPSPI transfer is busy.

enumerator kStatus\_LPSPI\_Error

LPSPI driver error.

enumerator kStatus\_LPSPI\_Idle

LPSPI is idle.

enumerator kStatus\_LPSPI\_OutOfRange

LPSPI transfer out Of range.

enumerator kStatus\_LPSPI\_Timeout

LPSPI timeout polling status flags.

enum \_lpspi\_flags

LPSPI status flags in SPIx\_SR register.

*Values:*

enumerator kLPSPI\_TxDataRequestFlag

Transmit data flag

enumerator kLPSPI\_RxDataReadyFlag

Receive data flag

enumerator kLPSPI\_WordCompleteFlag

Word Complete flag

enumerator kLPSPI\_FrameCompleteFlag

Frame Complete flag

enumerator kLPSPI\_TransferCompleteFlag

Transfer Complete flag

enumerator kLPSPI\_TransmitErrorFlag

Transmit Error flag (FIFO underrun)

enumerator kLPSPI\_ReceiveErrorFlag

Receive Error flag (FIFO overrun)

enumerator kLPSPI\_DataMatchFlag

Data Match flag

enumerator kLPSPI\_ModuleBusyFlag

Module Busy flag

enumerator kLPSPi\_AllStatusFlag

Used for clearing all w1c status flags

enum \_lpspi\_interrupt\_enable

LPSPi interrupt source.

*Values:*

enumerator kLPSPi\_TxInterruptEnable

Transmit data interrupt enable

enumerator kLPSPi\_RxInterruptEnable

Receive data interrupt enable

enumerator kLPSPi\_WordCompleteInterruptEnable

Word complete interrupt enable

enumerator kLPSPi\_FrameCompleteInterruptEnable

Frame complete interrupt enable

enumerator kLPSPi\_TransferCompleteInterruptEnable

Transfer complete interrupt enable

enumerator kLPSPi\_TransmitErrorInterruptEnable

Transmit error interrupt enable(FIFO underrun)

enumerator kLPSPi\_ReceiveErrorInterruptEnable

Receive Error interrupt enable (FIFO overrun)

enumerator kLPSPi\_DataMatchInterruptEnable

Data Match interrupt enable

enumerator kLPSPi\_AllInterruptEnable

All above interrupts enable.

enum \_lpspi\_dma\_enable

LPSPi DMA source.

*Values:*

enumerator kLPSPi\_TxDmaEnable

Transmit data DMA enable

enumerator kLPSPi\_RxDmaEnable

Receive data DMA enable

enum \_lpspi\_master\_slave\_mode

LPSPi master or slave mode configuration.

*Values:*

enumerator kLPSPi\_Master

LPSPi peripheral operates in master mode.

enumerator kLPSPi\_Slave

LPSPi peripheral operates in slave mode.

enum \_lpspi\_which\_pcs\_config

LPSPi Peripheral Chip Select (PCS) configuration (which PCS to configure).

*Values:*

enumerator kLPSPi\_Pcs0

PCS[0]

enumerator kLPSPI\_Pcs1  
PCS[1]

enumerator kLPSPI\_Pcs2  
PCS[2]

enumerator kLPSPI\_Pcs3  
PCS[3]

enum \_\_lpspi\_pcs\_polarity\_config  
LPSPI Peripheral Chip Select (PCS) Polarity configuration.  
*Values:*

enumerator kLPSPI\_PcsActiveHigh  
PCS Active High (idles low)

enumerator kLPSPI\_PcsActiveLow  
PCS Active Low (idles high)

enum \_\_lpspi\_pcs\_polarity  
LPSPI Peripheral Chip Select (PCS) Polarity.  
*Values:*

enumerator kLPSPI\_Pcs0ActiveLow  
Pcs0 Active Low (idles high).

enumerator kLPSPI\_Pcs1ActiveLow  
Pcs1 Active Low (idles high).

enumerator kLPSPI\_Pcs2ActiveLow  
Pcs2 Active Low (idles high).

enumerator kLPSPI\_Pcs3ActiveLow  
Pcs3 Active Low (idles high).

enumerator kLPSPI\_PcsAllActiveLow  
Pcs0 to Pcs5 Active Low (idles high).

enum \_\_lpspi\_clock\_polarity  
LPSPI clock polarity configuration.  
*Values:*

enumerator kLPSPI\_ClockPolarityActiveHigh  
CPOL=0. Active-high LPSPI clock (idles low)

enumerator kLPSPI\_ClockPolarityActiveLow  
CPOL=1. Active-low LPSPI clock (idles high)

enum \_\_lpspi\_clock\_phase  
LPSPI clock phase configuration.  
*Values:*

enumerator kLPSPI\_ClockPhaseFirstEdge  
CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.

enumerator kLPSPI\_ClockPhaseSecondEdge  
CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

enum `_lpspi_shift_direction`

LPSPi data shifter direction options.

*Values:*

enumerator `kLPSPi_MsbFirst`

Data transfers start with most significant bit.

enumerator `kLPSPi_LsbFirst`

Data transfers start with least significant bit.

enum `_lpspi_host_request_select`

LPSPi Host Request select configuration.

*Values:*

enumerator `kLPSPi_HostReqExtPin`

Host Request is an ext pin.

enumerator `kLPSPi_HostReqInternalTrigger`

Host Request is an internal trigger.

enum `_lpspi_match_config`

LPSPi Match configuration options.

*Values:*

enumerator `kLPSPi_MatchDisabled`

LPSPi Match Disabled.

enumerator `kLPSPi_1stWordEqualsM0orM1`

LPSPi Match Enabled.

enumerator `kLPSPi_AnyWordEqualsM0orM1`

LPSPi Match Enabled.

enumerator `kLPSPi_1stWordEqualsM0and2ndWordEqualsM1`

LPSPi Match Enabled.

enumerator `kLPSPi_AnyWordEqualsM0andNxtWordEqualsM1`

LPSPi Match Enabled.

enumerator `kLPSPi_1stWordAndM1EqualsM0andM1`

LPSPi Match Enabled.

enumerator `kLPSPi_AnyWordAndM1EqualsM0andM1`

LPSPi Match Enabled.

enum `_lpspi_pin_config`

LPSPi pin (SDO and SDI) configuration.

*Values:*

enumerator `kLPSPi_SdiInSdoOut`

LPSPi SDI input, SDO output.

enumerator `kLPSPi_SdiInSdiOut`

LPSPi SDI input, SDI output.

enumerator `kLPSPi_SdoInSdoOut`

LPSPi SDO input, SDO output.

enumerator `kLPSPi_SdoInSdiOut`

LPSPi SDO input, SDI output.

enum `_lpspi_data_out_config`

LPSPi data output configuration.

*Values:*

enumerator `kLpspiDataOutRetained`

Data out retains last value when chip select is de-asserted

enumerator `kLpspiDataOutTristate`

Data out is tristated when chip select is de-asserted

enum `_lpspi_transfer_width`

LPSPi transfer width configuration.

*Values:*

enumerator `kLPSPi_SingleBitXfer`

1-bit shift at a time, data out on SDO, in on SDI (normal mode)

enumerator `kLPSPi_TwoBitXfer`

2-bits shift out on SDO/SDI and in on SDO/SDI

enumerator `kLPSPi_FourBitXfer`

4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

enum `_lpspi_delay_type`

LPSPi delay type selection.

*Values:*

enumerator `kLPSPi_PcsToSck`

PCS-to-SCK delay.

enumerator `kLPSPi_LastSckToPcs`

Last SCK edge to PCS delay.

enumerator `kLPSPi_BetweenTransfer`

Delay between transfers.

enum `_lpspi_transfer_config_flag_for_master`

Use this enumeration for LPSPi master transfer configFlags.

*Values:*

enumerator `kLPSPi_MasterPcs0`

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS0 signal

enumerator `kLPSPi_MasterPcs1`

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS1 signal

enumerator `kLPSPi_MasterPcs2`

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS2 signal

enumerator `kLPSPi_MasterPcs3`

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS3 signal

enumerator `kLPSPi_MasterPcsContinuous`

Is PCS signal continuous

enumerator `kLPSPi_MasterByteSwap`

Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set `lpspi_shift_direction_t` to MSB).

- i. If you set `bitPerFrame = 8` , no matter the `kLPSPi_MasterByteSwap` you flag is used or not, the waveform is 1 2 3 4 5 6 7 8.

- ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPi\_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi\_MasterByteSwap flag.
- iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPi\_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi\_MasterByteSwap flag.

enum \_lpspi\_transfer\_config\_flag\_for\_slave

Use this enumeration for LPSPi slave transfer configFlags.

*Values:*

enumerator kLPSPi\_SlavePcs0

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS0 signal

enumerator kLPSPi\_SlavePcs1

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS1 signal

enumerator kLPSPi\_SlavePcs2

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS2 signal

enumerator kLPSPi\_SlavePcs3

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS3 signal

enumerator kLPSPi\_SlaveByteSwap

Is slave swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi\_shift\_direction\_t to MSB).

- i. If you set bitPerFrame = 8 , no matter the kLPSPi\_SlaveByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
- ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPi\_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi\_SlaveByteSwap flag.
- iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPi\_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi\_SlaveByteSwap flag.

enum \_lpspi\_transfer\_state

LPSPi transfer state, which is used for LPSPi transactional API state machine.

*Values:*

enumerator kLPSPi\_Idle

Nothing in the transmitter/receiver.

enumerator kLPSPi\_Busy

Transfer queue is not finished.

enumerator kLPSPi\_Error

Transfer error.

typedef enum \_lpspi\_master\_slave\_mode lpspi\_master\_slave\_mode\_t

LPSPi master or slave mode configuration.

typedef enum \_lpspi\_which\_pcs\_config lpspi\_which\_pcs\_t

LPSPi Peripheral Chip Select (PCS) configuration (which PCS to configure).

typedef enum \_lpspi\_pcs\_polarity\_config lpspi\_pcs\_polarity\_config\_t

LPSPi Peripheral Chip Select (PCS) Polarity configuration.

typedef enum *\_lpspi\_clock\_polarity* lpspi\_clock\_polarity\_t

LPSPI clock polarity configuration.

typedef enum *\_lpspi\_clock\_phase* lpspi\_clock\_phase\_t

LPSPI clock phase configuration.

typedef enum *\_lpspi\_shift\_direction* lpspi\_shift\_direction\_t

LPSPI data shifter direction options.

typedef enum *\_lpspi\_host\_request\_select* lpspi\_host\_request\_select\_t

LPSPI Host Request select configuration.

typedef enum *\_lpspi\_match\_config* lpspi\_match\_config\_t

LPSPI Match configuration options.

typedef enum *\_lpspi\_pin\_config* lpspi\_pin\_config\_t

LPSPI pin (SDO and SDI) configuration.

typedef enum *\_lpspi\_data\_out\_config* lpspi\_data\_out\_config\_t

LPSPI data output configuration.

typedef enum *\_lpspi\_transfer\_width* lpspi\_transfer\_width\_t

LPSPI transfer width configuration.

typedef enum *\_lpspi\_delay\_type* lpspi\_delay\_type\_t

LPSPI delay type selection.

typedef struct *\_lpspi\_master\_config* lpspi\_master\_config\_t

LPSPI master configuration structure.

typedef struct *\_lpspi\_slave\_config* lpspi\_slave\_config\_t

LPSPI slave configuration structure.

typedef struct *\_lpspi\_master\_handle* lpspi\_master\_handle\_t

Forward declaration of the *\_lpspi\_master\_handle* typedefs.

typedef struct *\_lpspi\_slave\_handle* lpspi\_slave\_handle\_t

Forward declaration of the *\_lpspi\_slave\_handle* typedefs.

typedef void (\*lpspi\_master\_transfer\_callback\_t)(LPSPI\_Type \*base, *lpspi\_master\_handle\_t* \*handle, *status\_t* status, void \*userData)

Master completion callback function pointer type.

**Param base**

LPSPI peripheral address.

**Param handle**

Pointer to the handle for the LPSPI master.

**Param status**

Success or error code describing whether the transfer is completed.

**Param userData**

Arbitrary pointer-dataSized value passed from the application.

typedef void (\*lpspi\_slave\_transfer\_callback\_t)(LPSPI\_Type \*base, *lpspi\_slave\_handle\_t* \*handle, *status\_t* status, void \*userData)

Slave completion callback function pointer type.

**Param base**

LPSPI peripheral address.

**Param handle**

Pointer to the handle for the LPSPI slave.



**Param status**

Success or error code describing whether the transfer is completed.

**Param userData**

Arbitrary pointer-dataSized value passed from the application.

`typedef struct _lpspi_transfer lpspi_transfer_t`

LPSPI master/slave transfer structure.

`volatile uint8_t g_lpspiDummyData[]`

Global variable for dummy data value setting.

`LPSPI_DUMMY_DATA`

LPSPI dummy data if no Tx data.

Dummy data used for tx if there is not txData.

`SPI_RETRY_TIMES`

Retry times for waiting flag.

`LPSPI_MASTER_PCS_SHIFT`

LPSPI master PCS shift macro , internal used.

`LPSPI_MASTER_PCS_MASK`

LPSPI master PCS shift macro , internal used.

`LPSPI_SLAVE_PCS_SHIFT`

LPSPI slave PCS shift macro , internal used.

`LPSPI_SLAVE_PCS_MASK`

LPSPI slave PCS shift macro , internal used.

`struct _lpspi_master_config`

`#include <fsl_lpspi.h>` LPSPI master configuration structure.

**Public Members**

`uint32_t baudRate`

Baud Rate for LPSPI.

`uint32_t bitsPerFrame`

Bits per frame, minimum 8, maximum 4096.

`lpspi_clock_polarity_t cpol`

Clock polarity.

`lpspi_clock_phase_t cpha`

Clock phase.

`lpspi_shift_direction_t direction`

MSB or LSB data shift direction.

`uint32_t pcsToSckDelayInNanoSec`

PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

`uint32_t lastSckToPcsDelayInNanoSec`

Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32\_t betweenTransferDelayInNanoSec

After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

lpspi\_which\_pcs\_t whichPcs

Desired Peripheral Chip Select (PCS).

lpspi\_pcs\_polarity\_config\_t pcsActiveHighOrLow

Desired PCS active high or low

lpspi\_pin\_config\_t pinCfg

Configures which pins are used for input and output data during single bit transfers.

lpspi\_data\_out\_config\_t dataOutConfig

Configures if the output data is tristated between accesses (LPSPi\_PCS is negated).

bool enableInputDelay

Enable master to sample the input data on a delayed SCK. This can help improve slave setup time. Refer to device data sheet for specific time length.

struct \_\_lpspi\_slave\_config

#include <fsl\_lpspi.h> LPSPi slave configuration structure.

### Public Members

uint32\_t bitsPerFrame

Bits per frame, minimum 8, maximum 4096.

lpspi\_clock\_polarity\_t cpol

Clock polarity.

lpspi\_clock\_phase\_t cpha

Clock phase.

lpspi\_shift\_direction\_t direction

MSB or LSB data shift direction.

lpspi\_which\_pcs\_t whichPcs

Desired Peripheral Chip Select (pcs)

lpspi\_pcs\_polarity\_config\_t pcsActiveHighOrLow

Desired PCS active high or low

lpspi\_pin\_config\_t pinCfg

Configures which pins are used for input and output data during single bit transfers.

lpspi\_data\_out\_config\_t dataOutConfig

Configures if the output data is tristated between accesses (LPSPi\_PCS is negated).

struct \_\_lpspi\_transfer

#include <fsl\_lpspi.h> LPSPi master/slave transfer structure.

### Public Members

const uint8\_t \*txData

Send buffer.

uint8\_t \*rxData

Receive buffer.

volatile size\_t dataSize

Transfer bytes.

uint32\_t configFlags

Transfer transfer configuration flags. Set from `_lpspi_transfer_config_flag_for_master` if the transfer is used for master or `_lpspi_transfer_config_flag_for_slave` enumeration if the transfer is used for slave.

struct `_lpspi_master_handle`

*#include <fsl\_lpspi.h>* LPSPi master transfer handle structure used for transactional API.

### Public Members

volatile bool isPcsContinuous

Is PCS continuous in transfer.

volatile bool writeTcrInIsr

A flag that whether should write TCR in ISR.

volatile bool isByteSwap

A flag that whether should byte swap.

volatile bool isTxMask

A flag that whether TCR[TXMSK] is set.

volatile uint16\_t bytesPerFrame

Number of bytes in each frame

volatile uint16\_t frameSize

Backup of TCR[FRAMESZ]

volatile uint8\_t fifoSize

FIFO dataSize.

volatile uint8\_t rxWatermark

Rx watermark.

volatile uint8\_t bytesEachWrite

Bytes for each write TDR.

volatile uint8\_t bytesEachRead

Bytes for each read RDR.

const uint8\_t \*volatile txData

Send buffer.

uint8\_t \*volatile rxData

Receive buffer.

volatile size\_t txRemainingByteCount

Number of bytes remaining to send.

volatile size\_t rxRemainingByteCount

Number of bytes remaining to receive.

volatile uint32\_t writeRegRemainingTimes

Write TDR register remaining times.

volatile uint32\_t readRegRemainingTimes

Read RDR register remaining times.

uint32\_t totalByteCount

Number of transfer bytes

uint32\_t txBuffIfNull

Used if the txData is NULL.

volatile uint8\_t state

LPSPi transfer state , \_lpspi\_transfer\_state.

*lpspi\_master\_transfer\_callback\_t* callback

Completion callback.

void \*userData

Callback user data.

struct \_lpspi\_slave\_handle

*#include <fsl\_lpspi.h>* LPSPi slave transfer handle structure used for transactional API.

### Public Members

volatile bool isByteSwap

A flag that whether should byte swap.

volatile uint8\_t fifoSize

FIFO dataSize.

volatile uint8\_t rxWatermark

Rx watermark.

volatile uint8\_t bytesEachWrite

Bytes for each write TDR.

volatile uint8\_t bytesEachRead

Bytes for each read RDR.

const uint8\_t \*volatile txData

Send buffer.

uint8\_t \*volatile rxData

Receive buffer.

volatile size\_t txRemainingByteCount

Number of bytes remaining to send.

volatile size\_t rxRemainingByteCount

Number of bytes remaining to receive.

volatile uint32\_t writeRegRemainingTimes

Write TDR register remaining times.

volatile uint32\_t readRegRemainingTimes

Read RDR register remaining times.

uint32\_t totalByteCount

Number of transfer bytes

volatile uint8\_t state

LPSPi transfer state , \_lpspi\_transfer\_state.

volatile uint32\_t errorCount

Error count for slave transfer.

*lpspi\_slave\_transfer\_callback\_t* callback

Completion callback.

void \*userData

Callback user data.

## 2.37 LPSPI eDMA Driver

FSL\_LPSPI\_EDMA\_DRIVER\_VERSION

LPSPI EDMA driver version.

DMA\_MAX\_TRANSFER\_COUNT

DMA max transfer size.

typedef struct *lpspi\_master\_edma\_handle* lpspi\_master\_edma\_handle\_t

Forward declaration of the *lpspi\_master\_edma\_handle* typedefs.

typedef struct *lpspi\_slave\_edma\_handle* lpspi\_slave\_edma\_handle\_t

Forward declaration of the *lpspi\_slave\_edma\_handle* typedefs.

typedef void (\*lpspi\_master\_edma\_transfer\_callback\_t)(LPSPI\_Type \*base, *lpspi\_master\_edma\_handle\_t* \*handle, *status\_t* status, void \*userData)

Completion callback function pointer type.

### Param base

LPSPI peripheral base address.

### Param handle

Pointer to the handle for the LPSPI master.

### Param status

Success or error code describing whether the transfer completed.

### Param userData

Arbitrary pointer-dataSized value passed from the application.

typedef void (\*lpspi\_slave\_edma\_transfer\_callback\_t)(LPSPI\_Type \*base, *lpspi\_slave\_edma\_handle\_t* \*handle, *status\_t* status, void \*userData)

Completion callback function pointer type.

### Param base

LPSPI peripheral base address.

### Param handle

Pointer to the handle for the LPSPI slave.

### Param status

Success or error code describing whether the transfer completed.

### Param userData

Arbitrary pointer-dataSized value passed from the application.

void LPSPI\_MasterTransferCreateHandleEDMA(LPSPI\_Type \*base, *lpspi\_master\_edma\_handle\_t* \*handle, *lpspi\_master\_edma\_transfer\_callback\_t* callback, void \*userData, *edma\_handle\_t* \*edmaRxRegToRxDataHandle, *edma\_handle\_t* \*edmaTxDataToTxRegHandle)

Initializes the LPSPI master eDMA handle.

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

Note that the LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx are the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for `edmaRxRegToRxDataHandle` and Tx DMAMUX source for `edmaTxDataToTxRegHandle`. (2) For a shared DMA request source, enable and set the Rx/Tx DMAMUX source for `edmaRxRegToRxDataHandle`.

#### Parameters

- `base` – LPSPI peripheral base address.
- `handle` – LPSPI handle pointer to `lpspi_master_edma_handle_t`.
- `callback` – LPSPI callback.
- `userData` – callback function parameter.
- `edmaRxRegToRxDataHandle` – `edmaRxRegToRxDataHandle` pointer to `edma_handle_t`.
- `edmaTxDataToTxRegHandle` – `edmaTxDataToTxRegHandle` pointer to `edma_handle_t`.

*status\_t* LPSPI\_MasterTransferEDMA(LPSPI\_Type \*base, *lpspi\_master\_edma\_handle\_t* \*handle, *lpspi\_transfer\_t* \*transfer)

LPSPI master transfer data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of `bytesPerFrame` if `bytesPerFrame` is less than or equal to 4. For `bytesPerFrame` greater than 4: The transfer data size should be equal to `bytesPerFrame` if the `bytesPerFrame` is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of `bytesPerFrame`.

#### Parameters

- `base` – LPSPI peripheral base address.
- `handle` – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.
- `transfer` – pointer to `lpspi_transfer_t` structure.

#### Returns

status of `status_t`.

*status\_t* LPSPI\_MasterTransferPrepareEDMALite(LPSPI\_Type \*base, *lpspi\_master\_edma\_handle\_t* \*handle, `uint32_t` configFlags)

LPSPI master config transfer parameter while using eDMA.

This function is preparing to transfer data using eDMA, work with `LPSPI_MasterTransferEDMALite`.

#### Parameters

- `base` – LPSPI peripheral base address.
- `handle` – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.
- `configFlags` – transfer configuration flags. `_lpspi_transfer_config_flag_for_master`.

#### Return values

- `kStatus_Success` – Execution successfully.
- `kStatus_LPSPI_Busy` – The LPSPI device is busy.

**Returns**

Indicates whether LPSPI master transfer was successful or not.

*status\_t* LPSPI\_MasterTransferEDMALite(LPSPI\_Type \*base, *lpspi\_master\_edma\_handle\_t* \*handle, *lpspi\_transfer\_t* \*transfer)

LPSPI master transfer data using eDMA without configs.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: This API is only for transfer through DMA without configuration. Before calling this API, you must call LPSPI\_MasterTransferPrepareEDMALite to configure it once. The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

**Parameters**

- base – LPSPI peripheral base address.
- handle – pointer to *lpspi\_master\_edma\_handle\_t* structure which stores the transfer state.
- transfer – pointer to *lpspi\_transfer\_t* structure, config field is not used.

**Return values**

- kStatus\_Success – Execution successfully.
- kStatus\_LPSPI\_Busy – The LPSPI device is busy.
- kStatus\_InvalidArgument – The transfer structure is invalid.

**Returns**

Indicates whether LPSPI master transfer was successful or not.

*void* LPSPI\_MasterTransferAbortEDMA(LPSPI\_Type \*base, *lpspi\_master\_edma\_handle\_t* \*handle)

LPSPI master aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

**Parameters**

- base – LPSPI peripheral base address.
- handle – pointer to *lpspi\_master\_edma\_handle\_t* structure which stores the transfer state.

*status\_t* LPSPI\_MasterTransferGetCountEDMA(LPSPI\_Type \*base, *lpspi\_master\_edma\_handle\_t* \*handle, *size\_t* \*count)

Gets the master eDMA transfer remaining bytes.

This function gets the master eDMA transfer remaining bytes.

**Parameters**

- base – LPSPI peripheral base address.
- handle – pointer to *lpspi\_master\_edma\_handle\_t* structure which stores the transfer state.
- count – Number of bytes transferred so far by the EDMA transaction.

**Returns**

status of *status\_t*.

```
void LPSPI_SlaveTransferCreateHandleEDMA(LPSPI_Type *base, lpspi_slave_edma_handle_t
                                         *handle, lpspi_slave_edma_transfer_callback_t
                                         callback, void *userData, edma_handle_t
                                         *edmaRxRegToRxDataHandle, edma_handle_t
                                         *edmaTxDataToTxRegHandle)
```

Initializes the LPSPI slave eDMA handle.

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

Note that LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx as the same source) DMA request source.

(1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for edmaRxRegToRxDataHandle .

#### Parameters

- base – LPSPI peripheral base address.
- handle – LPSPI handle pointer to *lpspi\_slave\_edma\_handle\_t*.
- callback – LPSPI callback.
- userData – callback function parameter.
- edmaRxRegToRxDataHandle – edmaRxRegToRxDataHandle pointer to *edma\_handle\_t*.
- edmaTxDataToTxRegHandle – edmaTxDataToTxRegHandle pointer to *edma\_handle\_t*.

```
status_t LPSPI_SlaveTransferEDMA(LPSPI_Type *base, lpspi_slave_edma_handle_t *handle,
                                  lpspi_transfer_t *transfer)
```

LPSPI slave transfers data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which return right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

#### Parameters

- base – LPSPI peripheral base address.
- handle – pointer to *lpspi\_slave\_edma\_handle\_t* structure which stores the transfer state.
- transfer – pointer to *lpspi\_transfer\_t* structure.

#### Returns

status of *status\_t*.

```
void LPSPI_SlaveTransferAbortEDMA(LPSPI_Type *base, lpspi_slave_edma_handle_t *handle)
```

LPSPI slave aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

#### Parameters

- base – LPSPI peripheral base address.



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

`status_t` LPSPI\_SlaveTransferGetCountEDMA(LPSPI\_Type \*base, *lpspi\_slave\_edma\_handle\_t* \*handle, `size_t` \*count)

Gets the slave eDMA transfer remaining bytes.

This function gets the slave eDMA transfer remaining bytes.

#### Parameters

- `base` – LPSPI peripheral base address.
- `handle` – pointer to `lpspi_slave_edma_handle_t` structure which stores the transfer state.
- `count` – Number of bytes transferred so far by the eDMA transaction.

#### Returns

status of `status_t`.

`struct _lpspi_master_edma_handle`

*#include <fsl\_lpspi\_edma.h>* LPSPI master eDMA transfer handle structure used for transactional API.

#### Public Members

`volatile bool` `isPcsContinuous`

Is PCS continuous in transfer.

`volatile bool` `isByteSwap`

A flag that whether should byte swap.

`volatile uint8_t` `fifoSize`

FIFO dataSize.

`volatile uint8_t` `rxWatermark`

Rx watermark.

`volatile uint8_t` `bytesEachWrite`

Bytes for each write TDR.

`volatile uint8_t` `bytesEachRead`

Bytes for each read RDR.

`volatile uint8_t` `bytesLastRead`

Bytes for last read RDR.

`volatile bool` `isThereExtraRxBytes`

Is there extra RX byte.

`const uint8_t *volatile` `txData`

Send buffer.

`uint8_t *volatile` `rxData`

Receive buffer.

`volatile size_t` `txRemainingByteCount`

Number of bytes remaining to send.

`volatile size_t` `rxRemainingByteCount`

Number of bytes remaining to receive.

`volatile uint32_t writeRegRemainingTimes`  
Write TDR register remaining times.

`volatile uint32_t readRegRemainingTimes`  
Read RDR register remaining times.

`uint32_t totalByteCount`  
Number of transfer bytes

`edma_tcd_t *lastTimeTCD`  
Pointer to the lastTime TCD

`bool isMultiDMATransmit`  
Is there multi DMA transmit

`volatile uint8_t dmaTransmitTime`  
DMA Transfer times.

`uint32_t lastTimeDataBytes`  
DMA transmit last Time data Bytes

`uint32_t dataBytesEveryTime`  
Bytes in a time for DMA transfer, default is DMA\_MAX\_TRANSFER\_COUNT

`edma_transfer_config_t transferConfigRx`  
Config of DMA rx channel.

`edma_transfer_config_t transferConfigTx`  
Config of DMA tx channel.

`uint32_t txBuffIfNull`  
Used if there is not txData for DMA purpose.

`uint32_t rxBuffIfNull`  
Used if there is not rxData for DMA purpose.

`uint32_t transmitCommand`  
Used to write TCR for DMA purpose.

`volatile uint8_t state`  
LPSPI transfer state , `_lpspi_transfer_state`.

`uint8_t nbytes`  
eDMA minor byte transfer count initially configured.

`lpspi_master_edma_transfer_callback_t callback`  
Completion callback.

`void *userData`  
Callback user data.

`edma_handle_t *edmaRxRegToRxDataHandle`  
edma\_handle\_t handle point used for RxReg to RxData buff

`edma_handle_t *edmaTxDataToTxRegHandle`  
edma\_handle\_t handle point used for TxData to TxReg buff

`edma_tcd_t lpspiSoftwareTCD[3]`  
SoftwareTCD, internal used

`struct _lpspi_slave_edma_handle`  
`#include <fsl_lpspi_edma.h>` LPSPI slave eDMA transfer handle structure used for transactional API.

## Public Members

volatile bool isByteSwap

A flag that whether should byte swap.

volatile uint8\_t fifoSize

FIFO dataSize.

volatile uint8\_t rxWatermark

Rx watermark.

volatile uint8\_t bytesEachWrite

Bytes for each write TDR.

volatile uint8\_t bytesEachRead

Bytes for each read RDR.

volatile uint8\_t bytesLastRead

Bytes for last read RDR.

volatile bool isThereExtraRxBytes

Is there extra RX byte.

uint8\_t nbytes

eDMA minor byte transfer count initially configured.

const uint8\_t \*volatile txData

Send buffer.

uint8\_t \*volatile rxData

Receive buffer.

volatile size\_t txRemainingByteCount

Number of bytes remaining to send.

volatile size\_t rxRemainingByteCount

Number of bytes remaining to receive.

volatile uint32\_t writeRegRemainingTimes

Write TDR register remaining times.

volatile uint32\_t readRegRemainingTimes

Read RDR register remaining times.

uint32\_t totalByteCount

Number of transfer bytes

uint32\_t txBuffIfNull

Used if there is not txData for DMA purpose.

uint32\_t rxBuffIfNull

Used if there is not rxData for DMA purpose.

volatile uint8\_t state

LPSPi transfer state.

uint32\_t errorCount

Error count for slave transfer.

*lpspi\_slave\_edma\_transfer\_callback\_t* callback

Completion callback.

`void *userData`

Callback user data.

`edma_handle_t *edmaRxRegToRxDataHandle`

`edma_handle_t` handle point used for RxReg to RxData buff

`edma_handle_t *edmaTxDataToTxRegHandle`

`edma_handle_t` handle point used for TxData to TxReg

`edma_tcd_t` `lpspiSoftwareTCD[2]`

SoftwareTCD, internal used

## 2.38 LPTMR: Low-Power Timer

`void LPTMR_Init(LPTMR_Type *base, const lptmr_config_t *config)`

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

---

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

---

### Parameters

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

`void LPTMR_Deinit(LPTMR_Type *base)`

Gates the LPTMR clock.

### Parameters

- `base` – LPTMR peripheral base address

`void LPTMR_GetDefaultConfig(lptmr_config_t *config)`

Fills in the LPTMR configuration structure with default settings.

The default values are as follows.

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

### Parameters

- `config` – A pointer to the LPTMR configuration structure.

`static inline void LPTMR_EnableInterrupts(LPTMR_Type *base, uint32_t mask)`

Enables the selected LPTMR interrupts.

### Parameters

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

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

Disables the selected LPTMR interrupts.

**Parameters**

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

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

Gets the enabled LPTMR interrupts.

**Parameters**

- base – LPTMR peripheral base address

**Returns**

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

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

Gets the LPTMR status flags.

**Parameters**

- base – LPTMR peripheral base address

**Returns**

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

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

Clears the LPTMR status flags.

**Parameters**

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

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

Sets the timer period in units of count.

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

---

**Note:**

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

**Parameters**

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

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

Reads the current timer counting value.

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

---

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

---

**Parameters**

- base – LPTMR peripheral base address

**Returns**

The current counter value in ticks

static inline void LPTMR\_StartTimer(LPTMR\_Type \*base)

Starts the timer.

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

**Parameters**

- base – LPTMR peripheral base address

static inline void LPTMR\_StopTimer(LPTMR\_Type \*base)

Stops the timer.

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

**Parameters**

- base – LPTMR peripheral base address

FSL\_LPTMR\_DRIVER\_VERSION

Driver Version

enum \_lptmr\_pin\_select

LPTMR pin selection used in pulse counter mode.

*Values:*

enumerator kLPTMR\_PinSelectInput\_0

Pulse counter input 0 is selected

enumerator kLPTMR\_PinSelectInput\_1

Pulse counter input 1 is selected

enumerator kLPTMR\_PinSelectInput\_2

Pulse counter input 2 is selected

enumerator kLPTMR\_PinSelectInput\_3

Pulse counter input 3 is selected

enum \_lptmr\_pin\_polarity

LPTMR pin polarity used in pulse counter mode.

*Values:*

enumerator kLPTMR\_PinPolarityActiveHigh

Pulse Counter input source is active-high

enumerator kLPTMR\_PinPolarityActiveLow

Pulse Counter input source is active-low

enum \_lptmr\_timer\_mode

LPTMR timer mode selection.

*Values:*

enumerator kLPTMR\_TimerModeTimeCounter

Time Counter mode

enumerator kLPTMR\_TimerModePulseCounter  
Pulse Counter mode

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

*Values:*

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

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

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

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

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

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

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

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

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

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

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

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

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

enumerator kLPTMR\_Prescale\_Glitch\_13  
Prescaler divide 16384, glitch filter 8192

enumerator kLPTMR\_Prescale\_Glitch\_14  
Prescaler divide 32768, glitch filter 16384

enumerator kLPTMR\_Prescale\_Glitch\_15  
Prescaler divide 65536, glitch filter 32768

enum \_lptmr\_prescaler\_clock\_select  
LPTMR prescaler/glitch filter clock select.

---

**Note:** Clock connections are SoC-specific

---

*Values:*

enum `_lptmr_interrupt_enable`

List of the LPTMR interrupts.

*Values:*

enumerator `kLPTMR_TimerInterruptEnable`

Timer interrupt enable

enum `_lptmr_status_flags`

List of the LPTMR status flags.

*Values:*

enumerator `kLPTMR_TimerCompareFlag`

Timer compare flag

typedef enum `_lptmr_pin_select` `lptmr_pin_select_t`

LPTMR pin selection used in pulse counter mode.

typedef enum `_lptmr_pin_polarity` `lptmr_pin_polarity_t`

LPTMR pin polarity used in pulse counter mode.

typedef enum `_lptmr_timer_mode` `lptmr_timer_mode_t`

LPTMR timer mode selection.

typedef enum `_lptmr_prescaler_glitch_value` `lptmr_prescaler_glitch_value_t`

LPTMR prescaler/glitch filter values.

typedef enum `_lptmr_prescaler_clock_select` `lptmr_prescaler_clock_select_t`

LPTMR prescaler/glitch filter clock select.

---

**Note:** Clock connections are SoC-specific

---

typedef enum `_lptmr_interrupt_enable` `lptmr_interrupt_enable_t`

List of the LPTMR interrupts.

typedef enum `_lptmr_status_flags` `lptmr_status_flags_t`

List of the LPTMR status flags.

typedef struct `_lptmr_config` `lptmr_config_t`

LPTMR config structure.

This structure holds the configuration settings for the LPTMR peripheral. To initialize this structure to reasonable defaults, call the `LPTMR_GetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration struct can be made constant so it resides in flash.

static inline void `LPTMR_EnableTimerDMA(LPTMR_Type *base, bool enable)`

Enable or disable timer DMA request.

#### Parameters

- `base` – base LPTMR peripheral base address
- `enable` – Switcher of timer DMA feature. “true” means to enable, “false” means to disable.

struct `_lptmr_config`

`#include <fsl_lptmr.h>` LPTMR config structure.

This structure holds the configuration settings for the LPTMR peripheral. To initialize this structure to reasonable defaults, call the `LPTMR_GetDefaultConfig()` function and pass a pointer to your configuration structure instance.



The configuration struct can be made constant so it resides in flash.

### Public Members

*lptmr\_timer\_mode\_t* timerMode

Time counter mode or pulse counter mode

*lptmr\_pin\_select\_t* pinSelect

LPTMR pulse input pin select; used only in pulse counter mode

*lptmr\_pin\_polarity\_t* pinPolarity

LPTMR pulse input pin polarity; used only in pulse counter mode

bool enableFreeRunning

True: enable free running, counter is reset on overflow False: counter is reset when the compare flag is set

bool bypassPrescaler

True: bypass prescaler; false: use clock from prescaler

*lptmr\_prescaler\_clock\_select\_t* prescalerClockSource

LPTMR clock source

*lptmr\_prescaler\_glitch\_value\_t* value

Prescaler or glitch filter value

## 2.39 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

### 2.40 LPUART Driver

static inline void LPUART\_SoftwareReset(LPUART\_Type \*base)

Resets the LPUART using software.

This function resets all internal logic and registers except the Global Register. Remains set until cleared by software.

#### Parameters

- base – LPUART peripheral base address.

*status\_t* LPUART\_Init(LPUART\_Type \*base, const *lpuart\_config\_t* \*config, uint32\_t srcClock\_Hz)

Initializes an LPUART instance with the user configuration structure and the peripheral clock.

This function configures the LPUART module with user-defined settings. Call the LPUART\_GetDefaultConfig() function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
lpuart_config_t lpuartConfig;
lpuartConfig.baudRate_Bps = 115200U;
lpuartConfig.parityMode = kLPUART_ParityDisabled;
lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
lpuartConfig.isMsb = false;
lpuartConfig.stopBitCount = kLPUART_OneStopBit;
lpuartConfig.txFifoWatermark = 0;
```

(continues on next page)

(continued from previous page)

```
lpuartConfig.rxFifoWatermark = 1;
LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
```

**Parameters**

- base – LPUART peripheral base address.
- config – Pointer to a user-defined configuration structure.
- srcClock\_Hz – LPUART clock source frequency in HZ.

**Return values**

- kStatus\_LPUART\_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus\_Success – LPUART initialize succeed

*status\_t* LPUART\_Deinit(LPUART\_Type \*base)

Deinitializes a LPUART instance.

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

**Parameters**

- base – LPUART peripheral base address.

**Return values**

- kStatus\_Success – Deinit is success.
- kStatus\_LPUART\_Timeout – Timeout during deinit.

void LPUART\_GetDefaultConfig(*lpuart\_config\_t* \*config)

Gets the default configuration structure.

This function initializes the LPUART configuration structure to a default value. The default values are: lpuartConfig->baudRate\_Bps = 115200U; lpuartConfig->parityMode = kLPUART\_ParityDisabled; lpuartConfig->dataBitsCount = kLPUART\_EightDataBits; lpuartConfig->isMsb = false; lpuartConfig->stopBitCount = kLPUART\_OneStopBit; lpuartConfig->txFifoWatermark = 0; lpuartConfig->rxFifoWatermark = 1; lpuartConfig->rxIdleType = kLPUART\_IdleTypeStartBit; lpuartConfig->rxIdleConfig = kLPUART\_IdleCharacter1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;

**Parameters**

- config – Pointer to a configuration structure.

*status\_t* LPUART\_SetBaudRate(LPUART\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)

Sets the LPUART instance baudrate.

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART\_Init.

```
LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
```

**Parameters**

- base – LPUART peripheral base address.
- baudRate\_Bps – LPUART baudrate to be set.
- srcClock\_Hz – LPUART clock source frequency in HZ.

**Return values**

- kStatus\_LPUART\_BaudrateNotSupport – Baudrate is not supported in the current clock source.
- kStatus\_Success – Set baudrate succeeded.

void LPUART\_Enable9bitMode(LPUART\_Type \*base, bool enable)

Enable 9-bit data mode for LPUART.

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

#### Parameters

- base – LPUART peripheral base address.
- enable – true to enable, false to disable.

static inline void LPUART\_SetMatchAddress(LPUART\_Type \*base, uint16\_t address1, uint16\_t address2)

Set the LPUART address.

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receives with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer; otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

---

**Note:** Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

---

#### Parameters

- base – LPUART peripheral base address.
- address1 – LPUART slave address1.
- address2 – LPUART slave address2.

static inline void LPUART\_EnableMatchAddress(LPUART\_Type \*base, bool match1, bool match2)

Enable the LPUART match address feature.

#### Parameters

- base – LPUART peripheral base address.
- match1 – true to enable match address1, false to disable.
- match2 – true to enable match address2, false to disable.

static inline void LPUART\_SetRxFifoWatermark(LPUART\_Type \*base, uint8\_t water)

Sets the rx FIFO watermark.

#### Parameters

- base – LPUART peripheral base address.
- water – Rx FIFO watermark.

static inline void LPUART\_SetTxFifoWatermark(LPUART\_Type \*base, uint8\_t water)

Sets the tx FIFO watermark.

#### Parameters

- base – LPUART peripheral base address.
- water – Tx FIFO watermark.

static inline void LPUART\_TransferEnable16Bit(*lpuart\_handle\_t* \*handle, bool enable)

Sets the LPUART using 16bit transmit, only for 9bit or 10bit mode.

This function Enable 16bit Data transmit in *lpuart\_handle\_t*.

#### Parameters

- handle – LPUART handle pointer.
- enable – true to enable, false to disable.

uint32\_t LPUART\_GetStatusFlags(LPUART\_Type \*base)

Gets LPUART status flags.

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators *\_lpuart\_flags*. To check for a specific status, compare the return value with enumerators in the *\_lpuart\_flags*. For example, to check whether the TX is empty:

```
if (kLPUART_TxDataRegEmptyFlag & LPUART_GetStatusFlags(LPUART1))
{
    ...
}
```

#### Parameters

- base – LPUART peripheral base address.

#### Returns

LPUART status flags which are Ored by the enumerators in the *\_lpuart\_flags*.

status\_t LPUART\_ClearStatusFlags(LPUART\_Type \*base, uint32\_t mask)

Clears status flags with a provided mask.

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only cleared or set by hardware are: *kLPUART\_TxDataRegEmptyFlag*, *kLPUART\_TransmissionCompleteFlag*, *kLPUART\_RxDataRegFullFlag*, *kLPUART\_RxActiveFlag*, *kLPUART\_NoiseErrorFlag*, *kLPUART\_ParityErrorFlag*, *kLPUART\_TxFifoEmptyFlag*, *kLPUART\_RxFifoEmptyFlag* Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

#### Parameters

- base – LPUART peripheral base address.
- mask – the status flags to be cleared. The user can use the enumerators in the *\_lpuart\_status\_flag\_t* to do the OR operation and get the mask.

#### Return values

- *kStatus\_LPUART\_FlagCannotClearManually* – The flag can't be cleared by this function but it is cleared automatically by hardware.
- *kStatus\_Success* – Status in the mask are cleared.

#### Returns

0 succeed, others failed.

void LPUART\_EnableInterrupts(LPUART\_Type \*base, uint32\_t mask)

Enables LPUART interrupts according to a provided mask.

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the *\_lpuart\_interrupt\_enable*. This examples shows how to enable TX empty interrupt and RX full interrupt:

```
LPUART_EnableInterrupts(LPUART1, kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↳ RxDataRegFullInterruptEnable);
```

### Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to enable. Logical OR of `_lpuart_interrupt_enable`.

```
void LPUART_DisableInterrupts(LPUART_Type *base, uint32_t mask)
```

Disables LPUART interrupts according to a provided mask.

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See `_lpuart_interrupt_enable`. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
LPUART_DisableInterrupts(LPUART1, kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↳ RxDataRegFullInterruptEnable);
```

### Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to disable. Logical OR of `_lpuart_interrupt_enable`.

```
uint32_t LPUART_GetEnabledInterrupts(LPUART_Type *base)
```

Gets enabled LPUART interrupts.

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators `_lpuart_interrupt_enable`. To check a specific interrupt enable status, compare the return value with enumerators in `_lpuart_interrupt_enable`. For example, to check whether the TX empty interrupt is enabled:

```
uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);

if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
{
    ...
}
```

### Parameters

- base – LPUART peripheral base address.

### Returns

LPUART interrupt flags which are logical OR of the enumerators in `_lpuart_interrupt_enable`.

```
static inline uintptr_t LPUART_GetDataRegisterAddress(LPUART_Type *base)
```

Gets the LPUART data register address.

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

### Parameters

- base – LPUART peripheral base address.

### Returns

LPUART data register addresses which are used both by the transmitter and receiver.

```
static inline void LPUART_EnableTxDMA(LPUART_Type *base, bool enable)
```

Enables or disables the LPUART transmitter DMA request.

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

**Parameters**

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

```
static inline void LPUART_EnableRxDMA(LPUART_Type *base, bool enable)
```

Enables or disables the LPUART receiver DMA.

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

**Parameters**

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

```
uint32_t LPUART_GetInstance(LPUART_Type *base)
```

Get the LPUART instance from peripheral base address.

**Parameters**

- base – LPUART peripheral base address.

**Returns**

LPUART instance.

```
static inline void LPUART_EnableTx(LPUART_Type *base, bool enable)
```

Enables or disables the LPUART transmitter.

This function enables or disables the LPUART transmitter.

**Parameters**

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

```
static inline void LPUART_EnableRx(LPUART_Type *base, bool enable)
```

Enables or disables the LPUART receiver.

This function enables or disables the LPUART receiver.

**Parameters**

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

```
static inline void LPUART_WriteByte(LPUART_Type *base, uint8_t data)
```

Writes to the transmitter register.

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

**Parameters**

- base – LPUART peripheral base address.
- data – Data write to the TX register.

static inline uint8\_t LPUART\_ReadByte(LPUART\_Type \*base)

Reads the receiver register.

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

**Parameters**

- base – LPUART peripheral base address.

**Returns**

Data read from data register.

static inline uint8\_t LPUART\_GetRxFifoCount(LPUART\_Type \*base)

Gets the rx FIFO data count.

**Parameters**

- base – LPUART peripheral base address.

**Returns**

rx FIFO data count.

static inline uint8\_t LPUART\_GetTxFifoCount(LPUART\_Type \*base)

Gets the tx FIFO data count.

**Parameters**

- base – LPUART peripheral base address.

**Returns**

tx FIFO data count.

void LPUART\_SendAddress(LPUART\_Type \*base, uint8\_t address)

Transmit an address frame in 9-bit data mode.

**Parameters**

- base – LPUART peripheral base address.
- address – LPUART slave address.

status\_t LPUART\_WriteBlocking(LPUART\_Type \*base, const uint8\_t \*data, size\_t length)

Writes to the transmitter register using a blocking method.

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the data to be sent out to the bus.

**Parameters**

- base – LPUART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

**Return values**

- kStatus\_LPUART\_Timeout – Transmission timed out and was aborted.
- kStatus\_Success – Successfully wrote all data.

status\_t LPUART\_WriteBlocking16bit(LPUART\_Type \*base, const uint16\_t \*data, size\_t length)

Writes to the transmitter register using a blocking method in 9bit or 10bit mode.

---

**Note:** This function only support 9bit or 10bit transfer. Please make sure only 10bit of data is valid and other bits are 0.

---

**Parameters**

- base – LPUART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

**Return values**

- kStatus\_LPUART\_Timeout – Transmission timed out and was aborted.
- kStatus\_Success – Successfully wrote all data.

*status\_t* LPUART\_ReadBlocking(LPUART\_Type \*base, uint8\_t \*data, size\_t length)

Reads the receiver data register using a blocking method.

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

**Parameters**

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data.
- length – Size of the buffer.

**Return values**

- kStatus\_LPUART\_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus\_LPUART\_NoiseError – Noise error happened while receiving data.
- kStatus\_LPUART\_FramingError – Framing error happened while receiving data.
- kStatus\_LPUART\_ParityError – Parity error happened while receiving data.
- kStatus\_LPUART\_Timeout – Transmission timed out and was aborted.
- kStatus\_Success – Successfully received all data.

*status\_t* LPUART\_ReadBlocking16bit(LPUART\_Type \*base, uint16\_t \*data, size\_t length)

Reads the receiver data register in 9bit or 10bit mode.

---

**Note:** This function only support 9bit or 10bit transfer.

---

**Parameters**

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data by 16bit, only 10bit is valid.
- length – Size of the buffer.

**Return values**

- kStatus\_LPUART\_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus\_LPUART\_NoiseError – Noise error happened while receiving data.
- kStatus\_LPUART\_FramingError – Framing error happened while receiving data.



- `kStatus_LPUART_ParityError` – Parity error happened while receiving data.
- `kStatus_LPUART_Timeout` – Transmission timed out and was aborted.
- `kStatus_Success` – Successfully received all data.

```
void LPUART__TransferCreateHandle(LPUART_Type *base, lpuart_handle_t *handle,  
                                lpuart_transfer_callback_t callback, void *userData)
```

Initializes the LPUART handle.

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

The LPUART driver supports the “background” receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn’t call the `LPUART_TransferReceiveNonBlocking()` API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing `NULL` as `ringBuffer`.

#### Parameters

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

```
status_t LPUART__TransferSendNonBlocking(LPUART_Type *base, lpuart_handle_t *handle,  
                                         lpuart_transfer_t *xfer)
```

Transmits a buffer of data using the interrupt method.

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the `kStatus_LPUART_TxIdle` as `status` parameter.

---

**Note:** The `kStatus_LPUART_TxIdle` is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the TX, check the `kLPUART_TransmissionCompleteFlag` to ensure that the transmit is finished.

---

#### Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.
- `xfer` – LPUART transfer structure, see `lpuart_transfer_t`.

#### Return values

- `kStatus_Success` – Successfully start the data transmission.
- `kStatus_LPUART_TxBusy` – Previous transmission still not finished, data not all written to the TX register.
- `kStatus_InvalidArgument` – Invalid argument.

```
void LPUART__TransferStartRingBuffer(LPUART_Type *base, lpuart_handle_t *handle, uint8_t  
                                     *ringBuffer, size_t ringBufferSize)
```

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the `UART_TransferReceiveNonBlocking()` API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

---

**Note:** When using RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, then only 31 bytes are used for saving data.

---

#### Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.
- `ringBuffer` – Start address of ring buffer for background receiving. Pass `NULL` to disable the ring buffer.
- `ringBufferSize` – size of the ring buffer.

`void LPUART__TransferStopRingBuffer(LPUART_Type *base, lpuart_handle_t *handle)`

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

#### Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.

`size_t LPUART__TransferGetRxRingBufferLength(LPUART_Type *base, lpuart_handle_t *handle)`

Get the length of received data in RX ring buffer.

#### Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.

#### Returns

Length of received data in RX ring buffer.

`void LPUART__TransferAbortSend(LPUART_Type *base, lpuart_handle_t *handle)`

Aborts the interrupt-driven data transmit.

This function aborts the interrupt driven data sending. The user can get the `remainBtyes` to find out how many bytes are not sent out.

#### Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.

`status_t LPUART__TransferGetSendCount(LPUART_Type *base, lpuart_handle_t *handle, uint32_t *count)`

Gets the number of bytes that have been sent out to bus.

This function gets the number of bytes that have been sent out to bus by an interrupt method.

#### Parameters

- `base` – LPUART peripheral base address.

- `handle` – LPUART handle pointer.
- `count` – Send bytes count.

#### Return values

- `kStatus_NoTransferInProgress` – No send in progress.
- `kStatus_InvalidArgument` – Parameter is invalid.
- `kStatus_Success` – Get successfully through the parameter `count`;

`status_t` LPUART\_TransferReceiveNonBlocking(LPUART\_Type \*base, *lpuart\_handle\_t* \*handle, *lpuart\_transfer\_t* \*xfer, `size_t` \*receivedBytes)

Receives a buffer of data using the interrupt method.

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter `kStatus_UART_RxIdle`. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to `xfer->data`, which returns with the parameter `receivedBytes` set to 5. For the remaining 5 bytes, the newly arrived data is saved from `xfer->data[5]`. When 5 bytes are received, the LPUART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to `xfer->data`. When all data is received, the upper layer is notified.

#### Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.
- `xfer` – LPUART transfer structure, see `uart_transfer_t`.
- `receivedBytes` – Bytes received from the ring buffer directly.

#### Return values

- `kStatus_Success` – Successfully queue the transfer into the transmit queue.
- `kStatus_LPUART_RxBusy` – Previous receive request is not finished.
- `kStatus_InvalidArgument` – Invalid argument.

`void` LPUART\_TransferAbortReceive(LPUART\_Type \*base, *lpuart\_handle\_t* \*handle)

Aborts the interrupt-driven data receiving.

This function aborts the interrupt-driven data receiving. The user can get the `remainBytes` to find out how many bytes not received yet.

#### Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.

`status_t` LPUART\_TransferGetReceiveCount(LPUART\_Type \*base, *lpuart\_handle\_t* \*handle, `uint32_t` \*count)

Gets the number of bytes that have been received.

This function gets the number of bytes that have been received.

#### Parameters

- `base` – LPUART peripheral base address.

- handle – LPUART handle pointer.
- count – Receive bytes count.

**Return values**

- kStatus\_NoTransferInProgress – No receive in progress.
- kStatus\_InvalidArgument – Parameter is invalid.
- kStatus\_Success – Get successfully through the parameter count;

void LPUART\_TransferHandleIRQ(LPUART\_Type \*base, void \*irqHandle)  
LPUART IRQ handle function.

This function handles the LPUART transmit and receive IRQ request.

**Parameters**

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

void LPUART\_TransferHandleErrorIRQ(LPUART\_Type \*base, void \*irqHandle)  
LPUART Error IRQ handle function.

This function handles the LPUART error IRQ request.

**Parameters**

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

void LPUART\_DriverIRQHandler(uint32\_t instance)  
LPUART driver IRQ handler common entry.

This function provides the common IRQ request entry for LPUART.

**Parameters**

- instance – LPUART instance.

FSL\_LPUART\_DRIVER\_VERSION  
LPUART driver version.

Error codes for the LPUART driver.

*Values:*

enumerator kStatus\_LPUART\_TxBusy  
TX busy

enumerator kStatus\_LPUART\_RxBusy  
RX busy

enumerator kStatus\_LPUART\_TxIdle  
LPUART transmitter is idle.

enumerator kStatus\_LPUART\_RxIdle  
LPUART receiver is idle.

enumerator kStatus\_LPUART\_TxWatermarkTooLarge  
TX FIFO watermark too large

enumerator kStatus\_LPUART\_RxWatermarkTooLarge  
RX FIFO watermark too large

enumerator kStatus\_LPUART\_FlagCannotClearManually  
Some flag can't manually clear

enumerator kStatus\_LPUART\_Error  
Error happens on LPUART.

enumerator kStatus\_LPUART\_RxRingBufferOverrun  
LPUART RX software ring buffer overrun.

enumerator kStatus\_LPUART\_RxHardwareOverrun  
LPUART RX receiver overrun.

enumerator kStatus\_LPUART\_NoiseError  
LPUART noise error.

enumerator kStatus\_LPUART\_FramingError  
LPUART framing error.

enumerator kStatus\_LPUART\_ParityError  
LPUART parity error.

enumerator kStatus\_LPUART\_BaudrateNotSupport  
Baudrate is not support in current clock source

enumerator kStatus\_LPUART\_IdleLineDetected  
IDLE flag.

enumerator kStatus\_LPUART\_Timeout  
LPUART times out.

enum \_lpuart\_parity\_mode  
LPUART parity mode.

*Values:*

enumerator kLPUART\_ParityDisabled  
Parity disabled

enumerator kLPUART\_ParityEven  
Parity enabled, type even, bit setting: PE | PT = 10

enumerator kLPUART\_ParityOdd  
Parity enabled, type odd, bit setting: PE | PT = 11

enum \_lpuart\_data\_bits  
LPUART data bits count.

*Values:*

enumerator kLPUART\_EightDataBits  
Eight data bit

enumerator kLPUART\_SevenDataBits  
Seven data bit

enum \_lpuart\_stop\_bit\_count  
LPUART stop bit count.

*Values:*

enumerator kLPUART\_OneStopBit  
One stop bit

enumerator kLPUART\_TwoStopBit

Two stop bits

enum \_lpuart\_transmit\_cts\_source

LPUART transmit CTS source.

*Values:*

enumerator kLPUART\_CtsSourcePin

CTS resource is the LPUART\_CTS pin.

enumerator kLPUART\_CtsSourceMatchResult

CTS resource is the match result.

enum \_lpuart\_transmit\_cts\_config

LPUART transmit CTS configure.

*Values:*

enumerator kLPUART\_CtsSampleAtStart

CTS input is sampled at the start of each character.

enumerator kLPUART\_CtsSampleAtIdle

CTS input is sampled when the transmitter is idle

enum \_lpuart\_idle\_type\_select

LPUART idle flag type defines when the receiver starts counting.

*Values:*

enumerator kLPUART\_IdleTypeStartBit

Start counting after a valid start bit.

enumerator kLPUART\_IdleTypeStopBit

Start counting after a stop bit.

enum \_lpuart\_idle\_config

LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

*Values:*

enumerator kLPUART\_IdleCharacter1

the number of idle characters.

enumerator kLPUART\_IdleCharacter2

the number of idle characters.

enumerator kLPUART\_IdleCharacter4

the number of idle characters.

enumerator kLPUART\_IdleCharacter8

the number of idle characters.

enumerator kLPUART\_IdleCharacter16

the number of idle characters.

enumerator kLPUART\_IdleCharacter32

the number of idle characters.

enumerator kLPUART\_IdleCharacter64

the number of idle characters.

enumerator kLPUART\_IdleCharacter128  
the number of idle characters.

enum \_lpuart\_interrupt\_enable

LPUART interrupt configuration structure, default settings all disabled.

This structure contains the settings for all LPUART interrupt configurations.

*Values:*

enumerator kLPUART\_LinBreakInterruptEnable  
LIN break detect. bit 7

enumerator kLPUART\_RxActiveEdgeInterruptEnable  
Receive Active Edge. bit 6

enumerator kLPUART\_TxDataRegEmptyInterruptEnable  
Transmit data register empty. bit 23

enumerator kLPUART\_TransmissionCompleteInterruptEnable  
Transmission complete. bit 22

enumerator kLPUART\_RxDataRegFullInterruptEnable  
Receiver data register full. bit 21

enumerator kLPUART\_IdleLineInterruptEnable  
Idle line. bit 20

enumerator kLPUART\_RxOverrunInterruptEnable  
Receiver Overrun. bit 27

enumerator kLPUART\_NoiseErrorInterruptEnable  
Noise error flag. bit 26

enumerator kLPUART\_FramingErrorInterruptEnable  
Framing error flag. bit 25

enumerator kLPUART\_ParityErrorInterruptEnable  
Parity error flag. bit 24

enumerator kLPUART\_Match1InterruptEnable  
Parity error flag. bit 15

enumerator kLPUART\_Match2InterruptEnable  
Parity error flag. bit 14

enumerator kLPUART\_TxFifoOverflowInterruptEnable  
Transmit FIFO Overflow. bit 9

enumerator kLPUART\_RxFifoUnderflowInterruptEnable  
Receive FIFO Underflow. bit 8

enumerator kLPUART\_AllInterruptEnable

enum \_lpuart\_flags

LPUART status flags.

This provides constants for the LPUART status flags for use in the LPUART functions.

*Values:*

enumerator kLPUART\_TxDataRegEmptyFlag  
Transmit data register empty flag, sets when transmit buffer is empty. bit 23

enumerator kLPUART\_TransmissionCompleteFlag

Transmission complete flag, sets when transmission activity complete. bit 22

enumerator kLPUART\_RxDataRegFullFlag

Receive data register full flag, sets when the receive data buffer is full. bit 21

enumerator kLPUART\_IdleLineFlag

Idle line detect flag, sets when idle line detected. bit 20

enumerator kLPUART\_RxOverrunFlag

Receive Overrun, sets when new data is received before data is read from receive register. bit 19

enumerator kLPUART\_NoiseErrorFlag

Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18

enumerator kLPUART\_FramingErrorFlag

Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17

enumerator kLPUART\_ParityErrorFlag

If parity enabled, sets upon parity error detection. bit 16

enumerator kLPUART\_LinBreakFlag

LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31

enumerator kLPUART\_RxActiveEdgeFlag

Receive pin active edge interrupt flag, sets when active edge detected. bit 30

enumerator kLPUART\_RxActiveFlag

Receiver Active Flag (RAF), sets at beginning of valid start. bit 24

enumerator kLPUART\_DataMatch1Flag

The next character to be read from LPUART\_DATA matches MA1. bit 15

enumerator kLPUART\_DataMatch2Flag

The next character to be read from LPUART\_DATA matches MA2. bit 14

enumerator kLPUART\_TxFifoEmptyFlag

TXEMPT bit, sets if transmit buffer is empty. bit 7

enumerator kLPUART\_RxFifoEmptyFlag

RXEMPT bit, sets if receive buffer is empty. bit 6

enumerator kLPUART\_TxFifoOverflowFlag

TXOF bit, sets if transmit buffer overflow occurred. bit 1

enumerator kLPUART\_RxFifoUnderflowFlag

RXUF bit, sets if receive buffer underflow occurred. bit 0

enumerator kLPUART\_AllClearFlags

enumerator kLPUART\_AllFlags

typedef enum *\_lpuart\_parity\_mode* lpuart\_parity\_mode\_t

LPUART parity mode.

typedef enum *\_lpuart\_data\_bits* lpuart\_data\_bits\_t

LPUART data bits count.

typedef enum *\_lpuart\_stop\_bit\_count* lpuart\_stop\_bit\_count\_t

LPUART stop bit count.



```
typedef enum _lpuart_transmit_cts_source lpuart_transmit_cts_source_t
    LPUART transmit CTS source.

typedef enum _lpuart_transmit_cts_config lpuart_transmit_cts_config_t
    LPUART transmit CTS configure.

typedef enum _lpuart_idle_type_select lpuart_idle_type_select_t
    LPUART idle flag type defines when the receiver starts counting.

typedef enum _lpuart_idle_config lpuart_idle_config_t
    LPUART idle detected configuration. This structure defines the number of idle characters
    that must be received before the IDLE flag is set.

typedef struct _lpuart_config lpuart_config_t
    LPUART configuration structure.

typedef struct _lpuart_transfer lpuart_transfer_t
    LPUART transfer structure.

typedef struct _lpuart_handle lpuart_handle_t

typedef void (*lpuart_transfer_callback_t)(LPUART_Type *base, lpuart_handle_t *handle,
status_t status, void *userData)
    LPUART transfer callback function.

typedef void (*lpuart_isr_t)(LPUART_Type *base, void *handle)

void *s_lpuartHandle[]

const IRQn_Type s_lpuartTxIRQ[]

lpuart_isr_t s_lpuartIsr[]

UART_RETRY_TIMES
    Retry times for waiting flag.

struct _lpuart_config
    #include <fsl_lpuart.h> LPUART configuration structure.
```

### Public Members

```
uint32_t baudRate_Bps
    LPUART baud rate

lpuart_parity_mode_t parityMode
    Parity mode, disabled (default), even, odd

lpuart_data_bits_t dataBitsCount
    Data bits count, eight (default), seven

bool isMsb
    Data bits order, LSB (default), MSB

lpuart_stop_bit_count_t stopBitCount
    Number of stop bits, 1 stop bit (default) or 2 stop bits

uint8_t txFifoWatermark
    TX FIFO watermark

uint8_t rxFifoWatermark
    RX FIFO watermark
```

`bool enableRxRTS`  
RX RTS enable

`bool enableTxCTS`  
TX CTS enable

`lpuart_transmit_cts_source_t txCtsSource`  
TX CTS source

`lpuart_transmit_cts_config_t txCtsConfig`  
TX CTS configure

`uint8_t rtsWatermark`  
RTS watermark

`lpuart_idle_type_select_t rxIdleType`  
RX IDLE type.

`lpuart_idle_config_t rxIdleConfig`  
RX IDLE configuration.

`bool enableTx`  
Enable TX

`bool enableRx`  
Enable RX

`bool swapTxdRxd`  
Swap TXD and RXD pins

`struct __lpuart_transfer`  
*#include <fsl\_lpuart.h>* LPUART transfer structure.

### Public Members

`size_t dataSize`  
The byte count to be transfer.

`struct __lpuart_handle`  
*#include <fsl\_lpuart.h>* LPUART handle structure.

### Public Members

`volatile size_t txDataSize`  
Size of the remaining data to send.

`size_t txDataSizeAll`  
Size of the data to send out.

`volatile size_t rxDataSize`  
Size of the remaining data to receive.

`size_t rxDataSizeAll`  
Size of the data to receive.

`size_t rxRingBufferSize`  
Size of the ring buffer.

`volatile uint16_t rxRingBufferHead`  
Index for the driver to store received data into ring buffer.

volatile uint16\_t rxRingBufferTail

Index for the user to get data from the ring buffer.

*lpuart\_transfer\_callback\_t* callback

Callback function.

void \*userData

LPUART callback function parameter.

volatile uint8\_t txState

TX transfer state.

volatile uint8\_t rxState

RX transfer state.

bool isSevenDataBits

Seven data bits flag.

bool is16bitData

16bit data bits flag, only used for 9bit or 10bit data

union \_\_\_\_unnamed65\_\_\_\_

### Public Members

uint8\_t \*data

The buffer of data to be transfer.

uint8\_t \*rxData

The buffer to receive data.

uint16\_t \*rxData16

The buffer to receive data.

const uint8\_t \*txData

The buffer of data to be sent.

const uint16\_t \*txData16

The buffer of data to be sent.

union \_\_\_\_unnamed67\_\_\_\_

### Public Members

const uint8\_t \*volatile txData

Address of remaining data to send.

const uint16\_t \*volatile txData16

Address of remaining data to send.

union \_\_\_\_unnamed69\_\_\_\_

### Public Members

uint8\_t \*volatile rxData

Address of remaining data to receive.

uint16\_t \*volatile rxData16

Address of remaining data to receive.

union \_\_\_\_unnamed71\_\_\_\_

### Public Members

`uint8_t *rxRingBuffer`  
Start address of the receiver ring buffer.

`uint16_t *rxRingBuffer16`  
Start address of the receiver ring buffer.

## 2.41 LPUART eDMA Driver

```
void LPUART__TransferCreateHandleEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle,  
                                     lpuart_edma_transfer_callback_t callback, void  
                                     *userData, edma_handle_t *txEdmaHandle,  
                                     edma_handle_t *rxEdmaHandle)
```

Initializes the LPUART handle which is used in transactional functions.

---

**Note:** This function disables all LPUART interrupts.

---

### Parameters

- `base` – LPUART peripheral base address.
- `handle` – Pointer to `lpuart_edma_handle_t` structure.
- `callback` – Callback function.
- `userData` – User data.
- `txEdmaHandle` – User requested DMA handle for TX DMA transfer.
- `rxEdmaHandle` – User requested DMA handle for RX DMA transfer.

```
status_t LPUART__SendEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle,  
                          lpuart_transfer_t *xfer)
```

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

### Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.
- `xfer` – LPUART eDMA transfer structure. See `lpuart_transfer_t`.

### Return values

- `kStatus_Success` – if succeed, others failed.
- `kStatus_LPUART_TxBusy` – Previous transfer on going.
- `kStatus_InvalidArgument` – Invalid argument.

```
status_t LPUART__ReceiveEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle,  
                             lpuart_transfer_t *xfer)
```

Receives data using eDMA.

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

### Parameters

- `base` – LPUART peripheral base address.
- `handle` – Pointer to `lpuart_edma_handle_t` structure.
- `xfer` – LPUART eDMA transfer structure, see `lpuart_transfer_t`.

**Return values**

- `kStatus__Success` – if succeed, others fail.
- `kStatus__LPUART_RxBusy` – Previous transfer ongoing.
- `kStatus__InvalidArgument` – Invalid argument.

`void LPUART__TransferAbortSendEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle)`  
 Aborts the sent data using eDMA.

This function aborts the sent data using eDMA.

**Parameters**

- `base` – LPUART peripheral base address.
- `handle` – Pointer to `lpuart_edma_handle_t` structure.

`void LPUART__TransferAbortReceiveEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle)`  
 Aborts the received data using eDMA.

This function aborts the received data using eDMA.

**Parameters**

- `base` – LPUART peripheral base address.
- `handle` – Pointer to `lpuart_edma_handle_t` structure.

`status_t LPUART__TransferGetSendCountEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle, uint32_t *count)`

Gets the number of bytes written to the LPUART TX register.

This function gets the number of bytes written to the LPUART TX register by DMA.

**Parameters**

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.
- `count` – Send bytes count.

**Return values**

- `kStatus__NoTransferInProgress` – No send in progress.
- `kStatus__InvalidArgument` – Parameter is invalid.
- `kStatus__Success` – Get successfully through the parameter `count`;

`status_t LPUART__TransferGetReceiveCountEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle, uint32_t *count)`

Gets the number of received bytes.

This function gets the number of received bytes.

**Parameters**

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.
- `count` – Receive bytes count.

**Return values**

- `kStatus_NoTransferInProgress` – No receive in progress.
- `kStatus_InvalidArgument` – Parameter is invalid.
- `kStatus_Success` – Get successfully through the parameter count;

`void LPUART__TransferEdmaHandleIRQ(LPUART_Type *base, void *lpuartEdmaHandle)`  
LPUART eDMA IRQ handle function.

This function handles the LPUART tx complete IRQ request and invoke user callback. It is not set to static so that it can be used in user application.

---

**Note:** This function is used as default IRQ handler by double weak mechanism. If user's specific IRQ handler is implemented, make sure this function is invoked in the handler.

---

### Parameters

- `base` – LPUART peripheral base address.
- `lpuartEdmaHandle` – LPUART handle pointer.

`FSL_LPUART_EDMA_DRIVER_VERSION`

LPUART EDMA driver version.

`typedef struct _lpuart_edma_handle lpuart_edma_handle_t`

`typedef void (*lpuart_edma_transfer_callback_t)(LPUART_Type *base, lpuart_edma_handle_t *handle, status_t status, void *userData)`

LPUART transfer callback function.

`struct _lpuart_edma_handle`

`#include <fsl_lpuart_edma.h>` LPUART eDMA handle.

### Public Members

`lpuart_edma_transfer_callback_t` callback

Callback function.

`void *userData`

LPUART callback function parameter.

`size_t rxDataSizeAll`

Size of the data to receive.

`size_t txDataSizeAll`

Size of the data to send out.

`edma_handle_t *txEdmaHandle`

The eDMA TX channel used.

`edma_handle_t *rxEdmaHandle`

The eDMA RX channel used.

`uint8_t nbytes`

eDMA minor byte transfer count initially configured.

`volatile uint8_t txState`

TX transfer state.

`volatile uint8_t rxState`

RX transfer state

## 2.42 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.
- bufferfault – Structure to store the result.

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.
- lmemFault – Structure to store the result.

```
static inline uint64_t MCM_GetLmemFaultData(MCM_Type *base)
```

Gets LMEM fault data.

#### Parameters

- base – MCM peripheral base address.

MCM\_LMFATR\_TYPE\_MASK

MCM\_LMFATR\_MODE\_MASK

MCM\_LMFATR\_BUFF\_MASK

MCM\_LMFATR\_CACH\_MASK

MCM\_ISCR\_STAT\_MASK

FSL\_COMPONENT\_ID

```
union _mcm_buffer_fault_attribute
```

*#include <fsl\_mcm.h>* The union of buffer fault attribute.

#### Public Members

uint32\_t attribute

Indicates the faulting attributes, when a properly-enabled cache write buffer error interrupt event is detected.

struct \_mcm\_buffer\_fault\_attribute, mcm\_buffer\_fault\_attribut attribute\_\_memory

```
struct _mcm_buffer_fault_attribut
#include <fsl_mcm.h>
```

### Public Members

uint32\_t busErrorDataAccessType

Indicates the type of cache write buffer access.

uint32\_t busErrorPrivilegeLevel

Indicates the privilege level of the cache write buffer access.

uint32\_t busErrorSize

Indicates the size of the cache write buffer access.

uint32\_t busErrorAccess

Indicates the type of system bus access.

uint32\_t busErrorMasterID

Indicates the crossbar switch bus master number of the captured cache write buffer bus error.

uint32\_t busErrorOverrun

Indicates if another cache write buffer bus error is detected.

union \_mcm\_lmem\_fault\_attribute

*#include <fsl\_mcm.h>* The union of LMEM fault attribute.

### Public Members

uint32\_t attribute

Indicates the attributes of the LMEM fault detected.

struct \_mcm\_lmem\_fault\_attribute.\_mcm\_lmem\_fault\_attribut attribute\_memory

struct \_mcm\_lmem\_fault\_attribut

*#include <fsl\_mcm.h>*

### Public Members

uint32\_t parityFaultProtectionSignal

Indicates the features of parity fault protection signal.

uint32\_t parityFaultMasterSize

Indicates the parity fault master size.

uint32\_t parityFaultWrite

Indicates the parity fault is caused by read or write.

uint32\_t backdoorAccess

Indicates the LMEM access fault is initiated by core access or backdoor access.

uint32\_t parityFaultSyndrome

Indicates the parity fault syndrome.

uint32\_t overrun

Indicates the number of faultss.

## 2.43 Mipi\_dsi

`void DSI_Init(MIPI_DSI_Type *base, dsi_config_t *config)`

Initializes the MIPI DSI host with the user configuration.

This function initializes the MIPI DSI host with the configuration, it should be called before other MIPI DSI driver functions.

### Parameters

- `base` – MIPI DSI host peripheral base address.
- `config` – Pointer to the user configuration structure.

`void DSI_Deinit(MIPI_DSI_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)`

Gets the default configuration to initialize the MIPI DSI host.

The default value is:

```
config->mode = kDSI_CommandMode;
config->packageFlags = kDSI_DpiEnableAll;
config->enableNoncontinuousClk = true;
config->HsRxDeviceReady_ByteClk = 0U;
config->lpRxDeviceReady_ByteClk = 0U;
config->HsTxDeviceReady_ByteClk = 0U;
config->lpTxDeviceReady_ByteClk = 0U;
```

### Parameters

- `config` – Pointer to a user-defined configuration structure.

`uint32_t DSI_DphyGetPlldivider(uint32_t *m, uint32_t *n, uint32_t refClkFreq_Hz, uint32_t desiredOutFreq_Hz)`

Calculates the D-PHY PLL dividers to generate the desired output frequency.

The phy byte clock frequency(byte count per second) is generated by multiplying the ref-ClkFreq\_Hz, the formula is as follows, m & n is configured by mediamix control block.

$\text{desiredOutFreq\_Hz} = \text{refClkFreq\_Hz} * (M + 2) / (N + 1)$ . M: 62 ~ 625 N: 0 ~ 15

### Parameters

- `m` – Control of the feedback multiplication ratio.
- `n` – Control of the input frequency division ratio.
- `refClkFreq_Hz` – The D-PHY input reference clock frequency (REF\_CLK).
- `desiredOutFreq_Hz` – Desired PLL output frequency.

### Returns

The actually output frequency using the returned dividers. If can not find suitable dividers, return 0.

`status_t DSI_PowerUp(MIPI_DSI_Type *base)`

Power up the DSI.

### Parameters

- base – MIPI DSI host peripheral base address.

**Return values**

- kStatus\_Success – Data transfer finished with no error.
- kStatus\_Timeout – Transfer failed because of timeout.

```
static inline void DSI_PowerDown(MIPI_DSI_Type *base)
```

Power down the DSI.

**Parameters**

- base – MIPI DSI host peripheral base address.

```
static inline void DSI_EnableInterrupts(MIPI_DSI_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_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_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.

```
void DSI_SetDpiConfig(MIPI_DSI_Type *base, const dsi_dpi_config_t *config, uint8_t laneNum)
```

Configure the DPI interface.

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.
- laneNum – How many lanes in use.

```
void DSI_SetCommandModeConfig(MIPI_DSI_Type *base, const dsi_command_config_t *config,
                               uint32_t phyByteClkFreq_Hz)
```

Configures the command mode configuration.

This function configures the timeout values for DSI command mode.

#### Parameters

- base – MIPI DSI host peripheral base address.
- config – Pointer to the command mode configuration structure.
- phyByteClkFreq\_Hz – Byte clock frequency in each lane.

```
static inline void DSI_EnableCommandMode(MIPI_DSI_Type *base, bool enable)
```

Enables the command mode.

This function configures the timeout values for DSI command mode.

#### Parameters

- base – MIPI DSI host peripheral base address.
- enable – true to enable command mode and disable video mode, vise versa.

```
static inline void DSI_EnableVpgEnMode(MIPI_DSI_Type *base, bool enable)
```

Enables the VPG mode.

This function configures video mode pattern generator.

#### Parameters

- base – MIPI DSI host peripheral base address.
- enable – true to enable video mode pattern generator.

```
void DSI_GetDefaultDphyConfig(dsi_dphy_config_t *config, uint32_t phyByteClkFreq_Hz, uint8_t
                              laneNum)
```

Gets the default D-PHY configuration.

Gets the default D-PHY configuration, the timing parameters are set according to D-PHY specification. User can use the configuration directly, or change the parameters according device specific requirements.

#### Parameters

- config – Pointer to the D-PHY configuration.
- phyByteClkFreq\_Hz – Byte clock frequency in each lane.
- laneNum – How may lanes in use.

```
void DSI_InitDphy(MIPI_DSI_Type *base, const dsi_dphy_config_t *config)
```

Initializes the D-PHY.

This function configures the D-PHY timing and setups the D-PHY PLL based on user configuration. The default configuration can be obtained by calling the function `DSI_GetDefaultDphyConfig`.

#### Parameters

- base – MIPI DSI host peripheral base address.
- config – Pointer to the D-PHY configuration.

```
void DSI_SetPacketControl(MIPI_DSI_Type *base, uint8_t flags)
```

Configures the APB packet to send.

This function configures the next APB packet transfer feature. After configuration, user can write the payload by calling `DSI_WriteTxPayload` then call `DSI_WriteTxHeader` to start the transfer or just call `DSI_WriteTxHeader` alone if it is a short packet.

**Parameters**

- base – MIPI DSI host peripheral base address.
- flags – The transfer control flags, see ref `_dsi_transfer_flags`.

`void DSI_WriteTxHeader(MIPI_DSI_Type *base, uint16_t wordCount, uint8_t virtualChannel, dsi_tx_data_type_t dataType)`

Writes tx header to command FIFO. This will trigger the packet 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  $(data1 \ll 8) | data0$ .
- virtualChannel – Virtual channel.
- dataType – The packet data type, (DI).

`void DSI_WriteTxPayload(MIPI_DSI_Type *base, const uint8_t *payload, uint16_t payloadSize)`

Fills 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_WriteTxPayloadExt(MIPI_DSI_Type *base, const uint8_t *payload, uint16_t payloadSize, bool sendDcsCmd, uint8_t dcsCmd)`

Writes payload data to generic payload FIFO.

Write the long packet payload to TX FIFO. This function could be used in two ways

- Include the DCS command in the 1st byte of 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_WriteTxPayload` when used in this way.
- The DCS command is not in 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_ReadRxData(MIPI_DSI_Type *base, uint8_t *payload, uint16_t payloadSize)`

Reads the long APB packet payload.

Read the long packet payload from RX FIFO. This function reads directly from RX FIFO status. Upper layer should make sure the whole rx packet has been received.

**Parameters**

- base – MIPI DSI host peripheral base address.
- payload – Pointer to the payload buffer.
- payloadSize – Payload size in byte.

*status\_t* DSI\_TransferBlocking(MIPI\_DSI\_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.

#### 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 ref DSI\_GetRxErrorStatus to check the error details.
- kStatus\_DSI\_PhyError – PHY error detected during transfer.
- kStatus\_DSI\_ErrorReportReceived – Error Report packet received, user could use ref DSI\_GetAndClearHostStatus to check the error report status.
- kStatus\_DSI\_NotSupported – Transfer format not supported.
- kStatus\_DSI\_Fail – Transfer failed for other reasons.

*uint16\_t* Pll\_Set\_Hs\_Freqrange(*uint32\_t* bnd\_width)

Lookup table method to obtain HS frequency range of operation selection override.

#### Parameters

- bnd\_width – band width frequency in Hz

#### Returns

the *hsfreqrange\_ovr*[6:0] value based on band width frequency in hz.

*uint16\_t* Pll\_Set\_Pll\_Prop\_Param(*uint32\_t* pll\_freq\_sel)

Lookup table method to obtain PLL Proportional Charge Pump control.

#### Parameters

- pll\_freq\_sel – PLL frequency in Mhz

#### Returns

the *pll\_prop\_cntrl\_rw*[5:0] value based on video Pll frequency in Mhz.

*uint16\_t* Pll\_Set\_Sr\_Osc\_Freq\_Target(*uint32\_t* pll\_freq\_sel)

Lookup table method to obtain DDL target oscillation frequency.

#### Parameters

- pll\_freq\_sel – PLL frequency in Mhz

#### Returns

the *sr\_osc\_freq\_target*[11:0] value based on video Pll frequency in Mhz.

*uint32\_t* Pll\_Set\_Pll\_Vco\_Freq(*uint32\_t* pll\_freq\_sel)

calculate VCO frequency.

#### Parameters

- pll\_freq\_sel – PLL frequency in Hz



**Returns**

the vco\_freq\_clk value

uint16\_t Pll\_Set\_Pll\_Vco\_Param(uint32\_t pll\_freq\_sel)

Lookup table method to obtain VCO parameter.

**Parameters**

- pll\_freq\_sel – PLL frequency in Mhz

**Returns**

the pll\_vco\_cntrl\_ovr\_rw[5:0] value based on video Pll frequency in Mhz. If can not find suitable value, return default value 63.

void DSI\_ConfigDphy(MIPI\_DSI\_Type \*base, uint32\_t phyRefClkFreq\_Hz, uint32\_t dataRateFreq\_Hz)

config to set Dphy.

**Parameters**

- phyRefClkFreq\_Hz – Dphy reference clock frequency in Hz
- dataRateFreq\_Hz – line rate clock frequency.

FSL\_MIPI\_DSI\_DRIVER\_VERSION

Error codes for the MIPI DSI driver.

*Values:*

enumerator kStatus\_DSI\_Busy

DSI is busy.

enumerator kStatus\_DSI\_EccMultiBitError

Multibit ECC error detected in rx packet.

enumerator kStatus\_DSI\_CrcError

CRC error detected in rx packet.

enumerator kStatus\_DSI\_PacketSizeError

Rx packet size error.

enumerator kStatus\_DSI\_EotMissingError

Received transmission does not end with an EoT packet.

enumerator kStatus\_DSI\_ErrorReportReceived

Error report package received.

enumerator kStatus\_DSI\_NotSupported

The transfer type not supported.

enumerator kStatus\_DSI\_PhyError

Physical layer error.

Status and interrupt mask of acknowledge error sent by device caused by host, belongs to interrupt group1. INT\_ST0 bit0-bit15.

*Values:*

enumerator kDSI\_ErrorReportSot

SoT error detected in device's acknowledge error report.

enumerator kDSI\_ErrorReportSotSync

SoT Sync error detected in device's acknowledge error report.

enumerator kDSI\_ErrorReportEotSync

EoT Sync error detected in device's acknowledge error report.

enumerator kDSI\_ErrorReportEscEntryCmd

Escape Mode Entry Command error detected in device's acknowledge error report.

enumerator kDSI\_ErrorReportLpSync

Low-power Transmit Sync error detected in device's acknowledge error report.

enumerator kDSI\_ErrorReportPeriphTo

Peripheral Timeout error detected in device's acknowledge error report.

enumerator kDSI\_ErrorReportFalseControl

False Control error detected in device's acknowledge error report.

enumerator kDSI\_ErrorReportDeviceSpecific1

The device specific error detected in device's acknowledge error report.

enumerator kDSI\_ErrorReportEccOneBit

Single-bit ECC error detected in device's acknowledge error report.

enumerator kDSI\_ErrorReportEccMultiBit

Multi-bit ECC error detected in device's acknowledge error report.

enumerator kDSI\_ErrorReportChecksum

Checksum error detected in device's acknowledge error report.

enumerator kDSI\_ErrorReportDataTypeUnrecognized

DSI data type not recognized error detected in device's acknowledge error report.

enumerator kDSI\_ErrorReportVcIdInvalid

Virtual channel ID invalid error detected in device's acknowledge error report.

enumerator kDSI\_ErrorReportTxLengthInvalid

Invalid transmission length error detected in device's acknowledge error report.

enumerator kDSI\_ErrorReportDeviceSpecific2

The device specific error detected in device's acknowledge error report.

enumerator kDSI\_ErrorReportProtocolViolation

Protocol violation error detected in device's acknowledge error report.

enumerator kDSI\_ErrorReportAll

Status and interrupt mask of error in phy layer, belongs to interrupt group1. INT\_ST0 bit16-bit20.

*Values:*

enumerator kDSI\_PhyErrorEscEntry

Escape entry error from Lane 0.

enumerator kDSI\_PhyErrorLpSync

Low-power data transmission synchronization error from Lane 0.

enumerator kDSI\_PhyErrorControl

Control error from Lane 0.

enumerator kDSI\_PhyErrorLp0Connection

LP0 connection error from Lane 0.

enumerator kDSI\_PhyErrorLp1Connection

LP1 connection error from Lane 0.

enumerator kDSI\_PhyErrorAll

Timeout error interrupt and status, belongs to interrupt group2. INT\_ST1 bit0-bit6.

*Values:*

enumerator kDSI\_TimeoutErrorHtx

High Speed forward TX timeout detected.

enumerator kDSI\_TimeoutErrorLrx

Reverse Low power data receive timeout detected.

Host receive packet error status, belongs to interrupt group2. INT\_ST1 bit0-bit6.

*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\_RxErrorPacketSize

Packet size error detected.

enumerator kDSI\_RxErrorEotMissing

Host receives a transmission that does not end with an EoT packet.

enumerator kDSI\_RxErrorAll

Host receive error status, belongs to interrupt group2. INT\_ST1 bit7-bit12 bit19?

*Values:*

enumerator kDSI\_DpiPayloadFifoOverflow

During a DPI pixel line storage, the payload FIFO overflow occurs.

enumerator kDSI\_GenericCommandFifoOverflow

System writes a command through the Generic interface while FIFO is full causing overflow.

enumerator kDSI\_GenericPayloadFifoOverflow

System writes a payload data through the Generic interface while FIFO is full causing payload FIFO overflow.

enumerator kDSI\_GenericPayloadFifoUnderflow

System writes the packet header before the packet payload is completed loaded into the payload FIFO during a packet build causing payload FIFO underflow.

enumerator kDSI\_GenericReadFifoUnderflow

System requests data before it is fully received causing underflow.

enumerator kDSI\_GenericReadFifoOverflow

The Read FIFO size is not correctly dimensioned for the max rx packet size causing generic read FIFO overflow.

`_dsi_dpi_package_flag` Flags for DPI package composition.

*Values:*

enumerator `kDSI_DpiEnableEotpTxHs`

Enables the EoTp transmission in high-speed.

enumerator `kDSI_DpiEnableEotpRx`

Enables the EoTp reception.

enumerator `kDSI_DpiEnableBta`

Enables the Bus Turn-Around (BTA) request.

enumerator `kDSI_DpiEnableEcc`

Enables the ECC reception, error correction, and reporting.

enumerator `kDSI_DpiEnableCrc`

Enables the CRC reception and error reporting.

enumerator `kDSI_DpiEnableEotpTxLp`

Enables the EoTp transmission in low-power.

enumerator `kDSI_DpiEnableAll`

enum `_dsi_operation_mode`

MIPI DSI operation mode.

*Values:*

enumerator `kDSI_VideoMode`

Video mode.

enumerator `kDSI_CommandMode`

Command mode.

enum `_dsi_dpi_color_coding`

MIPI DPI interface color coding.

*Values:*

enumerator `kDSI_DpiRGB16Bit`

16-bit configuration 1. RGB565: XXXXXXXX\_RRRRRGGG\_GGGBBBBB.

enumerator `kDSI_DpiRGB16BitLoose0`

16-bit configuration 2. RGB565: XXXRRRRR\_XXGGGGGG\_XXXBBBBB.

enumerator `kDSI_DpiRGB16BitLoose1`

16-bit configuration 3. RGB565: XXRRRRRX\_XXGGGGGG\_XXBBBBBX.

enumerator `kDSI_DpiRGB18Bit`

18-bit configuration 1. RGB666: XXXXXXRR\_RRRRGGGG\_GGBBBBBB.

enumerator `kDSI_DpiRGB18BitLoose`

18-bit configuration 2. RGB666: XXRRRRRR\_XXGGGGGG\_XXBBBBBB.

enumerator `kDSI_DpiRGB24Bit`

24-bit configuration. RGB888: RRRRRRRR\_GGGGGGGG\_BBBBBBBB.

enumerator `kDSI_DpiYCbCr20Bit`

20-bit configuration. YCbCr422 loosely packed.  
Cle1: YYYYYYYY\_YYXXCbCbCbCb\_CbCbCbCbCbCbXX,  
YYYYYYYY\_YYXXCrCrCrCr\_CrCrCrCrCrCrXX.

Cy-  
Cycle2:

enumerator kDSI\_DpiYCbCr24Bit

24-bit configuration. YCbCr422: CyCle1: YYYYYYYY\_YYYYCbCbCbCb\_CbCbCbCbCbCbCbCb,  
Cycle2: YYYYYYYY\_YYYYCrCrCrCr\_CrCrCrCrCrCrCrCr.

enumerator kDSI\_DpiYCbCr16Bit

16-bit configuration. YCbCr422 loosely packed. Cy-  
Cle1: YYYYYYYY\_YYXXCbCbCbCb\_CbCbCbCbCbCbXX, Cycle2:  
YYYYYYY\_XXXXCrCrCrCr\_CrCrCrCrXXXX.

enumerator kDSI\_DpiRGB30Bit

30-bit configuration. RGB10.10.10: XXXRRRRR\_RRRRGGGG\_GGGGGGBB\_BBBBBBBB.

enumerator kDSI\_DpiRGB36Bit

36-bit configuration. RGB12.12.12. CyCle1: XXXXXXRR\_RRRRRRRR\_RRGGGGGG, Cy-  
cle2: XXXXXXGG\_GGGGBBBB\_BBBBBBBB.

enumerator kDSI\_DpiYCbCr12Bit

12-bit configuration. YCbCr420: CyCle1: Y1Y1Y1Y1Y1Y1Y1\_Y0Y0Y0Y0Y0Y0Y0\_CbCbCbCbCbCbCbCb  
Cycle2: Y1Y1Y1Y1Y1Y1Y1Y1\_Y0Y0Y0Y0Y0Y0Y0Y0\_CrCrCrCrCrCrCrCr.

enumerator kDSI\_DpiDcs24Bit

24-bit configuration with no specific coding.

\_dsi\_dpi\_polarity\_flag Flags for DPI signal polarity.

*Values:*

enumerator kDSI\_DpiDataEnableActiveHigh

Data enable pin active high.

enumerator kDSI\_DpiVsyncActiveHigh

VSYNC active high.

enumerator kDSI\_DpiHsyncActiveHigh

HSYNC active high.

enumerator kDSI\_DpiShutDownActiveHigh

Shutdown pin active high.

enumerator kDSI\_DpiColorModeActiveHigh

Color mode pin active high.

enumerator kDSI\_DpiDataEnableActiveLow

Data enable pin active low.

enumerator kDSI\_DpiVsyncActiveLow

VSYNC active low.

enumerator kDSI\_DpiHsyncActiveLow

HSYNC active low.

enumerator kDSI\_DpiShutDownActiveLow

Shutdown pin active low.

enumerator kDSI\_DpiColorModeActiveLow

Color mode pin active low.

enum \_dsi\_video\_mode

DSI 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\_video\_pattern

*Values:*

enumerator kDSI\_PatternDisable

Color bar pattern mode disabled.

enumerator kDSI\_PatternVertical

Color bar pattern mode displayed vertically.

enumerator kDSI\_PatternHorizontal

Color bar pattern mode displayed horizontally.

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_TransferUseLowPower`

Use low power or not.

enumerator `kDSI_TransferPerformBTA`

Perform BTA or not at the end of a frame.

typedef enum `_dsi_operation_mode` `dsi_operation_mode_t`

MIPI DSI operation mode.

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_video_mode` `dsi_video_mode_t`

DSI video mode.

typedef enum `_dsi_video_pattern` `dsi_video_pattern_t`

typedef struct `_dsi_dpi_config` `dsi_dpi_config_t`

MIPI DSI controller DPI interface configuration.

typedef struct `_dsi_command_config` `dsi_command_config_t`

MIPI DSI command mode 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.
```

```
uint32_t DSI_GetInstance(MIPI_DSI_Type *base)
    Gets the MIPI DSI host controller instance from peripheral base address.
```

#### Parameters

- base – MIPI DSI peripheral base address.

#### Returns

MIPI DSI instance.

TX\_DPHY\_TX\_PLL\_1

TX\_DPHY\_TX\_PLL\_5

TX\_DPHY\_TX\_PLL\_9

TX\_DPHY\_TX\_PLL\_13

TX\_DPHY\_TX\_PLL\_17

TX\_DPHY\_TX\_PLL\_22

TX\_DPHY\_TX\_PLL\_23

TX\_DPHY\_TX\_PLL\_24

TX\_DPHY\_TX\_PLL\_25

TX\_DPHY\_TX\_PLL\_27

TX\_DPHY\_TX\_PLL\_28

TX\_DPHY\_TX\_PLL\_29

TX\_DPHY\_TX\_PLL\_30

TX\_DPHY\_TX\_PLL\_31

TX\_DPHY\_TX\_CB\_0

TX\_DPHY\_TX\_CB\_1

TX\_DPHY\_TX\_CB\_2

TX\_DPHY\_TX\_SLEW\_5

TX\_DPHY\_TX\_SLEW\_6

TX\_DPHY\_TX\_SLEW\_7

TX\_DPHY\_TX\_CLK\_TERMLOWCAP

```
struct _dsi_config
```

*#include <fsl\_mipi\_dsi.h>* MIPI DSI controller configuration.

## Public Members

*dsi\_operation\_mode\_t* mode

DSI operation mode. MODE\_CFG[cmd\_video\_mode]

uint8\_t packageFlags

OR'ed value of \_dsi\_dpi\_package\_flag that controls DPI package composition. PCK-HDL\_CFG

bool enableNoncontinuousClk

Enables the automatic mechanism to stop providing clock in the clock lane when time allows. LPCLK\_CTRL[auto\_clklane\_ctrl]

uint16\_t HsRxDeviceReady\_ByteClk

The min time the display device takes to process high-speed read from master before it can continue doing other stuff. The timer starts when D-PHY enters stop state and measured in lane byte clock. HS\_RD\_TO\_CNT[hs\_rd\_to\_cnt]

uint16\_t lpRxDeviceReady\_ByteClk

The min time the display device takes to process low-power read from master before it can continue doing other stuff. The timer starts when D-PHY enters stop state and measured in lane byte clock. LP\_RD\_TO\_CNT[lp\_rd\_to\_cnt]

uint16\_t HsTxDeviceReady\_ByteClk

The min time the display device takes to process high-speed write from master before it can continue doing other stuff. The timer starts when D-PHY enters stop state and measured in lane byte clock. HS\_WR\_TO\_CNT[hs\_wr\_to\_cnt]

uint16\_t lpTxDeviceReady\_ByteClk

The min time the display device takes to process low-power write from master before it can continue doing other stuff. The timer starts when D-PHY enters stop state and measured in lane byte clock. LP\_WR\_TO\_CNT[lp\_wr\_to\_cnt]

struct \_dsi\_dpi\_config

#include <fsl\_mipi\_dsi.h> MIPI DSI controller DPI interface configuration.

## Public Members

uint8\_t virtualChannel

Virtual channel. DPI\_VCID[dpi\_vcid]

*dsi\_dpi\_color\_coding\_t* colorCoding

DPI color coding. DPI\_COLOR\_CODING

uint8\_t polarityFlags

OR'ed value of \_dsi\_dpi\_polarity\_flag that controls signal polarity. DPI\_CFG\_POL

bool enablelpSwitch

Enable return to low-power inside the VSA/VBP/VFP/VACT/HBP/HFP period when timing allows. VID\_MODE\_CFG[bit8-13]

bool enableAck

Enable the request for an acknowledge response at the end of a frame. VID\_MODE\_CFG[frame\_bta\_ack\_en]

*dsi\_video\_mode\_t* videoMode

Video mode. VID\_MODE\_CFG[vid\_mode\_type]

uint16\_t pixelPayloadSize

Color bar pattern. VID\_MODE\_CFG[vpg\_orientation][vpg\_en][vpg\_mode=0] The number of pixels in a single video packet. For 18-bit not loosely packed data types, this number must be a multiple of 4, for YCbCr data types, it must be a multiple of 2. Recommended to set to the line size (in pixels). VID\_PKT\_SIZE

uint16\_t vsw

Number of lines in vertical sync width. VID\_VSA\_LINES

uint16\_t vbp

Number of lines in vertical back porch. VID\_VBP\_LINES

uint16\_t vfp

Number of lines in vertical front porch. VID\_VFP\_LINES

uint16\_t panelHeight

Number of lines in vertical active area. VID\_VACTIVE\_LINES

uint16\_t hsw

Horizontal sync width, in dpi pixel clock. VID\_HSA\_TIME

uint16\_t hbp

Horizontal back porch, in dpi pixel clock. VID\_HBP\_TIME

uint16\_t hfp

Horizontal front porch, in dpi pixel clock. VID\_HLINE\_TIME = (hsw+hbp+hfp+width)

struct \_\_dsi\_command\_config

*#include <fsl\_mipi\_dsi.h>* MIPI DSI command mode configuration.

### Public Members

uint32\_t escClkFreq\_Hz

Escape clock frequency in Hz.

uint16\_t lpRxTo\_Ns

Timeout value that triggers a low-power reception timeout contention detection. TO\_CNT\_CFG[lprx\_to\_cnt]

uint16\_t hsTxTo\_Ns

Timeout value that triggers a high-speed transmission timeout contention detection. In non-burst mode, the time should be larger than 1.1 times of one frame data transmission time, in burst mode it should be one line. TO\_CNT\_CFG[hstx\_to\_cnt]

uint16\_t btaTo\_Ns

The time period for which MIPI DSI host keeps the link still after completing a Bus Turnaround. BTA\_TO\_CNT[bta\_to\_cnt]

struct \_\_dsi\_dphy\_config

*#include <fsl\_mipi\_dsi.h>* MIPI DSI D-PHY configuration.

### Public Members

uint8\_t numLanes

Number of lanes. The value range is from 1-4, lane 0-3. PHY\_IF\_CFG[n\_lanes]

uint8\_t tStopState\_ByteClk

Minimum time that the PHY controller stays in stop state before a HS transmission. TODO in what unit? PHY\_IF\_CFG[phy\_stop\_wait\_time]

uint16\_t tClkHs2Lp\_ByteClk

Maximum time that the D-PHY clock lane takes to go from high-speed to low-power in lane byte clock. PHY\_TMR\_LPCLK\_CFG[phy\_clkhs2lp\_time]

uint16\_t tClkLp2Hs\_ByteClk

Maximum time that the D-PHY clock lane takes to go from low-power to high-speed in lane byte clock. PHY\_TMR\_LPCLK\_CFG[phy\_clklp2hs\_time]

uint16\_t tDataHs2Lp\_ByteClk

Maximum time that the D-PHY data lane takes to go from high-speed to low-power in lane byte clock. PHY\_TMR\_CFG[phy\_hs2lp\_time]

uint16\_t tDataLp2Hs\_ByteClk

Maximum time that the D-PHY data lane takes to go from low-power to high-speed in lane byte clock. PHY\_TMR\_CFG[phy\_lp2hs\_time]

uint16\_t maxRead\_ByteClk

Maximum time required to perform a read command in lane byte clock. PHY\_TMR\_RD\_CFG[max\_rd\_time]

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 TX 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.

## 2.44 MIPI\_DSI: MIPI DSI Host Controller

## 2.45 MU: Messaging Unit

```
uint32_t MU_GetInstance(MU_Type *base)
```

Get the MU instance index.

#### Parameters

- base – MU peripheral base address.

#### Returns

MU instance index.

```
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 `MU1_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 `MU1_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)`

brief 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 `MU1_BUSY_POLL_COUNT` is defined and non-zero, the function will timeout after the specified number of polling iterations and return `kStatus_Timeout`.

return status\_t retval `kStatus_Success` Flags were set successfully. `retval kStatus_Timeout` Timeout occurred while waiting for flags to update.

### Parameters

- `base` – MU peripheral base address.
- `flags` – The 3-bit MU flags to set.

static inline uint32\_t MU\_GetFlags(MU\_Type \*base)

Gets the current value of the 3-bit MU flags set by the other side.

This function gets the current 3-bit MU flags on the current side.

#### Parameters

- base – MU peripheral base address.

#### Returns

flags Current value of the 3-bit flags.

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);
}
```

If there are more than 4 general purpose interrupts, use `MU_GetGeneralPurposeStatusFlags`.

#### Parameters

- base – MU peripheral base address.

#### Returns

Bit mask of the MU status flags, see `_mu_status_flags`.

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 flags)

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);
```

If there are more than 4 general purpose interrupts, use `MU_ClearGeneralPurposeStatusFlags`.



**Parameters**

- base – MU peripheral base address.
- flags – Bit mask of the MU status flags. See `_mu_status_flags`. Only the following flags can be cleared by software (if applicable for particular device), other flags are cleared by hardware:
  - `kMU_GenInt0Flag`
  - `kMU_GenInt1Flag`
  - `kMU_GenInt2Flag`
  - `kMU_GenInt3Flag`
  - `kMU_MuResetInterruptFlag`
  - `kMU_OtherSideEnterRunInterruptFlag`
  - `kMU_OtherSideEnterHaltInterruptFlag`
  - `kMU_OtherSideEnterWaitInterruptFlag`
  - `kMU_OtherSideEnterStopInterruptFlag`
  - `kMU_OtherSideEnterPowerDownInterruptFlag`
  - `kMU_ResetAssertInterruptFlag`
  - `kMU_HardwareResetInterruptFlag`

```
static inline void MU_EnableInterrupts(MU_Type *base, uint32_t interrupts)
```

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);`

If there are more than 4 general purpose interrupts, use `MU_EnableGeneralPurposeInterrupts`.

**Parameters**

- base – MU peripheral base address.
- interrupts – Bit mask of the MU interrupts. See `_mu_interrupt_enable`.

```
static inline void MU_DisableInterrupts(MU_Type *base, uint32_t interrupts)
```

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);`

If there are more than 4 general purpose interrupts, use `MU_DisableGeneralPurposeInterrupts`.

**Parameters**

- base – MU peripheral base address.
- interrupts – Bit mask of the MU interrupts. See `_mu_interrupt_enable`.

*status\_t* MU\_TriggerInterrupts(MU\_Type \*base, uint32\_t interrupts)

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.
}
```

If there are more than 4 general purpose interrupts, use `MU_TriggerGeneralPurposeInterrupts`.

#### Parameters

- `base` – MU peripheral base address.
- `interrupts` – 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\_EnableGeneralPurposeInterrupts(MU\_Type \*base, uint32\_t interrupts)

Enables the MU general purpose interrupts.

This function enables the MU general purpose interrupts. The interrupts to enable should be passed in as bit mask of `mu_general_purpose_interrupt_t`. The function `MU_EnableInterrupts` only support general interrupt 0~3, this function supports all general interrupts.

For example, to enable general purpose interrupt 0 and 3, use like this:

```
MU_EnableGeneralPurposeInterrupts(MU, kMU_GeneralPurposeInterrupt0 | kMU_
↪ GeneralPurposeInterrupt3);
```

#### Parameters

- `base` – MU peripheral base address.
- `interrupts` – Bit mask of the MU general purpose interrupts, see `mu_general_purpose_interrupt_t`.

static inline void MU\_DisableGeneralPurposeInterrupts(MU\_Type \*base, uint32\_t interrupts)

Disables the MU general purpose interrupts.

This function disables the MU general purpose interrupts. The interrupts to disable should be passed in as bit mask of `mu_general_purpose_interrupt_t`. The function `MU_DisableInterrupts` only support general interrupt 0~3, this function supports all general interrupts.

For example, to disable general purpose interrupt 0 and 3, use like this:

```
MU_EnableGeneralPurposeInterrupts(MU, kMU_GeneralPurposeInterrupt0 | kMU_
↪ GeneralPurposeInterrupt3);
```

#### Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the MU general purpose interrupts. see `mu_general_purpose_interrupt_t`.

static inline uint32\_t MU\_GetGeneralPurposeStatusFlags(MU\_Type \*base)

Gets the MU general purpose interrupt status flags.

This function returns the bit mask of the MU general purpose interrupt status flags. `MU_GetStatusFlags` can only get general purpose interrupt status 0~3, this function can get all general purpose interrupts status.

This example shows to check whether general purpose interrupt 0 and 3 happened.

```
uint32_t flags;
flags = MU_GetGeneralPurposeStatusFlags(base);
if (kMU_GeneralPurposeInterrupt0 & flags)
{
}
if (kMU_GeneralPurposeInterrupt3 & flags)
{
}
```

### Parameters

- base – MU peripheral base address.

### Returns

Bit mask of the MU general purpose interrupt status flags.

static inline void MU\_ClearGeneralPurposeStatusFlags(MU\_Type \*base, uint32\_t flags)

Clear the MU general purpose interrupt status flags.

This function clears the specific MU general purpose interrupt status flags. The flags to clear should be passed in as bit mask. `mu_general_purpose_interrupt_t_mu_status_flags`.

Example to clear general purpose interrupt 0 and general interrupt 1 pending flags.

```
MU_ClearGeneralPurposeStatusFlags(base, kMU_GeneralPurposeInterrupt0 | kMU_
↪GeneralPurposeInterrupt1);
```

### Parameters

- base – MU peripheral base address.
- flags – Bit mask of the MU general purpose interrupt status flags. See `mu_general_purpose_interrupt_t`.

static inline uint32\_t MU\_GetRxStatusFlags(MU\_Type \*base)

Return the RX status flags in reverse numerical order.

This function return the RX status flags in reverse order. Note: RFn bits of SR[3-0](mu status register) are mapped in ascending numerical order: RF0 -> SR[0] RF1 -> SR[1] RF2 -> SR[2] RF3 -> SR[3] This function will return these bits in reverse numerical order(RF3->RF1) to comply with `MU_GetRxStatusFlags()` of mu driver. See `MU_GetRxStatusFlags()` from `drivers/mu/fsl_mu.h`

```
status_reg = MU_GetRxStatusFlags(base);
```

### Parameters

- base – MU peripheral base address.

### Returns

MU RX status flags in reverse order

*status\_t* MU\_TriggerGeneralPurposeInterrupts(MU\_Type \*base, uint32\_t interrupts)

Triggers general purpose interrupts to the other core.

This function triggers the specific general purpose interrupts to the other core. The interrupts to trigger are passed in as bit mask. See `mu_general_purpose_interrupt_t`. 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.

```
status_t status;
status = MU_TriggerGeneralPurposeInterrupts(base, kMU_GeneralPurposeInterrupt0 | kMU_
↪GeneralPurposeInterrupt2);

if (kStatus_Success != status)
{
    Previous general purpose interrupt 0 or general purpose interrupt 2
    has not been processed by the other core.
}
```

#### Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the interrupts to trigger. See `mu_general_purpose_interrupt_t`.

#### Return values

- kStatus\_Success – Interrupts have been triggered successfully.
- kStatus\_Fail – Previous interrupts have not been accepted.

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.

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 `MU1_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 MU1\_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);
```

---

**Note:** The feature waitReset, holdReset, and bootMode might be not supported for some platforms. waitReset is only available for platforms that FSL\_FEATURE\_MU\_NO\_CORE\_STATUS not defined as 1 and FSL\_FEATURE\_MU\_HAS\_RESET\_ASSERT\_INT not defined as 0. holdReset is only available for platforms that FSL\_FEATURE\_MU\_HAS\_RSTH not defined as 0. bootMode is only available for platforms that FSL\_FEATURE\_MU\_HAS\_BOOT not defined as 0.

---

**Parameters**

- base – MU peripheral base address.
- waitReset – Wait the other core enters reset. Only work when there is CSSR0[RAIP].
  - 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. Only work when there is CCR0[RSTH].

- 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

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\_RxFullPendingFlag  
Any RX full flag is pending.

enumerator kMU\_\_TxEmptyPendingFlag

Any TX empty flag is pending.

enumerator kMU\_\_GenIntPendingFlag

Any general interrupt flag is pending.

enumerator kMU\_\_EventPendingFlag

MU event pending.

enumerator kMU\_\_FlagsUpdatingFlag

MU flags update is on-going.

enumerator kMU\_\_MuInResetFlag

MU of any side is in reset.

enumerator kMU\_\_MuResetInterruptFlag

The other side initializes MU reset.

enum \_\_mu\_interrupt\_enable

MU interrupt source to enable.

*Values:*

enumerator kMU\_\_Tx0EmptyInterruptEnable

TX0 empty.

enumerator kMU\_\_Tx1EmptyInterruptEnable

TX1 empty.

enumerator kMU\_\_Tx2EmptyInterruptEnable

TX2 empty.

enumerator kMU\_\_Tx3EmptyInterruptEnable

TX3 empty.

enumerator kMU\_\_Rx0FullInterruptEnable

RX0 full.

enumerator kMU\_\_Rx1FullInterruptEnable

RX1 full.

enumerator kMU\_\_Rx2FullInterruptEnable

RX2 full.

enumerator kMU\_\_Rx3FullInterruptEnable

RX3 full.

enumerator kMU\_\_GenInt0InterruptEnable

General purpose interrupt 0.

enumerator kMU\_\_GenInt1InterruptEnable

General purpose interrupt 1.

enumerator kMU\_\_GenInt2InterruptEnable

General purpose interrupt 2.

enumerator kMU\_\_GenInt3InterruptEnable

General purpose interrupt 3.

enumerator kMU\_\_MuResetInterruptEnable

The other side initializes MU reset.

enum `_mu_interrupt_trigger`

MU interrupt that could be triggered to the other core.

*Values:*

enumerator `kMU_GenInt0InterruptTrigger`

General purpose interrupt 0.

enumerator `kMU_GenInt1InterruptTrigger`

General purpose interrupt 1.

enumerator `kMU_GenInt2InterruptTrigger`

General purpose interrupt 2.

enumerator `kMU_GenInt3InterruptTrigger`

General purpose interrupt 3.

enum `_mu_msg_reg_index`

MU message register index.

*Values:*

enumerator `kMU_MsgReg0`

Message register 0.

enumerator `kMU_MsgReg1`

Message register 1.

enumerator `kMU_MsgReg2`

Message register 2.

enumerator `kMU_MsgReg3`

Message register 3.

enum `_mu_general_purpose_interrupt`

MU general purpose interrupts.

*Values:*

enumerator `kMU_GeneralPurposeInterrupt0`

General purpose interrupt 0

enumerator `kMU_GeneralPurposeInterrupt1`

General purpose interrupt 1

enumerator `kMU_GeneralPurposeInterrupt2`

General purpose interrupt 2

enumerator `kMU_GeneralPurposeInterrupt3`

General purpose interrupt 3

typedef enum `_mu_msg_reg_index` `mu_msg_reg_index_t`

MU message register index.

typedef enum `_mu_general_purpose_interrupt` `mu_general_purpose_interrupt_t`

MU general purpose interrupts.

`MU_CORE_INTR(intr)`

`MU_MISC_INTR(intr)`

`MU_TX_INTR(intr)`

`MU_RX_INTR(intr)`



MU\_GI\_INTR(*intr*)  
MU\_GET\_CORE\_INTR(*intrs*)  
MU\_GET\_TX\_INTR(*intrs*)  
MU\_GET\_RX\_INTR(*intrs*)  
MU\_GET\_GI\_INTR(*intrs*)  
MU\_CORE\_FLAG(*flag*)  
MU\_STAT\_FLAG(*flag*)  
MU\_TX\_FLAG(*flag*)  
MU\_RX\_FLAG(*flag*)  
MU\_GI\_FLAG(*flag*)  
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*)  
MU1\_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.

## 2.46 OTFAD: On The Fly AES-128 Decryption Driver

void OTFAD\_GetDefaultConfig(*otfad\_config\_t* \**config*)

OTFAD module initialization function.

### Parameters

- *config* – OTFAD configuration.

void OTFAD\_Init(OTFAD\_Type \**base*, const *otfad\_config\_t* \**config*)

OTFAD module initialization function.

### Parameters

- *base* – OTFAD base address.
- *config* – OTFAD configuration.

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

Deinitializes the OTFAD.

static inline uint32\_t OTFAD\_GetOperateMode(OTFAD\_Type \*base)

OTFAD module get operate mode.

**Parameters**

- base – OTFAD base address.

static inline uint32\_t OTFAD\_GetStatus(OTFAD\_Type \*base)

OTFAD module get status.

**Parameters**

- base – OTFAD base address.

status\_t OTFAD\_SetEncryptionConfig(OTFAD\_Type \*base, const *otfad\_encryption\_config\_t* \*config)

OTFAD module set encryption configuration.

Note: if enable keyblob process, the first 256 bytes external memory is use for keyblob data, so this region shouldn't be in OTFAD region.

**Parameters**

- base – OTFAD base address.
- config – encryption configuration.

status\_t OTFAD\_GetEncryptionConfig(OTFAD\_Type \*base, *otfad\_encryption\_config\_t* \*config)

OTFAD module get encryption configuration.

Note: if enable keyblob process, the first 256 bytes external memory is use for keyblob data, so this region shouldn't be in OTFAD region.

**Parameters**

- base – OTFAD base address.
- config – encryption configuration.

status\_t OTFAD\_HitDetermination(OTFAD\_Type \*base, uint32\_t address, uint8\_t \*contextIndex)

OTFAD module do hit determination.

**Parameters**

- base – OTFAD base address.
- address – the physical address space assigned to the QuadSPI(FlexSPI) module.
- contextIndex – hitted context region index.

**Returns**

status, such as kStatus\_Success or kStatus\_OTFAD\_ResRegAccessMode.

FSL\_OTFAD\_DRIVER\_VERSION

Driver version.

Status codes for the OTFAD driver.

*Values:*

enumerator kStatus\_OTFAD\_ResRegAccessMode

Restricted register mode

enumerator kStatus\_OTFAD\_AddressError

End address less than start address

enumerator kStatus\_OTFAD\_RegionOverlap

the OTFAD does not support any form of memory region overlap, for system accesses that hit in multiple contexts or no contexts, the fetched data is simply bypassed

enumerator kStatus\_OTFAD\_RegionMiss

For accesses that hit in a single context, but not the selected one

OTFAD context type.

*Values:*

enumerator kOTFAD\_Context\_0

context 0

enumerator kOTFAD\_Context\_1

context 1

enumerator kOTFAD\_Context\_2

context 2

enumerator kOTFAD\_Context\_3

context 3

OTFAD operate mode.

*Values:*

enumerator kOTFAD\_NRM

Normal Mode

enumerator kOTFAD\_SVM

Security Violation Mode

enumerator kOTFAD\_LDM

Logically Disabled Mode

typedef struct *\_otfad\_encryption\_config* otfad\_encryption\_config\_t

OTFAD encryption configuration structure.

typedef struct *\_otfad\_config* otfad\_config\_t

OTFAD configuration structure.

struct *\_otfad\_encryption\_config*

*#include <fsl\_otfad.h>* OTFAD encryption configuration structure.

## Public Members

bool valid

The context is valid or not

bool AESdecryption

AES decryption enable

uint8\_t readOnly

read write attribute for the entire set of context registers

```
uint8_t contextIndex
    OTFAD context index
uint32_t startAddr
    Start address
uint32_t endAddr
    End address
uint32_t key[4]
    Encryption key
uint32_t counter[2]
    Encryption counter
struct _otfad_config
    #include <fsl_otfad.h> OTFAD configuration structure.
```

### Public Members

```
bool forceSVM
    Force entry into SVM after a write
bool forceLDM
    Force entry into LDM after a write
bool restrictedRegAccess
    Restricted register access enable
bool enableOTFAD
    OTFAD has decryption enabled
```

## 2.47 PDM: Microphone Interface

### 2.48 PDM Driver

```
void PDM_Init(PDM_Type *base, const pdm_config_t *config)
```

Initializes the PDM peripheral.

Ungates the PDM clock, resets the module, and configures PDM with a configuration structure. The configuration structure can be custom filled or set with default values by PDM\_GetDefaultConfig().

---

**Note:** This API should be called at the beginning of the application to use the PDM driver. Otherwise, accessing the PDM module can cause a hard fault because the clock is not enabled.

---

#### Parameters

- base – PDM base pointer
- config – PDM configuration structure.

`void PDM_Deinit(PDM_Type *base)`

De-initializes the PDM peripheral.

This API gates the PDM clock. The PDM module can't operate unless PDM\_Init is called to enable the clock.

**Parameters**

- base – PDM base pointer

`static inline void PDM_Reset(PDM_Type *base)`

Resets the PDM module.

**Parameters**

- base – PDM base pointer

`static inline void PDM_Enable(PDM_Type *base, bool enable)`

Enables/disables PDM interface.

**Parameters**

- base – PDM base pointer
- enable – True means PDM interface is enabled, false means PDM interface is disabled.

`static inline void PDM_EnableDebugMode(PDM_Type *base, bool enable)`

Enables/disables debug mode for PDM. The PDM interface cannot enter debug mode once in Disable/Low Leakage or Low Power mode.

**Parameters**

- base – PDM base pointer
- enable – True means PDM interface enter debug mode, false means PDM interface in normal mode.

`static inline void PDM_EnableInDebugMode(PDM_Type *base, bool enable)`

Enables/disables PDM interface in debug mode.

**Parameters**

- base – PDM base pointer
- enable – True means PDM interface is enabled debug mode, false means PDM interface is disabled after after completing the current frame in debug mode.

`static inline void PDM_EnterLowLeakageMode(PDM_Type *base, bool enable)`

Enables/disables PDM interface disable/Low Leakage mode.

**Parameters**

- base – PDM base pointer
- enable – True means PDM interface is in disable/low leakage mode, False means PDM interface is in normal mode.

`static inline void PDM_EnableChannel(PDM_Type *base, uint8_t channel, bool enable)`

Enables/disables the PDM channel.

**Parameters**

- base – PDM base pointer
- channel – PDM channel number need to enable or disable.
- enable – True means enable PDM channel, false means disable.

```
void PDM_SetChannelConfig(PDM_Type *base, uint32_t channel, const pdm_channel_config_t
                        *config)
```

PDM one channel configurations.

#### Parameters

- base – PDM base pointer
- config – PDM channel configurations.
- channel – channel number. after completing the current frame in debug mode.

```
status_t PDM_SetSampleRateConfig(PDM_Type *base, uint32_t sourceClock_HZ, uint32_t
                                sampleRate_HZ)
```

PDM set sample rate.

---

**Note:** This function is depend on the configuration of the PDM and PDM channel, so the correct call sequence is

```
PDM_Init(base, pdmConfig)
PDM_SetChannelConfig(base, channel, &channelConfig)
PDM_SetSampleRateConfig(base, source, sampleRate)
```

---

#### Parameters

- base – PDM base pointer
- sourceClock\_HZ – PDM source clock frequency.
- sampleRate\_HZ – PDM sample rate.

```
status_t PDM_SetSampleRate(PDM_Type *base, uint32_t enableChannelMask,
                            pdm_df_quality_mode_t qualityMode, uint8_t osr, uint32_t clkDiv)
```

PDM set sample rate.

#### Deprecated:

Do not use this function. It has been superceded by PDM\_SetSampleRateConfig

#### Parameters

- base – PDM base pointer
- enableChannelMask – PDM channel enable mask.
- qualityMode – quality mode.
- osr – cic oversample rate
- clkDiv – clock divider

```
uint32_t PDM_GetInstance(PDM_Type *base)
```

Get the instance number for PDM.

#### Parameters

- base – PDM base pointer.

```
static inline uint32_t PDM_GetStatus(PDM_Type *base)
```

Gets the PDM internal status flag. Use the Status Mask in `_pdm_internal_status` to get the status value needed.

#### Parameters

- base – PDM base pointer

**Returns**

PDM status flag value.

```
static inline uint32_t PDM_GetFifoStatus(PDM_Type *base)
```

Gets the PDM FIFO status flag. Use the Status Mask in `_pdm_fifo_status` to get the status value needed.

**Parameters**

- base – PDM base pointer

**Returns**

FIFO status.

```
static inline uint32_t PDM_GetRangeStatus(PDM_Type *base)
```

Gets the PDM Range status flag. Use the Status Mask in `_pdm_range_status` to get the status value needed.

**Parameters**

- base – PDM base pointer

**Returns**

output status.

```
static inline void PDM_ClearStatus(PDM_Type *base, uint32_t mask)
```

Clears the PDM Tx status.

**Parameters**

- base – PDM base pointer
- mask – State mask. It can be a combination of the status between `kPDM_StatusFrequencyLow` and `kPDM_StatusCh7FifoDataAvaliable`.

```
static inline void PDM_ClearFIFOStatus(PDM_Type *base, uint32_t mask)
```

Clears the PDM Tx status.

**Parameters**

- base – PDM base pointer
- mask – State mask. It can be a combination of the status in `_pdm_fifo_status`.

```
static inline void PDM_ClearRangeStatus(PDM_Type *base, uint32_t mask)
```

Clears the PDM range status.

**Parameters**

- base – PDM base pointer
- mask – State mask. It can be a combination of the status in `_pdm_range_status`.

```
void PDM_EnableInterrupts(PDM_Type *base, uint32_t mask)
```

Enables the PDM interrupt requests.

**Parameters**

- base – PDM base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - `kPDM_ErrorInterruptEnable`
  - `kPDM_FIFOInterruptEnable`

```
static inline void PDM_DisableInterrupts(PDM_Type *base, uint32_t mask)
```

Disables the PDM interrupt requests.

**Parameters**

- base – PDM base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kPDM\_ErrorInterruptEnable
  - kPDM\_FIFOInterruptEnable

```
static inline void PDM_EnableDMA(PDM_Type *base, bool enable)
```

Enables/disables the PDM DMA requests.

**Parameters**

- base – PDM base pointer
- enable – True means enable DMA, false means disable DMA.

```
static inline uint32_t PDM_GetDataRegisterAddress(PDM_Type *base, uint32_t channel)
```

Gets the PDM data register address.

This API is used to provide a transfer address for the PDM DMA transfer configuration.

**Parameters**

- base – PDM base pointer.
- channel – Which data channel used.

**Returns**

data register address.

```
void PDM_ReadFifo(PDM_Type *base, uint32_t startChannel, uint32_t channelNums, void  
                 *buffer, size_t size, uint32_t dataWidth)
```

PDM read fifo.

---

**Note:** : This function support 16 bit only for IP version that only supports 16bit.

---

**Parameters**

- base – PDM base pointer.
- startChannel – start channel number.
- channelNums – total enabled channelnums.
- buffer – received buffer address.
- size – number of samples to read.
- dataWidth – sample width.

```
void PDM_SetChannelGain(PDM_Type *base, uint32_t channel, pdm_df_output_gain_t gain)
```

Set the PDM channel gain.

Please note for different quality mode, the valid gain value is different, reference RM for detail.

**Parameters**

- base – PDM base pointer.
- channel – PDM channel index.



- gain – channel gain, the register gain value range is 0 - 15.

```
void PDM_TransferCreateHandle(PDM_Type *base, pdm_handle_t *handle,  
                             pdm_transfer_callback_t callback, void *userData)
```

Initializes the PDM handle.

This function initializes the handle for the PDM transactional APIs. Call this function once to get the handle initialized.

#### Parameters

- base – PDM base pointer.
- handle – PDM handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function.

```
status_t PDM_TransferSetChannelConfig(PDM_Type *base, pdm_handle_t *handle, uint32_t  
                                     channel, const pdm_channel_config_t *config, uint32_t  
                                     format)
```

PDM set channel transfer config.

#### Parameters

- base – PDM base pointer.
- handle – PDM handle pointer.
- channel – PDM channel.
- config – channel config.
- format – data format, support data width configurations, \_pdm\_data\_width.

#### Return values

kStatus\_PDM\_ChannelConfig\_Failed – or kStatus\_Success.

```
status_t PDM_TransferReceiveNonBlocking(PDM_Type *base, pdm_handle_t *handle,  
                                         pdm_transfer_t *xfer)
```

Performs an interrupt non-blocking receive transfer on PDM.

---

**Note:** This API returns immediately after the transfer initiates. Call the PDM\_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus\_PDM\_Busy, the transfer is finished.

---

#### Parameters

- base – PDM base pointer
- handle – Pointer to the pdm\_handle\_t structure which stores the transfer state.
- xfer – Pointer to the pdm\_transfer\_t structure.

#### Return values

- kStatus\_Success – Successfully started the data receive.
- kStatus\_PDM\_Busy – Previous receive still not finished.

```
void PDM_TransferAbortReceive(PDM_Type *base, pdm_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 – PDM base pointer
- handle – Pointer to the `pdm_handle_t` structure which stores the transfer state.

`void PDM_TransferHandleIRQ(PDM_Type *base, pdm_handle_t *handle)`

Tx interrupt handler.

### Parameters

- base – PDM base pointer.
- handle – Pointer to the `pdm_handle_t` structure.

FSL\_PDM\_DRIVER\_VERSION

Version 2.9.3

PDM return status.

*Values:*

enumerator `kStatus_PDM_Busy`

PDM is busy.

enumerator `kStatus_PDM_CLK_LOW`

PDM clock frequency low

enumerator `kStatus_PDM_FIFO_ERROR`

PDM FIFO underrun or overflow

enumerator `kStatus_PDM_QueueFull`

PDM FIFO underrun or overflow

enumerator `kStatus_PDM_Idle`

PDM is idle

enumerator `kStatus_PDM_Output_ERROR`

PDM is output error

enumerator `kStatus_PDM_ChannelConfig_Failed`

PDM channel config failed

enum `_pdm_interrupt_enable`

The PDM interrupt enable flag.

*Values:*

enumerator `kPDM_ErrorInterruptEnable`

PDM channel error interrupt enable.

enumerator `kPDM_FIFOInterruptEnable`

PDM channel FIFO interrupt

enum `_pdm_internal_status`

The PDM status.

*Values:*

enumerator kPDM\_StatusDfBusyFlag  
Decimation filter is busy processing data

enumerator kPDM\_StatusFrequencyLow  
Mic app clock frequency not high enough

enumerator kPDM\_StatusCh0FifoDataAvaliable  
channel 0 fifo data reached watermark level

enumerator kPDM\_StatusCh1FifoDataAvaliable  
channel 1 fifo data reached watermark level

enumerator kPDM\_StatusCh2FifoDataAvaliable  
channel 2 fifo data reached watermark level

enumerator kPDM\_StatusCh3FifoDataAvaliable  
channel 3 fifo data reached watermark level

enum \_\_pdm\_channel\_enable\_mask

PDM channel enable mask.

*Values:*

enumerator kPDM\_EnableChannel0  
channgel 0 enable mask

enumerator kPDM\_EnableChannel1  
channgel 1 enable mask

enumerator kPDM\_EnableChannel2  
channgel 2 enable mask

enumerator kPDM\_EnableChannel3  
channgel 3 enable mask

enumerator kPDM\_EnableChannelAll

enum \_\_pdm\_fifo\_status

The PDM fifo status.

*Values:*

enumerator kPDM\_FifoStatusUnderflowCh0  
channel0 fifo status underflow

enumerator kPDM\_FifoStatusUnderflowCh1  
channel1 fifo status underflow

enumerator kPDM\_FifoStatusUnderflowCh2  
channel2 fifo status underflow

enumerator kPDM\_FifoStatusUnderflowCh3  
channel3 fifo status underflow

enumerator kPDM\_FifoStatusOverflowCh0  
channel0 fifo status overflow

enumerator kPDM\_FifoStatusOverflowCh1  
channel1 fifo status overflow

enumerator kPDM\_FifoStatusOverflowCh2  
channel2 fifo status overflow

enumerator kPDM\_FifoStatusOverflowCh3  
channel3 fifo status overflow

enum \_\_pdm\_range\_status

The PDM output status.

*Values:*

enumerator kPDM\_RangeStatusUnderFlowCh0  
channel0 range status underflow

enumerator kPDM\_RangeStatusUnderFlowCh1  
channel1 range status underflow

enumerator kPDM\_RangeStatusUnderFlowCh2  
channel2 range status underflow

enumerator kPDM\_RangeStatusUnderFlowCh3  
channel3 range status underflow

enumerator kPDM\_RangeStatusOverFlowCh0  
channel0 range status overflow

enumerator kPDM\_RangeStatusOverFlowCh1  
channel1 range status overflow

enumerator kPDM\_RangeStatusOverFlowCh2  
channel2 range status overflow

enumerator kPDM\_RangeStatusOverFlowCh3  
channel3 range status overflow

enum \_\_pdm\_dc\_removal

PDM DC removal configurations.

*Values:*

enumerator kPDM\_DcRemovalCutOff20Hz  
DC removal cut off 20HZ

enumerator kPDM\_DcRemovalCutOff13Hz  
DC removal cut off 13.3HZ

enumerator kPDM\_DcRemovalCutOff40Hz  
DC removal cut off 40HZ

enumerator kPDM\_DcRemovalBypass  
DC removal bypass

enum \_\_pdm\_df\_quality\_mode

PDM decimation filter quality mode.

*Values:*

enumerator kPDM\_QualityModeMedium  
quality mode medium

enumerator kPDM\_QualityModeHigh  
quality mode high

enumerator kPDM\_QualityModeLow  
quality mode low

enumerator kPDM\_QualityModeVeryLow0  
quality mode very low0

enumerator kPDM\_QualityModeVeryLow1  
quality mode very low1

enumerator kPDM\_QualityModeVeryLow2  
quality mode very low2

enum \_\_pdm\_qulaity\_\_mode\_k\_factor  
PDM quality mode K factor.

*Values:*

enumerator kPDM\_QualityModeHighKFactor  
high quality mode K factor = 1 / 2

enumerator kPDM\_QualityModeMediumKFactor  
medium/very low0 quality mode K factor = 2 / 2

enumerator kPDM\_QualityModeLowKFactor  
low/very low1 quality mode K factor = 4 / 2

enumerator kPDM\_QualityModeVeryLow2KFactor  
very low2 quality mode K factor = 8 / 2

enum \_\_pdm\_df\_output\_gain  
PDM decimation filter output gain.

*Values:*

enumerator kPDM\_DfOutputGain0  
Decimation filter output gain 0

enumerator kPDM\_DfOutputGain1  
Decimation filter output gain 1

enumerator kPDM\_DfOutputGain2  
Decimation filter output gain 2

enumerator kPDM\_DfOutputGain3  
Decimation filter output gain 3

enumerator kPDM\_DfOutputGain4  
Decimation filter output gain 4

enumerator kPDM\_DfOutputGain5  
Decimation filter output gain 5

enumerator kPDM\_DfOutputGain6  
Decimation filter output gain 6

enumerator kPDM\_DfOutputGain7  
Decimation filter output gain 7

enumerator kPDM\_DfOutputGain8  
Decimation filter output gain 8

enumerator kPDM\_DfOutputGain9  
Decimation filter output gain 9

enumerator kPDM\_DfOutputGain10  
Decimation filter output gain 10

enumerator kPDM\_DfOutputGain11  
Decimation filter output gain 11

enumerator kPDM\_DfOutputGain12  
Decimation filter output gain 12

enumerator kPDM\_DfOutputGain13  
Decimation filter output gain 13

enumerator kPDM\_DfOutputGain14  
Decimation filter output gain 14

enumerator kPDM\_DfOutputGain15  
Decimation filter output gain 15

enum \_\_pdm\_data\_width  
PDM data width.

*Values:*

enumerator kPDM\_DataWwidth24  
PDM data width 24bit

enumerator kPDM\_DataWwidth32  
PDM data width 32bit

typedef enum *\_pdm\_dc\_removal* pdm\_dc\_removal\_t  
PDM DC removal configurations.

typedef enum *\_pdm\_df\_quality\_mode* pdm\_df\_quality\_mode\_t  
PDM decimation filter quality mode.

typedef enum *\_pdm\_df\_output\_gain* pdm\_df\_output\_gain\_t  
PDM decimation filter output gain.

typedef struct *\_pdm\_channel\_config* pdm\_channel\_config\_t  
PDM channel configurations.

typedef struct *\_pdm\_config* pdm\_config\_t  
PDM user configuration structure.

typedef struct *\_pdm\_transfer* pdm\_transfer\_t  
PDM SDMA transfer structure.

typedef struct *\_pdm\_handle* pdm\_handle\_t  
PDM handle.

typedef void (\*pdm\_transfer\_callback\_t)(PDM\_Type \*base, *pdm\_handle\_t* \*handle, *status\_t* status, void \*userData)  
PDM transfer callback prototype.

PDM\_XFER\_QUEUE\_SIZE  
PDM XFER QUEUE SIZE.

struct \_\_pdm\_channel\_config  
*#include <fsl\_pdm.h>* PDM channel configurations.

### Public Members

*pdm\_dc\_removal\_t* outputCutOffFreq  
PDM output DC removal cut off frequency

*pdm\_df\_output\_gain\_t* gain  
Decimation Filter Output Gain

struct \_\_pdm\_config  
#include <fsl\_pdm.h> PDM user configuration structure.

### Public Members

bool enableDoze  
This module will enter disable/low leakage mode if DOZEN is active with ipg\_doze is asserted

bool enableFilterBypass  
Switchable bypass path for the decimation filter

uint8\_t fifoWatermark  
Watermark value for FIFO

*pdm\_df\_quality\_mode\_t* qualityMode  
Quality mode

uint8\_t cicOverSampleRate  
CIC filter over sampling rate

struct \_\_pdm\_transfer  
#include <fsl\_pdm.h> PDM SDMA transfer structure.

### Public Members

volatile uint8\_t \*data  
Data start address to transfer.

volatile size\_t dataSize  
Total Transfer bytes size.

struct \_\_pdm\_handle  
#include <fsl\_pdm.h> PDM handle structure.

### Public Members

uint32\_t state  
Transfer status

*pdm\_transfer\_callback\_t* callback  
Callback function called at transfer event

void \*userData  
Callback parameter passed to callback function

*pdm\_transfer\_t* pdmQueue[(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

`uint32_t format`  
data format

`uint8_t watermark`  
Watermark value

`uint8_t startChannel`  
end channel

`uint8_t channelNums`  
Enabled channel number

## 2.49 PDM EDMA Driver

`void PDM_TransferInstallEDMATCDMemory(pdm_edma_handle_t *handle, void *tcdAddr, size_t tcdNum)`

Install EDMA descriptor memory.

### Parameters

- `handle` – Pointer to EDMA channel transfer handle.
- `tcdAddr` – EDMA head descriptor address.
- `tcdNum` – EDMA link descriptor address.

`void PDM_TransferCreateHandleEDMA(PDM_Type *base, pdm_edma_handle_t *handle, pdm_edma_callback_t callback, void *userData, edma_handle_t *dmaHandle)`

Initializes the PDM Rx eDMA handle.

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

### Parameters

- `base` – PDM base pointer.
- `handle` – PDM eDMA handle pointer.
- `callback` – Pointer to user callback function.
- `userData` – User parameter passed to the callback function.
- `dmaHandle` – eDMA handle pointer, this handle shall be static allocated by users.

`void PDM_TransferSetMultiChannelInterleaveType(pdm_edma_handle_t *handle, pdm_edma_multi_channel_interleave_t multiChannelInterleaveType)`

Initializes the multi PDM channel interleave type.

This function initializes the PDM DMA handle member `interleaveType`, it shall be called only when application would like to use type `kPDM_EDMAMultiChannelInterleavePerChannelBlock`, since the default `interleaveType` is `kPDM_EDMAMultiChannelInterleavePerChannelSample` always

### Parameters

- `handle` – PDM eDMA handle pointer.



- multiChannelInterleaveType – Multi channel interleave type.

```
void PDM_TransferSetChannelConfigEDMA(PDM_Type *base, pdm_edma_handle_t *handle,
                                     uint32_t channel, const pdm_channel_config_t
                                     *config)
```

Configures the PDM channel.

#### Parameters

- base – PDM base pointer.
- handle – PDM eDMA handle pointer.
- channel – channel index.
- config – pdm channel configurations.

```
status_t PDM_TransferReceiveEDMA(PDM_Type *base, pdm_edma_handle_t *handle,
                                 pdm_edma_transfer_t *xfer)
```

Performs a non-blocking PDM receive using eDMA.

Mcaro MCUX\_SDK\_PDM\_EDMA\_PDM\_ENABLE\_INTERNAL can control whether PDM is enabled internally or externally.

- Scatter gather case: This function support dynamic scatter gather and static scatter gather, a. for the dynamic scatter gather case: Application should call PDM\_TransferReceiveEDMA function continuously to make sure new receive request is submit before the previous one finish. b. for the static scatter gather case: Application should use the link transfer feature and make sure a loop link transfer is provided, such as:

```
pdm_edma_transfer_t pdmXfer[2] =
{
    {
        .data = s_buffer,
        .dataSize = BUFFER_SIZE,
        .linkTransfer = &pdmXfer[1],
    },
    {
        .data = &s_buffer[BUFFER_SIZE],
        .dataSize = BUFFER_SIZE,
        .linkTransfer = &pdmXfer[0]
    },
};
```

- Multi channel case: This function support receive multi pdm channel data, for example, if two channel is requested,

```
PDM_TransferSetChannelConfigEDMA(DEMO_PDM, &s_pdmRxHandle_0, DEMO_PDM_
↳ENABLE_CHANNEL_0, &channelConfig);
PDM_TransferSetChannelConfigEDMA(DEMO_PDM, &s_pdmRxHandle_0, DEMO_PDM_
↳ENABLE_CHANNEL_1, &channelConfig);
PDM_TransferReceiveEDMA(DEMO_PDM, &s_pdmRxHandle_0, pdmXfer);
```

The output data will be formatted as below if handle->interleaveType =

**Note:** This interface returns immediately after the transfer initiates. Call the PDM\_GetReceiveRemainingBytes to poll the transfer status and check whether the PDM transfer is finished.

`void PDM_TransferTerminateReceiveEDMA(PDM_Type *base, pdm_edma_handle_t *handle)`  
Terminate all PDM receive.

This function will clear all transfer slots buffered in the pdm queue. If users only want to abort the current transfer slot, please call `PDM_TransferAbortReceiveEDMA`.

#### Parameters

- base – PDM base pointer.
- handle – PDM eDMA handle pointer.

`void PDM_TransferAbortReceiveEDMA(PDM_Type *base, pdm_edma_handle_t *handle)`  
Aborts a PDM receive using eDMA.

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call `PDM_TransferTerminateReceiveEDMA`.

#### Parameters

- base – PDM base pointer
- handle – PDM eDMA handle pointer.

`status_t PDM_TransferGetReceiveCountEDMA(PDM_Type *base, pdm_edma_handle_t *handle, size_t *count)`

Gets byte count received by PDM.

#### Parameters

- base – PDM base pointer
- handle – PDM eDMA handle pointer.
- count – Bytes count received by PDM.

#### Return values

- `kStatus_Success` – Succeed get the transfer count.
- `kStatus_NoTransferInProgress` – There is no non-blocking transaction in progress.

`FSL_PDM_EDMA_DRIVER_VERSION`  
Version 2.6.5

`enum _pdm_edma_multi_channel_interleave`  
pdm multi channel interleave type

*Values:*

enumerator `kPDM_EDMAMultiChannelInterleavePerChannelSample`

enumerator `kPDM_EDMAMultiChannelInterleavePerChannelBlock`

`typedef struct _pdm_edma_handle` `pdm_edma_handle_t`  
PDM edma handler.

`typedef enum _pdm_edma_multi_channel_interleave` `pdm_edma_multi_channel_interleave_t`  
pdm multi channel interleave type

`typedef struct _pdm_edma_transfer` `pdm_edma_transfer_t`  
PDM edma transfer.

`typedef void (*pdm_edma_callback_t)(PDM_Type *base, pdm_edma_handle_t *handle, status_t status, void *userData)`

PDM eDMA transfer callback function for finish and error.

MCUX\_SDK\_PDM\_EDMA\_PDM\_ENABLE\_INTERNAL

the PDM enable position When calling PDM\_TransferReceiveEDMA

struct \_\_pdm\_edma\_transfer

*#include <fsl\_pdm\_edma.h>* PDM edma transfer.

### Public Members

volatile uint8\_t \*data

Data start address to transfer.

volatile size\_t dataSize

Total Transfer bytes size.

struct \_\_pdm\_edma\_transfer \*linkTransfer

linked transfer configurations

struct \_\_pdm\_edma\_handle

*#include <fsl\_pdm\_edma.h>* PDM DMA transfer handle, users should not touch the content of the handle.

### Public Members

*edma\_handle\_t* \*dmaHandle

DMA handler for PDM send

uint8\_t count

The transfer data count in a DMA request

uint32\_t receivedBytes

total transfer count

uint32\_t state

Internal state for PDM eDMA transfer

*pdm\_edma\_callback\_t* callback

Callback for users while transfer finish or error occurs

bool isLoopTransfer

loop transfer

void \*userData

User callback parameter

*edma\_tcd\_t* \*tcd

TCD pool for eDMA transfer.

uint32\_t tcdNum

TCD number

uint32\_t tcdUser

Index for user to queue transfer.

uint32\_t tcdDriver

Index for driver to get the transfer data and size

volatile uint32\_t tcdUsedNum

Index for user to queue transfer.

*pdm\_edma\_multi\_channel\_interleave\_t* *interleaveType*  
multi channel transfer interleave type

*uint8\_t* *endChannel*  
The last enabled channel

*uint8\_t* *channelNums*  
total channel numbers

## 2.50 RGPIO: Rapid General-Purpose Input/Output Driver

FSL\_RGPIODRIVER\_VERSION

RGPIO driver version 2.2.0.

enum *\_rgpio\_pin\_direction*

RGPIO direction definition.

*Values:*

enumerator *kRGPIO\_DigitalInput*  
Set current pin as digital input

enumerator *kRGPIO\_DigitalOutput*  
Set current pin as digital output

enum *\_rgpio\_checker\_attribute*

RGPIO checker attribute.

*Values:*

enumerator *kRGPIO\_UsernonsecureRWUsersecureRWPrivilegedsecureRW*  
User nonsecure:Read+Write; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator *kRGPIO\_UsernonsecureRUsersecureRWPrivilegedsecureRW*  
User nonsecure:Read; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator *kRGPIO\_UsernonsecureNUsersecureRWPrivilegedsecureRW*  
User nonsecure:None; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator *kRGPIO\_UsernonsecureRUsersecureRPrivilegedsecureRW*  
User nonsecure:Read; User Secure:Read; Privileged Secure:Read+Write

enumerator *kRGPIO\_UsernonsecureNUsersecureRPrivilegedsecureRW*  
User nonsecure:None; User Secure:Read; Privileged Secure:Read+Write

enumerator *kRGPIO\_UsernonsecureNUsersecureNPrivilegedsecureRW*  
User nonsecure:None; User Secure:None; Privileged Secure:Read+Write

enumerator *kRGPIO\_UsernonsecureNUsersecureNPrivilegedsecureR*  
User nonsecure:None; User Secure:None; Privileged Secure:Read

enumerator *kRGPIO\_UsernonsecureNUsersecureNPrivilegedsecureN*  
User nonsecure:None; User Secure:None; Privileged Secure:None

enumerator *kRGPIO\_IgnoreAttributeCheck*  
Ignores the attribute check

enum *\_rgpio\_interrupt\_sel*

Configures the interrupt generation condition.

*Values:*

enumerator kRGPIO\_InterruptOutput0  
Interrupt/DMA request/trigger output 0.

enumerator kRGPIO\_InterruptOutput1  
Interrupt/DMA request/trigger output 1.

enumerator kRGPIO\_InterruptOutput2  
Interrupt/DMA request/trigger output 2.

enumerator kRGPIO\_InterruptOutput3  
Interrupt/DMA request/trigger output 3.

enum \_rgpio\_interrupt\_config  
Configures the interrupt generation condition.

*Values:*

enumerator kRGPIO\_InterruptOrDMADisabled  
Interrupt/DMA request is disabled.

enumerator kRGPIO\_DMARisingEdge  
DMA request on rising edge.

enumerator kRGPIO\_DMAFallingEdge  
DMA request on falling edge.

enumerator kRGPIO\_DMAEitherEdge  
DMA request on either edge.

enumerator kRGPIO\_FlagRisingEdge  
Flag sets on rising edge.

enumerator kRGPIO\_FlagFallingEdge  
Flag sets on falling edge.

enumerator kRGPIO\_FlagEitherEdge  
Flag sets on either edge.

enumerator kRGPIO\_InterruptLogicZero  
Interrupt when logic zero.

enumerator kRGPIO\_InterruptRisingEdge  
Interrupt on rising edge.

enumerator kRGPIO\_InterruptFallingEdge  
Interrupt on falling edge.

enumerator kRGPIO\_InterruptEitherEdge  
Interrupt on either edge.

enumerator kRGPIO\_InterruptLogicOne  
Interrupt when logic one.

enumerator kRGPIO\_ActiveHighTriggerOutputEnable  
Enable active high-trigger output.

enumerator kRGPIO\_ActiveLowTriggerOutputEnable  
Enable active low-trigger output.

typedef enum \_rgpio\_pin\_direction rgpio\_pin\_direction\_t  
RGPIO direction definition.

```
typedef enum _rgpio_checker_attribute rgpio_checker_attribute_t
    RGPIO checker attribute.
```

```
typedef enum _rgpio_interrupt_sel rgpio_interrupt_sel_t
    Configures the interrupt generation condition.
```

```
typedef enum _rgpio_interrupt_config rgpio_interrupt_config_t
    Configures the interrupt generation condition.
```

```
typedef struct _rgpio_pin_config rgpio_pin_config_t
    The RGPIO pin configuration structure.
```

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT\_SetPinConfig().

```
struct _rgpio_pin_config
    #include <fsl_rgpio.h> The RGPIO pin configuration structure.
```

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT\_SetPinConfig().

### Public Members

```
rgpio_pin_direction_t pinDirection
    RGPIO direction, input or output
```

```
uint8_t outputLogic
    Set a default output logic, which has no use in input
```

## 2.51 RGPIO Driver

```
void RGPIO_PinInit(RGPIO_Type *base, uint32_t pin, const rgpio_pin_config_t *config)
```

Initializes a RGPIO pin used by the board.

To initialize the RGPIO, define a pin configuration, as either input or output, in the user file. Then, call the RGPIO\_PinInit() function.

This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
rgpio_pin_config_t config =
{
    kRGPIO_DigitalInput,
    0,
}
Define a digital output pin configuration,
rgpio_pin_config_t config =
{
    kRGPIO_DigitalOutput,
    0,
}
```

### Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- pin – RGPIO port pin number

- config – RGPIO pin configuration pointer

uint32\_t RGPIO\_GetInstance(RGPIO\_Type \*base)

Gets the RGPIO instance according to the RGPIO base.

#### Parameters

- base – RGPIO peripheral base pointer(PTA, PTB, PTC, etc.)

#### Return values

RGPIO – instance

static inline *rgpio\_pin\_direction\_t* RGPIO\_GetPinDirection(RGPIO\_Type \*base, uint32\_t pin)

Gets the current direction of a RGPIO pin.

#### Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- pin – RGPIO port pin number

#### Return values

RGPIO – pin direction

- kRGPIO\_DigitalInput: pin is configured as digital input.
- kRGPIO\_DigitalOutput: pin is configured as digital output.

static inline void RGPIO\_PinWrite(RGPIO\_Type \*base, uint32\_t pin, uint8\_t output)

Sets the output level of the multiple RGPIO pins to the logic 1 or 0.

#### Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- pin – RGPIO pin number
- output – RGPIO pin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

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

Sets the output level of the multiple RGPIO pins to the logic 1 or 0.

#### Deprecated:

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

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

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

#### Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number macro

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

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

#### Deprecated:

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

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

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

**Parameters**

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number macro

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

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

*Deprecated:*

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

**Parameters**

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number macro

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

Reverses the current output logic of the multiple RGPIO pins.

**Parameters**

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number macro

static inline void RGPIO\_TogglePinsOutput(RGPIO\_Type \*base, uint32\_t mask)

Reverses the current output logic of the multiple RGPIO pins.

*Deprecated:*

Do not use this function. It has been superceded by RGPIO\_PortToggle.

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

Reads the current input value of the RGPIO port.

**Parameters**

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- pin – RGPIO pin number

**Return values**

RGPIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

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

Reads the current input value of the RGPIO port.

*Deprecated:*

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



```
static inline void RGPIO_EnablePortInput(RGPIO_Type *base, uint32_t mask, bool enable)
```

#### Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number mask
- enable – RGPIO digital input enable/disable flag.

```
void RGPIO_CheckAttributeBytes(RGPIO_Type *base, rgpio_checker_attribute_t attribute)
```

The RGPIO module supports a device-specific number of data ports, organized as 32-bit words. Each 32-bit data port includes a GACR register, which defines the byte-level attributes required for a successful access to the RGPIO programming model. The attribute controls for the 4 data bytes in the GACR follow a standard little endian data convention.

#### Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number macro

```
static inline void RGPIO_SetPinInterruptConfig(RGPIO_Type *base, uint32_t pin,
                                             rgpio_interrupt_sel_t sel,
                                             rgpio_interrupt_config_t config)
```

Configures the gpio pin interrupt/DMA request.

#### Parameters

- base – RGPIO peripheral base pointer.
- pin – RGPIO pin number.
- sel – RGPIO pin interrupt selection(0-3).
- config – RGPIO pin interrupt configuration.

```
static inline void _SetMultipleInterruptPinsConfig(RGPIO_Type *base, uint32_t mask,
                                                  rgpio_interrupt_sel_t sel,
                                                  rgpio_interrupt_config_t config)
```

Sets the gpio interrupt configuration in ICR register for multiple pins.

#### Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number macro.
- sel – RGPIO pin interrupt selection(0-3).
- config – RGPIO pin interrupt configuration.

```
static inline uint32_t RGPIO_GetPinsInterruptFlags(RGPIO_Type *base, rgpio_interrupt_sel_t sel)
```

Reads the whole gpio status flag.

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

#### Parameters

- base – RGPIO peripheral base pointer.
- sel – RGPIO pin interrupt selection(0-3).

**Returns**

Current gpio interrupt status flags, for example, 0x00010001 means the pin 0 and 16 have the interrupt.

```
static inline void RGPIO_ClearPinsInterruptFlags(RGPIO_Type *base, rgpio_interrupt_sel_t sel,
                                                uint32_t mask)
```

Clears the multiple pin interrupt status flag.

**Parameters**

- base – RGPIO peripheral base pointer.
- sel – RGPIO pin interrupt selection(0-3).
- mask – RGPIO pin number macro.

## 2.52 SAI: Serial Audio Interface

### 2.53 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_SetMasterClockConfig(I2S_Type *base, sai_master_clock_t *config)
```

Master clock configurations.

#### Parameters

- base – SAI base pointer.
- config – master clock configurations.

```
void SAI_TxSetFifoConfig(I2S_Type *base, sai_fifo_t *config)
```

SAI transmitter fifo configurations.

#### Parameters

- base – SAI base pointer.
- config – fifo configurations.

```
void SAI_RxSetFifoConfig(I2S_Type *base, sai_fifo_t *config)
```

SAI receiver fifo configurations.

#### Parameters

- base – SAI base pointer.
- config – fifo configurations.

```
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 similiar 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 –

void SAI\_TxSetFIFOPacking(I2S\_Type \*base, *sai\_fifo\_packing\_t* pack)

Set Tx FIFO packing feature.

**Parameters**

- base – SAI base pointer.
- pack – FIFO pack type. It is element of *sai\_fifo\_packing\_t*.

void SAI\_RxSetFIFOPacking(I2S\_Type \*base, *sai\_fifo\_packing\_t* pack)

Set Rx FIFO packing feature.

**Parameters**

- base – SAI base pointer.
- pack – FIFO pack type. It is element of *sai\_fifo\_packing\_t*.

static inline void SAI\_TxSetFIFOErrorContinue(I2S\_Type \*base, bool isEnabled)

Set Tx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

**Parameters**

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI\_RxSetFIFOErrorContinue(I2S\_Type \*base, bool isEnabled)

Set Rx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

**Parameters**

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

```
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)

enumerator kSAI\_ModeSyncWithOtherTx

Synchronous with another SAI transmit

enumerator kSAI\_ModeSyncWithOtherRx

Synchronous with another SAI receiver

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

enumerator kSAI\_FIFORequestInterruptEnable  
FIFO request, means reached watermark

\_sai\_dma\_enable\_t, The DMA request sources

*Values:*

enumerator kSAI\_FIFOWarningDMAEnable  
FIFO warning caused by the DMA request

enumerator kSAI\_FIFORequestDMAEnable  
FIFO request 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\_FIFORequestFlag  
FIFO request 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\_fifo\_packing

The SAI packing mode The mode includes 8 bit and 16 bit packing.

*Values:*

enumerator kSAI\_FifoPackingDisabled  
Packing disabled

enumerator kSAI\_FifoPacking8bit  
8 bit packing enabled

enumerator kSAI\_FifoPacking16bit  
16bit packing enabled

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\_data\_pin\_state

sai data pin state definition

*Values:*

enumerator kSAI\_DataPinStateTriState

transmit data pins are tri-stated when slots are masked or channels are disabled

enumerator kSAI\_DataPinStateOutputZero

transmit data pins are never tri-stated and will output zero when slots are masked or channel disabled

enum \_sai\_fifo\_combine

sai fifo combine mode definition

*Values:*

enumerator kSAI\_FifoCombineDisabled

sai TX/RX fifo combine mode disabled

enumerator kSAI\_FifoCombineModeEnabledOnRead

sai TX fifo combine mode enabled on FIFO reads

enumerator kSAI\_FifoCombineModeEnabledOnWrite

sai TX fifo combine mode enabled on FIFO write

enumerator kSAI\_RxFifoCombineModeEnabledOnWrite

sai RX fifo combine mode enabled on FIFO write

enumerator kSAI\_RXFifoCombineModeEnabledOnRead

sai RX fifo combine mode enabled on FIFO reads

enumerator kSAI\_FifoCombineModeEnabledOnReadWrite

sai TX/RX fifo combined mode enabled on FIFO read/writes

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_bclk_source sai_bclk_source_t`  
Bit clock source.

`typedef enum _sai_reset_type sai_reset_type_t`  
The reset type.

`typedef enum _sai_fifo_packing sai_fifo_packing_t`  
The SAI packing mode The mode includes 8 bit and 16 bit packing.

`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_data_pin_state sai_data_pin_state_t`  
sai data pin state definition

`typedef enum _sai_fifo_combine sai_fifo_combine_t`  
sai fifo combine mode definition

`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_master_clock sai_master_clock_t`  
master clock configurations

`typedef struct _sai_fifo sai_fifo_t`  
sai fifo configurations

`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\_transciever\_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

bool mclkOutputEnable

Master clock output enable, true means master clock divider enabled

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



`uint32_t` masterClockHz  
Master clock frequency in Hz

`uint8_t` watermark  
Watermark value

`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_master_clock`  
*#include <fsl\_sai.h>* master clock configurations

### Public Members

`bool` mclkOutputEnable  
master clock output enable

`uint32_t` mclkHz  
target mclk frequency

`uint32_t` mclkSourceClkHz  
mclk source frequency

`struct _sai_fifo`  
*#include <fsl\_sai.h>* sai fifo configurations

### Public Members

`bool` fifoContinueOnError  
fifo continues when error occur

`sai_fifo_combine_t` fifoCombine  
fifo combine mode

`sai_fifo_packing_t` fifoPacking  
fifo packing mode

`uint8_t` fifoWatermark  
fifo watermark

`struct _sai_bit_clock`  
*#include <fsl\_sai.h>* sai bit clock configurations

### Public Members

bool bclkSrcSwap

bit clock source swap

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

bool frameSyncGenerateOnDemand

internal frame sync is generated when FIFO waring flag is clear

*sai\_clock\_polarity\_t* frameSyncPolarity

frame sync polarity

struct *\_sai\_serial\_data*

*#include <fsl\_sai.h>* sai serial data configurations

### Public Members

*sai\_data\_pin\_state\_t* dataMode

sai data pin state when slots masked or channel disabled

*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\_fifo\_t fifo  
    fifo 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

uint8\_t watermark  
Watermark value

## 2.54 SAI EDMA Driver

```
void SAI_TransferTxCreateHandleEDMA(I2S_Type *base, sai_edma_handle_t *handle,  
                                   sai_edma_callback_t callback, void *userData,  
                                   edma_handle_t *txDmaHandle)
```

Initializes the SAI eDMA 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 eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- txDmaHandle – eDMA handle pointer; this handle shall be static allocated by users.

```
void SAI_TransferRxCreateHandleEDMA(I2S_Type *base, sai_edma_handle_t *handle,
                                   sai_edma_callback_t callback, void *userData,
                                   edma_handle_t *rxDmaHandle)
```

Initializes the SAI Rx eDMA 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 eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- rxDmaHandle – eDMA handle pointer, this handle shall be static allocated by users.

```
void SAI_TransferSetInterleaveType(sai_edma_handle_t *handle, sai_edma_interleave_t
                                   interleaveType)
```

Initializes the SAI interleave type.

This function initializes the SAI DMA handle member interleaveType, it shall be called only when application would like to use type kSAI\_EDMAInterleavePerChannelBlock, since the default interleaveType is kSAI\_EDMAInterleavePerChannelSample always

#### Parameters

- handle – SAI eDMA handle pointer.
- interleaveType – Multi channel interleave type.

```
void SAI_TransferTxSetConfigEDMA(I2S_Type *base, sai_edma_handle_t *handle,
                                 sai_transceiver_t *saiConfig)
```

Configures the SAI Tx.

**Note:** SAI eDMA supports data transfer in a multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of sai\_transceiver\_t with the corresponding values of \_sai\_channel\_mask enum, enable the FIFO Combine mode by assigning kSAI\_FifoCombineModeEnabledOnWrite to the fifoCombine member of sai\_fifo\_combine\_t which is a member of sai\_transceiver\_t. This is an example of multi-channel data transfer configuration step.

```
sai_transceiver_t config;
SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits, kSAI_Stereo, kSAI_Channel0Mask|kSAI_
↪Channel1Mask);
config.fifo.fifoCombine = kSAI_FifoCombineModeEnabledOnWrite;
SAI_TransferTxSetConfigEDMA(I2S0, &edmaHandle, &config);
```

#### Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- saiConfig – sai configurations.

```
void SAI_TransferRxSetConfigEDMA(I2S_Type *base, sai_edma_handle_t *handle,  
                                sai_transceiver_t *saiConfig)
```

Configures the SAI Rx.

---

**Note:** SAI eDMA supports data transfer in a multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of sai\_transceiver\_t with the corresponding values of \_sai\_channel\_mask enum, enable the FIFO Combine mode by assigning kSAI\_FifoCombineModeEnabledOnRead to the fifoCombine member of sai\_fifo\_combine\_t which is a member of sai\_transceiver\_t. This is an example of multi-channel data transfer configuration step.

```
sai_transceiver_t config;  
SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits, kSAI_Stereo, kSAI_Channel0Mask|kSAI_  
↪Channel1Mask);  
config.fifo.fifoCombine = kSAI_FifoCombineModeEnabledOnRead;  
SAI_TransferRxSetConfigEDMA(I2S0, &edmaHandle, &config);
```

---

### Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- saiConfig – sai configurations.

```
status_t SAI_TransferSendEDMA(I2S_Type *base, sai_edma_handle_t *handle, sai_transfer_t  
                             *xfer)
```

Performs a non-blocking SAI transfer using DMA.

This function support multi channel transfer,

- a. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers
- b. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

---

**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 eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure.

### Return values

- kStatus\_\_Success – Start a SAI eDMA send successfully.
- kStatus\_\_InvalidArgument – The input argument is invalid.
- kStatus\_\_TxBusy – SAI is busy sending data.

*status\_t* SAI\_TransferReceiveEDMA(I2S\_Type \*base, *sai\_edma\_handle\_t* \*handle, *sai\_transfer\_t* \*xfer)

Performs a non-blocking SAI receive using eDMA.

This function support multi channel transfer,

- a. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers
- b. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

---

**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 eDMA handle pointer.
- xfer – Pointer to DMA transfer structure.

#### Return values

- kStatus\_Success – Start a SAI eDMA receive successfully.
- kStatus\_InvalidArgument – The input argument is invalid.
- kStatus\_RxBusy – SAI is busy receiving data.

*status\_t* SAI\_TransferSendLoopEDMA(I2S\_Type \*base, *sai\_edma\_handle\_t* \*handle, *sai\_transfer\_t* \*xfer, *uint32\_t* loopTransferCount)

Performs a non-blocking SAI loop transfer using eDMA.

Once the loop transfer start, application can use function SAI\_TransferAbortSendEDMA to stop the loop transfer.

---

**Note:** This function support loop transfer only,such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in *sai\_edma\_handle\_t*, so application could redefine the SAI\_XFER\_QUEUE\_SIZE to determine the proper TCD pool size. This function support one sai channel only.

---

#### Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure, should be a array with elements counts >=1(loopTransferCount).
- loopTransferCount – the counts of xfer array.

#### Return values

- kStatus\_Success – Start a SAI eDMA send successfully.
- kStatus\_InvalidArgument – The input argument is invalid.

```
status_t SAI_TransferReceiveLoopEDMA(I2S_Type *base, sai_edma_handle_t *handle,  
                                     sai_transfer_t *xfer, uint32_t loopTransferCount)
```

Performs a non-blocking SAI loop transfer using eDMA.

Once the loop transfer start, application can use function SAI\_TransferAbortReceiveEDMA to stop the loop transfer.

---

**Note:** This function support loop transfer only, such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in sai\_edma\_handle\_t, so application could redefine the SAI\_XFER\_QUEUE\_SIZE to determine the proper TCD pool size. This function support one sai channel only.

---

#### Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure, should be a array with elements counts >=1(loopTransferCount).
- loopTransferCount – the counts of xfer array.

#### Return values

- kStatus\_Success – Start a SAI eDMA receive successfully.
- kStatus\_InvalidArgument – The input argument is invalid.

```
void SAI_TransferTerminateSendEDMA(I2S_Type *base, sai_edma_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\_TransferAbortSendEDMA.

#### Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

```
void SAI_TransferTerminateReceiveEDMA(I2S_Type *base, sai_edma_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\_TransferAbortReceiveEDMA.

#### Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

```
void SAI_TransferAbortSendEDMA(I2S_Type *base, sai_edma_handle_t *handle)
```

Aborts a SAI transfer using eDMA.

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI\_TransferTerminateSendEDMA.

#### Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.



`void SAI_TransferAbortReceiveEDMA(I2S_Type *base, sai_edma_handle_t *handle)`

Aborts a SAI receive using eDMA.

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call `SAI_TransferTerminateReceiveEDMA`.

**Parameters**

- `base` – SAI base pointer
- `handle` – SAI eDMA handle pointer.

`status_t SAI_TransferGetSendCountEDMA(I2S_Type *base, sai_edma_handle_t *handle, size_t *count)`

Gets byte count sent by SAI.

**Parameters**

- `base` – SAI base pointer.
- `handle` – SAI eDMA handle pointer.
- `count` – Bytes count sent by SAI.

**Return values**

- `kStatus_Success` – Succeed get the transfer count.
- `kStatus_NoTransferInProgress` – There is no non-blocking transaction in progress.

`status_t SAI_TransferGetReceiveCountEDMA(I2S_Type *base, sai_edma_handle_t *handle, size_t *count)`

Gets byte count received by SAI.

**Parameters**

- `base` – SAI base pointer
- `handle` – SAI eDMA handle pointer.
- `count` – Bytes count received by SAI.

**Return values**

- `kStatus_Success` – Succeed get the transfer count.
- `kStatus_NoTransferInProgress` – There is no non-blocking transaction in progress.

`uint32_t SAI_TransferGetValidTransferSlotsEDMA(I2S_Type *base, sai_edma_handle_t *handle)`

Gets valid transfer slot.

This function can be used to query the valid transfer request slot that the application can submit. It should be called in the critical section, that means the application could call it in the corresponding callback function or disable IRQ before calling it in the application, otherwise, the returned value may not correct.

**Parameters**

- `base` – SAI base pointer
- `handle` – SAI eDMA handle pointer.

**Return values**

`valid` – slot count that application submit.

`FSL_SAI_EDMA_DRIVER_VERSION`

Version 2.7.3

enum *\_sai\_edma\_interleave*

sai interleave type

*Values:*

enumerator kSAI\_EDMAInterleavePerChannelSample

enumerator kSAI\_EDMAInterleavePerChannelBlock

typedef struct *sai\_edma\_handle* *sai\_edma\_handle\_t*

typedef void (\**sai\_edma\_callback\_t*)(I2S\_Type \**base*, *sai\_edma\_handle\_t* \**handle*, *status\_t* *status*, void \**userData*)

SAI eDMA transfer callback function for finish and error.

typedef enum *\_sai\_edma\_interleave* *sai\_edma\_interleave\_t*

sai interleave type

MCUX\_SDK\_SAI\_EDMA\_RX\_ENABLE\_INTERNAL

the SAI enable position When calling SAI\_TransferReceiveEDMA

MCUX\_SDK\_SAI\_EDMA\_TX\_ENABLE\_INTERNAL

the SAI enable position When calling SAI\_TransferSendEDMA

struct *sai\_edma\_handle*

*#include <fsl\_sai\_edma.h>* SAI DMA transfer handle, users should not touch the content of the handle.

### Public Members

*edma\_handle\_t* \**dmaHandle*

DMA handler for SAI send

uint8\_t *nbytes*

eDMA minor byte transfer count initially configured.

uint8\_t *bytesPerFrame*

Bytes in a frame

uint8\_t *channelMask*

Enabled channel mask value, reference *\_sai\_channel\_mask*

uint8\_t *channelNums*

total enabled channel nums

uint8\_t *channel*

Which data channel

uint8\_t *count*

The transfer data count in a DMA request

uint32\_t *state*

Internal state for SAI eDMA transfer

*sai\_edma\_callback\_t* *callback*

Callback for users while transfer finish or error occurs

void \**userData*

User callback parameter

uint8\_t *tcd*[((4U) + 1U) \* sizeof(*edma\_tcd\_t*)]

TCD pool for eDMA transfer.

*sai\_transfer\_t* saiQueue[(4U)]  
 Transfer queue storing queued transfer.

*size\_t* transferSize[(4U)]  
 Data bytes need to transfer

*sai\_edma\_interleave\_t* interleaveType  
 Transfer interleave type

*volatile uint8\_t* queueUser  
 Index for user to queue transfer.

*volatile uint8\_t* queueDriver  
 Index for driver to get the transfer data and size

## 2.55 SAR\_ADC: SAR\_ADC Module

`void ADC_GetDefaultConfig(adc_config_t *config)`

This function is used to get available predefined configurations for the ADC initialization.

### Parameters

- *config* – Pointer to the ADC configuration structure, please refer to *adc\_config\_t* for details.

`void ADC_Init(ADC_Type *base, const adc_config_t *config)`

This function is used to initialize the ADC.

### Parameters

- *base* – ADC peripheral base address.
- *config* – Pointer to the ADC configuration structure, please refer to *adc\_config\_t* for details.

`void ADC_Deinit(ADC_Type *base)`

This function is used to de-initialize the ADC.

### Parameters

- *base* – ADC peripheral base address.

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

This function is used to enter or exit power-down mode.

After the release of the reset, the ADC analog module will be kept in power-down mode by default. The power-down mode can be set anytime. However, ADC can enter the power-down mode successfully only after completion of an ongoing conversion (if there is one). In scan mode, the ongoing operation should be aborted manually before or after switching mode. If the power-down mode is entered by setting MCR[PWDN], the process running in the previous mode must be restarted manually (by setting the appropriate START bit in the MCR register) after exiting power-down mode.

---

**Note:** After setting the ADC mode, it is recommended to use the function `ADC_GetAdcState` to query whether the ADC has correctly entered the mode.

---

### Parameters

- *base* – ADC peripheral base address.
- *enable* – Indicates whether to enter or exit power-down mode.

- **true** Request to enter power-down mode.
- **false** When ADC status is in power-down mode (MSR[ADCSTATUS] = 001b), start ADC transition to IDLE mode.

static inline void ADC\_SetOperatingClock(ADC\_Type \*base, *adc\_clock\_frequency\_t* clockSelect)

This function is used to select the ADC operating clock.

---

**Note:** Needs to enter power-down mode before changing the ADC internal operating clock.

---

#### Parameters

- base – ADC peripheral base address.
- clockSelect – ADC clock frequency selection, please refer to *adc\_clock\_frequency\_t* for details.

static inline *adc\_state\_t* ADC\_GetAdcState(ADC\_Type \*base)

This function is used to get the ADC state.

#### Parameters

- base – ADC peripheral base address.

#### Returns

ADC state, for possible states, please refer to *adc\_state\_t* for details.

static inline bool ADC\_CheckAutoClockOffEnabled(ADC\_Type \*base)

This function is used to check whether the ADC auto clock-off feature has been enabled or not.

#### Parameters

- base – ADC peripheral base address.

#### Returns

ADC auto clock-off feature status.

- **true** Auto clock-off feature has been enabled.
- **false** Auto clock-off feature has not been enabled.

static inline uint8\_t ADC\_GetCurrentConvertedChannelId(ADC\_Type \*base)

This function is used to get the ID of the channel that is currently being converted.

#### Parameters

- base – ADC peripheral base address.

#### Returns

ADC channel ID that is currently being converted.

static inline bool ADC\_CheckSelfTestConvInProgress(ADC\_Type \*base)

This function is used to check whether the self-test conversion is in process or not.

#### Parameters

- base – ADC peripheral base address.

#### Returns

Self-test conversion status.

- **true** Self-test conversion is in process.
- **false** Self-test conversion is not in process.

static inline bool ADC\_CheckInjectConvInProgress(ADC\_Type \*base)

This function is used to check whether the inject conversion is in process or not.

**Parameters**

- base – ADC peripheral base address.

**Returns**

Inject conversion status.

- **true** Inject conversion is in process.
- **false** Inject conversion is not in process.

static inline bool ADC\_CheckInjectConvAborted(ADC\_Type \*base)

This function is used to check whether the inject conversion has been aborted or not.

**Parameters**

- base – ADC peripheral base address.

**Returns**

Inject conversion abort status.

- **true** Injected conversion has been aborted.
- **false** Injected conversion has not been aborted.

static inline bool ADC\_CheckNormalConvInProgress(ADC\_Type \*base)

This function is used to check whether the normal conversion is in process or not.

**Parameters**

- base – ADC peripheral base address.

**Returns**

Normal conversion status.

- **true** Normal conversion is in process.
- **false** Normal conversion is not in process.

static inline bool ADC\_CheckCalibrationBusy(ADC\_Type \*base)

This function is used to check whether the ADC is executing calibration or ready for use.

**Parameters**

- base – ADC peripheral base address.

**Returns**

Calibration process status.

- **true** ADC is busy in a calibration process.
- **false** ADC is ready for use.

static inline bool ADC\_CheckCalibrationFailed(ADC\_Type \*base)

This function is used to check whether the calibration has failed or passed.

---

**Note:** When the user clears the calibration failed status and then reads the status, it will display the calibration passed. At this time, the calibration may not be successful. The user must read the MSR[CALBUSY] bit by function ADC\_CheckCalibrationBusy to perform a double check.

---

**Returns**

Normal conversion status.

- **true** Calibration failed.

- **false** Calibration passed (must be checked with CALBUSY = 0b).

static inline void ADC\_ClearCalibrationFailedFlag(ADC\_Type \*base)

This function is used to clear the flag of calibration.

#### Parameters

- base – ADC peripheral base address.

static inline bool ADC\_CheckCalibrationSuccessful(ADC\_Type \*base)

This function is used to check whether the calibration is successful or not.

#### Parameters

- base – ADC peripheral base address.

#### Returns

Normal conversion status.

- **true** Calibrated or calibration successful.
- **false** Uncalibrated or calibration unsuccessful.

void ADC\_SetConvChainConfig(ADC\_Type \*base, const *adc\_chain\_config\_t* \*config)

This function is used to configure the chain.

#### Parameters

- base – ADC peripheral base address.
- config – Pointer to the chain configuration structure, please refer to *adc\_chain\_config\_t* for details.

static inline void ADC\_EnableSpecificChannelNormalConv(ADC\_Type \*base, uint8\_t  
channelIndex)

This function is used to enable the specific ADC channel to execute normal conversion.

#### Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to enable the normal conversion.

static inline void ADC\_DisableSpecificChannelNormalConv(ADC\_Type \*base, uint8\_t  
channelIndex)

This function is used to disable the specific ADC channel to execute the normal conversion.

#### Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to disable the normal conversion.

static inline void ADC\_EnableSpecificChannelInjectConv(ADC\_Type \*base, uint8\_t channelIndex)

This function is used to enable the specific ADC channel to execute the inject conversion.

#### Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to enable the inject conversion.

static inline void ADC\_DisableSpecificChannelInjectConv(ADC\_Type \*base, uint8\_t channelIndex)

This function is used to disable the specific ADC channel to execute the inject conversion.

#### Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to disable the inject conversion.

static inline void ADC\_SetConvMode(ADC\_Type \*base, *adc\_conv\_mode\_t* convMode)

This function is used to set the ADC conversion mode.

---

**Note:** Before setting the conversion mode, users need to check whether the ADC is in the idle status through the function ADC\_GetAdcState.

---

#### Parameters

- base – ADC peripheral base address.
- convMode – ADC conversion mode, please refer to *adc\_conv\_mode\_t* for details.

static inline void ADC\_StartConvChain(ADC\_Type \*base, *adc\_conv\_mode\_t* convMode)

This function is used to start the ADC conversion chain to execute the conversion.

---

**Note:** Normal conversion supports two conversion modes, one is one-shot conversion mode, and the other is scan conversion mode. Normal conversion should usually be used to convert analog samples most of the time in an application, unless there is a special need. Inject conversion has a higher priority than normal conversion and it runs in one-shot mode only, it can be started in IDLE condition or when normal conversion is in process.

---

#### Parameters

- base – ADC peripheral base address.
- convMode – Pointer to the ADC conversion chain, please refer to *adc\_conv\_mode\_t* for details.

static inline void ADC\_StopConvChain(ADC\_Type \*base)

This function is used to stop scan in normal conversion scan operation mode.

---

**Note:** In scan operation mode, the MCR[NSTART] field remains high after setting. Clearing this field in scan operation mode causes the current chain conversion to finish, then stop the scan.

---

#### Parameters

- base – ADC peripheral base address.

static inline void ADC\_AbortCurrentConvChain(ADC\_Type \*base)

This function is used to abort the conversion chain.

Abort the current chain of conversions by setting MCR[ABORTCHAIN]. In that case, the behavior of the ADC depends on MCR[MODE] (one-shot/scan conversion modes). In one-shot mode, MSR[NSTART] is automatically reset together with MCR[ABORTCHAIN], an end-of-chain interrupt is not generated in the case of an abort chain. In scan mode, a new chain is started. The end-of-conversion interrupt of the current aborted conversion is not generated but an end-of-chain interrupt is generated.

---

**Note:** Setting this field in an IDLE state (for example, no normal/inject conversion is in process) has no effect. In this case, the MCR[ABORTCHAIN] field is reset immediately.

---

#### Parameters

- base – ADC peripheral base address.

```
static inline void ADC_AbortCurrentConv(ADC_Type *base)
```

This function is used to abort the conversion channel.

Abort the current conversion and immediately start the conversion of the next channel of the chain. In the case of an abort operation, MSR[NSTART/JSTART] remains set if not in the last channel of the chain, and MCR[ABORT] is reset as soon as the channel is aborted. The end-of-conversion interrupt corresponds to the aborted channel is not generated. This behavior is true for normal or inject conversion modes. If the last channel of a chain is aborted, and the end-of-chain is reported, then an end-of-chain interrupt will be generated.

---

**Note:** This conversion can not abort while the self-test channel conversion is in process.

---

#### Parameters

- base – ADC peripheral base address.

```
static inline void ADC_EnableConvInt(ADC_Type *base, uint32_t mask)
```

This function is used to enable the ADC end-of-conversion and end-of-chain interrupts.

#### Parameters

- base – ADC peripheral base address.
- mask – Mask value to enable the ADC end-of-conversion and end-of-chain interrupts, please refer to `_adc_conv_int_enable` for details.

```
static inline void ADC_DisableConvInt(ADC_Type *base, uint32_t mask)
```

This function is used to disable the ADC end-of-conversion and end-of-chain interrupts.

#### Parameters

- base – ADC peripheral base address.
- mask – Mask value to disable the ADC end-of-conversion and end-of-chain interrupts, please refer to `_adc_conv_int_enable` for details.

```
static inline uint32_t ADC_GetConvIntStatus(ADC_Type *base)
```

This function is used to get the ADC end-of-conversion and end-of-chain interrupts status.

#### Parameters

- base – ADC peripheral base address.

#### Returns

ADC end-of-conversion and end-of-chain interrupts status mask.

```
static inline void ADC_ClearConvIntStatus(ADC_Type *base, uint32_t mask)
```

This function is used to clear the ADC end-of-conversion and end-of-chain interrupts status.

#### Parameters

- base – ADC peripheral base address.
- mask – Mask value for flags to be cleared, please refer to `_adc_conv_int_flag` for details.

```
static inline void ADC_EnableSpecificConvChannelInt(ADC_Type *base, uint8_t channelIndex)
```

This function is used to enable the specific ADC channel end-of-conversion interrupt.

---

**Note:** This function can only turn on the interrupt of a specific channel, if the interrupt occurs, the user also needs to turn on the global end-of-conversion interrupt by using function `ADC_EnableConvInt`

---



**Parameters**

- base – ADC peripheral base address.
- channelIndex – Channel index to enable the end-of-conversion interrupt.

static inline void ADC\_DisableSpecificConvChannelInt(ADC\_Type \*base, uint8\_t channelIndex)

This function is used to disable the specific ADC channel end-of-conversion interrupt.

**Parameters**

- base – ADC peripheral base address.
- channelIndex – Channel index to disable the end-of-conversion interrupt.

static inline bool ADC\_CheckSpecificConvChannelInt(ADC\_Type \*base, uint8\_t channelIndex)

This function is used to check whether the specific conversion channel's end-of-conversion interrupt has been occurred.

**Parameters**

- base – ADC peripheral base address.
- channelIndex – Channel index to check the end-of-conversion interrupt status.

**Returns**

Channel end-of-conversion interrupt status flag of the specific channel.

- **true** Channel end-of-conversion interrupt has been occurred.
- **false** Channel end-of-conversion interrupt has not been occurred.

static inline void ADC\_ClearSpecificConvChannelInt(ADC\_Type \*base, uint8\_t channelIndex)

This function is used to clear specific channel end-of-conversion interrupt flag.

**Parameters**

- base – ADC peripheral base address.
- channelIndex – Channel index to clear the end-of-conversion interrupt flag.

static inline void ADC\_EnableDmaTransfer(ADC\_Type \*base)

This function is used to enable the DMA transfer function.

---

**Note:** This function is a master switch used to control whether the data converted by the ADC is transmitted through DMA. If the user configures DMA to transmit the conversion data of the channel during the chain channel configuration process, then the main switch of the DMA transmission must be turned on, otherwise, data transmission cannot be performed.

---

**Parameters**

- base – ADC peripheral base address.

static inline void ADC\_DisableDmaTransfer(ADC\_Type \*base)

This function is used to disable the DMA transfer function.

**Parameters**

- base – ADC peripheral base address.

static inline void ADC\_EnableSpecificConvChannelDmaTransfer(ADC\_Type \*base, uint8\_t channelIndex)

This function is used to enable the specific ADC conversion channel's DMA transfer function.

---

**Note:** This function can only turn on the DMA transfer of a specific channel, before using the DMA transfer, the user needs to turn on the global DMA transfer by using the function `ADC_EnableDmaTransfer`.

---

#### Parameters

- `base` – ADC peripheral base address.
- `channelIndex` – Channel index to enable the DMA transfer function.

```
static inline void ADC__DisableSpecificConvChannelDmaTransfer(ADC_Type *base, uint8_t  
                                                             channelIndex)
```

This function is used to disable the specific ADC conversion channel's DMA transfer function.

#### Parameters

- `base` – ADC peripheral base address.
- `channelIndex` – Channel index to disable the DMA transfer function.

```
static inline void ADC__EnableSpecificConvChannelPresample(ADC_Type *base, uint8_t  
                                                           channelIndex)
```

This function is used to enable the specific ADC conversion channel's pre-sample function.

#### Parameters

- `base` – ADC peripheral base address.
- `channelIndex` – Channel index to enable the pre-sample function.

```
static inline void ADC__DisableSpecificConvChannelPresample(ADC_Type *base, uint8_t  
                                                           channelIndex)
```

This function is used to disable the specific ADC conversion channel's pre-sample function.

#### Parameters

- `base` – ADC peripheral base address.
- `channelIndex` – Channel index to disable the pre-sample function.

```
void ADC_SetAnalogWdgConfig(ADC_Type *base, const adc_wdg_config_t *config)
```

This function is used to configure the analog watchdog.

The analog watchdogs are used to monitor the conversion result to see if it is within defined limits, specified by a higher and a lower threshold value. After the conversion of the selected channel, a comparison is performed between the converted value and the threshold values. If the converted value is outside the threshold values, then a corresponding threshold violation interrupt is generated.

#### Parameters

- `base` – ADC peripheral base address.
- `config` – Pointer to the analog watchdog configuration structure, please refer to `adc_wdg_config_t` for details.

```
static inline void ADC__EnableSpecificConvChannelAnalogWdg(ADC_Type *base, uint8_t  
                                                           channelIndex)
```

This function is used to enable the specific ADC conversion channel's analog watchdog function.

#### Parameters

- `base` – ADC peripheral base address.

- `channelIndex` – Conversion channel index to enable the analog watchdog function.

```
static inline void ADC__DisableSpecificConvChannelAnalogWdg(ADC_Type *base, uint8_t  
                                                           channelIndex)
```

This function is used to disable the specific ADC conversion channel's analog watchdog function.

#### Parameters

- `base` – ADC peripheral base address.
- `channelIndex` – Conversion channel index to disable the analog watchdog function.

```
static inline bool ADC__CheckSpecificConvChannelOutOfRange(ADC_Type *base, uint8_t  
                                                           channelIndex)
```

This function is used to check whether the specific conversion channel's converted data is out of range.

#### Parameters

- `base` – ADC peripheral base address.
- `channelIndex` – Channel index to check the converted data out of range status.

#### Returns

Converted data out of range status of the specific conversion channel.

- **true** Channel converted data is out of range.
- **false** Channel converted data is in range.

```
static inline void ADC__ClearSpecificConvChannelOutOfRange(ADC_Type *base, uint8_t  
                                                           channelIndex)
```

This function is used to clear the specific conversion channel's converted data out-of-range flag.

#### Parameters

- `base` – ADC peripheral base address.
- `channelIndex` – Channel index to clear the converted data out of range flag.

```
static inline void ADC__EnableWdgThresholdInt(ADC_Type *base, uint32_t mask)
```

This function is used to enable the analog watchdog threshold low or/and high interrupts.

#### Parameters

- `base` – ADC peripheral base address.
- `mask` – Mask value to enable the analog watchdog threshold low or/and high interrupts, please refer to `_adc_wdg_threshold_int_enable` for details.

```
static inline void ADC__DisableWdgThresholdInt(ADC_Type *base, uint32_t mask)
```

This function is used to disable the analog watchdog threshold low or/and high interrupts.

#### Parameters

- `base` – ADC peripheral base address.
- `mask` – Mask value to disable the analog watchdog threshold low or/and high interrupts, please refer to `_adc_wdg_threshold_int_enable` for details.

```
static inline uint32_t ADC__GetWdgThresholdIntStatus(ADC_Type *base)
```

This function is used to get the analog watchdog threshold interrupts status.

#### Parameters

- `base` – ADC peripheral base address.

**Returns**

Analog watchdog threshold interrupts status mask.

```
static inline void ADC_ClearWdgThresholdIntStatus(ADC_Type *base, uint32_t mask)
```

This function is used to clear the analog watchdog threshold low or/and high interrupts status.

**Parameters**

- `base` – ADC peripheral base address.
- `mask` – Mask value for flags to be cleared, please refer to `_adc_wdg_threshold_int_flag` for details.

```
bool ADC_DoCalibration(ADC_Type *base, const adc_calibration_config_t *config)
```

This function is used to do the calibration.

The calibration is used to reduce or eliminate the various errors. In the calibration process, the calibration values for offset, gain, and capacitor mismatch are obtained. These calibration values (except gain calibration) are used in a result post-processing step to reduce or eliminate the various errors contribution effects. The gain calibration is used during the sample phase to define the additional charge to be loaded in order to compensate for the gain failure. Calibration must be performed after every power-up reset and whenever required in runtime operation. It is also recommended to run calibration if the operating conditions (particularly `VrefH`) change. Never apply functional reset during the calibration process. If applied, calibration must be rerun after exiting a reset condition; otherwise, the calibration-generated values and conversion results may be unspecified.

---

**Note:** This function executes a calibration sequence, it is recommended to run this sequence before using the ADC converter. The maximum clock frequency for the calibration is 40 MHz. Before calling this function, the user needs to ensure that the input clock is within 40MHz. The results of individual steps are also updated in the CALSTAT register (CALSTAT[STAT\_n]). The result of the last failed step is dynamically updated in the same register.

---

**Parameters**

- `base` – ADC peripheral base address.
- `config` – Pointer to the calibration configuration structure, please refer to `adc_calibration_config_t` for details.

**Returns**

Status whether calibration is running passed or failed.

- **true** Calibration successful.
- **false** Calibration unsuccessful.

```
void ADC_SetUserOffsetAndGainConfig(ADC_Type *base, const adc_user_offset_gain_config_t *config)
```

This function is used to configure the user gain and offset.

**Parameters**

- `base` – ADC peripheral base address.
- `config` – Pointer to the user offset and gain configuration structure, please refer to `adc_user_offset_gain_config_t` for details.

void ADC\_SetSelfTestConfig(ADC\_Type \*base, const *adc\_self\_test\_config\_t* \*config)

This function is used to configure the ADC self-test.

The self-test is used to check at regular intervals whether ADC is operating correctly. When self-test is enabled, ADC automatically checks its components and flags any errors it finds. The test can be enabled to check the supply voltage (VDD), reference voltage (VrefH), and calibrated values.

---

**Note:** Before calling this function, please ensure the functional conversion is one-shot conversion mode normal conversion type and the operating clock are equal to bus frequency. ADC self-test should be run with MCR[ADCLKSE] bit set to 1. Self-test with ADCLKSE bit set to 0 can give erroneous results.

---

#### Parameters

- base – ADC peripheral base address.
- config – Pointer to the self-test configuration structure, please refer to *adc\_self\_test\_config\_t* for details.

void ADC\_SetSelfTestWdgConfig(ADC\_Type \*base, const *adc\_self\_test\_wdg\_config\_t* \*config)

This function is used to configure the ADC self-test watchdog.

#### Parameters

- base – ADC peripheral base address.
- config – Pointer to the self-test watchdog configuration structure, please refer to *adc\_self\_test\_wdg\_config\_t* for details.

void ADC\_GetCalibrationLastFailedTestResult(ADC\_Type \*base, int16\_t \*result)

This function is used to get the test result for the last failed test.

#### Parameters

- base – ADC peripheral base address.
- result – Points to a 16-bit signed variable, and it is used to store the test result for the last failing test.

static inline uint16\_t ADC\_GetCalibrationStepsStatus(ADC\_Type \*base)

This function is used to get the status of the calibration steps.

---

**Note:** The status of calibration steps (step 0 to step 12) is stored in the CALSTAT register, and only the lower 12 bits are available in the returned result.

---

#### Parameters

- base – ADC peripheral base address.

#### Returns

ADC self-test interrupt status mask.

static inline void ADC\_EnableSelfTest(ADC\_Type \*base)

This function is used to enable the ADC self-test.

Decides whether to enable the ADC self-test. The self-test test is enabled by setting STCR2[EN], this field must be set before starting normal conversion and should not be changed while the conversion is in process. This field should only be reset after the end-of-conversion of the last self-test channel has been received.

---

**Note:** ADC self-test should be run with MCR[ADCLKSE] bit set to 1. Self-test with ADCLKSE bit set to 0 can give erroneous results. In the case of Inject Conversion mode, test channel conversion is not performed. It is performed only during normal conversions.

---

### Parameters

- base – ADC peripheral base address.

static inline void ADC\_DisableSelfTest(ADC\_Type \*base)

This function is used to disable the ADC self-test.

### Parameters

- base – ADC peripheral base address.

static inline void ADC\_EnableSelfTestWdgThreshold(ADC\_Type \*base,  
adc\_self\_test\_wdg\_threshold\_t wdgID)

This function is used to enable the ADC self-test watchdog threshold for algorithm S step 0/1/2 and algorithm C.

The user can pass kADC\_SelfTestWdgThresholdForAlgSStep0 as parameter 'wdgID' to enable the self-test watchdog threshold function for algorithm S step 0; pass kADC\_SelfTestWdgThresholdForAlgSStep1Integer as parameter 'wdgID' to enable the self-test watchdog threshold function for algorithm S step 1; pass kADC\_SelfTestWdgThresholdForAlgSStep2 as parameter 'wdgID' to enable the self-test watchdog threshold function for algorithm S step 2; pass kADC\_SelfTestWdgThresholdForAlgCStep0 as parameter 'wdgID' to enable the self-test watchdog threshold function for algorithm C; Other enumerations in adc\_self\_test\_wdg\_threshold\_t have no use.

### Parameters

- base – ADC peripheral base address.
- wdgID – Watchdog threshold index to enable, please refer to adc\_self\_test\_wdg\_threshold\_t for details.

static inline void ADC\_DisableSelfTestWdgThreshold(ADC\_Type \*base,  
adc\_self\_test\_wdg\_threshold\_t wdgID)

This function is used to disable the ADC self-test watchdog threshold for algorithm S step 0/1/2 and algorithm C.

The user can pass kADC\_SelfTestWdgThresholdForAlgSStep0 as parameter 'wdgID' to disable the self-test watchdog threshold function for algorithm S step 0; pass kADC\_SelfTestWdgThresholdForAlgSStep1Integer as parameter 'wdgID' to disable the self-test watchdog threshold function for algorithm S step 1; pass kADC\_SelfTestWdgThresholdForAlgSStep2 as parameter 'wdgID' to disable the self-test watchdog threshold function for algorithm S step 2; pass kADC\_SelfTestWdgThresholdForAlgCStep0 as parameter 'wdgID' to disable the self-test watchdog threshold function for algorithm C; Other enumerations in adc\_self\_test\_wdg\_threshold\_t have no use.

### Parameters

- base – ADC peripheral base address.
- wdgID – Watchdog threshold index to disable, please refer to adc\_self\_test\_wdg\_threshold\_t for details.

static inline void ADC\_EnableSelfTestWdgTimer(ADC\_Type \*base, adc\_alg\_type\_t  
wdgTimerType)

This function is used to enable the ADC self-test watchdog timer for algorithm S or/and algorithm C.

The user can pass `kADC_SelfTestForAlgS` as parameter 'wdgTimerType' to enable the watchdog timer for algorithm S; pass `kADC_SelfTestForAlgC` as parameter 'wdgTimerType' to enable the watchdog timer for algorithm C; pass `kADC_SelfTestForAlgSAndC` as parameter 'wdgTimerType' to enable the watchdog timer for algorithm S and C.

#### Parameters

- `base` – ADC peripheral base address.
- `wdgTimerType` – Watchdog timer type to enable, please refer to `adc_alg_type_t` for details.

```
static inline void ADC_DisableSelfTestWdgTimer(ADC_Type *base, adc_alg_type_t
                                              wdgTimerType)
```

This function is used to disable the ADC self-test watchdog timer.

The user can pass `kADC_SelfTestForAlgS` as parameter 'wdgTimerType' to disable the watchdog timer for algorithm S; pass `kADC_SelfTestForAlgC` as parameter 'wdgTimerType' to disable the watchdog timer for algorithm C; pass `kADC_SelfTestForAlgSAndC` as parameter 'wdgTimerType' to disable the watchdog timer for algorithm S and C.

#### Parameters

- `base` – ADC peripheral base address.
- `wdgTimerType` – Watchdog timer type to disable, please refer to `adc_alg_type_t` for details.

```
static inline void ADC_SetSelfTestWdgTimerVal(ADC_Type *base, adc_wdg_timer_val_t
                                              wdgTimerVal)
```

This function is used to set the ADC self-test watchdog timer value.

#### Parameters

- `base` – ADC peripheral base address.
- `wdgTimerVal` – Watchdog timer value, please refer to `adc_wdg_timer_val_t` for details.

```
static inline uint16_t ADC_GetSelfTestChannelConvFailedData(ADC_Type *base,
                                                            adc_self_test_wdg_threshold_t
                                                            type)
```

This function is used to get the ADC self-test channel converted data when `ERR_S0/ERR_S1_INTEGER/ERR_S1_FRACTION/ERR_S2/ERR_C` occurred.

---

**Note:** The user can pass `kADC_SelfTestWdgThresholdForAlgSStep0` as parameter 'type' to get the converted data when `ERR_S0` occurred; pass `kADC_SelfTestWdgThresholdForAlgSStep1Integer` as parameter 'type' to get the converted data when `ERR_S1_INTEGER` occurred; pass `kADC_SelfTestWdgThresholdForAlgSStep1Fraction` as parameter 'type' to get the converted data when `ERR_S1_FRACTION` occurred; pass `kADC_SelfTestWdgThresholdForAlgSStep2` as parameter 'type' to get the converted data when `ERR_S2` occurred; pass `kADC_SelfTestWdgThresholdForAlgCStep0` as parameter 'type' to get the converted data when `ERR_C` occurred; Other enumerations in `adc_self_test_wdg_threshold_t` have no use.

---

#### Parameters

- `base` – ADC peripheral base address.
- `type` – Watchdog threshold index to get the ADC self-test channel converted data, please refer to `adc_self_test_wdg_threshold_t` for details.

```
static inline void ADC_EnableSelfTestInt(ADC_Type *base, uint32_t mask)
```

This function is used to enable the ADC self-test-related interrupts.

---

**Note:** Watchdog timer feature is applicable only for scan operation mode and not for one-shot operation mode.

---

#### Parameters

- `base` – ADC peripheral base address.
- `mask` – Mask value to enable the ADC self-test related interrupts, please refer to `_adc_self_test_int_enable` for details.

```
static inline void ADC_DisableSelfTestInt(ADC_Type *base, uint32_t mask)
```

This function is used to disable the ADC self-test interrupt.

#### Parameters

- `base` – ADC peripheral base address.
- `mask` – Mask value to disable the ADC self-test related interrupts, please refer to `_adc_self_test_int_enable` for details.

```
static inline uint32_t ADC_GetSelfTestIntStatus(ADC_Type *base)
```

This function is used to get the ADC self-test interrupts status.

#### Parameters

- `base` – ADC peripheral base address.

#### Returns

ADC self-test related interrupts status mask.

```
static inline void ADC_ClearSelfTestIntStatus(ADC_Type *base, uint32_t mask)
```

This function is used to clear the ADC self-test interrupts status.

#### Parameters

- `base` – ADC peripheral base address.
- `mask` – Mask value for flags to be cleared, please refer to `_adc_self_test_int_flag` for details.

```
bool ADC_GetChannelConvResult(ADC_Type *base, adc_conv_result_t *result, uint8_t  
                             channelIndex)
```

This function is used to get the specific ADC channel's conversion result.

CDR[VALID] indicates whether a new conversion is available, this field is automatically reset to 0 when the data is read. CDR[OVERW] Indicates whether the previous conversion data was overwritten without having been read, in which case the overwritten data is lost.

#### Parameters

- `base` – SAR ADC peripheral base address.
- `result` – Pointer to SAR ADC channels conversion result structure, please refer to `adc_conv_result_t` for details.
- `channelIndex` – Channel index to get the conversion result.

#### Returns

Indicates whether the acquisition of the specific channel conversion result is successful or not.

- **true** Obtaining the specific channel conversion result successfully, and the conversion result is stored in the input parameter result.



- **false** Obtaining the specific channel conversion result failed.

bool ADC\_GetSelfTestChannelConvData(ADC\_Type \*base, *adc\_self\_test\_conv\_result\_t* \*result)

This function is used to get the test channel converted data when algorithm S step 0, algorithm S step 2, or algorithm C step executes.

#### Parameters

- *base* – ADC peripheral base address.
- *result* – Pointer to the SAR ADC self-test channel conversion result structure, please refer to *adc\_self\_test\_conv\_result\_t* for details.

#### Returns

Indicates whether the acquisition of the self-test channel conversion result is successful or not.

- **true** Obtaining the self-test channel conversion result successfully, and the conversion result is stored in the input parameter 'result'.
- **false** Obtaining the self-test channel conversion result failed.

bool ADC\_GetSelfTestChannelConvDataForAlgSStep1(ADC\_Type \*base,  
*adc\_self\_test\_conv\_result\_t* \*result)

This function is used to get the test channel converted data when algorithm S step 1 executes.

#### Parameters

- *base* – ADC peripheral base address.
- *result* – Pointer to the SAR ADC self-test channel conversion result structure, please refer to *adc\_self\_test\_conv\_result\_t* for details.

#### Returns

Indicates whether the acquisition of the self-test channel conversion result is successful or not.

- **true** Obtaining the self-test channel conversion result successfully, and the conversion result is stored in the input parameter 'result'.
- **false** Obtaining the self-test channel conversion result failed.

FSL\_SAR\_ADC\_DRIVER\_VERSION

SAR ADC driver version 2.3.0.

enum \_adc\_conv\_int\_enable

This enumeration provides the mask for the ADC end-of-conversion and end-of-chain interrupts enabling.

*Values:*

enumerator kADC\_NormalConvChainEndIntEnable

Enable end of normal chain conversion interrupt.

enumerator kADC\_NormalConvEndIntEnable

Enable end of normal conversion interrupt.

enumerator kADC\_InjectConvChainEndIntEnable

Enable end of inject chain conversion interrupt.

enumerator kADC\_InjectConvEndIntEnable

Enable end of inject conversion interrupt.

enum \_adc\_wdg\_threshold\_int\_enable

This enumeration provides the mask for the ADC analog watchdog threshold interrupts enabling.

*Values:*

enumerator kADC\_wdg0LowThresholdIntEnable  
Enable watchdog 0 low threshold interrupt.

enumerator kADC\_wdg0HighThresholdIntEnable  
Enable watchdog 0 high threshold interrupt.

enumerator kADC\_wdg1LowThresholdIntEnable  
Enable watchdog 1 low threshold interrupt.

enumerator kADC\_wdg1HighThresholdIntEnable  
Enable watchdog 1 high threshold interrupt.

enumerator kADC\_wdg2LowThresholdIntEnable  
Enable watchdog 2 low threshold interrupt.

enumerator kADC\_wdg2HighThresholdIntEnable  
Enable watchdog 2 high threshold interrupt.

enumerator kADC\_wdg3LowThresholdIntEnable  
Enable watchdog 3 low threshold interrupt.

enumerator kADC\_wdg3HighThresholdIntEnable  
Enable watchdog 3 high threshold interrupt.

enumerator kADC\_wdg4LowThresholdIntEnable  
Enable watchdog 4 low threshold interrupt.

enumerator kADC\_wdg4HighThresholdIntEnable  
Enable watchdog 4 high threshold interrupt.

enumerator kADC\_wdg5LowThresholdIntEnable  
Enable watchdog 5 low threshold interrupt.

enumerator kADC\_wdg5HighThresholdIntEnable  
Enable watchdog 5 high threshold interrupt.

enumerator kADC\_wdg6LowThresholdIntEnable  
Enable watchdog 6 low threshold interrupt.

enumerator kADC\_wdg6HighThresholdIntEnable  
Enable watchdog 6 high threshold interrupt.

enumerator kADC\_wdg7LowThresholdIntEnable  
Enable watchdog 7 low threshold interrupt.

enumerator kADC\_wdg7HighThresholdIntEnable  
Enable watchdog 7 high threshold interrupt.

enum \_adc\_self\_test\_int\_enable

This enumeration provides the mask for the ADC self-test related interrupts enabling.

*Values:*

enumerator kADC\_AlgSStep0ErrIntEnable  
Enable self-test algorithm S step0 error interrupt.

enumerator kADC\_AlgSStep1ErrIntEnable  
Enable self-test algorithm S step1 error interrupt.

enumerator kADC\_\_AlgSStep2ErrIntEnable  
Enable self-test algorithm S step2 error interrupt.

enumerator kADC\_\_AlgCErrIntEnable  
Enable self-test algorithm C error interrupt.

enumerator kADC\_\_AlgSEndIntEnable  
Enable self-test algorithm S end interrupt.

enumerator kADC\_\_AlgCEndIntEnable  
Enable self-test algorithm C end interrupt.

enumerator kADC\_\_ConvEndIntEnable  
Enable self-test conversion end interrupt.

enumerator kADC\_\_WdgTimeErrIntEnable  
Enable watchdog time error interrupt.

enumerator kADC\_\_WdgSequenceErrIntEnable  
Enable watchdog sequence error interrupt.

enum \_adc\_conv\_int\_flag

This enumeration provides the mask for the ADC end-of-conversion and end-of-chain interrupts flag.

*Values:*

enumerator kADC\_\_NormalConvChainEndIntFlag  
Indicates whether the end of normal chain conversion interrupt has occurred.

enumerator kADC\_\_NormalConvEndIntFlag  
Indicates whether the end of conversion interrupt has occurred.

enumerator kADC\_\_InjectConvChainEndIntFlag  
Indicates whether the end of inject chain conversion interrupt has occurred.

enumerator kADC\_\_InjectConvEndIntFlag  
Indicates whether the end of inject conversion interrupt has occurred.

enum \_adc\_wdg\_threshold\_int\_flag

This enumeration provides the mask for the ADC analog watchdog threshold interrupts flag.

*Values:*

enumerator kADC\_\_wdg0LowThresholdIntFlag  
Indicates whether the watchdog 0 low threshold interrupt has occurred.

enumerator kADC\_\_wdg0HighThresholdIntFlag  
Indicates whether the watchdog 0 high threshold interrupt has occurred.

enumerator kADC\_\_wdg1LowThresholdIntFlag  
Indicates whether the watchdog 1 low threshold interrupt has occurred.

enumerator kADC\_\_wdg1HighThresholdIntFlag  
Indicates whether the watchdog 1 high threshold interrupt has occurred.

enumerator kADC\_\_wdg2LowThresholdIntFlag  
Indicates whether the watchdog 2 low threshold interrupt has occurred.

enumerator kADC\_\_wdg2HighThresholdIntFlag  
Indicates whether the watchdog 2 high threshold interrupt has occurred.

enumerator kADC\_\_wdg3LowThresholdIntFlag

Indicates whether the watchdog 3 low threshold interrupt has occurred.

enumerator kADC\_\_wdg3HighThresholdIntFlag

Indicates whether the watchdog 3 high threshold interrupt has occurred.

enumerator kADC\_\_wdg4LowThresholdIntFlag

Indicates whether the watchdog 4 low threshold interrupt has occurred.

enumerator kADC\_\_wdg4HighThresholdIntFlag

Indicates whether the watchdog 4 high threshold interrupt has occurred.

enumerator kADC\_\_wdg5LowThresholdIntFlag

Indicates whether the watchdog 5 low threshold interrupt has occurred.

enumerator kADC\_\_wdg5HighThresholdIntFlag

Indicates whether the watchdog 5 high threshold interrupt has occurred.

enumerator kADC\_\_wdg6LowThresholdIntFlag

Indicates whether the watchdog 6 low threshold interrupt has occurred.

enumerator kADC\_\_wdg6HighThresholdIntFlag

Indicates whether the watchdog 6 high threshold interrupt has occurred.

enumerator kADC\_\_wdg7LowThresholdIntFlag

Indicates whether the watchdog 7 low threshold interrupt has occurred.

enumerator kADC\_\_wdg7HighThresholdIntFlag

Indicates whether the watchdog 7 high threshold interrupt has occurred.

enum \_adc\_self\_test\_int\_flag

This enumeration provides the mask for the ADC self-test-related interrupts flag.

*Values:*

enumerator kADC\_\_AlgSStep0ErrIntFlag

Indicates whether the self-test algorithm S step0 error interrupt has occurred.

enumerator kADC\_\_AlgSStep1ErrIntFlag

Indicates whether the self-test algorithm S step1 error interrupt has occurred.

enumerator kADC\_\_AlgSStep2ErrIntFlag

Indicates whether the self-test algorithm S step2 error interrupt has occurred.

enumerator kADC\_\_AlgCErrIntFlag

Indicates whether the self-test algorithm C error interrupt has occurred.

enumerator kADC\_\_AlgSEndIntFlag

Indicates whether the algorithm S end interrupt has completed.

enumerator kADC\_\_AlgCEndIntFlag

Indicates whether the algorithm C end interrupt has completed.

enumerator kADC\_\_SelfTestConvEndIntFlag

Indicates whether the self-test end-of-conversion interrupt has completed.

enumerator kADC\_\_OverWriteErrIntFlag

Indicates whether the overwrite error interrupt has occurred.

enumerator kADC\_\_WdgTimeErrIntFlag

Indicates whether the watchdog time error interrupt has occurred.

enumerator kADC\_WdgSequenceErrIntFlag

Indicates whether the watchdog sequence error interrupt has occurred.

enum \_adc\_ext\_trig

This enumeration provides the selection of the ADC external trigger type.

*Values:*

enumerator kADC\_ExtTrigDisable

Normal trigger input does not start a conversion.

enumerator kADC\_ExtTrigFallingEdge

Normal trigger (falling edge) input starts a conversion.

enumerator kADC\_ExtTrigRisingEdge

Normal trigger (rising edge) input starts a conversion.

enum \_adc\_state

This enumeration provides the selection of the ADC state.

*Values:*

enumerator kADC\_AdcIdle

Indicates the ADC is in the IDLE state.

enumerator kADC\_AdcPowerdown

Indicates the ADC is in the power-down state.

enumerator kADC\_AdcWait

Indicates the ADC is in the wait state.

enumerator kADC\_AdcBusyInCalibration

Indicates the ADC is in the calibration busy state.

enumerator kADC\_AdcSample

Indicates the ADC is in the sample state.

enumerator kADC\_AdcConv

Indicates the ADC is in the conversion state.

enum \_adc\_conv\_mode

This enumeration provides the selection of the ADC conversion mode, including normal conversion one-shot mode, normal conversion scan mode, and inject conversion one-shot mode.

*Values:*

enumerator kADC\_NormalConvOneShotMode

Normal conversion one-shot mode.

enumerator kADC\_NormalConvScanMode

Normal conversion scan mode.

enumerator kADC\_InjectConvOneShotMode

Inject conversion one-shot mode.

enum \_adc\_clock\_frequency

This enumeration provides the selection of the ADC operating clock frequency, including half-bus frequency and full bus frequency.

*Values:*

enumerator kADC\_HalfBusFrequency

Half of bus clock frequency.

enumerator kADC\_FullBusFrequency

Equal to bus clock frequency.

enum \_adc\_conv\_data\_align

This enumeration provides the selection of the ADC conversion data alignment, including the right alignment and left alignment.

*Values:*

enumerator kADC\_ConvDataRightAlign

Conversion data is right aligned.

enumerator kADC\_ConvDataLeftAlign

Conversion data is left aligned.

enum \_adc\_presample\_voltage\_src

This enumeration provides the selection of the ADC internal analog input voltage sources for pre-sample, including DVDD0P8/2, AVDD1P8/4, VREFL\_1p8 and VREFH\_1p8.

*Values:*

enumerator kADC\_PresampleVoltageSrcVREL

Use VREL as pre-sample voltage source.

enumerator kADC\_PresampleVoltageSrcVREH

Use VREH as pre-sample voltage source.

enum \_adc\_dma\_request\_clear\_src

This enumeration provides the selection of the DMA request clear sources, including clear by acknowledgment from the DMA controller and clear on a read of the data register.

*Values:*

enumerator kADC\_DMAResultClearByAck

DMA request cleared by acknowledgment from DMA controller.

enumerator kADC\_DMAResultClearOnRead

DMA request cleared on a read of the data register.

enum \_adc\_average\_sample\_numbers

This enumeration provides the selection of the ADC calibration averaging sample numbers, including 16, 32, 128, and 512 averaging samples.

*Values:*

enumerator kADC\_AverageSampleNumbers16

Use 16 averaging samples during calibration.

enumerator kADC\_AverageSampleNumbers32

Use 32 averaging samples during calibration.

enumerator kADC\_AverageSampleNumbers128

Use 128 averaging samples during calibration.

enumerator kADC\_AverageSampleNumbers512

Use 512 averaging samples during calibration.

enum \_adc\_sample\_time

This enumeration provides the selection of the ADC sample time of calibration conversions, including 22, 8, 16 and 32 cycles of ADC\_CLK.

*Values:*

enumerator kADC\_SampleTime22

Use 22 cycles of ADC\_CLK as sample time of calibration conversions.

enumerator kADC\_SampleTime8

Use 8 cycles of ADC\_CLK as sample time of calibration conversions.

enumerator kADC\_SampleTime16

Use 16 cycles of ADC\_CLK as sample time of calibration conversions.

enumerator kADC\_SampleTime32

Use 32 cycles of ADC\_CLK as sample time of calibration conversions.

enum \_adc\_wdg\_threshold\_int

This enumeration provides the selection of the ADC analog watchdog threshold low and high interrupt enable.

*Values:*

enumerator kADC\_LowHighThresholdIntDisable

Enable the ADC analog watchdog low and high threshold interrupts.

enumerator kADC\_LowThresholdIntEnable

Enable the ADC analog watchdog low threshold interrupt.

enumerator kADC\_HighThresholdIntEnable

Enable the ADC analog watchdog high threshold interrupt.

enumerator kADC\_LowHighThresholdIntEnable

Enable the ADC analog watchdog low and high threshold interrupts.

enum \_adc\_alg\_type

This enumeration provides the selection of the ADC self-test algorithm type, including algorithm S, algorithm C and algorithm S and C.

---

**Note:** The meaning of enumeration member 'kADC\_SelfTestForAlgSAndC' in different conversion modes is different, in the one-shot conversion mode, it means executing algorithm S; in the scan conversion mode, it means executing algorithm S and C.

---

*Values:*

enumerator kADC\_SelfTestForAlgS

Use algorithm S for self-test.

enumerator kADC\_SelfTestForAlgC

Use algorithm C for self-test.

enumerator kADC\_SelfTestForAlgSAndC

Use algorithm S for one-shot conversion mode self-test, use algorithm S and algorithm C for scan conversion mode self-test.

enum \_adc\_self\_test\_wdg\_threshold

This enumeration provides the selection of the ADC self-test watchdog thresholds for algorithm S step 0 - 2, and watchdog thresholds for algorithm C step 0 and step x (x = 1 - 11).

*Values:*

enumerator kADC\_SelfTestWdgThresholdForAlgSStep0

Self-test watchdog threshold for the algorithm S step 0.

enumerator kADC\_SelfTestWdgThresholdForAlgSStep1

Self-test watchdog threshold for the algorithm S step 1 fraction part.

enumerator kADC\_SelfTestWdgThresholdForAlgSStep2  
Self-test watchdog threshold for the algorithm S step 2.

enumerator kADC\_SelfTestWdgThresholdForAlgCStep0  
Self-test watchdog threshold for the algorithm C step 0.

enumerator kADC\_SelfTestWdgThresholdForAlgCStepx  
Self-test watchdog threshold for the algorithm C step x.

enum \_adc\_wdg\_timer\_val

This enumeration provides the selection of the ADC self-test watchdog timer value, including 0.1ms, 0.5ms, 1ms, 2ms, 5ms, 10ms, 20ms and 50ms.

*Values:*

enumerator kADC\_SelfTestWdgTimerVal0  
0.1ms ((0008h × Prescaler) cycles at 80 MHz).

enumerator kADC\_SelfTestWdgTimerVal1  
0.5ms ((0027h × Prescaler) cycles at 80 MHz).

enumerator kADC\_SelfTestWdgTimerVal2  
1ms ((004Eh × Prescaler) cycles at 80 MHz).

enumerator kADC\_SelfTestWdgTimerVal3  
2ms ((009Ch × Prescaler) cycles at 80 MHz).

enumerator kADC\_SelfTestWdgTimerVal4  
5ms ((0187h × Prescaler) cycles at 80 MHz).

enumerator kADC\_SelfTestWdgTimerVal5  
10ms ((030Dh × Prescaler) cycles at 80 MHz).

enumerator kADC\_SelfTestWdgTimerVal6  
20ms ((061Ah × Prescaler) cycles at 80 MHz).

enumerator kADC\_SelfTestWdgTimerVal7  
50ms ((0F42h × Prescaler) cycles at 80 MHz).

typedef enum \_adc\_ext\_trig adc\_ext\_trig\_t

This enumeration provides the selection of the ADC external trigger type.

typedef enum \_adc\_state adc\_state\_t

This enumeration provides the selection of the ADC state.

typedef enum \_adc\_conv\_mode adc\_conv\_mode\_t

This enumeration provides the selection of the ADC conversion mode, including normal conversion one-shot mode, normal conversion scan mode, and inject conversion one-shot mode.

typedef enum \_adc\_clock\_frequency adc\_clock\_frequency\_t

This enumeration provides the selection of the ADC operating clock frequency, including half-bus frequency and full bus frequency.

typedef enum \_adc\_conv\_data\_align adc\_conv\_data\_align\_t

This enumeration provides the selection of the ADC conversion data alignment, including the right alignment and left alignment.

typedef enum \_adc\_presample\_voltage\_src adc\_presample\_voltage\_src\_t

This enumeration provides the selection of the ADC internal analog input voltage sources for pre-sample, including DVDD0P8/2, AVDD1P8/4, VREFL\_1p8 and VREFH\_1p8.



typedef enum *\_adc\_dma\_request\_clear\_src* adc\_dma\_request\_clear\_src\_t

This enumeration provides the selection of the DMA request clear sources, including clear by acknowledgment from the DMA controller and clear on a read of the data register.

typedef enum *\_adc\_average\_sample\_numbers* adc\_average\_sample\_numbers\_t

This enumeration provides the selection of the ADC calibration averaging sample numbers, including 16, 32, 128, and 512 averaging samples.

typedef enum *\_adc\_sample\_time* adc\_sample\_time\_t

This enumeration provides the selection of the ADC sample time of calibration conversions, including 22, 8, 16 and 32 cycles of ADC\_CLK.

typedef enum *\_adc\_wdg\_threshold\_int* adc\_wdg\_threshold\_int\_t

This enumeration provides the selection of the ADC analog watchdog threshold low and high interrupt enable.

typedef enum *\_adc\_alg\_type* adc\_alg\_type\_t

This enumeration provides the selection of the ADC self-test algorithm type, including algorithm S, algorithm C and algorithm S and C.

---

**Note:** The meaning of enumeration member 'kADC\_SelfTestForAlgSAndC' in different conversion modes is different, in the one-shot conversion mode, it means executing algorithm S; in the scan conversion mode, it means executing algorithm S and C.

---

typedef enum *\_adc\_self\_test\_wdg\_threshold* adc\_self\_test\_wdg\_threshold\_t

This enumeration provides the selection of the ADC self-test watchdog thresholds for algorithm S step 0 - 2, and watchdog thresholds for algorithm C step 0 and step x (x = 1 - 11).

typedef enum *\_adc\_wdg\_timer\_val* adc\_wdg\_timer\_val\_t

This enumeration provides the selection of the ADC self-test watchdog timer value, including 0.1ms, 0.5ms, 1ms, 2ms, 5ms, 10ms, 20ms and 50ms.

typedef struct *\_adc\_config* adc\_config\_t

This structure is used to configure the ADC module.

typedef struct *\_adc\_channel\_config* adc\_channel\_config\_t

This structure is used to configure the ADC conversion channel.

typedef struct *\_adc\_chain\_config* adc\_chain\_config\_t

This structure is used to configure the ADC conversion chain.

typedef struct *\_adc\_wdg\_config* adc\_wdg\_config\_t

This structure is used to configure the ADC analog watchdog.

typedef struct *\_adc\_calibration\_config* adc\_calibration\_config\_t

This structure is used to configure the ADC calibration.

typedef struct *\_adc\_user\_offset\_gain\_config* adc\_user\_offset\_gain\_config\_t

This structure is used to configure the ADC user offset and gain.

typedef struct *\_adc\_self\_test\_config* adc\_self\_test\_config\_t

This structure is used to configure the ADC self-test.

typedef struct *\_adc\_self\_test\_wdg\_config* adc\_self\_test\_wdg\_config\_t

This structure is used to configure the ADC self-test watchdog for algorithm steps.

---

**Note:** The algorithm S step 2 only has the 'LowThrsholdVal'.

---

```
typedef struct _adc_conv_result adc_conv_result_t
```

This structure is used to save the result information when obtaining the conversion result.

```
typedef struct _adc_self_test_conv_result adc_self_test_conv_result_t
```

This structure is used to save the result information when obtaining the self-test channel conversion result.

---

**Note:** The member ‘convData’ is used to store self-test channel conversion results. Only when executing step 1 of algorithm S, member ‘convDataFraction’ will be used to store the fractional part data. When executing other algorithms, this member will not be used.

---

ADC\_GROUP\_COUNTS

ADC\_THRESHOLD\_COUNTS

ADC\_SELF\_TEST\_THRESHOLD\_COUNTS

GET\_REGINDEX(channelIndex)

GET\_BITINDEX(channelIndex)

REGISTER\_READWRITE(baseRegister, shiftIndex)

REGISTER\_READONLY(baseRegister, shiftIndex)

NCMR\_IO(base, registerIndex)

JCMR\_IO(base, registerIndex)

PSR\_IO(base, registerIndex)

DMAR\_IO(base, registerIndex)

CWSELR\_IO(base, registerIndex)

CWENR\_IO(base, registerIndex)

CIMR\_IO(base, registerIndex)

CEOCFR\_IO(base, registerIndex)

AWORR\_IO(base, registerIndex)

STAWR\_IO(base, registerIndex)

CEOCFR\_I(base, registerIndex)

AWORR\_I(base, registerIndex)

CDR\_I(base, registerIndex)

WDG\_SELECT\_MASK(shiftIndex)

WDG\_SELECT\_SHIFT(shiftIndex)

WDG\_SELECT(val, shiftIndex)

ADC\_CDR\_VALID\_MASK

ADC\_CDR\_VALID\_SHIFT

ADC\_CDR\_OVERW\_MASK

ADC\_CDR\_OVERW\_SHIFT  
ADC\_CDR\_RESULT\_MASK  
ADC\_CDR\_RESULT\_SHIFT  
ADC\_CDR\_CDATA\_MASK  
ADC\_CDR\_CDATA\_SHIFT  
ADC\_STAWR\_AWDE\_MASK  
ADC\_STAWR\_THRL\_MASK  
ADC\_STAWR\_THRL\_SHIFT  
ADC\_STAWR\_THRL(val)  
ADC\_STAWR\_THRH\_MASK  
ADC\_STAWR\_THRH\_SHIFT  
ADC\_STAWR\_THRH(val)  
ADC\_CALSTAT\_MAX  
ADC\_CALSTAT\_SIGN

struct \_adc\_config

*#include <fsl\_sar\_adc.h>* This structure is used to configure the ADC module.

### Public Members

bool enableAutoClockOff

Decides whether to enable the ADC auto clock-off function, when set to true, the internal ADC clock is automatically switched off during IDLE mode to reduce power consumption (without going into power-down mode).

bool enableOverWrite

Decides whether to enable the latest conversion to overwrite the current value in the data registers.

bool enableConvertPresampleVal

Decides whether to convert the pre-sampled value, if enabled, pre-sampling is followed by the conversion, sampling will be bypassed and conversion of the pre-sampled data will be done.

*adc\_ext\_trig\_t* extTrig

Specifies whether the normal trigger (with trigger type) input starts a conversion.

*adc\_conv\_data\_align\_t* convDataAlign

Selects the conversion data alignment.

*adc\_clock\_frequency\_t* clockFrequency

Selects the ADC clock frequency.

*adc\_dma\_request\_clear\_src\_t* dmaRequestClearSrc

Selects DMA request clear source.

*adc\_presample\_voltage\_src\_t* presampleVoltageSrc[1]

Selects analog input voltages for group 0 (corresponding to channel 0 to channel 31) and group 32 (corresponding to channel 32 to channel 63) pre-sampling.

uint8\_t samplePhaseDuration[1]

Sets the sample phase duration in terms of the ADC controller clock for group 0 (corresponding to channel 0 to channel 31) and group 32 (corresponding to channel 32 to channel 63), the minimum acceptable value is 8, configuring to a value lower than 8 sets the sample period to 8 cycles.

struct \_\_adc\_channel\_config

*#include <fsl\_sar\_adc.h>* This structure is used to configure the ADC conversion channel.

### Public Members

uint8\_t channelIndex

Sets the conversion channel index.

bool enableInt

Decides Whether to enable the interrupt function of the current conversion channel.

bool enablePresample

Decides whether to enable the pre-sample function of the current conversion channel.

bool enableDmaTransfer

Decides whether to enable the DMA transfer function of the current conversion channel.

bool enableWdg

Decides whether to enable the analog watchdog function of the current conversion channel.

uint8\_t wdgIndex

Indicates which analog watchdog to provide the low and high threshold value.

struct \_\_adc\_chain\_config

*#include <fsl\_sar\_adc.h>* This structure is used to configure the ADC conversion chain.

### Public Members

adc\_conv\_mode\_t convMode

Selects conversion mode.

bool enableGlobalChannelConvEndInt

Global control function to determine whether to enable the interrupt function of conversion channels.

bool enableChainConvEndInt

Decides whether to enable the current chain end-of-conversion interrupt.

uint8\_t channelCount

Indicates the channel counts.

adc\_channel\_config\_t \*channelConfig

Chain channels configuration.

struct \_\_adc\_wdg\_config

*#include <fsl\_sar\_adc.h>* This structure is used to configure the ADC analog watchdog.

**Public Members**

uint8\_t wdgIndex

Indicates the analog watchdog index

adc\_wdg\_threshold\_int\_t wdgThresholdInt

Selects watchdog threshold low or/and high interrupt to enable/disable.

uint16\_t lowThresholdVal

Sets the ADC analog watchdog low threshold value.

uint16\_t highThresholdVal

Sets the ADC analog watchdog high threshold value.

struct \_adc\_calibration\_config

*#include <fsl\_sar\_adc.h>* This structure is used to configure the ADC calibration.

**Public Members**

bool enableAverage

Decides whether to enable averaging of calibration time.

adc\_sample\_time\_t sampleTime

Selects sample time of calibration conversions.

adc\_average\_sample\_numbers\_t averageSampleNumbers

Selects calibration averaging sample numbers.

struct \_adc\_user\_offset\_gain\_config

*#include <fsl\_sar\_adc.h>* This structure is used to configure the ADC user offset and gain.

**Public Members**

int8\_t userOffset

Sets user defined gain value.

int16\_t userGain

Sets user defined offset value.

struct \_adc\_self\_test\_config

*#include <fsl\_sar\_adc.h>* This structure is used to configure the ADC self-test.

**Public Members**

adc\_alg\_type\_t algType

Selects the self-test algorithm.

uint8\_t algSteps

Sets the self-test algorithm steps, it should be programmed to zero in scan mode.

uint8\_t algSSamplePhaseDuration

Sets the self-test algorithm S conversion sampling phase duration.

uint8\_t algCSamplePhaseDuration

Sets the self-test algorithm C conversion sampling phase duration.

uint8\_t baudRate

Sets the baud rate for the selected algorithm in scan mode, must write to this field before enabling a self-test channel.

struct `_adc_self_test_wdg_config`

*#include <fsl\_sar\_adc.h>* This structure is used to configure the ADC self-test watchdog for algorithm steps.

---

**Note:** The algorithm S step 2 only has the ‘LowThrsholdVal’.

---

### Public Members

`adc_self_test_wdg_threshold_t` `wdgThresholdId`

Indicates the self-test watchdog index

`uint16_t` `lowThrsholdVal`

Sets the self-test watchdog low threshold value.

`uint16_t` `highThrsholdVal`

Sets the self-test watchdog high threshold value.

struct `_adc_conv_result`

*#include <fsl\_sar\_adc.h>* This structure is used to save the result information when obtaining the conversion result.

### Public Members

`bool` `overWrittenFlag`

Indicates when conversion data was overwritten by a newer result, the new data is written or discarded according to MCR[OWREN].

`uint8_t` `convMode`

Indicates the mode of conversion for the corresponding channel.

`uint16_t` `convData`

Stores the conversion data corresponding to the internal channel.

struct `_adc_self_test_conv_result`

*#include <fsl\_sar\_adc.h>* This structure is used to save the result information when obtaining the self-test channel conversion result.

---

**Note:** The member ‘convData’ is used to store self-test channel conversion results. Only when executing step 1 of algorithm S, member ‘convDataFraction’ will be used to store the fractional part data. When executing other algorithms, this member will not be used.

---

### Public Members

`bool` `overWrittenFlag`

Indicates when conversion data is overwritten by a newer result.

`uint16_t` `convData`

Stores the conversion data corresponding to the internal self-test channel.

`uint16_t` `convDataFraction`

This field is only used to store the fractional part conversion result.

## 2.56 SEMA42: Hardware Semaphores Driver

FSL\_SEMA42\_DRIVER\_VERSION

SEMA42 driver version.

SEMA42 status return codes.

*Values:*

enumerator kStatus\_SEMA42\_Busy

SEMA42 gate has been locked by other processor.

enumerator kStatus\_SEMA42\_Reseting

SEMA42 gate resetting is ongoing.

enum \_sema42\_gate\_status

SEMA42 gate lock status.

*Values:*

enumerator kSEMA42\_Unlocked

The gate is unlocked.

enumerator kSEMA42\_LockedByProc0

The gate is locked by processor 0.

enumerator kSEMA42\_LockedByProc1

The gate is locked by processor 1.

enumerator kSEMA42\_LockedByProc2

The gate is locked by processor 2.

enumerator kSEMA42\_LockedByProc3

The gate is locked by processor 3.

enumerator kSEMA42\_LockedByProc4

The gate is locked by processor 4.

enumerator kSEMA42\_LockedByProc5

The gate is locked by processor 5.

enumerator kSEMA42\_LockedByProc6

The gate is locked by processor 6.

enumerator kSEMA42\_LockedByProc7

The gate is locked by processor 7.

enumerator kSEMA42\_LockedByProc8

The gate is locked by processor 8.

enumerator kSEMA42\_LockedByProc9

The gate is locked by processor 9.

enumerator kSEMA42\_LockedByProc10

The gate is locked by processor 10.

enumerator kSEMA42\_LockedByProc11

The gate is locked by processor 11.

enumerator kSEMA42\_LockedByProc12

The gate is locked by processor 12.

enumerator kSEMA42\_LockedByProc13

The gate is locked by processor 13.

enumerator kSEMA42\_LockedByProc14

The gate is locked by processor 14.

typedef enum *\_sema42\_gate\_status* sema42\_gate\_status\_t  
SEMA42 gate lock status.

void SEMA42\_Init(SEMA42\_Type \*base)

Initializes the SEMA42 module.

This function initializes the SEMA42 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either SEMA42\_ResetGate or SEMA42\_ResetAllGates function.

#### Parameters

- base – SEMA42 peripheral base address.

void SEMA42\_Deinit(SEMA42\_Type \*base)

De-initializes the SEMA42 module.

This function de-initializes the SEMA42 module. It only disables the clock.

#### Parameters

- base – SEMA42 peripheral base address.

*status\_t* SEMA42\_TryLock(SEMA42\_Type \*base, uint8\_t gateNum, uint8\_t procNum)

Tries to lock the SEMA42 gate.

This function tries to lock the specific SEMA42 gate. If the gate has been locked by another processor, this function returns an error code.

#### Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

#### Return values

- kStatus\_Success – Lock the sema42 gate successfully.
- kStatus\_SEMA42\_Busy – Sema42 gate has been locked by another processor.

*status\_t* SEMA42\_Lock(SEMA42\_Type \*base, uint8\_t gateNum, uint8\_t procNum)

Locks the SEMA42 gate.

This function locks the specific SEMA42 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

If SEMA42\_BUSY\_POLL\_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus\_Timeout.

#### Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

#### Return values

- kStatus\_Success – The gate was successfully locked.



- `kStatus_Timeout` – Timeout occurred while waiting for the gate to be unlocked.

**Returns**

`status_t`

`static inline void SEMA42_Unlock(SEMA42_Type *base, uint8_t gateNum)`

Unlocks the SEMA42 gate.

This function unlocks the specific SEMA42 gate. It only writes unlock value to the SEMA42 gate register. However, it does not check whether the SEMA42 gate is locked by the current processor or not. As a result, if the SEMA42 gate is not locked by the current processor, this function has no effect.

**Parameters**

- `base` – SEMA42 peripheral base address.
- `gateNum` – Gate number to unlock.

`static inline sema42_gate_status_t SEMA42_GetGateStatus(SEMA42_Type *base, uint8_t gateNum)`

Gets the status of the SEMA42 gate.

This function checks the lock status of a specific SEMA42 gate.

**Parameters**

- `base` – SEMA42 peripheral base address.
- `gateNum` – Gate number.

**Returns**

`status_t` Current status.

`status_t SEMA42_ResetGate(SEMA42_Type *base, uint8_t gateNum)`

Resets the SEMA42 gate to an unlocked status.

This function resets a SEMA42 gate to an unlocked status.

**Parameters**

- `base` – SEMA42 peripheral base address.
- `gateNum` – Gate number.

**Return values**

- `kStatus_Success` – SEMA42 gate is reset successfully.
- `kStatus_SEMA42_Reseting` – Some other reset process is ongoing.

`static inline status_t SEMA42_ResetAllGates(SEMA42_Type *base)`

Resets all SEMA42 gates to an unlocked status.

This function resets all SEMA42 gate to an unlocked status.

**Parameters**

- `base` – SEMA42 peripheral base address.

**Return values**

- `kStatus_Success` – SEMA42 is reset successfully.
- `kStatus_SEMA42_Reseting` – Some other reset process is ongoing.

`SEMA42_GATE_NUM_RESET_ALL`

The number to reset all SEMA42 gates.

SEMA42\_GATE<sub>n</sub>(base, n)

SEMA42 gate n register address.

The SEMA42 gates are sorted in the order 3, 2, 1, 0, 7, 6, 5, 4, ... not in the order 0, 1, 2, 3, 4, 5, 6, 7, ... The macro SEMA42\_GATE<sub>n</sub> gets the SEMA42 gate based on the gate index.

The input gate index is XOR'ed with 3U:  $0 \wedge 3 = 3$   $1 \wedge 3 = 2$   $2 \wedge 3 = 1$   $3 \wedge 3 = 0$   $4 \wedge 3 = 7$   $5 \wedge 3 = 6$   $6 \wedge 3 = 5$   $7 \wedge 3 = 4$  ...

SEMA42\_BUSY\_POLL\_COUNT

Maximum polling iterations for SEMA42 waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the SEMA42 driver code before timing out and returning an error.

It applies to all waiting loops in SEMA42 driver, such as waiting for a gate to be unlocked, waiting for a reset to complete, or waiting for a resource to become available.

This is a count of loop iterations, not a time-based value.

If defined as 0, polling loops will continue indefinitely until their exit condition is met, which could potentially cause the system to hang if hardware doesn't respond or if a resource is never released.

## 2.57 TMU: Thermal Management Unit Driver

enum \_\_tmu\_monitor\_site

TMU monitor site.

*Values:*

enumerator kTMU\_MonitorSite0

TMU monitoring site 0.

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.

enumerator kTMU\_RisingTemperatureCriticalInterruptEnable

Rising temperature critical threshold exceeded interrupt enable.

enumerator kTMU\_FallingTemperatureCriticalInterruptEnable

Falling temperature critical threshold exceeded interrupt enable.

enum \_\_tmu\_interrupt\_status\_flags

TMU interrupt status flags.

*Values:*

enumerator kTMU\_ImmediateTemperatureStatusFlags

Immediate temperature threshold exceeded(IHTT).

enumerator kTMU\_AverageTemperatureStatusFlags

Average temperature threshold exceeded(AHTT).

enumerator kTMU\_AverageTemperatureCriticalStatusFlags

Average temperature critical threshold exceeded.(AHTCT)

enumerator kTMU\_RisingTemperatureCriticalStatusFlags

Rising temperature critical threshold exceeded.(RTRCT)

enumerator kTMU\_FallingTemperatureCriticalStatusFlags

Falling temperature critical threshold exceeded.(FTRCT)

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 160 °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 0-160 °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 0-160 °C for an enabled monitored site.

**Returns**

Execution status.

`status_t TMU_GetImmediateTemperature(TMU_Type *base, float *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.
- temperature – Last immediate temperature reading at site 0.

**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 \*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.
- temperature – Last average temperature reading at site 0.

**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 thresold.

**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 2.1.0.

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.

bool risingCriticalThresoldEnable

Enable rising temperature rate critical threshold.

bool fallingCriticalThresoldEnable

Enable rising temperature rate critical threshold.

uint8\_t immediateThresoldValue

Range:0U-160U. 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-160U. Valid when corresponding thresold is enabled. High temperature average threshold value. Determines the average upper temperature threshold, for any enabled monitored site.

uint8\_t averageCriticalThresoldValue

Range:0U-160U. Valid when corresponding thresold is enabled. High temperature average critical threshold value. Determines the average upper critical temperature threshold, for any enabled monitored site.

uint8\_t risingfallingCriticalThresoldValue

Range:0U-160U. Valid when corresponding thresold is enabled. Rising temperature rate critical threshold value. Determines the rising 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 IHTT event. Please refer to “\_tmu\_monitor\_site” enumeration.

uint16\_t averageInterruptsSiteMask

The mask of the temperature sensor site associated with a detected AHTT event. Please refer to “\_tmu\_monitor\_site” enumeration.

uint16\_t averageCriticalInterruptsSiteMask

The mask of the temperature sensor site associated with a detected AHTCT event. Please refer to “\_tmu\_monitor\_site” enumeration.

uint16\_t risingCriticalInterruptsSiteMask

The mask of the temperature sensor site associated with a detected RTRCT event. Please refer to “\_tmu\_monitor\_site” enumeration.

uint16\_t fallingCriticalInterruptsSiteMask

The mask of the temperature sensor site associated with a detected FTRCT 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.

## 2.58 TPM: Timer PWM Module

uint32\_t TPM\_GetInstance(TPM\_Type \*base)

Gets the instance from the base address.

### Parameters

- base – TPM peripheral base address

### Returns

The TPM instance

void TPM\_Init(TPM\_Type \*base, const tpm\_config\_t \*config)

Ungates the TPM clock and configures the peripheral for basic operation.

---

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

---

### Parameters

- base – TPM peripheral base address
- config – Pointer to user's TPM config structure.

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

Stops the counter and gates the TPM clock.

### Parameters

- base – TPM peripheral base address

void TPM\_GetDefaultConfig(tpm\_config\_t \*config)

Fill in the TPM config struct with the default settings.

The default values are:

```
config->prescale = kTPM_Prescale_Divide_1;
config->useGlobalTimeBase = false;
config->syncGlobalTimeBase = false;
config->dozeEnable = false;
config->dbgMode = false;
config->enableReloadOnTrigger = false;
config->enableStopOnOverflow = false;
config->enableStartOnTrigger = false;
#if FSL_FEATURE_TPM_HAS_PAUSE_COUNTER_ON_TRIGGER
config->enablePauseOnTrigger = false;
#endif
config->triggerSelect = kTPM_Trigger_Select_0;
#if FSL_FEATURE_TPM_HAS_EXTERNAL_TRIGGER_SELECTION
config->triggerSource = kTPM_TriggerSource_External;
config->extTriggerPolarity = kTPM_ExtTrigger_Active_High;
```

(continues on next page)



(continued from previous page)

```
#endif
#if defined(FSL_FEATURE_TPM_HAS_POL) && FSL_FEATURE_TPM_HAS_POL
    config->chnlPolarity = 0U;
#endif
```

### Parameters

- config – Pointer to user's TPM config structure.

*tpm\_clock\_prescale\_t* TPM\_CalculateCounterClkDiv(TPM\_Type \*base, uint32\_t counterPeriod\_Hz, uint32\_t srcClock\_Hz)

Calculates the counter clock prescaler.

This function calculates the values for SC[PS].

return Calculated clock prescaler value.

### Parameters

- base – TPM peripheral base address
- counterPeriod\_Hz – The desired frequency in Hz which corresponding to the time when the counter reaches the mod value
- srcClock\_Hz – TPM counter clock in Hz

static inline void TPM\_Reset(TPM\_Type \*base)

Performs a software reset on the TPM module.

Reset all internal logic and registers, except the Global Register. Remains set until cleared by software.

---

**Note:** TPM software reset is available on certain SoC's only

---

### Parameters

- base – TPM peripheral base address

*status\_t* TPM\_SetupPwm(TPM\_Type \*base, const *tpm\_chnl\_pwm\_signal\_param\_t* \*chnlParams, uint8\_t numOfChnls, *tpm\_pwm\_mode\_t* mode, uint32\_t pwmFreq\_Hz, uint32\_t srcClock\_Hz)

Configures the PWM signal parameters.

User calls this function to configure the PWM signals period, mode, dutycycle and edge. Use this function to configure all the TPM channels that will be used to output a PWM signal

### Parameters

- base – TPM peripheral base address
- chnlParams – Array of PWM channel parameters to configure the channel(s)
- numOfChnls – Number of channels to configure, this should be the size of the array passed in
- mode – PWM operation mode, options available in enumeration *tpm\_pwm\_mode\_t*
- pwmFreq\_Hz – PWM signal frequency in Hz
- srcClock\_Hz – TPM counter clock in Hz

**Returns**

kStatus\_Success PWM setup successful  
kStatus\_Error PWM setup failed  
kStatus\_Timeout PWM setup timeout when write register CnV or MOD

*status\_t* TPM\_UpdatePwmDutycycle(TPM\_Type \*base, *tpm\_chnl\_t* chnlNumber, *tpm\_pwm\_mode\_t* currentPwmMode, uint8\_t dutyCyclePercent)

Update the duty cycle of an active PWM signal.

**Parameters**

- base – TPM peripheral base address
- chnlNumber – The channel number. In combined mode, this represents the channel pair number
- currentPwmMode – The current PWM mode set during PWM setup
- dutyCyclePercent – New PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle)

**Returns**

kStatus\_Success if the PWM setup was successful, kStatus\_Error on failure

void TPM\_UpdateChnlEdgeLevelSelect(TPM\_Type \*base, *tpm\_chnl\_t* chnlNumber, uint8\_t level)

Update the edge level selection for a channel.

---

**Note:** When the TPM has PWM pause level select feature (FSL\_FEATURE\_TPM\_HAS\_PAUSE\_LEVEL\_SELECT = 1), the PWM output cannot be turned off by selecting the output level. In this case, must use TPM\_DisableChannel API to close the PWM output.

---

**Parameters**

- base – TPM peripheral base address
- chnlNumber – The channel number
- level – The level to be set to the ELSnB:ELSnA field; valid values are 00, 01, 10, 11. See the appropriate SoC reference manual for details about this field.

static inline uint8\_t TPM\_GetChannelControlBits(TPM\_Type \*base, *tpm\_chnl\_t* chnlNumber)

Get the channel control bits value (mode, edge and level bit fields).

This function disable the channel by clear all mode and level control bits.

**Parameters**

- base – TPM peripheral base address
- chnlNumber – The channel number

**Returns**

The control bits value. This is the logical OR of members of the enumeration *tpm\_chnl\_control\_bit\_mask\_t*.

static inline *status\_t* TPM\_DisableChannel(TPM\_Type \*base, *tpm\_chnl\_t* chnlNumber)

Disable the channel.

This function disable the channel by clear all mode and level control bits.

**Parameters**

- base – TPM peripheral base address
- chnlNumber – The channel number

**Returns**

kStatus\_Success PWM setup successful kStatus\_Timeout PWM setup timeout when write register CnSC

```
static inline status_t TPM_EnableChannel(TPM_Type *base, tpm_chnl_t chnlNumber, uint8_t control)
```

Enable the channel according to mode and level configs.

This function enable the channel output according to input mode/level config parameters.

**Parameters**

- base – TPM peripheral base address
- chnlNumber – The channel number
- control – The control bits value. This is the logical OR of members of the enumeration *tpm\_chnl\_control\_bit\_mask\_t*.

**Returns**

kStatus\_Success PWM setup successful kStatus\_Timeout PWM setup timeout when write register CnSC

```
void TPM_SetupInputCapture(TPM_Type *base, tpm_chnl_t chnlNumber, tpm_input_capture_edge_t captureMode)
```

Enables capturing an input signal on the channel using the function parameters.

When the edge specified in the captureMode argument occurs on the channel, the TPM counter is captured into the CnV register. The user has to read the CnV register separately to get this value.

**Parameters**

- base – TPM peripheral base address
- chnlNumber – The channel number
- captureMode – Specifies which edge to capture

```
status_t TPM_SetupOutputCompare(TPM_Type *base, tpm_chnl_t chnlNumber, tpm_output_compare_mode_t compareMode, uint32_t compareValue)
```

Configures the TPM to generate timed pulses.

When the TPM counter matches the value of compareVal argument (this is written into CnV reg), the channel output is changed based on what is specified in the compareMode argument.

**Parameters**

- base – TPM peripheral base address
- chnlNumber – The channel number
- compareMode – Action to take on the channel output when the compare condition is met
- compareValue – Value to be programmed in the CnV register.

**Returns**

kStatus\_Success PWM setup successful kStatus\_Timeout PWM setup timeout when write register CnV

```
void TPM_SetupDualEdgeCapture(TPM_Type *base, tpm_chnl_t chnlPairNumber, const tpm_dual_edge_capture_param_t *edgeParam, uint32_t filterValue)
```

Configures the dual edge capture mode of the TPM.

This function allows to measure a pulse width of the signal on the input of channel of a channel pair. The filter function is disabled if the filterVal argument passed is zero.

**Parameters**

- base – TPM peripheral base address
- chnlPairNumber – The TPM channel pair number; options are 0, 1, 2, 3
- edgeParam – Sets up the dual edge capture function
- filterValue – Filter value, specify 0 to disable filter.

```
void TPM_SetupQuadDecode(TPM_Type *base, const tpm_phase_params_t *phaseAParams,  
                        const tpm_phase_params_t *phaseBParams,  
                        tpm_quad_decode_mode_t quadMode)
```

Configures the parameters and activates the quadrature decode mode.

**Parameters**

- base – TPM peripheral base address
- phaseAParams – Phase A configuration parameters
- phaseBParams – Phase B configuration parameters
- quadMode – Selects encoding mode used in quadrature decoder mode

```
static inline void TPM_SetChannelPolarity(TPM_Type *base, tpm_chnl_t chnlNumber, bool  
                                         enable)
```

Set the input and output polarity of each of the channels.

**Parameters**

- base – TPM peripheral base address
- chnlNumber – The channel number
- enable – true: Set the channel polarity to active high; false: Set the channel polarity to active low;

```
static inline void TPM_EnableChannelExtTrigger(TPM_Type *base, tpm_chnl_t chnlNumber, bool  
                                              enable)
```

Enable external trigger input to be used by channel.

In input capture mode, configures the trigger input that is used by the channel to capture the counter value. In output compare or PWM mode, configures the trigger input used to modulate the channel output. When modulating the output, the output is forced to the channel initial value whenever the trigger is not asserted.

---

**Note:** No matter how many external trigger sources there are, only input trigger 0 and 1 are used. The even numbered channels share the input trigger 0 and the odd numbered channels share the second input trigger 1.

---

**Parameters**

- base – TPM peripheral base address
- chnlNumber – The channel number
- enable – true: Configures trigger input 0 or 1 to be used by channel; false: Trigger input has no effect on the channel

```
void TPM_EnableInterrupts(TPM_Type *base, uint32_t mask)
```

Enables the selected TPM interrupts.

**Parameters**

- base – TPM peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration `tpm_interrupt_enable_t`

`void TPM_DisableInterrupts(TPM_Type *base, uint32_t mask)`

Disables the selected TPM interrupts.

#### Parameters

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

`uint32_t TPM_GetEnabledInterrupts(TPM_Type *base)`

Gets the enabled TPM interrupts.

#### Parameters

- base – TPM peripheral base address

#### Returns

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

`void TPM_RegisterCallback(TPM_Type *base, tpm_callback_t callback)`

Register callback.

If channel or overflow interrupt is enabled by the user, then a callback can be registered which will be invoked when the interrupt is triggered.

#### Parameters

- base – TPM peripheral base address
- callback – Callback function

`void TPM_DriverIRQHandler(uint32_t instance)`

TPM driver IRQ handler common entry.

This function provides the common IRQ request entry for TPM.

#### Parameters

- instance – TPM instance.

`static inline uint32_t TPM_GetChannelValue(TPM_Type *base, tpm_chnl_t chnlNumber)`

Gets the TPM channel value.

---

**Note:** The TPM channel value contain the captured TPM counter value for the input modes or the match value for the output modes.

---

#### Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number

#### Returns

The channle CnV regisyer value.

`static inline uint32_t TPM_GetStatusFlags(TPM_Type *base)`

Gets the TPM status flags.

#### Parameters

- base – TPM peripheral base address

**Returns**

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

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

Clears the TPM status flags.

**Parameters**

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

static inline *status\_t* TPM\_SetTimerPeriod(TPM\_Type \*base, uint32\_t ticks)

Sets the timer period in units of ticks.

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

---

**Note:**

- This API allows the user to use the TPM module as a timer. Do not mix usage of this API with TPM's PWM setup API's.
  - Call the utility macros provided in the `fsl_common.h` to convert usec or msec to ticks.
- 

**Parameters**

- `base` – TPM peripheral base address
- `ticks` – A timer period in units of ticks, which should be equal or greater than 1.

**Returns**

`kStatus_Success` PWM setup successful  
`kStatus_Timeout` PWM setup timeout when write register CnSC

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

Reads the current timer counting value.

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

---

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

---

**Parameters**

- `base` – TPM peripheral base address

**Returns**

The current counter value in ticks

static inline void TPM\_StartTimer(TPM\_Type \*base, *tpm\_clock\_source\_t* clockSource)

Starts the TPM counter.

**Parameters**

- `base` – TPM peripheral base address
- `clockSource` – TPM clock source; once clock source is set the counter will start running

static inline *status\_t* TPM\_StopTimer(TPM\_Type \*base)

Stops the TPM counter.

**Parameters**

- base – TPM peripheral base address

**Returns**

kStatus\_Success PWM setup successful  
kStatus\_Timeout PWM setup timeout  
when write register CnSC

FSL\_TPM\_DRIVER\_VERSION

TPM driver version 2.4.0.

enum *\_tpm\_chnl*

List of TPM channels.

---

**Note:** Actual number of available channels is SoC dependent

---

*Values:*

enumerator kTPM\_Chnl\_0  
TPM channel number 0

enumerator kTPM\_Chnl\_1  
TPM channel number 1

enumerator kTPM\_Chnl\_2  
TPM channel number 2

enumerator kTPM\_Chnl\_3  
TPM channel number 3

enumerator kTPM\_Chnl\_4  
TPM channel number 4

enumerator kTPM\_Chnl\_5  
TPM channel number 5

enumerator kTPM\_Chnl\_6  
TPM channel number 6

enumerator kTPM\_Chnl\_7  
TPM channel number 7

enum *\_tpm\_pwm\_mode*

TPM PWM operation modes.

*Values:*

enumerator kTPM\_EdgeAlignedPwm  
Edge aligned PWM

enumerator kTPM\_CenterAlignedPwm  
Center aligned PWM

enumerator kTPM\_CombinedPwm  
Combined PWM (Edge-aligned, center-aligned, or asymmetrical PWMs can be obtained in combined mode using different software configurations)

enum \_tpm\_pwm\_level\_select

TPM PWM output pulse mode: high-true, low-true or no output.

---

**Note:** When the TPM has PWM pause level select feature, the PWM output cannot be turned off by selecting the output level. In this case, the channel must be closed to close the PWM output.

---

*Values:*

enumerator kTPM\_HighTrue

High true pulses

enumerator kTPM\_LowTrue

Low true pulses

enum \_tpm\_pwm\_pause\_level\_select

TPM PWM output when first enabled or paused: set or clear.

*Values:*

enumerator kTPM\_ClearOnPause

Clear Output when counter first enabled or paused.

enumerator kTPM\_SetOnPause

Set Output when counter first enabled or paused.

enum \_tpm\_chnl\_control\_bit\_mask

List of TPM channel modes and level control bit mask.

*Values:*

enumerator kTPM\_ChnlELSnAMask

Channel ELSA bit mask.

enumerator kTPM\_ChnlELSnBMask

Channel EL SB bit mask.

enumerator kTPM\_ChnlMSAMask

Channel MSA bit mask.

enumerator kTPM\_ChnlMSBMask

Channel MSB bit mask.

enum \_tpm\_trigger\_select

Trigger sources available.

This is used for both internal & external trigger sources (external trigger sources available in certain SoC's)

---

**Note:** The actual trigger sources available is SoC-specific.

---

*Values:*

enumerator kTPM\_Trigger\_Select\_0

enumerator kTPM\_Trigger\_Select\_1

enumerator kTPM\_Trigger\_Select\_2

enumerator kTPM\_Trigger\_Select\_3



enumerator kTPM\_Trigger\_Select\_4  
enumerator kTPM\_Trigger\_Select\_5  
enumerator kTPM\_Trigger\_Select\_6  
enumerator kTPM\_Trigger\_Select\_7  
enumerator kTPM\_Trigger\_Select\_8  
enumerator kTPM\_Trigger\_Select\_9  
enumerator kTPM\_Trigger\_Select\_10  
enumerator kTPM\_Trigger\_Select\_11  
enumerator kTPM\_Trigger\_Select\_12  
enumerator kTPM\_Trigger\_Select\_13  
enumerator kTPM\_Trigger\_Select\_14  
enumerator kTPM\_Trigger\_Select\_15

enum \_tpm\_trigger\_source

Trigger source options available.

---

**Note:** This selection is available only on some SoC's. For SoC's without this selection, the only trigger source available is internal trigger.

---

*Values:*

enumerator kTPM\_TriggerSource\_External

Use external trigger input

enumerator kTPM\_TriggerSource\_Internal

Use internal trigger (channel pin input capture)

enum \_tpm\_ext\_trigger\_polarity

External trigger source polarity.

---

**Note:** Selects the polarity of the external trigger source.

---

*Values:*

enumerator kTPM\_ExtTrigger\_Active\_High

External trigger input is active high

enumerator kTPM\_ExtTrigger\_Active\_Low

External trigger input is active low

enum \_tpm\_output\_compare\_mode

TPM output compare modes.

*Values:*

enumerator kTPM\_NoOutputSignal

No channel output when counter reaches CnV

enumerator kTPM\_ToggleOnMatch

Toggle output

enumerator kTPM\_ClearOnMatch

Clear output

enumerator kTPM\_SetOnMatch

Set output

enumerator kTPM\_HighPulseOutput

Pulse output high

enumerator kTPM\_LowPulseOutput

Pulse output low

enum \_tpm\_input\_capture\_edge

TPM input capture edge.

*Values:*

enumerator kTPM\_RisingEdge

Capture on rising edge only

enumerator kTPM\_FallingEdge

Capture on falling edge only

enumerator kTPM\_RiseAndFallEdge

Capture on rising or falling edge

enum \_tpm\_quad\_decode\_mode

TPM quadrature decode modes.

---

**Note:** This mode is available only on some SoC's.

---

*Values:*

enumerator kTPM\_QuadPhaseEncode

Phase A and Phase B encoding mode

enumerator kTPM\_QuadCountAndDir

Count and direction encoding mode

enum \_tpm\_phase\_polarity

TPM quadrature phase polarities.

*Values:*

enumerator kTPM\_QuadPhaseNormal

Phase input signal is not inverted

enumerator kTPM\_QuadPhaseInvert

Phase input signal is inverted

enum \_tpm\_clock\_source

TPM clock source selection.

*Values:*

enumerator kTPM\_SystemClock

System clock

enumerator kTPM\_ExternalClock

External TPM\_EXTCLK pin clock

enumerator kTPM\_ExternalInputTriggerClock  
Selected external input trigger clock

enum \_tpm\_clock\_prescale  
TPM prescale value selection for the clock source.

*Values:*

enumerator kTPM\_Prescale\_Divide\_1  
Divide by 1

enumerator kTPM\_Prescale\_Divide\_2  
Divide by 2

enumerator kTPM\_Prescale\_Divide\_4  
Divide by 4

enumerator kTPM\_Prescale\_Divide\_8  
Divide by 8

enumerator kTPM\_Prescale\_Divide\_16  
Divide by 16

enumerator kTPM\_Prescale\_Divide\_32  
Divide by 32

enumerator kTPM\_Prescale\_Divide\_64  
Divide by 64

enumerator kTPM\_Prescale\_Divide\_128  
Divide by 128

enum \_tpm\_interrupt\_enable  
List of TPM interrupts.

*Values:*

enumerator kTPM\_Chnl0InterruptEnable  
Channel 0 interrupt.

enumerator kTPM\_Chnl1InterruptEnable  
Channel 1 interrupt.

enumerator kTPM\_Chnl2InterruptEnable  
Channel 2 interrupt.

enumerator kTPM\_Chnl3InterruptEnable  
Channel 3 interrupt.

enumerator kTPM\_Chnl4InterruptEnable  
Channel 4 interrupt.

enumerator kTPM\_Chnl5InterruptEnable  
Channel 5 interrupt.

enumerator kTPM\_Chnl6InterruptEnable  
Channel 6 interrupt.

enumerator kTPM\_Chnl7InterruptEnable  
Channel 7 interrupt.

enumerator kTPM\_TimeOverflowInterruptEnable  
Time overflow interrupt.

enum *\_tpm\_status\_flags*

List of TPM flags.

*Values:*

enumerator *kTPM\_Chnl0Flag*

Channel 0 flag

enumerator *kTPM\_Chnl1Flag*

Channel 1 flag

enumerator *kTPM\_Chnl2Flag*

Channel 2 flag

enumerator *kTPM\_Chnl3Flag*

Channel 3 flag

enumerator *kTPM\_Chnl4Flag*

Channel 4 flag

enumerator *kTPM\_Chnl5Flag*

Channel 5 flag

enumerator *kTPM\_Chnl6Flag*

Channel 6 flag

enumerator *kTPM\_Chnl7Flag*

Channel 7 flag

enumerator *kTPM\_TimeOverflowFlag*

Time overflow flag

typedef enum *\_tpm\_chnl* *tpm\_chnl\_t*

List of TPM channels.

---

**Note:** Actual number of available channels is SoC dependent

---

typedef enum *\_tpm\_pwm\_mode* *tpm\_pwm\_mode\_t*

TPM PWM operation modes.

typedef enum *\_tpm\_pwm\_level\_select* *tpm\_pwm\_level\_select\_t*

TPM PWM output pulse mode: high-true, low-true or no output.

---

**Note:** When the TPM has PWM pause level select feature, the PWM output cannot be turned off by selecting the output level. In this case, the channel must be closed to close the PWM output.

---

typedef enum *\_tpm\_pwm\_pause\_level\_select* *tpm\_pwm\_pause\_level\_select\_t*

TPM PWM output when first enabled or paused: set or clear.

typedef enum *\_tpm\_chnl\_control\_bit\_mask* *tpm\_chnl\_control\_bit\_mask\_t*

List of TPM channel modes and level control bit mask.

typedef struct *\_tpm\_chnl\_pwm\_signal\_param* *tpm\_chnl\_pwm\_signal\_param\_t*

Options to configure a TPM channel's PWM signal.

```
typedef enum _tpm_trigger_select tpm_trigger_select_t
```

Trigger sources available.

This is used for both internal & external trigger sources (external trigger sources available in certain SoC's)

---

**Note:** The actual trigger sources available is SoC-specific.

---

```
typedef enum _tpm_trigger_source tpm_trigger_source_t
```

Trigger source options available.

---

**Note:** This selection is available only on some SoC's. For SoC's without this selection, the only trigger source available is internal trigger.

---

```
typedef enum _tpm_ext_trigger_polarity tpm_ext_trigger_polarity_t
```

External trigger source polarity.

---

**Note:** Selects the polarity of the external trigger source.

---

```
typedef enum _tpm_output_compare_mode tpm_output_compare_mode_t
```

TPM output compare modes.

```
typedef enum _tpm_input_capture_edge tpm_input_capture_edge_t
```

TPM input capture edge.

```
typedef struct _tpm_dual_edge_capture_param tpm_dual_edge_capture_param_t
```

TPM dual edge capture parameters.

---

**Note:** This mode is available only on some SoC's.

---

```
typedef enum _tpm_quad_decode_mode tpm_quad_decode_mode_t
```

TPM quadrature decode modes.

---

**Note:** This mode is available only on some SoC's.

---

```
typedef enum _tpm_phase_polarity tpm_phase_polarity_t
```

TPM quadrature phase polarities.

```
typedef struct _tpm_phase_param tpm_phase_params_t
```

TPM quadrature decode phase parameters.

```
typedef enum _tpm_clock_source tpm_clock_source_t
```

TPM clock source selection.

```
typedef enum _tpm_clock_prescale tpm_clock_prescale_t
```

TPM prescale value selection for the clock source.

```
typedef struct _tpm_config tpm_config_t
```

TPM config structure.

This structure holds the configuration settings for the TPM peripheral. To initialize this structure to reasonable defaults, call the `TPM_GetDefaultConfig()` function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

`typedef enum _tpm_interrupt_enable tpm_interrupt_enable_t`

List of TPM interrupts.

`typedef enum _tpm_status_flags tpm_status_flags_t`

List of TPM flags.

`typedef void (*tpm_callback_t)(TPM_Type *base)`

TPM callback function pointer.

**Param base**

TPM peripheral base address.

`TPM_TIMEOUT`

Max loops to wait for writing register.

When writing MOD CnV CnSC and SC register, driver will wait until register is updated. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

`TPM_MAX_COUNTER_VALUE(x)`

Help macro to get the max counter value.

`struct _tpm_chnl_pwm_signal_param`

*#include <fsl\_tpm.h>* Options to configure a TPM channel's PWM signal.

**Public Members**

`tpm_chnl_t` chnlNumber

TPM channel to configure. In combined mode (available in some SoC's), this represents the channel pair number

`tpm_pwm_pause_level_select_t` pauseLevel

PWM output level when counter first enabled or paused

`tpm_pwm_level_select_t` level

PWM output active level select

`uint8_t` dutyCyclePercent

PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=always active signal (100% duty cycle)

`uint8_t` firstEdgeDelayPercent

Used only in combined PWM mode to generate asymmetrical PWM. Specifies the delay to the first edge in a PWM period. If unsure, leave as 0. Should be specified as percentage of the PWM period, (dutyCyclePercent + firstEdgeDelayPercent) value should be not greater than 100.

`bool` enableComplementary

Used only in combined PWM mode. true: The combined channels output complementary signals; false: The combined channels output same signals;

`tpm_pwm_pause_level_select_t` secPauseLevel

Used only in combined PWM mode. Define the second channel output level when counter first enabled or paused

`uint8_t` deadTimeValue[2]

The dead time value for channel n and n+1 in combined complementary PWM mode. Deadtime insertion is disabled when this value is zero, otherwise deadtime insertion for channel n/n+1 is configured as (deadTimeValue \* 4) clock cycles. deadTimeValue's available range is 0 ~ 15.

struct `_tpm_dual_edge_capture_param`  
*#include <fsl\_tpm.h>* TPM dual edge capture parameters.

---

**Note:** This mode is available only on some SoC's.

---

### Public Members

bool enableSwap  
true: Use channel n+1 input, channel n input is ignored; false: Use channel n input, channel n+1 input is ignored

*tpm\_input\_capture\_edge\_t* currChanEdgeMode  
Input capture edge select for channel n

*tpm\_input\_capture\_edge\_t* nextChanEdgeMode  
Input capture edge select for channel n+1

struct `_tpm_phase_param`  
*#include <fsl\_tpm.h>* TPM quadrature decode phase parameters.

### Public Members

uint32\_t phaseFilterVal  
Filter value, filter is disabled when the value is zero

*tpm\_phase\_polarity\_t* phasePolarity  
Phase polarity

struct `_tpm_config`  
*#include <fsl\_tpm.h>* TPM config structure.

This structure holds the configuration settings for the TPM peripheral. To initialize this structure to reasonable defaults, call the `TPM_GetDefaultConfig()` function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

### Public Members

*tpm\_clock\_prescale\_t* prescale  
Select TPM clock prescale value

bool useGlobalTimeBase  
true: The TPM channels use an external global time base (the local counter still use for generate overflow interrupt and DMA request); false: All TPM channels use the local counter as their timebase

bool syncGlobalTimeBase  
true: The TPM counter is synchronized to the global time base; false: disabled

*tpm\_trigger\_select\_t* triggerSelect  
Input trigger to use for controlling the counter operation

*tpm\_trigger\_source\_t* triggerSource  
Decides if we use external or internal trigger.

*tpm\_ext\_trigger\_polarity\_t* extTriggerPolarity

when using external trigger source, need selects the polarity of it.

bool enableDoze

true: TPM counter is paused in doze mode; false: TPM counter continues in doze mode

bool enableDebugMode

true: TPM counter continues in debug mode; false: TPM counter is paused in debug mode

bool enableReloadOnTrigger

true: TPM counter is reloaded on trigger; false: TPM counter not reloaded

bool enableStopOnOverflow

true: TPM counter stops after overflow; false: TPM counter continues running after overflow

bool enableStartOnTrigger

true: TPM counter only starts when a trigger is detected; false: TPM counter starts immediately

bool enablePauseOnTrigger

true: TPM counter will pause while trigger remains asserted; false: TPM counter continues running

uint8\_t chnlPolarity

Defines the input/output polarity of the channels in POL register

## 2.59 TRDC: Trusted Resource Domain Controller

void TRDC\_\_Init(*TRDC\_Type* \*base)

Initializes the TRDC module.

This function enables the TRDC clock.

### Parameters

- base – TRDC peripheral base address.

void TRDC\_\_Deinit(*TRDC\_Type* \*base)

De-initializes the TRDC module.

This function disables the TRDC clock.

### Parameters

- base – TRDC peripheral base address.

static inline uint8\_t TRDC\_\_GetCurrentMasterDomainId(*TRDC\_Type* \*base)

Gets the domain ID of the current bus master.

### Parameters

- base – TRDC peripheral base address.

### Returns

Domain ID of current bus master.

void TRDC\_\_GetHardwareConfig(*TRDC\_Type* \*base, *trdc\_hardware\_config\_t* \*config)

Gets the TRDC hardware configuration.

This function gets the TRDC hardware configurations, including number of bus masters, number of domains, number of MRCs and number of PACs.



**Parameters**

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

static inline void TRDC\_SetDacGlobalValid(*TRDC\_Type* \*base)

Sets the TRDC DAC(Domain Assignment Controllers) global valid.

Once enabled, it will remain enabled until next reset.

**Parameters**

- base – TRDC peripheral base address.

static inline void TRDC\_LockMasterDomainAssignment(*TRDC\_Type* \*base, uint8\_t master, uint8\_t regNum)

Locks the bus master domain assignment register.

This function locks the master domain assignment. After it is locked, the register can't be changed until next reset.

**Parameters**

- base – TRDC peripheral base address.
- master – Which master to configure, refer to trdcx\_master\_t in processor header file, x is trdc instance.
- regNum – Which register to configure, processor master can have more than one register for the MDAC configuration.
- assignIndex – Which assignment register to lock.

static inline void TRDC\_SetMasterDomainAssignmentValid(*TRDC\_Type* \*base, uint8\_t master, uint8\_t regNum, bool valid)

Sets the master domain assignment as valid or invalid.

This function sets the master domain assignment as valid or invalid.

**Parameters**

- base – TRDC peripheral base address.
- master – Which master to configure.
- regNum – Which register to configure, processor master can have more than one register for the MDAC configuration.
- assignIndex – Index for the domain assignment register.
- valid – True to set valid, false to set invalid.

void TRDC\_GetDefaultProcessorDomainAssignment(*trdc\_processor\_domain\_assignment\_t* \*domainAssignment)

Gets the default master domain assignment for the processor bus master.

This function gets the default master domain assignment for the processor bus master. It should only be used for the processor bus masters, such as CORE0. This function sets the assignment as follows:

```
assignment->domainId      = 0U;
assignment->domainIdSelect = kTRDC_DidMda;
assignment->lock           = 0U;
```

**Parameters**

- domainAssignment – Pointer to the assignment structure.

```
void TRDC_GetDefaultNonProcessorDomainAssignment(trdc_non_processor_domain_assignment_t
                                                *domainAssignment)
```

Gets the default master domain assignment for non-processor bus master.

This function gets the default master domain assignment for non-processor bus master. It should only be used for the non-processor bus masters, such as DMA. This function sets the assignment as follows:

```
assignment->domainId      = 0U;
assignment->privilegeAttr  = kTRDC_ForceUser;
assignment->secureAttr     = kTRDC_ForceSecure;
assignment->bypassDomainId = 0U;
assignment->lock           = 0U;
```

### Parameters

- domainAssignment – Pointer to the assignment structure.

```
void TRDC_SetProcessorDomainAssignment(TRDC_Type *base, uint8_t master, uint8_t regNum,
                                       const trdc_processor_domain_assignment_t
                                       *domainAssignment)
```

Sets the processor bus master domain assignment.

This function sets the processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter `assignIndex` specifies which assignment register to set.

Example: Set domain assignment for core 0.

```
trdc_processor_domain_assignment_t processorAssignment;

TRDC_GetDefaultProcessorDomainAssignment(&processorAssignment);

processorAssignment.domainId = 0;
processorAssignment.xxx      = xxx;
TRDC_SetMasterDomainAssignment(TRDC, &processorAssignment);
```

### Parameters

- base – TRDC peripheral base address.
- master – Which master to configure, refer to `trdc_master_t` in processor header file.
- regNum – Which register to configure, processor master can have more than one register for the MDAC configuration.
- domainAssignment – Pointer to the assignment structure.

```
void TRDC_SetNonProcessorDomainAssignment(TRDC_Type *base, uint8_t master, const
                                          trdc_non_processor_domain_assignment_t
                                          *domainAssignment)
```

Sets the non-processor bus master domain assignment.

This function sets the non-processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter `assignIndex` specifies which assignment register to set.

Example: Set domain assignment for DMA0.

```
trdc_non_processor_domain_assignment_t nonProcessorAssignment;

TRDC_GetDefaultNonProcessorDomainAssignment(&nonProcessorAssignment);
```

(continues on next page)

(continued from previous page)

```

nonProcessorAssignment.domainId = 1;
nonProcessorAssignment.xxx      = xxx;

TRDC_SetMasterDomainAssignment(TRDC, kTrdcMasterDma0, 0U, &nonProcessorAssignment);

```

**Parameters**

- base – TRDC peripheral base address.
- master – Which master to configure, refer to `trdc_master_t` in processor header file.
- domainAssignment – Pointer to the assignment structure.

static inline uint64\_t TRDC\_GetActiveMasterPidMap(*TRDC\_Type* \*base)

Gets the bit map of the bus master(s) that is(are) sourcing a PID register.

This function sets the non-processor master domain assignment as valid.

**Parameters**

- base – TRDC peripheral base address.

**Returns**

the bit map of the master(s). Bit 1 sets indicates bus master 1.

void TRDC\_SetPid(*TRDC\_Type* \*base, uint8\_t master, const *trdc\_pid\_config\_t* \*pidConfig)

Sets the current Process identifier(PID) for processor core.

Each processor has a corresponding process identifier (PID) which can be used to group tasks into different domains. Secure privileged software saves and restores the PID as part of any context switch. This data structure defines an array of 32-bit values, one per MDA module, that define the PID. Since this register resource is only applicable to processor cores, the data structure is typically sparsely populated. The `HWCFG[2-3]` registers provide a bitmap of the implemented PIDn registers. This data structure is indexed using the corresponding MDA instance number. Depending on the operating clock domain of each DAC instance, there may be optional information stored in the corresponding PIDm register to properly implement the LK2 = 2 functionality.

**Parameters**

- base – TRDC peripheral base address.
- master – Which processor master to configure, refer to `trdc_master_t` in processor header file.
- pidConfig – Pointer to the configuration structure.

void TRDC\_GetDefaultIDAUConfig(*trdc\_idau\_config\_t* \*idauConfiguration)

Gets the default IDAU(Implementation-Defined Attribution Unit) configuration.

```

config->lockSecureVTOR   = false;
config->lockNonsecureVTOR = false;
config->lockSecureMPU     = false;
config->lockNonsecureMPU  = false;
config->lockSAU           = false;

```

**Parameters**

- domainAssignment – Pointer to the configuration structure.

void TRDC\_SetIDAU(*TRDC\_Type* \*base, const *trdc\_idau\_config\_t* \*idauConfiguration)

Sets the IDAU(Implementation-Defined Attribution Unit) control configuration.

Example: Lock the secure and non-secure MPU registers.

```
trdc_idau_config_t idauConfiguration;

TRDC_GetDefaultIDAUConfig(&idauConfiguration);

idauConfiguration.lockSecureMPU = true;
idauConfiguration.lockNonsecureMPU = true;
TRDC_SetIDAU(TRDC, &idauConfiguration);
```

### Parameters

- base – TRDC peripheral base address.
- domainAssignment – Pointer to the configuration structure.

static inline void TRDC\_EnableFlashLogicalWindow(*TRDC\_Type* \*base, bool enable)

Enables/disables the FLW(flash logical window) function.

### Parameters

- base – TRDC peripheral base address.
- enable – True to enable, false to disable.

static inline void TRDC\_LockFlashLogicalWindow(*TRDC\_Type* \*base)

Locks FLW registers. Once locked the registers can not be updated until next reset.

### Parameters

- base – TRDC peripheral base address.

static inline uint32\_t TRDC\_GetFlashLogicalWindowPbase(*TRDC\_Type* \*base)

Gets the FLW physical base address.

### Parameters

- base – TRDC peripheral base address.

### Returns

Physical address of the FLW function.

static inline void TRDC\_GetSetFlashLogicalWindowSize(*TRDC\_Type* \*base, uint16\_t size)

Sets the FLW size.

### Parameters

- base – TRDC peripheral base address.
- size – Size of the FLW in unit of 32k bytes.

void TRDC\_GetDefaultFlashLogicalWindowConfig(*trdc\_flw\_config\_t* \*flwConfiguration)

Gets the default FLW(Flash Logical Window) configuration.

```
config->blockCount = false;
config->arrayBaseAddr = false;
config->lock = false;
config->enable = false;
```

### Parameters

- flwConfiguration – Pointer to the configuration structure.

void TRDC\_SetFlashLogicalWindow(*TRDC\_Type* \*base, const *trdc\_flw\_config\_t* \*flwConfiguration)

Sets the FLW function's configuration.

```
trdc_flw_config_t flwConfiguration;

TRDC_GetDefaultIDAUConfig(&flwConfiguration);

flwConfiguration.blockCount = 32U;
flwConfiguration.arrayBaseAddr = 0xFFFFFFFF;
TRDC_SetIDAU(TRDC, &flwConfiguration);
```

### Parameters

- base – TRDC peripheral base address.
- flwConfiguration – Pointer to the configuration structure.

*status\_t* TRDC\_GetAndClearFirstDomainError(*TRDC\_Type* \*base, *trdc\_domain\_error\_t* \*error)

Gets and clears the first domain error of the current domain.

This function gets the first access violation information for the current domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

### Parameters

- base – TRDC peripheral base address.
- error – Pointer to the error information.

### Returns

If the access violation is captured, this function returns the *kStatus\_Success*. The error information can be obtained from the parameter error. If no access violation is captured, this function returns the *kStatus\_NoData*.

*status\_t* TRDC\_GetAndClearFirstSpecificDomainError(*TRDC\_Type* \*base, *trdc\_domain\_error\_t* \*error, *uint8\_t* domainId)

Gets and clears the first domain error of the specific domain.

This function gets the first access violation information for the specific domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

### Parameters

- base – TRDC peripheral base address.
- error – Pointer to the error information.
- domainId – The error of which domain to get and clear.

### Returns

If the access violation is captured, this function returns the *kStatus\_Success*. The error information can be obtained from the parameter error. If no access violation is captured, this function returns the *kStatus\_NoData*.

static inline void TRDC\_SetMrcGlobalValid(*TRDC\_Type* \*base)

Sets the TRDC MRC(Memory Region Checkers) global valid.

Once enabled, it will remain enabled until next reset.

### Parameters

- base – TRDC peripheral base address.

static inline *uint8\_t* TRDC\_GetMrcRegionNumber(*TRDC\_Type* \*base, *uint8\_t* mrcIdx)

Gets the TRDC MRC(Memory Region Checkers) region number valid.

### Parameters

- base – TRDC peripheral base address.

**Returns**

the region number of the given MRC instance

```
void TRDC_MrcSetMemoryAccessConfig(TRDC_Type *base, const  
                                   trdc_memory_access_control_config_t *config, uint8_t  
                                   mrcIdx, uint8_t regIdx)
```

Sets the memory access configuration for one of the access control register of one MRC.

Example: Enable the secure operations and lock the configuration for MRC0 region 1.

```
trdc_memory_access_control_config_t config;  
  
config.securePrivX = true;  
config.securePrivW = true;  
config.securePrivR = true;  
config.lock = true;  
TRDC_SetMrcMemoryAccess(TRDC, &config, 0, 1);
```

**Parameters**

- base – TRDC peripheral base address.
- config – Pointer to the configuration structure.
- mrcIdx – MRC index.
- regIdx – Register number.

```
void TRDC_MrcEnableDomainNseUpdate(TRDC_Type *base, uint8_t mrcIdx, uint16_t  
                                   domianMask, bool enable)
```

Enables the update of the selected domians.

After the domians' update are enabled, their regions' NSE bits can be set or clear.

**Parameters**

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- domianMask – Bit mask of the domains to be enabled.
- enable – True to enable, false to disable.

```
void TRDC_MrcRegionNseSet(TRDC_Type *base, uint8_t mrcIdx, uint16_t regionMask)
```

Sets the NSE bits of the selected regions for domains.

This function sets the NSE bits for the selected regions for the domains whose update are enabled.

**Parameters**

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- regionMask – Bit mask of the regions whose NSE bits to set.

```
void TRDC_MrcRegionNseClear(TRDC_Type *base, uint8_t mrcIdx, uint16_t regionMask)
```

Clears the NSE bits of the selected regions for domains.

This function clears the NSE bits for the selected regions for the domains whose update are enabled.

**Parameters**

- base – TRDC peripheral base address.
- mrcIdx – MRC index.

- `regionMask` – Bit mask of the regions whose NSE bits to clear.

`void TRDC__MrcDomainNseClear(TRDC_Type *base, uint8_t mrcIdx, uint16_t domainMask)`

Clears the NSE bits for all the regions of the selected domains.

This function clears the NSE bits for all regions of selected domains whose update are enabled.

#### Parameters

- `base` – TRDC peripheral base address.
- `mrcIdx` – MRC index.
- `domainMask` – Bit mask of the domains whose NSE bits to clear.

`void TRDC__MrcSetRegionDescriptorConfig(TRDC_Type *base, const trdc_mrc_region_descriptor_config_t *config)`

Sets the configuration for one of the region descriptor per domain per MRC instance.

This function sets the configuration for one of the region descriptor, including the start and end address of the region, memory access control policy and valid.

#### Parameters

- `base` – TRDC peripheral base address.
- `config` – Pointer to region descriptor configuration structure.

`static inline void TRDC__SetMbcGlobalValid(TRDC_Type *base)`

Sets the TRDC MBC(Memory Block Checkers) global valid.

Once enabled, it will remain enabled until next reset.

#### Parameters

- `base` – TRDC peripheral base address.

`void TRDC__GetMbcHardwareConfig(TRDC_Type *base, trdc_slave_memory_hardware_config_t *config, uint8_t mbcIdx, uint8_t slvIdx)`

Gets the hardware configuration of the one of two slave memories within each MBC(memory block checker).

#### Parameters

- `base` – TRDC peripheral base address.
- `config` – Pointer to the structure to get the configuration.
- `mbcIdx` – MBC number.
- `slvIdx` – Slave number.

`void TRDC__MbcSetNseUpdateConfig(TRDC_Type *base, const trdc_mbc_nse_update_config_t *config, uint8_t mbcIdx)`

Sets the NSR update configuration for one of the MBC instance.

After set the NSE configuration, the configured memory area can be update by NSE set/clear.

#### Parameters

- `base` – TRDC peripheral base address.
- `config` – Pointer to NSE update configuration structure.
- `mbcIdx` – MBC index.

`void TRDC__MbcWordNseSet(TRDC_Type *base, uint8_t mbcIdx, uint32_t bitMask)`

Sets the NSE bits of the selected configuration words according to NSE update configuration.

This function sets the NSE bits of the word for the configured region, memory.

**Parameters**

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- bitMask – Mask of the bits whose NSE bits to set.

void TRDC\_\_MbcWordNseClear(*TRDC\_Type* \*base, uint8\_t mbcIdx, uint32\_t bitMask)

Clears the NSE bits of the selected configuration words according to NSE update configuration.

This function sets the NSE bits of the word for the configured regio, memory.

**Parameters**

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- bitMask – Mask of the bits whose NSE bits to clear.

void TRDC\_\_MbcNseClearAll(*TRDC\_Type* \*base, uint8\_t mbcIdx, uint16\_t domainMask, uint8\_t slave)

Clears all configuration words' NSE bits of the selected domain and memory.

**Parameters**

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- domainMask – Mask of the domains whose NSE bits to clear, 0b110 means clear domain 1&2.
- slaveMask – Mask of the slaves whose NSE bits to clear, 0x11 means clear all slave 0&1's NSE bits.

void TRDC\_\_MbcSetMemoryAccessConfig(*TRDC\_Type* \*base, const *trdc\_memory\_access\_control\_config\_t* \*config, uint8\_t mbcIdx, uint8\_t rgdIdx)

Sets the memory access configuration for one of the region descriptor of one MBC.

Example: Enable the secure operations and lock the configuration for MRC0 region 1.

```
trdc_memory_access_control_config_t config;

config.securePrivX = true;
config.securePrivW = true;
config.securePrivR = true;
config.lock = true;
TRDC_SetMbcMemoryAccess(TRDC, &config, 0, 1);
```

**Parameters**

- base – TRDC peripheral base address.
- config – Pointer to the configuration structure.
- mbcIdx – MBC index.
- rgdIdx – Region descriptor number.

void TRDC\_\_MbcSetMemoryBlockConfig(*TRDC\_Type* \*base, const *trdc\_mbc\_memory\_block\_config\_t* \*config)

Sets the configuration for one of the memory block per domain per MBC instance.

This function sets the configuration for one of the memory block, including the memory access control policy and nse enable.



**Parameters**

- base – TRDC peripheral base address.
- config – Pointer to memory block configuration structure.

enum `_trdc_did_sel`

TRDC domain ID select method, the register bit `TRDC_MDA_W0_0_DFMT0[DIDS]`, used for domain hit evaluation.

*Values:*

enumerator `kTRDC_DidMda`

Use `MDAn[2:0]` as DID.

enumerator `kTRDC_DidInput`

Use the input DID (`DID_in`) as DID.

enumerator `kTRDC_DidMdaAndInput`

Use `MDAn[2]` concatenated with `DID_in[1:0]` as DID.

enumerator `kTRDC_DidReserved`

Reserved.

enum `_trdc_secure_attr`

TRDC secure attribute, the register bit `TRDC_MDA_W0_0_DFMT0[SA]`, used for bus master domain assignment.

*Values:*

enumerator `kTRDC_ForceSecure`

Force the bus attribute for this master to secure.

enumerator `kTRDC_ForceNonSecure`

Force the bus attribute for this master to non-secure.

enumerator `kTRDC_MasterSecure`

Use the bus master's secure/nonsecure attribute directly.

enumerator `kTRDC_MasterSecure1`

Use the bus master's secure/nonsecure attribute directly.

enum `_trdc_pid_domain_hit_config`

The configuration of domain hit evaluation of PID.

*Values:*

enumerator `kTRDC_pidDomainHitNone0`

No PID is included in the domain hit evaluation.

enumerator `kTRDC_pidDomainHitNone1`

No PID is included in the domain hit evaluation.

enumerator `kTRDC_pidDomainHitInclusive`

The PID is included in the domain hit evaluation when  $(PID \& \sim PIDM)$ .

enumerator `kTRDC_pidDomainHitExclusive`

The PID is included in the domain hit evaluation when  $\sim(PID \& \sim PIDM)$ .

enum `_trdc_privilege_attr`

TRDC privileged attribute, the register bit `TRDC_MDA_W0_x_DFMT1[PA]`, used for non-processor bus master domain assignment.

*Values:*

enumerator kTRDC\_ForceUser

Force the bus attribute for this master to user.

enumerator kTRDC\_ForcePrivilege

Force the bus attribute for this master to privileged.

enumerator kTRDC\_MasterPrivilege

Use the bus master's attribute directly.

enumerator kTRDC\_MasterPrivilege1

Use the bus master's attribute directly.

enum \_trdc\_pid\_lock

PID lock configuration.

*Values:*

enumerator kTRDC\_PidUnlocked0

The PID value can be updated by any secure privileged write.

enumerator kTRDC\_PidUnlocked1

The PID value can be updated by any secure privileged write.

enumerator kTRDC\_PidUnlocked2

The PID value can be updated by any secure privileged write from the bus master that first configured this register.

enumerator kTRDC\_PidLocked

The PID value is locked until next reset.

enum \_trdc\_controller

TRDC controller definition for domain error check. Each TRDC instance may have different MRC or MBC count, call TRDC\_GetHardwareConfig to get the actual count.

*Values:*

enumerator kTRDC\_MemBlockController0

Memory block checker 0.

enumerator kTRDC\_MemBlockController1

Memory block checker 1.

enumerator kTRDC\_MemBlockController2

Memory block checker 2.

enumerator kTRDC\_MemBlockController3

Memory block checker 3.

enumerator kTRDC\_MemRegionChecker0

Memory region checker 0.

enumerator kTRDC\_MemRegionChecker1

Memory region checker 1.

enumerator kTRDC\_MemRegionChecker2

Memory region checker 2.

enumerator kTRDC\_MemRegionChecker3

Memory region checker 3.

enumerator kTRDC\_MemRegionChecker4

Memory region checker 4.

enumerator kTRDC\_MemRegionChecker5

Memory region checker 5.

enumerator kTRDC\_MemRegionChecker6

Memory region checker 6.

enum \_trdc\_error\_state

|                         |            |                        |    |
|-------------------------|------------|------------------------|----|
| TRDC domain error state | definition | TRDC_MBCn_DERR_W1[EST] | or |
| TRDC_MRCn_DERR_W1[EST]. |            |                        |    |

*Values:*

enumerator kTRDC\_ErrorStateNone

No access violation detected.

enumerator kTRDC\_ErrorStateNone1

No access violation detected.

enumerator kTRDC\_ErrorStateSingle

Single access violation detected.

enumerator kTRDC\_ErrorStateMulti

Multiple access violation detected.

enum \_trdc\_error\_attr

|                             |            |                         |    |
|-----------------------------|------------|-------------------------|----|
| TRDC domain error attribute | definition | TRDC_MBCn_DERR_W1[EATR] | or |
| TRDC_MRCn_DERR_W1[EATR].    |            |                         |    |

*Values:*

enumerator kTRDC\_ErrorSecureUserInst

Secure user mode, instruction fetch access.

enumerator kTRDC\_ErrorSecureUserData

Secure user mode, data access.

enumerator kTRDC\_ErrorSecurePrivilegeInst

Secure privileged mode, instruction fetch access.

enumerator kTRDC\_ErrorSecurePrivilegeData

Secure privileged mode, data access.

enumerator kTRDC\_ErrorNonSecureUserInst

NonSecure user mode, instruction fetch access.

enumerator kTRDC\_ErrorNonSecureUserData

NonSecure user mode, data access.

enumerator kTRDC\_ErrorNonSecurePrivilegeInst

NonSecure privileged mode, instruction fetch access.

enumerator kTRDC\_ErrorNonSecurePrivilegeData

NonSecure privileged mode, data access.

enum \_trdc\_error\_type

TRDC domain error access type definition TRDC\_DERR\_W1\_n[ERW].

*Values:*

enumerator kTRDC\_ErrorTypeRead

Error occurs on read reference.

enumerator kTRDC\_ErrorTypeWrite

Error occurs on write reference.

enum `_trdc_region_descriptor`

The region descriptor enumeration, used to form a mask to set/clear the NSE bits for one or several regions.

*Values:*

enumerator `kTRDC_RegionDescriptor0`

Region descriptor 0.

enumerator `kTRDC_RegionDescriptor1`

Region descriptor 1.

enumerator `kTRDC_RegionDescriptor2`

Region descriptor 2.

enumerator `kTRDC_RegionDescriptor3`

Region descriptor 3.

enumerator `kTRDC_RegionDescriptor4`

Region descriptor 4.

enumerator `kTRDC_RegionDescriptor5`

Region descriptor 5.

enumerator `kTRDC_RegionDescriptor6`

Region descriptor 6.

enumerator `kTRDC_RegionDescriptor7`

Region descriptor 7.

enumerator `kTRDC_RegionDescriptor8`

Region descriptor 8.

enumerator `kTRDC_RegionDescriptor9`

Region descriptor 9.

enumerator `kTRDC_RegionDescriptor10`

Region descriptor 10.

enumerator `kTRDC_RegionDescriptor11`

Region descriptor 11.

enumerator `kTRDC_RegionDescriptor12`

Region descriptor 12.

enumerator `kTRDC_RegionDescriptor13`

Region descriptor 13.

enumerator `kTRDC_RegionDescriptor14`

Region descriptor 14.

enumerator `kTRDC_RegionDescriptor15`

Region descriptor 15.

enum `_trdc_MRC_domain`

The MRC domain enumeration, used to form a mask to enable/disable the update or clear all NSE bits of one or several domains.

*Values:*

enumerator `kTRDC_MrcDomain0`

Domain 0.

enumerator kTRDC\_MrcDomain1

Domain 1.

enumerator kTRDC\_MrcDomain2

Domain 2.

enumerator kTRDC\_MrcDomain3

Domain 3.

enumerator kTRDC\_MrcDomain4

Domain 4.

enumerator kTRDC\_MrcDomain5

Domain 5.

enumerator kTRDC\_MrcDomain6

Domain 6.

enumerator kTRDC\_MrcDomain7

Domain 7.

enumerator kTRDC\_MrcDomain8

Domain 8.

enumerator kTRDC\_MrcDomain9

Domain 9.

enumerator kTRDC\_MrcDomain10

Domain 10.

enumerator kTRDC\_MrcDomain11

Domain 11.

enumerator kTRDC\_MrcDomain12

Domain 12.

enumerator kTRDC\_MrcDomain13

Domain 13.

enumerator kTRDC\_MrcDomain14

Domain 14.

enumerator kTRDC\_MrcDomain15

Domain 15.

enum \_trdc\_MBC\_domain

The MBC domain enumeration, used to form a mask to enable/disable the update or clear NSE bits of one or several domains.

*Values:*

enumerator kTRDC\_MbcDomain0

Domain 0.

enumerator kTRDC\_MbcDomain1

Domain 1.

enumerator kTRDC\_MbcDomain2

Domain 2.

enumerator kTRDC\_MbcDomain3

Domain 3.

enumerator kTRDC\_MbcDomain4  
Domain 4.

enumerator kTRDC\_MbcDomain5  
Domain 5.

enumerator kTRDC\_MbcDomain6  
Domain 6.

enumerator kTRDC\_MbcDomain7  
Domain 7.

enum \_trdc\_MBC\_memory

The MBC slave memory enumeration, used to form a mask to enable/disable the update or clear NSE bits of one or several memory block.

*Values:*

enumerator kTRDC\_MbcSlaveMemory0  
Memory 0.

enumerator kTRDC\_MbcSlaveMemory1  
Memory 1.

enumerator kTRDC\_MbcSlaveMemory2  
Memory 2.

enumerator kTRDC\_MbcSlaveMemory3  
Memory 3.

enum \_trdc\_MBC\_bit

The MBC bit enumeration, used to form a mask to set/clear configured words' NSE.

*Values:*

enumerator kTRDC\_MbcBit0  
Bit 0.

enumerator kTRDC\_MbcBit1  
Bit 1.

enumerator kTRDC\_MbcBit2  
Bit 2.

enumerator kTRDC\_MbcBit3  
Bit 3.

enumerator kTRDC\_MbcBit4  
Bit 4.

enumerator kTRDC\_MbcBit5  
Bit 5.

enumerator kTRDC\_MbcBit6  
Bit 6.

enumerator kTRDC\_MbcBit7  
Bit 7.

enumerator kTRDC\_MbcBit8  
Bit 8.

enumerator kTRDC\_MbcBit9  
Bit 9.

enumerator kTRDC\_MbcBit10  
Bit 10.

enumerator kTRDC\_MbcBit11  
Bit 11.

enumerator kTRDC\_MbcBit12  
Bit 12.

enumerator kTRDC\_MbcBit13  
Bit 13.

enumerator kTRDC\_MbcBit14  
Bit 14.

enumerator kTRDC\_MbcBit15  
Bit 15.

enumerator kTRDC\_MbcBit16  
Bit 16.

enumerator kTRDC\_MbcBit17  
Bit 17.

enumerator kTRDC\_MbcBit18  
Bit 18.

enumerator kTRDC\_MbcBit19  
Bit 19.

enumerator kTRDC\_MbcBit20  
Bit 20.

enumerator kTRDC\_MbcBit21  
Bit 21.

enumerator kTRDC\_MbcBit22  
Bit 22.

enumerator kTRDC\_MbcBit23  
Bit 23.

enumerator kTRDC\_MbcBit24  
Bit 24.

enumerator kTRDC\_MbcBit25  
Bit 25.

enumerator kTRDC\_MbcBit26  
Bit 26.

enumerator kTRDC\_MbcBit27  
Bit 27.

enumerator kTRDC\_MbcBit28  
Bit 28.

enumerator kTRDC\_MbcBit29  
Bit 29.

enumerator kTRDC\_MbcBit30  
Bit 30.

enumerator kTRDC\_MbcBit31

Bit 31.

typedef struct *\_trdc\_hardware\_config* trdc\_hardware\_config\_t

TRDC hardware configuration.

typedef struct *\_trdc\_slave\_memory\_hardware\_config* trdc\_slave\_memory\_hardware\_config\_t

Hardware configuration of the two slave memories within each MBC(memory block checker).

typedef enum *\_trdc\_did\_sel* trdc\_did\_sel\_t

TRDC domain ID select method, the register bit TRDC\_MDA\_W0\_0\_DFMT0[DIDS], used for domain hit evaluation.

typedef enum *\_trdc\_secure\_attr* trdc\_secure\_attr\_t

TRDC secure attribute, the register bit TRDC\_MDA\_W0\_0\_DFMT0[SA], used for bus master domain assignment.

typedef enum *\_trdc\_pid\_domain\_hit\_config* trdc\_pid\_domain\_hit\_config\_t

The configuration of domain hit evaluation of PID.

typedef struct *\_trdc\_processor\_domain\_assignment* trdc\_processor\_domain\_assignment\_t

Domain assignment for the processor bus master.

typedef enum *\_trdc\_privilege\_attr* trdc\_privilege\_attr\_t

TRDC privileged attribute, the register bit TRDC\_MDA\_W0\_x\_DFMT1[PA], used for non-processor bus master domain assignment.

typedef struct *\_trdc\_non\_processor\_domain\_assignment*

trdc\_non\_processor\_domain\_assignment\_t

Domain assignment for the non-processor bus master.

typedef enum *\_trdc\_pid\_lock* trdc\_pid\_lock\_t

PID lock configuration.

typedef struct *\_trdc\_pid\_config* trdc\_pid\_config\_t

Process identifier(PID) configuration for processor cores.

typedef struct *\_trdc\_idau\_config* trdc\_idau\_config\_t

IDAU(Implementation-Defined Attribution Unit) configuration for TZ-M function control.

typedef struct *\_trdc\_flw\_config* trdc\_flw\_config\_t

FLW(Flash Logical Window) configuration.

typedef enum *\_trdc\_controller* trdc\_controller\_t

TRDC controller definition for domain error check. Each TRDC instance may have different MRC or MBC count, call TRDC\_GetHardwareConfig to get the actual count.

typedef enum *\_trdc\_error\_state* trdc\_error\_state\_t

TRDC domain error state definition TRDC\_MBCn\_DERR\_W1[EST] or TRDC\_MRCn\_DERR\_W1[EST].

typedef enum *\_trdc\_error\_attr* trdc\_error\_attr\_t

TRDC domain error attribute definition TRDC\_MBCn\_DERR\_W1[EATR] or TRDC\_MRCn\_DERR\_W1[EATR].

typedef enum *\_trdc\_error\_type* trdc\_error\_type\_t

TRDC domain error access type definition TRDC\_DERR\_W1\_n[ERW].

typedef struct *\_trdc\_domain\_error* trdc\_domain\_error\_t

TRDC domain error definition.



`typedef struct _trdc_memory_access_control_config trdc_memory_access_control_config_t`  
Memory access control configuration for MBC/MRC.

`typedef struct _trdc_mrc_region_descriptor_config trdc_mrc_region_descriptor_config_t`  
The configuration of each region descriptor per domain per MRC instance.

`typedef struct _trdc_mbc_nse_update_config trdc_mbc_nse_update_config_t`  
The configuration of MBC NSE update.

`typedef struct _trdc_mbc_memory_block_config trdc_mbc_memory_block_config_t`  
The configuration of each memory block per domain per MBC instance.

`FSL_TRDC_DRIVER_VERSION`

`struct _trdc_hardware_config`  
`#include <fsl_trdc.h>` TRDC hardware configuration.

### Public Members

`uint8_t masterNumber`  
Number of bus masters.

`uint8_t domainNumber`  
Number of domains.

`uint8_t mbcNumber`  
Number of MBCs.

`uint8_t mrcNumber`  
Number of MRCs.

`struct _trdc_slave_memory_hardware_config`  
`#include <fsl_trdc.h>` Hardware configuration of the two slave memories within each MBC(memory block checker).

### Public Members

`uint32_t blockNum`  
Number of blocks.

`uint32_t blockSize`  
Block size.

`struct _trdc_processor_domain_assignment`  
`#include <fsl_trdc.h>` Domain assignment for the processor bus master.

### Public Members

`uint32_t domainId`  
Domain ID.

`uint32_t domainIdSelect`  
Domain ID select method, see `trdc_did_sel_t`.

`uint32_t pidDomainHitConfig`  
The configuration of the domain hit evaluation for PID, see `trdc_pid_domain_hit_config_t`.

uint32\_t pidMask

The mask combined with PID, so multiple PID can be included as part of the domain hit determination. Set to 0 to disable.

uint32\_t secureAttr

Secure attribute, see trdc\_secure\_attr\_t.

uint32\_t pid

The process identifier, combined with pidMask to form the domain hit determination.

uint32\_t \_\_pad0\_\_

Reserved.

uint32\_t lock

Lock the register.

uint32\_t \_\_pad1\_\_

Reserved.

struct \_\_trdc\_non\_processor\_domain\_assignment

*#include <fsl\_trdc.h>* Domain assignment for the non-processor bus master.

### Public Members

uint32\_t domainId

Domain ID.

uint32\_t privilegeAttr

Privileged attribute, see trdc\_privilege\_attr\_t.

uint32\_t secureAttr

Secure attribute, see trdc\_secure\_attr\_t.

uint32\_t bypassDomainId

Bypass domain ID.

uint32\_t \_\_pad0\_\_

Reserved.

uint32\_t lock

Lock the register.

uint32\_t \_\_pad1\_\_

Reserved.

struct \_\_trdc\_pid\_config

*#include <fsl\_trdc.h>* Process identifier(PID) configuration for processor cores.

### Public Members

uint32\_t pid

The process identifier of the executing task. The highest bit can be used to define secure/nonsecure attribute of the task.

uint32\_t \_\_pad0\_\_

Reserved.

uint32\_t lock

How to lock the register, see trdc\_pid\_lock\_t.

uint32\_t \_\_pad1\_\_  
Reserved.

struct \_\_trdc\_idau\_config  
#include <fsl\_trdc.h> IDAU(Implementation-Defined Attribution Unit) configuration for TZ-M function control.

### Public Members

uint32\_t \_\_pad0\_\_  
Reserved.

uint32\_t lockSecureVTOR  
Disable writes to secure VTOR(Vector Table Offset Register).

uint32\_t lockNonsecureVTOR  
Disable writes to non-secure VTOR, Application interrupt and Reset Control Registers.

uint32\_t lockSecureMPU  
Disable writes to secure MPU(Memory Protection Unit) from software or from a debug agent connected to the processor in Secure state.

uint32\_t lockNonsecureMPU  
Disable writes to non-secure MPU(Memory Protection Unit) from software or from a debug agent connected to the processor.

uint32\_t lockSAU  
Disable writes to SAU(Security Attribution Unit) registers.

uint32\_t \_\_pad1\_\_  
Reserved.

struct \_\_trdc\_flw\_config  
#include <fsl\_trdc.h> FLW(Flash Logical Window) configuration.

### Public Members

uint16\_t blockCount  
Block count of the Flash Logic Window in 32KByte blocks.

uint32\_t arrayBaseAddr  
Flash array base address of the Flash Logical Window.

bool lock  
Disable writes to FLW registers.

bool enable  
Enable FLW function.

struct \_\_trdc\_domain\_error  
#include <fsl\_trdc.h> TRDC domain error definition.

### Public Members

trdc\_controller\_t controller  
Which controller captured access violation.

uint32\_t address

Access address that generated access violation.

trdc\_error\_state\_t errorState

Error state.

trdc\_error\_attr\_t errorAttr

Error attribute.

trdc\_error\_type\_t errorType

Error type.

uint8\_t errorPort

Error port.

uint8\_t domainId

Domain ID.

uint8\_t slaveMemoryIdx

The slave memory index. Only apply when violation in MBC.

struct \_trdc\_memory\_access\_control\_config

#include <fsl\_trdc.h> Memory access control configuration for MBC/MRC.

### Public Members

uint32\_t nonsecureUsrX

Allow nonsecure user execute access.

uint32\_t nonsecureUsrW

Allow nonsecure user write access.

uint32\_t nonsecureUsrR

Allow nonsecure user read access.

uint32\_t \_\_pad0\_\_

Reserved.

uint32\_t nonsecurePrivX

Allow nonsecure privilege execute access.

uint32\_t nonsecurePrivW

Allow nonsecure privilege write access.

uint32\_t nonsecurePrivR

Allow nonsecure privilege read access.

uint32\_t \_\_pad1\_\_

Reserved.

uint32\_t secureUsrX

Allow secure user execute access.

uint32\_t secureUsrW

Allow secure user write access.

uint32\_t secureUsrR

Allow secure user read access.

uint32\_t \_\_pad2\_\_

Reserved.

uint32\_t securePrivX

Allownsecure privilege execute access.

uint32\_t securePrivW

Allownsecure privilege write access.

uint32\_t securePrivR

Allownsecure privilege read access.

uint32\_t \_\_pad3\_\_

Reserved.

uint32\_t lock

Lock the configuration until next reset, only apply to access control register 0.

struct \_trdc\_mrc\_region\_descriptor\_config

*#include <fsl\_trdc.h>* The configuration of each region descriptor per domain per MRC instance.

### Public Members

uint8\_t memoryAccessControlSelect

Select one of the 8 access control policies for this region, for access cotrol policies see trdc\_memory\_access\_control\_config\_t.

uint32\_t startAddr

Physical start address.

bool valid

Lock the register.

bool nseEnable

Enable non-secure accesses and disable secure accesses.

uint32\_t endAddr

Physical start address.

uint8\_t mrcIdx

The index of the MRC for this configuration to take effect.

uint8\_t domainIdx

The index of the domain for this configuration to take effect.

uint8\_t regionIdx

The index of the region for this configuration to take effect.

struct \_trdc\_mbc\_nse\_update\_config

*#include <fsl\_trdc.h>* The configuration of MBC NSE update.

### Public Members

uint32\_t \_\_pad0\_\_

Reserved.

uint32\_t wordIdx

MBC configuration word index to be updated.

uint32\_t \_\_pad1\_\_

Reserved.

uint32\_t memorySelect

Bit mask of the selected memory to be updated. \_trdc\_MBC\_memory.

uint32\_t \_\_pad2\_\_

Reserved.

uint32\_t domainSelect

Bit mask of the selected domain to be updated. \_trdc\_MBC\_domain.

uint32\_t \_\_pad3\_\_

Reserved.

uint32\_t autoIncrement

Whether to increment the word index after current word is updated using this configuration.

struct \_trdc\_mbc\_memory\_block\_config

*#include <fsl\_trdc.h>* The configuration of each memory block per domain per MBC instance.

### Public Members

uint32\_t memoryAccessControlSelect

Select one of the 8 access control policies for this memory block, for access control policies see trdc\_memory\_access\_control\_config\_t.

uint32\_t nseEnable

Enable non-secure accesses and disable secure accesses.

uint32\_t mbcIdx

The index of the MBC for this configuration to take effect.

uint32\_t domainIdx

The index of the domain for this configuration to take effect.

uint32\_t slaveMemoryIdx

The index of the slave memory for this configuration to take effect.

uint32\_t memoryBlockIdx

The index of the memory block for this configuration to take effect.

## 2.60 Trdc\_core

typedef struct *\_TRDC\_General\_Type* TRDC\_General\_Type

TRDC general configuration register definition.

typedef struct *\_TRDC\_FLW\_Type* TRDC\_FLW\_Type

TRDC flash logical control register definition.

typedef struct *\_TRDC\_DomainError\_Type* TRDC\_DomainError\_Type

TRDC domain error register definition.

typedef struct *\_TRDC\_DomainAssignment\_Type* TRDC\_DomainAssignment\_Type

TRDC master domain assignment register definition.

typedef struct *\_TRDC\_MBC\_Type* TRDC\_MBC\_Type

TRDC MBC control register definition.

```
typedef struct _TRDC_MRC_Type TRDC_MRC_Type
    TRDC MRC control register definition. MRC_DOM0_RGD_W[region][word].
TRDC_GENERAL_BASE(base)
    TRDC base address convert macro.
TRDC_FLW_BASE(base)
TRDC_DOMAIN_ERROR_BASE(base)
TRDC_DOMAIN_ASSIGNMENT_BASE(base)
TRDC_MBC_BASE(base, instance)
TRDC_MRC_BASE(base, instance)
struct _TRDC_General_Type
    #include <fsl_trdc_core.h> TRDC general configuration register definition.
```

### Public Members

```
__IO uint32_t TRDC_CR
    TRDC Register, offset: 0x0
__I uint32_t TRDC_HWCFG0
    TRDC Hardware Configuration Register 0, offset: 0xF0
__I uint32_t TRDC_HWCFG1
    TRDC Hardware Configuration Register 1, offset: 0xF4
__I uint32_t TRDC_HWCFG2
    TRDC Hardware Configuration Register 2, offset: 0xF8
__I uint32_t TRDC_HWCFG3
    TRDC Hardware Configuration Register 3, offset: 0xFC
__I uint8_t DACFG [8]
    Domain Assignment Configuration Register, array offset: 0x100, array step: 0x1
__IO uint32_t TRDC_IDAU_CR
    TRDC IDAU Control Register, offset: 0x1C0
struct _TRDC_FLW_Type
    #include <fsl_trdc_core.h> TRDC flash logical control register definition.
```

### Public Members

```
__IO uint32_t TRDC_FLW_CTL
    TRDC FLW Control, offset: 0x1E0
__I uint32_t TRDC_FLW_PBASE
    TRDC FLW Physical Base, offset: 0x1E4
__IO uint32_t TRDC_FLW_ABASE
    TRDC FLW Array Base, offset: 0x1E8
__IO uint32_t TRDC_FLW_BCNT
    TRDC FLW Block Count, offset: 0x1EC
struct _TRDC_DomainError_Type
    #include <fsl_trdc_core.h> TRDC domain error register definition.
```

### Public Members

\_\_IO uint32\_t TRDC\_FDID  
TRDC Fault Domain ID, offset: 0x1FC

\_\_IO uint32\_t TRDC\_DERRLOC [16]  
TRDC Domain Error Location Register, array offset: 0x200, array step: 0x4

struct \_TRDC\_DomainAssignment\_Type  
*#include <fsl\_trdc\_core.h>* TRDC master domain assignment register definition.

### Public Members

\_\_IO uint32\_t PID [8]  
Process Identifier, array offset: 0x700, array step: 0x4

struct \_TRDC\_MBC\_Type  
*#include <fsl\_trdc\_core.h>* TRDC MBC control register definition.

### Public Members

\_\_IO uint32\_t MBC\_MEM\_GLB\_CFG [4]  
MBC Global Configuration Register, array offset: 0x10000, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_NSE\_BLK\_INDEX  
MBC NonSecure Enable Block Index, array offset: 0x10010, array step: 0x2000

\_\_IO uint32\_t MBC\_NSE\_BLK\_SET  
MBC NonSecure Enable Block Set, array offset: 0x10014, array step: 0x2000

\_\_IO uint32\_t MBC\_NSE\_BLK\_CLR  
MBC NonSecure Enable Block Clear, array offset: 0x10018, array step: 0x2000

\_\_IO uint32\_t MBC\_NSE\_BLK\_CLR\_ALL  
MBC NonSecure Enable Block Clear All, array offset: 0x1001C, array step: 0x2000

\_\_IO uint32\_t MBC\_MEMN\_GLBAC [8]  
MBC Global Access Control, array offset: 0x10020, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM0\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x10040, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM0\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10140, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM0\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10180, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM0\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x101A0, array step: index\*0x2000, index2\*0x4

\_\_IO uint32\_t MBC\_DOM0\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x101A8, array step: index\*0x2000, index2\*0x4



- \_\_\_IO uint32\_t MBC\_DOM0\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x101C8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM0\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x101D0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM0\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x101F0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM1\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x10240, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM1\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10340, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM1\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10380, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM1\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x103A0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM1\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x103A8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM1\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x103C8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM1\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x103D0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM1\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x103F0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM2\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x10440, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM2\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10540, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM2\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10580, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM2\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x105A0, array step: index\*0x2000, index2\*0x4

- \_\_\_IO uint32\_t MBC\_DOM2\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x105A8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM2\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x105C8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM2\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x105D0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM2\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x105F0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM3\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x10640, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM3\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10740, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM3\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10780, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM3\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x107A0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM3\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x107A8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM3\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x107C8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM3\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x107D0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM3\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x107F0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM4\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x10840, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM4\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10940, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM4\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10980, array step: index\*0x2000, index2\*0x4

- \_\_\_IO uint32\_t MBC\_DOM4\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x109A0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM4\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x109A8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM4\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x109C8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM4\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x109D0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM4\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x109F0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM5\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x10A40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM5\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10B40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM5\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10B80, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM5\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10BA0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM5\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10BA8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM5\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10BC8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM5\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10BD0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM5\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10BF0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM6\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x10C40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM6\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10D40, array step: index\*0x2000, index2\*0x4

- \_\_\_IO uint32\_t MBC\_DOM6\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10D80, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM6\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10DA0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM6\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10DA8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM6\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10DC8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM6\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10DD0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM6\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10DF0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM7\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x10E40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM7\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10F40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM7\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10F80, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM7\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10FA0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM7\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10FA8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM7\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10FC8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM7\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x10FD0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM7\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x10FF0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM8\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x11040, array step: index\*0x2000, index2\*0x4

- \_\_\_IO uint32\_t MBC\_DOM8\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11140, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM8\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11180, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM8\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x111A0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM8\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x111A8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM8\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x111C8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM8\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x111D0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM8\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x111F0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM9\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x11240, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM9\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11340, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM9\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11380, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM9\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x113A0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM9\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x113A8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM9\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x113C8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM9\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x113D0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM9\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x113F0, array step: index\*0x2000, index2\*0x4

- \_\_\_IO uint32\_t MBC\_DOM10\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x11440, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM10\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11540, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM10\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11580, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM10\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x115A0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM10\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x115A8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM10\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x115C8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM10\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x115D0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM10\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x115F0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM11\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x11640, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM11\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11740, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM11\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11780, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM11\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x117A0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM11\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x117A8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM11\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x117C8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM11\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x117D0, array step: index\*0x2000, index2\*0x4

- \_\_\_IO uint32\_t MBC\_DOM11\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x117F0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM12\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x11840, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM12\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11940, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM12\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11980, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM12\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x119A0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM12\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x119A8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM12\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x119C8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM12\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x119D0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM12\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x119F0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM13\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x11A40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM13\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11B40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM13\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11B80, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM13\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11BA0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM13\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11BA8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM13\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11BC8, array step: index\*0x2000, index2\*0x4

- \_\_\_IO uint32\_t MBC\_DOM13\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11BD0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM13\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11BF0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM14\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x11C40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM14\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11D40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM14\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11D80, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM14\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11DA0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM14\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11DA8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM14\_MEM2\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11DC8, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM14\_MEM3\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11DD0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM14\_MEM3\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11DF0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM15\_MEM0\_BLK\_CFG\_W [64]  
MBC Memory Block Configuration Word, array offset: 0x11E40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM15\_MEM0\_BLK\_NSE\_W [16]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11F40, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM15\_MEM1\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11F80, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM15\_MEM1\_BLK\_NSE\_W [2]  
MBC Memory Block NonSecure Enable Word, array offset: 0x11FA0, array step: index\*0x2000, index2\*0x4
- \_\_\_IO uint32\_t MBC\_DOM15\_MEM2\_BLK\_CFG\_W [8]  
MBC Memory Block Configuration Word, array offset: 0x11FA8, array step: index\*0x2000, index2\*0x4



```

__IO uint32_t MBC_DOM15_MEM2_BLK_NSE_W [2]
    MBC Memory Block NonSecure Enable Word, array offset: 0x11FC8, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM3_BLK_CFG_W [8]
    MBC Memory Block Configuration Word, array offset: 0x11FD0, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM3_BLK_NSE_W [2]
    MBC Memory Block NonSecure Enable Word, array offset: 0x11FF0, array step: index*0x2000, index2*0x4

struct _TRDC_MRC_Type
#include <fsl_trdc_core.h> TRDC MRC control register definition.
MRC_DOM0_RGD_W[region][word].

```

### Public Members

```

__I uint32_t MRC_GLBCFG
    MRC Global Configuration Register, array offset: 0x14000, array step: 0x1000

__IO uint32_t MRC_NSE_RGN_INDIRECT
    MRC NonSecure Enable Region Indirect, array offset: 0x14010, array step: 0x1000

__O uint32_t MRC_NSE_RGN_SET
    MRC NonSecure Enable Region Set, array offset: 0x14014, array step: 0x1000

__O uint32_t MRC_NSE_RGN_CLR
    MRC NonSecure Enable Region Clear, array offset: 0x14018, array step: 0x1000

__O uint32_t MRC_NSE_RGN_CLR_ALL
    MRC NonSecure Enable Region Clear All, array offset: 0x1401C, array step: 0x1000

__IO uint32_t MRC_GLBAC [8]
    MRC Global Access Control, array offset: 0x14020, array step: index*0x1000, index2*0x4

__IO uint32_t MRC_DOM0_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14040, array step: index*0x1000, index2*0x8, index3*0x4

__IO uint32_t MRC_DOM0_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x140C0, array step: 0x1000

__IO uint32_t MRC_DOM1_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14140, array step: index*0x1000, index2*0x8, index3*0x4

__IO uint32_t MRC_DOM1_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x141C0, array step: 0x1000

__IO uint32_t MRC_DOM2_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14240, array step: index*0x1000, index2*0x8, index3*0x4

__IO uint32_t MRC_DOM2_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x142C0, array step: 0x1000

__IO uint32_t MRC_DOM3_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14340, array step: index*0x1000, index2*0x8, index3*0x4

```

\_\_IO uint32\_t MRC\_DOM3\_RGD\_NSE  
MRC Region Descriptor NonSecure Enable, array offset: 0x143C0, array step: 0x1000

\_\_IO uint32\_t MRC\_DOM4\_RGD\_W [16][2]  
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14440, array step: index\*0x1000, index2\*0x8, index3\*0x4

\_\_IO uint32\_t MRC\_DOM4\_RGD\_NSE  
MRC Region Descriptor NonSecure Enable, array offset: 0x144C0, array step: 0x1000

\_\_IO uint32\_t MRC\_DOM5\_RGD\_W [16][2]  
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14540, array step: index\*0x1000, index2\*0x8, index3\*0x4

\_\_IO uint32\_t MRC\_DOM5\_RGD\_NSE  
MRC Region Descriptor NonSecure Enable, array offset: 0x145C0, array step: 0x1000

\_\_IO uint32\_t MRC\_DOM6\_RGD\_W [16][2]  
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14640, array step: index\*0x1000, index2\*0x8, index3\*0x4

\_\_IO uint32\_t MRC\_DOM6\_RGD\_NSE  
MRC Region Descriptor NonSecure Enable, array offset: 0x146C0, array step: 0x1000

\_\_IO uint32\_t MRC\_DOM7\_RGD\_W [16][2]  
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14740, array step: index\*0x1000, index2\*0x8, index3\*0x4

\_\_IO uint32\_t MRC\_DOM7\_RGD\_NSE  
MRC Region Descriptor NonSecure Enable, array offset: 0x147C0, array step: 0x1000

\_\_IO uint32\_t MRC\_DOM8\_RGD\_W [16][2]  
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14840, array step: index\*0x1000, index2\*0x8, index3\*0x4

\_\_IO uint32\_t MRC\_DOM8\_RGD\_NSE  
MRC Region Descriptor NonSecure Enable, array offset: 0x148C0, array step: 0x1000

\_\_IO uint32\_t MRC\_DOM9\_RGD\_W [16][2]  
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14940, array step: index\*0x1000, index2\*0x8, index3\*0x4

\_\_IO uint32\_t MRC\_DOM9\_RGD\_NSE  
MRC Region Descriptor NonSecure Enable, array offset: 0x149C0, array step: 0x1000

\_\_IO uint32\_t MRC\_DOM10\_RGD\_W [16][2]  
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14A40, array step: index\*0x1000, index2\*0x8, index3\*0x4

\_\_IO uint32\_t MRC\_DOM10\_RGD\_NSE  
MRC Region Descriptor NonSecure Enable, array offset: 0x14AC0, array step: 0x1000

\_\_IO uint32\_t MRC\_DOM11\_RGD\_W [16][2]  
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14B40, array step: index\*0x1000, index2\*0x8, index3\*0x4

\_\_IO uint32\_t MRC\_DOM11\_RGD\_NSE  
MRC Region Descriptor NonSecure Enable, array offset: 0x14BC0, array step: 0x1000

\_\_IO uint32\_t MRC\_DOM12\_RGD\_W [16][2]  
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14C40, array step: index\*0x1000, index2\*0x8, index3\*0x4

```

__IO uint32_t MRC_DOM12_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x14CC0, array step: 0x1000
__IO uint32_t MRC_DOM13_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14D40,
    array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM13_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x14DC0, array step: 0x1000
__IO uint32_t MRC_DOM14_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14E40,
    array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM14_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x14EC0, array step: 0x1000
__IO uint32_t MRC_DOM15_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14F40,
    array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM15_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x14FC0, array step: 0x1000
struct MBC_DERR

```

### Public Members

```

__I uint32_t W0
    MBC Domain Error Word0 Register, array offset: 0x400, array step: 0x10
__I uint32_t W1
    MBC Domain Error Word1 Register, array offset: 0x404, array step: 0x10
__O uint32_t W3
    MBC Domain Error Word3 Register, array offset: 0x40C, array step: 0x10
struct MRC_DERR

```

### Public Members

```

__I uint32_t W0
    MRC Domain Error Word0 Register, array offset: 0x480, array step: 0x10
__I uint32_t W1
    MRC Domain Error Word1 Register, array offset: 0x484, array step: 0x10
__O uint32_t W3
    MRC Domain Error Word3 Register, array offset: 0x48C, array step: 0x10
union __unnamed95__

```

### Public Members

```

struct _TRDC_DomainAssignment_Type MDA_DFMT0[8]
struct _TRDC_DomainAssignment_Type MDA_DFMT1[8]
struct MDA_DFMT0

```

### Public Members

\_\_IO uint32\_t MDA\_W\_DFMT0 [8]

DAC Master Domain Assignment Register, array offset: 0x800, array step: index\*0x20, index2\*0x4

struct MDA\_DFMT1

### Public Members

\_\_IO uint32\_t MDA\_W\_DFMT1 [1]

DAC Master Domain Assignment Register, array offset: 0x800, array step: index\*0x20, index2\*0x4

## 2.61 Trdc\_soc

enum \_trdc\_master

Enumeration for TRDC master mapping.

Defines the enumeration for TRDC master resource collections.

*Values:*

enumerator kTRDC1\_MasterReserved  
Reserved

enumerator kTRDC1\_MasterCM33  
CM33

enumerator kTRDC1\_MasterEDMA1  
EDMA1

enumerator kTRDC1\_MasterMTR\_FBX  
MTR FBX

enumerator kTRDC1\_MasterMTR  
MTR

enumerator kTRDC2\_MasterReserved0  
Reserved0

enumerator kTRDC2\_MasterReserved1  
Reserved1

enumerator kTRDC2\_MasterDAP  
DAP AHB\_AP\_SYS

enumerator kTRDC2\_MasterCoreSight  
CoreSight

enumerator kTRDC2\_MasterEDMA2  
EDMA2

enumerator kTRDC3\_MasterUSDHC1  
uSDHC1

enumerator kTRDC3\_MasterUSDHC2  
uSDHC2

```
enumerator kTRDC3_MasterTestPort
    Test port
enumerator kTRDC3_MasterUSDHC3
    USDHC3
enumerator kTRDC3_MasterENET0
    ENET0
enumerator kTRDC3_MasterENET1
    ENET1
enumerator kTRDC3_MasterENETQos
    ENET Qos
enumerator kTRDC3_MasterCA55Read
    CA55 read channel
enumerator kTRDC3_MasterCA55Write
    CA55 write channel
enumerator kTRDC3_MasterNPUm0
    NPU m0
enumerator kTRDC3_MasterNPUm1
    NPU m1
enumerator kTRDC_MediaMix_MasterISI_M
    ISI M
enumerator kTRDC_MediaMix_MasterISI_U_V
    ISI U&V
enumerator kTRDC_MediaMix_MasterPXP
    PXP
enumerator kTRDC_MediaMix_MasterLCDIF
    Lcdif
enumerator kTRDC_HSIOMix_MasterUSB1
    USB1
enumerator kTRDC_HSIOMix_MasterUSB2
    USB2
typedef void TRDC_Type
    TRDC typedef.
typedef enum _trdc_master trdc_master_t
    Enumeration for TRDC master mapping.
    Defines the enumeration for TRDC master resource collections.
FSL_TRDC_SOC_DRIVER_VERSION
    Driver version 2.0.0.
TRDC_DACFG_NCM_MASK
TRDC_MBC_MEM_GLBCFG_NBLKS_MASK
TRDC_MBC_MEM_GLBCFG_SIZE_LOG2_MASK
TRDC_MBC_MEM_GLBCFG_SIZE_LOG2_SHIFT
```

TRDC\_MBC\_NSE\_BLK\_CLR\_ALL\_DID\_SEL0\_SHIFT  
TRDC\_MDA\_W\_DFMT0\_LK1\_MASK  
TRDC\_MDA\_W\_DFMT0\_VLD\_MASK  
TRDC\_MDA\_W\_DFMT1\_LK1\_MASK  
TRDC\_MDA\_W\_DFMT1\_VLD\_MASK  
TRDC\_MRC\_DOM0\_RGD\_W\_MRACSEL\_MASK  
TRDC\_MRC\_DOM0\_RGD\_W\_NSE\_MASK  
TRDC\_MRC\_DOM0\_RGD\_W\_VLD\_MASK  
TRDC\_MRC\_DOM0\_RGD\_W\_END\_ADDR\_MASK  
TRDC\_MRC\_DOM0\_RGD\_W\_STRT\_ADDR\_MASK  
TRDC\_MRC\_GLBCFG\_NRGNS\_MASK  
TRDC\_MRC\_GLBCFG\_NRGNS\_SHIFT  
TRDC\_TRDC\_CR\_GVLDB\_MASK  
TRDC\_TRDC\_CR\_GVLDM\_MASK  
TRDC\_TRDC\_CR\_GVLDR\_MASK  
TRDC\_TRDC\_FLW\_CTL\_LK\_MASK  
TRDC\_TRDC\_FLW\_CTL\_V\_MASK  
TRDC\_TRDC\_HWCFG0\_NDID\_MASK  
TRDC\_TRDC\_HWCFG0\_NDID\_SHIFT  
TRDC\_TRDC\_HWCFG0\_NMBC\_MASK  
TRDC\_TRDC\_HWCFG0\_NMBC\_SHIFT  
TRDC\_TRDC\_HWCFG0\_NMRC\_MASK  
TRDC\_TRDC\_HWCFG0\_NMRC\_SHIFT  
TRDC\_TRDC\_HWCFG0\_NMSTR\_MASK  
TRDC\_TRDC\_HWCFG0\_NMSTR\_SHIFT  
TRDC\_TRDC\_HWCFG1\_DID\_MASK  
TRDC\_TRDC\_HWCFG1\_DID\_SHIFT  
TRDC\_TRDC\_IDAU\_CR\_VLD\_MASK  
TRDC\_W1\_EATR\_MASK  
TRDC\_W1\_EATR\_SHIFT  
TRDC\_W1\_EDID\_MASK  
TRDC\_W1\_EPORT\_MASK  
TRDC\_W1\_EPORT\_SHIFT

TRDC\_W1\_ERW\_MASK  
TRDC\_W1\_ERW\_SHIFT  
TRDC\_W1\_EST\_MASK  
TRDC\_W1\_EST\_SHIFT  
TRDC\_MBC\_NSE\_BLK\_CLR\_ALL\_MEMSEL  
TRDC\_MRC\_DOM0\_RGD\_W\_MRACSEL  
TRDC\_MRC\_DOM0\_RGD\_W\_NSE  
TRDC\_MRC\_DOM0\_RGD\_W\_VLD  
TRDC\_TRDC\_FDID\_FDID  
TRDC\_TRDC\_FLW\_CTL\_LK  
TRDC\_TRDC\_FLW\_CTL\_V  
TRDC\_W3\_RECR  
TRDC\_BASE\_PTRS  
    TRDC base table.  
TRDC\_GENERAL\_OFFSET  
    TRDC base address convert macro.  
TRDC\_FLW\_OFFSET  
TRDC\_DOMAIN\_ERROR\_OFFSET  
TRDC\_DOMAIN\_ASSIGNMENT\_OFFSET  
TRDC\_MBC\_OFFSET(**x**)  
TRDC\_MBC\_ARRAY\_STEP  
TRDC\_MRC\_OFFSET(**x**)  
TRDC\_MRC\_ARRAY\_STEP

## 2.62 TSTMR: Timestamp Timer Driver

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

Init TSTMR.

This function initializes the TSTMR module.

### Parameters

- base – TSTMR peripheral base address.

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

Deinit TSTMR.

This function deinitializes the TSTMR module.

### Parameters

- base – TSTMR peripheral base address.

FSL\_TSTMR\_DRIVER\_VERSION

Version 2.1.0

`static inline uint64_t TSTMR_ReadTimeStamp(TSTMR_Type *base)`

Reads the time stamp.

This function reads the low and high registers and returns the 56-bit free running counter value. This can be read by software at any time to determine the software ticks. TSTMR registers can be read with 32-bit accesses only. The TSTMR LOW read should occur first, followed by the TSTMR HIGH read.

**Parameters**

- base – TSTMR peripheral base address.

**Returns**

The 56-bit time stamp value.

`void TSTMR_DelayUs(TSTMR_Type *base, uint64_t delayInUs)`

Delays for a specified number of microseconds.

This function repeatedly reads the timestamp register and waits for the user-specified delay value.

**Parameters**

- base – TSTMR peripheral base address.
- delayInUs – Delay value in microseconds.

## 2.63 WDOG32: 32-bit Watchdog Timer

`void WDOG32_GetDefaultConfig(wdog32_config_t *config)`

Initializes the WDOG32 configuration structure.

This function initializes the WDOG32 configuration structure to default values. The default values are:

```
wdog32Config->enableWdog32 = true;
wdog32Config->clockSource = kWDog32_ClockSource1;
wdog32Config->prescaler = kWDog32_ClockPrescalerDivide1;
wdog32Config->workMode.enableWait = true;
wdog32Config->workMode.enableStop = false;
wdog32Config->workMode.enableDebug = false;
wdog32Config->testMode = kWDog32_TestModeDisabled;
wdog32Config->enableUpdate = true;
wdog32Config->enableInterrupt = false;
wdog32Config->enableWindowMode = false;
wdog32Config->windowValue = 0U;
wdog32Config->timeoutValue = 0xFFFFU;
```

**See also:**`wdog32_config_t`**Parameters**

- config – Pointer to the WDOG32 configuration structure.



*status\_t* WDOG32\_Init(WDOG\_Type \*base, const *wdog32\_config\_t* \*config)

Initializes the WDOG32 module.

This function initializes the WDOG32. To reconfigure the WDOG32 without forcing a reset first, enableUpdate must be set to true in the configuration.

Example:

```
wdog32_config_t config;
WDOG32_GetDefaultConfig(&config);
config.timeoutValue = 0x7ffU;
config.enableUpdate = true;
WDOG32_Init(wdog_base,&config);
```

---

**Note:** If there is errata ERR010536 (FSL\_FEATURE\_WDOG\_HAS\_ERRATA\_010536 defined as 1), then after calling this function, user need delay at least 4 LPO clock cycles before accessing other WDOG32 registers.

---

#### Parameters

- base – WDOG32 peripheral base address.
- config – The configuration of the WDOG32.

#### Return values

- kStatus\_Success – The initialization was successful
- kStatus\_Timeout – The initialization timed out

*status\_t* WDOG32\_Deinit(WDOG\_Type \*base)

De-initializes the WDOG32 module.

This function shuts down the WDOG32. Ensure that the WDOG\_CS.UPDATE is 1, which means that the register update is enabled.

#### Parameters

- base – WDOG32 peripheral base address.

#### Return values

- kStatus\_Success – The de-initialization was successful
- kStatus\_Timeout – The de-initialization timed out

*status\_t* WDOG32\_Unlock(WDOG\_Type \*base)

Unlocks the WDOG32 register written.

This function unlocks the WDOG32 register written.

Before starting the unlock sequence and following the configuration, disable the global interrupts. Otherwise, an interrupt could effectively invalidate the unlock sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

#### Parameters

- base – WDOG32 peripheral base address

#### Return values

- kStatus\_Success – The unlock sequence was successful
- kStatus\_Timeout – The unlock sequence timed out

`void WDOG32_Enable(WDOG_Type *base)`

Enables the WDOG32 module.

This function writes a value into the WDOG\_CS register to enable the WDOG32. The WDOG\_CS register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32\_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

#### Parameters

- base – WDOG32 peripheral base address.

`void WDOG32_Disable(WDOG_Type *base)`

Disables the WDOG32 module.

This function writes a value into the WDOG\_CS register to disable the WDOG32. The WDOG\_CS register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32\_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

#### Parameters

- base – WDOG32 peripheral base address

`void WDOG32_EnableInterrupts(WDOG_Type *base, uint32_t mask)`

Enables the WDOG32 interrupt.

This function writes a value into the WDOG\_CS register to enable the WDOG32 interrupt. The WDOG\_CS register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32\_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

#### Parameters

- base – WDOG32 peripheral base address.
- mask – The interrupts to enable. The parameter can be a combination of the following source if defined:
  - kWDOG32\_InterruptEnable

`void WDOG32_DisableInterrupts(WDOG_Type *base, uint32_t mask)`

Disables the WDOG32 interrupt.

This function writes a value into the WDOG\_CS register to disable the WDOG32 interrupt. The WDOG\_CS register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32\_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

#### Parameters

- base – WDOG32 peripheral base address.
- mask – The interrupts to disabled. The parameter can be a combination of the following source if defined:
  - kWDOG32\_InterruptEnable

`static inline uint32_t WDOG32_GetStatusFlags(WDOG_Type *base)`

Gets the WDOG32 all status flags.

This function gets all status flags.

Example to get the running flag:

```
uint32_t status;
status = WDOG32_GetStatusFlags(wdog_base) & kWDOG32_RunningFlag;
```

**See also:**

`_wdog32_status_flags_t`

- true: related status flag has been set.
- false: related status flag is not set.

**Parameters**

- base – WDOG32 peripheral base address

**Returns**

State of the status flag: asserted (true) or not-asserted (false).

```
void WDOG32_ClearStatusFlags(WDOG_Type *base, uint32_t mask)
```

Clears the WDOG32 flag.

This function clears the WDOG32 status flag.

Example to clear an interrupt flag:

```
WDOG32_ClearStatusFlags(wdog_base, kWDOG32_InterruptFlag);
```

**Parameters**

- base – WDOG32 peripheral base address.
- mask – The status flags to clear. The parameter can be any combination of the following values:
  - kWDOG32\_InterruptFlag

```
void WDOG32_SetTimeoutValue(WDOG_Type *base, uint16_t timeoutCount)
```

Sets the WDOG32 timeout value.

This function writes a timeout value into the WDOG\_TOVAL register. The WDOG\_TOVAL register is a write-once register. To ensure the reconfiguration fits the timing of WCT, unlock function will be called inline.

**Parameters**

- base – WDOG32 peripheral base address
- timeoutCount – WDOG32 timeout value, count of WDOG32 clock ticks.

```
void WDOG32_SetWindowValue(WDOG_Type *base, uint16_t windowValue)
```

Sets the WDOG32 window value.

This function writes a window value into the WDOG\_WIN register. The WDOG\_WIN register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32\_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

**Parameters**

- base – WDOG32 peripheral base address.
- windowValue – WDOG32 window value.

```
static inline void WDOG32_Refresh(WDOG_Type *base)
```

Refreshes the WDOG32 timer.

This function feeds the WDOG32. This function should be called before the Watchdog timer is in timeout. Otherwise, a reset is asserted.

**Parameters**

- base – WDOG32 peripheral base address

static inline uint16\_t WDOG32\_GetCounterValue(WDOG\_Type \*base)

Gets the WDOG32 counter value.

This function gets the WDOG32 counter value.

**Parameters**

- base – WDOG32 peripheral base address.

**Returns**

Current WDOG32 counter value.

WDOG\_FIRST\_WORD\_OF\_UNLOCK

First word of unlock sequence

WDOG\_SECOND\_WORD\_OF\_UNLOCK

Second word of unlock sequence

WDOG\_FIRST\_WORD\_OF\_REFRESH

First word of refresh sequence

WDOG\_SECOND\_WORD\_OF\_REFRESH

Second word of refresh sequence

FSL\_WDOG32\_DRIVER\_VERSION

WDOG32 driver version.

enum \_\_wdog32\_clock\_source

Max loops to wait for WDOG32 unlock sequence complete.

This is the maximum number of loops to wait for the wdog32 unlock sequence to complete. If set to 0, it will wait indefinitely until the unlock sequence is complete.

Max loops to wait for WDOG32 reconfiguration complete.

This is the maximum number of loops to wait for the wdog32 reconfiguration to complete. If set to 0, it will wait indefinitely until the reconfiguration is complete.

Describes WDOG32 clock source.

*Values:*

enumerator kWDG32\_ClockSource0

Clock source 0

enumerator kWDG32\_ClockSource1

Clock source 1

enumerator kWDG32\_ClockSource2

Clock source 2

enumerator kWDG32\_ClockSource3

Clock source 3

enum \_\_wdog32\_clock\_prescaler

Describes the selection of the clock prescaler.

*Values:*

enumerator kWDG32\_ClockPrescalerDivide1

Divided by 1

enumerator kWDOG32\_ClockPrescalerDivide256

Divided by 256

enum \_\_wdog32\_test\_mode

Describes WDOG32 test mode.

*Values:*

enumerator kWDOG32\_TestModeDisabled

Test Mode disabled

enumerator kWDOG32\_UserModeEnabled

User Mode enabled

enumerator kWDOG32\_LowByteTest

Test Mode enabled, only low byte is used

enumerator kWDOG32\_HighByteTest

Test Mode enabled, only high byte is used

enum \_\_wdog32\_interrupt\_enable\_t

WDOG32 interrupt configuration structure.

This structure contains the settings for all of the WDOG32 interrupt configurations.

*Values:*

enumerator kWDOG32\_InterruptEnable

Interrupt is generated before forcing a reset

enum \_\_wdog32\_status\_flags\_t

WDOG32 status flags.

This structure contains the WDOG32 status flags for use in the WDOG32 functions.

*Values:*

enumerator kWDOG32\_RunningFlag

Running flag, set when WDOG32 is enabled

enumerator kWDOG32\_InterruptFlag

Interrupt flag, set when interrupt occurs

typedef enum \_\_wdog32\_clock\_source wdog32\_clock\_source\_t

Max loops to wait for WDOG32 unlock sequence complete.

This is the maximum number of loops to wait for the wdog32 unlock sequence to complete. If set to 0, it will wait indefinitely until the unlock sequence is complete.

Max loops to wait for WDOG32 reconfiguration complete.

This is the maximum number of loops to wait for the wdog32 reconfiguration to complete. If set to 0, it will wait indefinitely until the reconfiguration is complete.

Describes WDOG32 clock source.

typedef enum \_\_wdog32\_clock\_prescaler wdog32\_clock\_prescaler\_t

Describes the selection of the clock prescaler.

typedef struct \_\_wdog32\_work\_mode wdog32\_work\_mode\_t

Defines WDOG32 work mode.

typedef enum \_\_wdog32\_test\_mode wdog32\_test\_mode\_t

Describes WDOG32 test mode.

```
typedef struct _wdog32_config wdog32_config_t
```

Describes WDOG32 configuration structure.

```
struct _wdog32_work_mode
```

*#include <fsl\_wdog32.h>* Defines WDOG32 work mode.

### Public Members

**bool** enableWait  
Enables or disables WDOG32 in wait mode

**bool** enableStop  
Enables or disables WDOG32 in stop mode

**bool** enableDebug  
Enables or disables WDOG32 in debug mode

```
struct _wdog32_config
```

*#include <fsl\_wdog32.h>* Describes WDOG32 configuration structure.

### Public Members

**bool** enableWdog32  
Enables or disables WDOG32

*wdog32\_clock\_source\_t* clockSource  
Clock source select

*wdog32\_clock\_prescaler\_t* prescaler  
Clock prescaler value

*wdog32\_work\_mode\_t* workMode  
Configures WDOG32 work mode in debug stop and wait mode

*wdog32\_test\_mode\_t* testMode  
Configures WDOG32 test mode

**bool** enableUpdate  
Update write-once register enable

**bool** enableInterrupt  
Enables or disables WDOG32 interrupt

**bool** enableWindowMode  
Enables or disables WDOG32 window mode

**uint16\_t** windowValue  
Window value

**uint16\_t** timeoutValue  
Timeout value

## 2.64 CACHE: CACHE Memory Controller

uint32\_t XCACHE\_GetInstanceByAddr(uint32\_t address)

brief Returns an instance number given physical memory address.

param address The physical memory address.

**Returns**

XCACHE instance number starting from 0.

void XCACHE\_EnableCache(XCACHE\_Type \*base)

Enables the cache.

**Parameters**

- base – XCACHE peripheral base address.

void XCACHE\_DisableCache(XCACHE\_Type \*base)

Disables the cache.

**Parameters**

- base – XCACHE peripheral base address.

void XCACHE\_InvalidateCache(XCACHE\_Type \*base)

Invalidates the cache.

**Parameters**

- base – XCACHE peripheral base address.

void XCACHE\_InvalidateCacheByRange(uint32\_t address, uint32\_t size\_byte)

Invalidates cache by range.

---

**Note:** Address and size should be aligned to “XCACHE\_LINESIZE\_BYTE”. The startAddr here will be forced to align to XCACHE\_LINESIZE\_BYTE if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

---

**Parameters**

- address – The physical address of cache.
- size\_byte – size of the memory to be invalidated, should be larger than 0, better to align with cache line size.

void XCACHE\_CleanCache(XCACHE\_Type \*base)

Cleans the cache.

**Parameters**

- base – XCACHE peripheral base address.

void XCACHE\_CleanCacheByRange(uint32\_t address, uint32\_t size\_byte)

Cleans cache by range.

---

**Note:** Address and size should be aligned to “XCACHE\_LINESIZE\_BYTE”. The startAddr here will be forced to align to XCACHE\_LINESIZE\_BYTE if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

---

**Parameters**

- address – The physical address of cache.

- `size_byte` – size of the memory to be cleaned, should be larger than 0, better to align with cache line size.

`void XCACHE_CleanInvalidateCache(XCACHE_Type *base)`

Cleans and invalidates the cache.

#### Parameters

- `base` – XCACHE peripheral base address.

`void XCACHE_CleanInvalidateCacheByRange(uint32_t address, uint32_t size_byte)`

Cleans and invalidate cache by range.

---

**Note:** Address and size should be aligned to “XCACHE\_LINESIZE\_BYTE”. The `startAddr` here will be forced to align to XCACHE\_LINESIZE\_BYTE if `startAddr` is not aligned. For the `size_byte`, application should make sure the alignment or make sure the right operation order if the `size_byte` is not aligned.

---

#### Parameters

- `address` – The physical address of cache.
- `size_byte` – size of the memory to be Cleaned and Invalidated, should be larger than 0, better to align with cache line size.

`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 XCACHE\_LINESIZE\_BYTE due to the cache operation unit FSL\_FEATURE\_XCACHE\_LINESIZE\_BYTE. The `startAddr` here will be forced to align to the cache line size if `startAddr` is not aligned. For the `size_byte`, application should make sure the alignment or make sure the right operation order if the `size_byte` is not aligned.

---

#### Parameters

- `address` – The physical address.
- `size_byte` – size of the memory to be invalidated, should be larger than 0, better to align with cache line size.

`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 XCACHE\_LINESIZE\_BYTE due to the cache operation unit FSL\_FEATURE\_XCACHE\_LINESIZE\_BYTE. The `startAddr` here will be forced to align to the cache line size if `startAddr` is not aligned. For the `size_byte`, application should make sure the alignment or make sure the right operation order if the `size_byte` is not aligned.

---

#### Parameters

- `address` – The physical address.
- `size_byte` – size of the memory to be invalidated, should be larger than 0, better to align with cache line size.



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 XCACHE\_LINESIZE\_BYTE due to the cache operation unit FSL\_FEATURE\_XCACHE\_LINESIZE\_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

---

#### Parameters

- address – The physical address.
- size\_byte – size of the memory to be cleaned, should be larger than 0, better to align with cache line size.

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 XCACHE\_LINESIZE\_BYTE due to the cache operation unit FSL\_FEATURE\_XCACHE\_LINESIZE\_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

---

#### Parameters

- address – The physical address.
- size\_byte – size of the memory to be Cleaned and Invalidated, should be larger than 0, better to align with cache line size.

FSL\_CACHE\_DRIVER\_VERSION  
cache driver version.

XCACHE\_LINESIZE\_BYTE  
cache line size.



# Chapter 3

## Middleware

### 3.1 Connectivity

#### 3.1.1 lwIP

**This is the NXP fork of the [lwIP networking stack](#).**

- For details about changes and additions made by NXP, see CHANGELOG.
- For details about the NXP porting layer, see *The NXP lwIP Port*.
- For usage and API of lwIP, use official documentation at <http://www.nongnu.org/lwip/>.

#### The NXP lwIP Port

Below is description of possible settings of the port layer and an overview of a few helper functions.

The best place for redefinition of any mentioned macro is `lwipopts.h`.

The declaration of every mentioned function is in `ethernetif.h`. Please check the doxygen comments of those functions before.

**Link state** Physical link state (up/down) and its speed and duplex must be read out from PHY over MDIO bus. Especially link information is useful for lwIP stack so it can for example send DHCP discovery immediately when a link becomes up.

To simplify this port layer offers a function `ethernetif_probe_link()` which reads those data from PHY and forwards them into lwIP stack.

In almost all examples this function is called every `ETH_LINK_POLLING_INTERVAL_MS` (1500ms) by a function `probe_link_cyclic()`.

By setting `ETH_LINK_POLLING_INTERVAL_MS` to 0 polling will be disabled. On FreeRTOS, `probe_link_cyclic()` will be then called on an interrupt generated by PHY. GPIO port and pin for the interrupt line must be set in the `ethernetifConfig` struct passed to `ethernetif_init()`. On bare metal interrupts are not supported right now.

**Rx task** To improve the reaction time of the app, reception of packets is done in a dedicated task. The rx task stack size can be set by `ETH_RX_TASK_STACK_SIZE` macro, its priority by `ETH_RX_TASK_PRIO`.

If you want to save memory you can set reception to be done in an interrupt by setting `ETH_DO_RX_IN_SEPARATE_TASK` macro to 0.

**Disabling Rx interrupt when out of buffers** If `ETH_DISABLE_RX_INT_WHEN_OUT_OF_BUFFERS` is set to 1, then when the port gets out of Rx buffers, Rx enet interrupt will be disabled for a particular controller. Everytime Rx buffer is freed, Rx interrupt will be enabled.

This prevents your app from never getting out of Rx interrupt when the network is flooded with traffic.

`ETH_DISABLE_RX_INT_WHEN_OUT_OF_BUFFERS` is by default turned on, on FreeRTOS and off on bare metal.

**Limit the number of packets read out from the driver at once on bare metal.** You may define macro `ETH_MAX_RX_PKTS_AT_ONCE` to limit the number of received packets read out from the driver at once.

In case of heavy Rx traffic, lowering this number improves the realtime behaviour of an app. Increasing improves Rx throughput.

Setting it to value  $< 1$  or not defining means “no limit”.

**Helper functions** If your application needs to wait for the link to become up you can use one of the following functions:

- `ethernetif_wait_linkup()` - Blocks until the link on the passed netif is not up.
- `ethernetif_wait_linkup_array()` - Blocks until the link on at least one netif from the passed list of netifs becomes up.

If your app needs to wait for the IPv4 address on a particular netif to become different than “ANY” address (255.255.255.255) function `ethernetif_wait_ipv4_valid()` does this.

## 3.2 Motor Control

### 3.2.1 FreeMASTER

*Communication Driver User Guide*

#### Introduction

**What is FreeMASTER?** FreeMASTER is a PC-based application developed by NXP for NXP customers. It is a versatile tool usable as a real-time monitor, visualization tool, and a graphical control panel of embedded applications based on the NXP processing units.

This document describes the embedded-side software driver which implements an interface between the application and the host PC. The interface covers the following communication:

- **Serial** UART communication either over plain RS232 interface or more typically over a USB-to-Serial either external or built in a debugger probe.
- **USB** direct connection to target microcontroller
- **CAN bus**
- **TCP/IP network** wired or WiFi
- **Segger J-Link RTT**

- JTAG debug port communication
- ...and all of the above also using a **Zephyr** generic drivers.

The driver also supports so-called “packet-driven BDM” interface which enables a protocol-based communication over a debugging port. The BDM stands for Background Debugging Module and its physical implementation is different on each platform. Some platforms leverage a semi-standard JTAG interface, other platforms provide a custom implementation called BDM. Regardless of the name, this debugging interface enables non-intrusive access to the memory space while the target CPU is running. For basic memory read and write operations, there is no communication driver required on the target when communicating with the host PC. Use this driver to get more advanced FreeMASTER protocol features over the BDM interface. The driver must be configured for the packet-driven BDM mode, in which the host PC uses the debugging interface to write serial command frames directly to the target memory buffer. The same method is then used to read response frames from that memory buffer.

Similar to “packet-driven BDM”, the FreeMASTER also supports a communication over [J-Link RTT](<https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/>) interface defined by SEGGER Microcontroller GmbH for ARM CortexM-based microcontrollers. This method also uses JTAG physical interface and enables high-speed real time communication to run over the same channel as used for application debugging.

**Driver version 3** This document describes version 3 of the FreeMASTER Communication Driver. This version features the implementation of the new Serial Protocol, which significantly extends the features and security of its predecessor. The new protocol internal number is v4 and its specification is available in the documentation accompanying the driver code.

Driver V3 is deployed to modern 32-bit MCU platforms first, so the portfolio of supported platforms is smaller than for the previous V2 versions. It is recommended to keep using the V2 driver for legacy platforms, such as S08, S12, ColdFire, or Power Architecture. Reach out to [FreeMASTER community](#) or to the local NXP representative with requests for more information or to port the V3 driver to legacy MCU devices.

Thanks to a layered approach, the new driver simplifies the porting of the driver to new UART, CAN or networking communication interfaces significantly. Users are encouraged to port the driver to more NXP MCU platforms and contribute the code back to NXP for integration into future releases. Existing code and low-level driver layers may be used as an example when porting to new targets.

**Note:** Using the FreeMASTER tool and FreeMASTER Communication Driver is only allowed in systems based on NXP microcontroller or microprocessor unit. Use with non-NXP MCU platforms is **not permitted** by the license terms.

**Target platforms** The driver implementation uses the following abstraction mechanisms which simplify driver porting and supporting new communication modules:

- **General CPU Platform** (see source code in the `src/platforms` directory). The code in this layer is only specific to native data type sizes and CPU architectures (for example; alignment-aware memory copy routines). This driver version brings two generic implementations of 32-bit platforms supporting both little-endian and big-endian architectures. There are also implementations customized for the 56F800E family of digital signal controllers and S12Z MCUs. **Zephyr** is treated as a specific CPU platform as it brings unified user configuration (Kconfig) and generic hardware device drivers. With Zephyr, the transport layer and low-level communication layers described below are configured automatically using Kconfig and Device Tree technologies.
- **Transport Communication Layer** - The Serial, CAN, Networking, PD-BDM, and other methods of transport logic are implemented as a driver layer called FMSTR\_TRANSPORT with a uniform API. A support of the Network transport also extends single-client modes of operation which are native for Serial, USB and CAN by a concept of multiple client sessions.

- **Low-level Communication Driver** - Each type of transport further defines a low-level API used to access the physical communication module. For example, the Serial transport defines a character-oriented API implemented by different serial communication modules like UART, LPUART, USART, and also USB-CDC. Similarly, the CAN transport defines a message-oriented API implemented by the FlexCAN or MCAN modules. Moreover, there are multiple different implementations for the same kind of communication peripherals. The difference between the implementation is in the way the low-level hardware registers are accessed. The *mcuxsdk* folder contains implementations which use MCUXpresso SDK drivers. These drivers should be used in applications based on the NXP MCUXpresso SDK. The “ampsdk” drivers target automotive-specific MCUs and their respective SDKs. The “dreg” implementations use a plain C-language access to hardware register addresses which makes it a universal and the most portable solution. In this case, users are encouraged to add more drivers for other communication modules or other respective SDKs and contribute the code back to NXP for integration.

The low-level drivers defined for the Networking transport enable datagram-oriented UDP and stream TCP communication. This implementation is demonstrated using the lwIP software stack but shall be portable to other TCP/IP stacks. It may sound surprisingly, but also the Segger J-Link RTT communication driver is linked to the Networking transport (RTT is stream oriented communication handled similarly to TCP).

**Replacing existing drivers** For all supported platforms, the driver described in this document replaces the V2 implementation and also older driver implementations that were available separately for individual platforms (PC Master SCI drivers).

**Clocks, pins, and peripheral initialization** The FreeMASTER communication driver is only responsible for runtime processing of the communication and must be integrated with an user application code to function properly. The user application code is responsible for general initialization of clock sources, pin multiplexers, and peripheral registers related to the communication speed. Such initialization should be done before calling the `FMSTR_Init` function.

It is recommended to develop the user application using one of the Software Development Kits (SDKs) available from third parties or directly from NXP, such as MCUXpresso SDK, MCUXpresso IDE, and related tools. This approach simplifies the general configuration process significantly.

**MCUXpresso SDK** The MCUXpresso SDK is a software package provided by NXP which contains the device initialization code, linker files, and software drivers with example applications for the NXP family of MCUs. The MCUXpresso Config Tools may be used to generate the clock-setup and pin-multiplexer setup code suitable for the selected processor.

The MCUXpresso SDK also contains this FreeMASTER communication driver as a “middleware” component which may be downloaded along with the example applications from <https://mcuxpresso.nxp.com/en/welcome>.

**MCUXpresso SDK on GitHub** The FreeMASTER communication driver is also released as one of the middleware components of the MCUXpresso SDK on the GitHub. This release enables direct integration of the FreeMASTER source code Git repository into a target applications including Zephyr applications.

Related links:

- [The official FreeMASTER middleware repository.](#)
- [Online version of this document](#)

**FreeMASTER in Zephyr** The FreeMASTER middleware repository can be used with MCUXpresso SDK as well as a Zephyr module. Zephyr-specific samples which include examples of Kconfig and Device Tree configurations for Serial, USB and Network communications are available in separate repository. West manifest in this sample repository fetches the full Zephyr package including the FreeMASTER middleware repository used as a Zephyr module.

## Example applications

**MCUX SDK Example applications** There are several example applications available for each supported MCU platform.

- **fmstr\_uart** demonstrates a plain serial transmission, typically connecting to a computer's physical or virtual COM port. The typical transmission speed is 115200 bps.
- **fmstr\_can** demonstrates CAN bus communication. This requires a suitable CAN interface connected to the computer and interconnected with the target MCU using a properly terminated CAN bus. The typical transmission speed is 500 kbps. A FreeMASTER-over-CAN communication plug-in must be used.
- **fmstr\_usb\_cdc** uses an on-chip USB controller to implement a CDC communication class. It is connected directly to a computer's USB port and creates a virtual COM port device. The typical transmission speed is above 1 Mbps.
- **fmstr\_net** demonstrates the Network communication over UDP or TCP protocol. Existing examples use lwIP stack to implement the communication, but in general, it shall be possible to use any other TCP/IP stack to achieve the same functionality.
- **fmstr\_wifi** is the fmstr\_net application modified to use a WiFi network interface instead of a wired Ethernet connection.
- **fmstr\_rtt** demonstrates the communication over SEGGER J-Link RTT interface. Both fmstr\_net and fmstr\_rtt examples require the FreeMASTER TCP/UDP communication plug-in to be used on the PC host side.
- **fmstr\_eonce** uses the real-time data unit on the JTAG EOnCE module of the 56F800E family to implement pseudo-serial communication over the JTAG port. The typical transmission speed is around 10 kbps. This communication requires FreeMASTER JTAG/EOnCE communication plug-in.
- **fmstr\_pdbdm** uses JTAG or BDM debugging interface to access the target RAM directly while the CPU is running. Note that such approach can be used with any MCU application, even without any special driver code. The computer reads from and writes into the RAM directly without CPU intervention. The Packet-Driven BDM (PD-BDM) communication uses the same memory access to exchange command and response frames. With PD-BDM, the FreeMASTER tool is able to go beyond basic memory read/write operations and accesses also advanced features like Recorder, TSA, or Pipes. The typical transmission speed is around 10 kbps. A PD-BDM communication plug-in must be used in FreeMASTER and configured properly for the selected debugging interface. Note that this communication cannot be used while a debugging interface is used by a debugger session.
- **fmstr\_any** is a special example application which demonstrates how the NXP MCUXpresso Config Tools can be used to configure pins, clocks, peripherals, interrupts, and even the FreeMASTER "middleware" driver features in a graphical and user friendly way. The user can switch between the Serial, CAN, and other ways of communication and generate the required initialization code automatically.

**Zephyr sample applications** Zephyr sample applications demonstrate Kconfig and Device Tree configuration which configure the FreeMASTER middleware module for a selected communication option (Serial, CAN, Network or RTT).

Refer to *readme.md* files in each sample directory for description of configuration options required to implement FreeMASTER connectivity.

## Description

This section shows how to add the FreeMASTER Communication Driver into application and how to configure the connection to the FreeMASTER visualization tool.

**Features** The FreeMASTER driver implements the FreeMASTER protocol V4 and provides the following features which may be accessed using the FreeMASTER visualization tool:

- Read/write access to any memory location on the target.
- Optional password protection of the read, read/write, and read/write/flash access levels.
- Atomic bit manipulation on the target memory (bit-wise write access).
- Optimal size-aligned access to memory which is also suitable to access the peripheral register space.
- Oscilloscope access—real-time access to target variables. The sample rate may be limited by the communication speed.
- Recorder— access to the fast transient recorder running on the board as a part of the FreeMASTER driver. The sample rate is only limited by the MCU CPU speed. The length of the data recorded depends on the amount of available memory.
- Multiple instances of Oscilloscopes and Recorders without the limitation of maximum number of variables.
- Application commands—high-level message delivery from the PC to the application.
- TSA tables—describing the data types, variables, files, or hyperlinks exported by the target application. The TSA newly supports also non-memory mapped resources like external EEPROM or SD Card files.
- Pipes—enabling the buffered stream-oriented data exchange for a general-purpose terminal-like communication, diagnostic data streaming, or other data exchange.

The FreeMASTER driver features:

- Full FreeMASTER protocol V4 implementation with a new V4 style of CRC used.
- Layered approach supporting Serial, CAN, Network, PD-BDM, and other transports.
- Layered low-level Serial transport driver architecture enabling to select UART, LPUART, USART, and other physical implementations of serial interfaces, including USB-CDC.
- Layered low-level CAN transport driver architecture enabling to select FlexCAN, msCAN, MCAN, and other physical implementations of the CAN interface.
- Layered low-level Networking transport enabling to select TCP, UDP or J-Link RTT communication.
- TSA support to write-protect memory regions or individual variables and to deny the access to the unsafe memory.
- The pipe callback handlers are invoked whenever new data is available for reading from the pipe.
- Two Serial Single-Wire modes of operation are enabled. The “external” mode has the RX and TX shorted on-board. The “true” single-wire mode interconnects internally when the MCU or UART modules support it.



The following sections briefly describe all FreeMASTER features implemented by the driver. See the PC-based FreeMASTER User Manual for more details on how to use the features to monitor, tune, or control an embedded application.

**Board Detection** The FreeMASTER protocol V4 defines the standard set of configuration values which the host PC tool reads to identify the target and to access other target resources properly. The configuration includes the following parameters:

- Version of the driver and the version of the protocol implemented.
- MTU as the Maximum size of the Transmission Unit (for example; communication buffer size).
- Application name, description, and version strings.
- Application build date and time as a string.
- Target processor byte ordering (little/big endian).
- Protection level that requires password authentication.
- Number of the Recorder and Oscilloscope instances.
- RAM Base Address for optimized memory access commands.

**Memory Read** This basic feature enables the host PC to read any data memory location by specifying the address and size of the required memory area. The device response frame must be shorter than the MTU to fit into the outgoing communication buffer. To read a device memory of any size, the host uses the information retrieved during the Board Detection and splits the large-block request to multiple partial requests.

The driver uses size-aligned operations to read the target memory (for example; uses proper read-word instruction when an address is aligned to 4 bytes).

**Memory Write** Similarly to the Memory Read operation, the Memory Write feature enables to write to any RAM memory location on the target device. A single write command frame must be shorter than the MTU to fit into the target communication buffer. Larger requests must be split into smaller ones.

The driver uses size-aligned operations to write to the target memory (for example; uses proper write-word instruction when an address is aligned to 4 bytes).

**Masked Memory Write** To implement the write access to a single bit or a group of bits of target variables, the Masked Memory Write feature is available in the FreeMASTER protocol and it is supported by the driver using the Read-Modify-Write approach.

Be careful when writing to bit fields of volatile variables that are also modified in an application interrupt. The interrupt may be serviced in the middle of a read-modify-write operation and it may cause data corruption.

**Oscilloscope** The protocol and driver enables any number of variables to be read at once with a single request from the host. This feature is called Oscilloscope and the FreeMASTER tool uses it to display a real-time graph of variable values.

The driver can be configured to support any number of Oscilloscope instances and enable simultaneously running graphs to be displayed on the host computer screen.

**Recorder** The protocol enables the host to select target variables whose values are then periodically recorded into a dedicated on-board memory buffer. After such data sampling stops (either on a host request or by evaluating a threshold-crossing condition), the data buffer is downloaded to the host and displayed as a graph. The data sampling rate is not limited by the speed of the communication line, so it enables displaying the variable transitions in a very high resolution.

The driver can be configured to support multiple Recorder instances and enable multiple recorder graphs to be displayed on the host screen. Having multiple recorders also enables setting the recording point differently for each instance. For example; one instance may be recording data in a general timer interrupt while another instance may record at a specific control algorithm time in the PWM interrupt.

**TSA** With the TSA feature, data types and variables can be described directly in the application source code. Such information is later provided to the FreeMASTER tool which may use it instead of reading symbol data from the application ELF executable file.

The information is encoded as so-called TSA tables which become direct part of the application code. The TSA tables contain descriptors of variables that shall be visible to the host tool. The descriptors can describe the memory areas by specifying the address and size of the memory block or more conveniently using the C variable names directly. Different set of TSA descriptors can be used to encode information about the structure types, unions, enumerations, or arrays.

The driver also supports special types of TSA table entries to describe user resources like external EEPROM and SD Card files, memory-mapped files, virtual directories, web URL hyperlinks, and constant enumerations.

**TSA Safety** When the TSA is enabled in the application, the TSA Safety can be enabled and validate the memory accesses directly by the embedded-side driver. When the TSA Safety is turned on, any memory request received from the host is validated and accepted only if it belongs to a TSA-described object. The TSA entries can be declared as Read-Write or Read-Only so that the driver can actively deny the write access to the Read-Only objects.

**Application commands** The Application Commands are high-level messages that can be delivered from the PC Host to the embedded application for further processing. The embedded application can either poll the status, or be called back when a new Application Command arrives to be processed. After the embedded application acknowledges that the command is handled, the host receives the Result Code and reads the other return data from memory. Both the Application Commands and the Result Codes are specific to a given application and it is user's responsibility to define them. The FreeMASTER protocol and the FreeMASTER driver only implement the delivery channel and a set of API calls to enable the Application Command processing in general.

**Pipes** The Pipes enable buffered and stream-oriented data exchange between the PC Host and the target application. Any pipe can be written to and read from at both ends (either on the PC or the MCU). The data transmission is acknowledged using the special FreeMASTER protocol commands. It is guaranteed that the data bytes are delivered from the writer to the reader in a proper order and without losses.

**Serial single-wire operation** The MCU Serial Communication Driver natively supports normal dual-wire operation. Because the protocol is half-duplex only, the driver can also operate in two single-wire modes:

- “External” single-wire operation where the Receiver and Transmitter pins are shorted on the board. This mode is supported by default in the MCU driver because the Receiver and Transmitter units are enabled or disabled whenever needed. It is also easy to extend this operation for the RS485 communication.

- “True” single-wire mode which uses only a single pin and the direction switching is made by the UART module. This mode of operation must be enabled by defining the FMSTR\_SERIAL\_SINGLEWIRE configuration option.

**Multi-session support** With networking interface it is possible for multiple clients to access the target MCU simultaneously. Reading and writing of target memory is processed atomically so there is no risk of data corruption. The state-full resources such as Recorders or Oscilloscopes are locked to a client session upon first use and access is denied to other clients until lock is released..

## Zephyr-specific

**Dedicated communication task** FreeMASTER communication may run isolated in a dedicated task. The task automates the FMSTR\_Init and FMSTR\_Poll calls together with periodic activities enabling the FreeMASTER UI to fetch information about tasks and CPU utilization. The task can be started automatically or manually, and it must be assigned a priority to be able to react on interrupts and other communication events. Refer to Zephyr FreeMASTER sample applications which all use this communication task.

**Zephyr shell and logging over FreeMASTER pipe** FreeMASTER implements a shell backend which may use FreeMASTER pipe as a I/O terminal and logging output. Refer to Zephyr FreeMASTER sample applications which all use this feature.

**Automatic TSA tables** TSA tables can be declared as “automatic” in Zephyr which make them automatically registered in the table list. This may be very useful when there are many TSA tables or when the tables are defined in different (often unrelated) libraries linked together. In this case user does not need to build a list of all tables manually.

**Driver files** The driver source files can be found in a top-level src folder, further divided into the sub-folders:

- **src/platforms** platform-specific folder—one folder exists for each supported processor platform (for example; 32-bit Little Endian platform). Each such folder contains a platform header file with data types and a code which implements the potentially platform-specific operations, such as aligned memory access.
- **src/common** folder—contains the common driver source files shared by the driver for all supported platforms. All the .c files must be added to the project, compiled, and linked together with the application.
  - *freemaster.h* - master driver header file, which declares the common data types, macros, and prototypes of the FreeMASTER driver API functions.
  - *freemaster\_cfg.h.example* - this file can serve as an example of the FreeMASTER driver configuration file. Save this file into a project source code folder and rename it to *freemaster\_cfg.h*. The FreeMASTER driver code includes this file to get the project-specific configuration options and to optimize the compilation of the driver.
  - *freemaster\_defcfg.h* - defines the default values for each FreeMASTER configuration option if the option is not set in the *freemaster\_cfg.h* file.
  - *freemaster\_protocol.h* - defines the FreeMASTER protocol constants used internally by the driver.
  - *freemaster\_protocol.c* - implements the FreeMASTER protocol decoder and handles the basic Get Configuration Value, Memory Read, and Memory Write commands.

- *freemaster\_rec.c* - handles the Recorder-specific commands and implements the Recorder sampling and triggering routines. When the Recorder is disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
- *freemaster\_scope.c* - handles the Oscilloscope-specific commands. If the Oscilloscope is disabled by the FreeMASTER driver configuration file, this file compiles as void.
- *freemaster\_pipes.c* - implements the Pipes functionality when the Pipes feature is enabled.
- *freemaster\_appcmd.c* - handles the communication commands used to deliver and execute the Application Commands within the context of the embedded application. When the Application Commands are disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
- *freemaster\_tsa.c* - handles the commands specific to the TSA feature. This feature enables the FreeMASTER host tool to obtain the TSA memory descriptors declared in the embedded application. If the TSA is disabled by the FreeMASTER driver configuration file, this file compiles as void.
- *freemaster\_tsa.h* - contains the declaration of the macros used to define the TSA memory descriptors. This file is indirectly included into the user application code (via *freemaster.h*).
- *freemaster\_sha.c* - implements the SHA-1 hash code used in the password authentication algorithm.
- *freemaster\_private.h* - contains the declarations of functions and data types used internally in the driver. It also contains the C pre-processor statements to perform the compile-time verification of the user configuration provided in the *freemaster\_cfg.h* file.
- *freemaster\_serial.c* - implements the serial protocol logic including the CRC, FIFO queuing, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a character-oriented API exported by the specific low-level driver.
- *freemaster\_serial.h* - defines the low-level character-oriented Serial API.
- *freemaster\_can.c* - implements the CAN protocol logic including the CAN message preparation, signalling using the first data byte in the CAN frame, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a message-oriented API exported by the specific low-level driver.
- *freemaster\_can.h* - defines the low-level message-oriented CAN API.
- *freemaster\_net.c* - implements the Network protocol transport logic including multiple session management code.
- *freemaster\_net.h* - definitions related to the Network transport.
- *freemaster\_pdbdm.c* - implements the packet-driven BDM communication buffer and other communication-related operations.
- *freemaster\_utils.c* - aligned memory copy routines, circular buffer management and other utility functions
- *freemaster\_utils.h* - definitions related to utility code.
- **src/drivers/[sdk]/serial** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster\_serial\_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the UART, LPUART, USART, and other kinds of Serial communication modules.

- ***src/drivers/[sdk]/can*** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster\_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the FlexCAN, msCAN, MCAN, and other kinds of CAN communication modules.
- ***src/drivers/[sdk]/network*** - contains low-level code adapting the FreeMASTER Network transport to an underlying TCP/IP or RTT stack.
  - *freemaster\_net\_lwip\_tcp.c* and *\_udp.c* - default networking implementation of TCP and UDP transports using lwIP stack.
  - *freemaster\_net\_segger\_rtt.c* - implementation of network transport using Segger J-Link RTT interface

**Driver configuration** The driver is configured using a single header file (*freemaster\_cfg.h*). Create this file and save it together with other project source files before compiling the driver code. All FreeMASTER driver source files include the *freemaster\_cfg.h* file and use the macros defined here for the conditional and parameterized compilation. The C compiler must locate the configuration file when compiling the driver files. Typically, it can be achieved by putting this file into a folder where the other project-specific included files are stored.

As a starting point to create the configuration file, get the *freemaster\_cfg.h.example* file, rename it to *freemaster\_cfg.h*, and save it into the project area.

**Note:** It is NOT recommended to leave the *freemaster\_cfg.h* file in the FreeMASTER driver source code folder. The configuration file must be placed at a project-specific location, so that it does not affect the other applications that use the same driver.

**Configurable items** This section describes the configuration options which can be defined in *freemaster\_cfg.h*.

### Interrupt modes

```
#define FMSTR_LONG_INTR    [0|1]
#define FMSTR_SHORT_INTR  [0|1]
#define FMSTR_POLL_DRIVEN [0|1]
```

**Value Type** boolean (0 or 1)

**Description** Exactly one of the three macros must be defined to non-zero. The others must be defined to zero or left undefined. The non-zero-defined constant selects the interrupt mode of the driver. See [Driver interrupt modes](#).

- FMSTR\_LONG\_INTR — long interrupt mode
- FMSTR\_SHORT\_INTR — short interrupt mode
- FMSTR\_POLL\_DRIVEN — poll-driven mode

**Note:** Some options may not be supported by all communication interfaces. For example, the FMSTR\_SHORT\_INTR option is not supported by the USB\_CDC interface.

### Protocol transport

```
#define FMSTR_TRANSPORT [identifier]
```

**Value Type** Driver identifiers are structure instance names defined in FreeMASTER source code. Specify one of existing instances to make use of the protocol transport.

**Description** Use one of the pre-defined constants, as implemented by the FreeMASTER code. The current driver supports the following transports:

- **FMSTR\_SERIAL** - serial communication protocol
- **FMSTR\_CAN** - using CAN communication
- **FMSTR\_PDBDM** - using packet-driven BDM communication
- **FMSTR\_NET** - network communication using TCP or UDP protocol

**Serial transport** This section describes configuration parameters used when serial transport is used:

```
#define FMSTR_TRANSPORT FMSTR_SERIAL
```

**FMSTR\_SERIAL\_DRV** Select what low-level driver interface will be used when implementing the Serial communication.

```
#define FMSTR_SERIAL_DRV [identifier]
```

**Value Type** Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing serial driver instances.

**Description** When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/serial* implementation):

- **FMSTR\_SERIAL\_MCUX\_UART** - UART driver
- **FMSTR\_SERIAL\_MCUX\_LPUART** - LPUART driver
- **FMSTR\_SERIAL\_MCUX\_USART** - USART driver
- **FMSTR\_SERIAL\_MCUX\_MINIUSART** - miniUSART driver
- **FMSTR\_SERIAL\_MCUX\_QSCI** - DSC QSCI driver
- **FMSTR\_SERIAL\_MCUX\_USB** - USB/CDC class driver (also see code in the */support/mcuxsdk\_usb* folder)
- **FMSTR\_SERIAL\_56F800E\_EONCE** - DSC JTAG EOnCE driver

Other SDKs or BSPs may define custom low-level driver interface structure which may be used as **FMSTR\_SERIAL\_DRV**. For example:

- **FMSTR\_SERIAL\_DREG\_UART** - demonstrates the low-level interface implemented without the MCUXpresso SDK and using direct access to peripheral registers.

### **FMSTR\_SERIAL\_BASE**

```
#define FMSTR_SERIAL_BASE [address|symbol]
```

**Value Type** Optional address value (numeric or symbolic)

**Description** Specify the base address of the UART, LPUART, USART, or other serial peripheral module to be used for the communication. This value is not defined by default. User application should call `FMSTR_SetSerialBaseAddress()` to select the peripheral module.

#### FMSTR\_COMM\_BUFFER\_SIZE

```
#define FMSTR_COMM_BUFFER_SIZE [number]
```

**Value Type** 0 or a value in range 32...255

**Description** Specify the size of the communication buffer to be allocated by the driver. Default value, which suits all driver features, is used when this option is defined as 0.

#### FMSTR\_COMM\_QUEUE\_SIZE

```
#define FMSTR_COMM_QUEUE_SIZE [number]
```

**Value Type** Value in range 0...255

**Description** Specify the size of the FIFO receiver queue used to quickly receive and store characters in the `FMSTR_SHORT_INTR` interrupt mode. The default value is 32 B.

#### FMSTR\_SERIAL\_SINGLEWIRE

```
#define FMSTR_SERIAL_SINGLEWIRE [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Set to non-zero to enable the “True” single-wire mode which uses a single MCU pin to communicate. The low-level driver enables the pin direction switching when the MCU peripheral supports it.

**CAN Bus transport** This section describes configuration parameters used when CAN transport is used:

```
#define FMSTR_TRANSPORT FMSTR_CAN
```

**FMSTR\_CAN\_DRV** Select what low-level driver interface will be used when implementing the CAN communication.

```
#define FMSTR_CAN_DRV [identifier]
```

**Value Type** Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing CAN driver instances.



**Description** When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/can implementation*):

- **FMSTR\_CAN\_MCUX\_FLEXCAN** - FlexCAN driver
- **FMSTR\_CAN\_MCUX\_MCAN** - MCAN driver
- **FMSTR\_CAN\_MCUX\_MSCAN** - msCAN driver
- **FMSTR\_CAN\_MCUX\_DSCFLEXCAN** - DSC FlexCAN driver
- **FMSTR\_CAN\_MCUX\_DSCMSCAN** - DSC msCAN driver

Other SDKs or BSPs may define the custom low-level driver interface structure which may be used as FMSTR\_CAN\_DRV.

### FMSTR\_CAN\_BASE

```
#define FMSTR_CAN_BASE [address|symbol]
```

**Value Type** Optional address value (numeric or symbolic)

**Description** Specify the base address of the FlexCAN, msCAN, or other CAN peripheral module to be used for the communication. This value is not defined by default. User application should call FMSTR\_SetCanBaseAddress() to select the peripheral module.

### FMSTR\_CAN\_CMDID

```
#define FMSTR_CAN_CMDID [number]
```

**Value Type** CAN identifier (11-bit or 29-bit number)

**Description** CAN message identifier used for FreeMASTER commands (direction from PC Host tool to target application). When declaring 29-bit identifier, combine the numeric value with FMSTR\_CAN\_EXTID bit. Default value is 0x7AA.

### FMSTR\_CAN\_RSPID

```
#define FMSTR_CAN_RSPID [number]
```

**Value Type** CAN identifier (11-bit or 29-bit number)

**Description** CAN message identifier used for responding messages (direction from target application to PC Host tool). When declaring 29-bit identifier, combine the numeric value with FMSTR\_CAN\_EXTID bit. Note that both *CMDID* and *RSPID* values may be the same. Default value is 0x7AA.

### FMSTR\_FLEXCAN\_TXMB

```
#define FMSTR_FLEXCAN_TXMB [number]
```

**Value Type** Number in range of 0..N where N is number of CAN message-buffers supported by HW module.



**Description** Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame transmission. Default value is 0.

#### FMSTR\_FLEXCAN\_RXMB

```
#define FMSTR_FLEXCAN_RXMB [number]
```

**Value Type** Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

**Description** Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame reception. Note that the FreeMASTER driver may also operate with a common message buffer used by both TX and RX directions. Default value is 1.

**Network transport** This section describes configuration parameters used when Network transport is used:

```
#define FMSTR_TRANSPORT FMSTR_NET
```

**FMSTR\_NET\_DRV** Select network interface implementation.

```
#define FMSTR_NET_DRV [identifier]
```

**Value Type** Identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing NET driver instances.

**Description** When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/network implementation*):

- **FMSTR\_NET\_LWIP\_TCP** - TCP communication using lwIP stack
- **FMSTR\_NET\_LWIP\_UDP** - UDP communication using lwIP stack
- **FMSTR\_NET\_SEGGER\_RTT** - Communication using SEGGER J-Link RTT interface

Other SDKs or BSPs may define the custom networking interface which may be used as FMSTR\_CAN\_DRV.

Add another row below:

#### FMSTR\_NET\_PORT

```
#define FMSTR_NET_PORT [number]
```

**Value Type** TCP or UDP port number (short integer)

**Description** Specifies the server port number used by TCP or UDP protocols.

#### FMSTR\_NET\_BLOCKING\_TIMEOUT

```
#define FMSTR_NET_BLOCKING_TIMEOUT [number]
```

**Value Type** Timeout as number of milliseconds

**Description** This value specifies a timeout in milliseconds for which the network socket operations may block the execution inside *FMSTR\_Poll*. This may be set high (e.g. 250) when a dedicated RTOS task is used to handle FreeMASTER protocol polling. Set to a lower value when the polling task is also responsible for other operations. Set to 0 to attempt to use non-blocking socket operations.

#### FMSTR\_NET\_AUTODISCOVERY

```
#define FMSTR_NET_AUTODISCOVERY [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** This option enables the FreeMASTER driver to use a separate UDP socket to broadcast auto-discovery messages to network. This helps the FreeMASTER tool to discover the target device address, port and protocol options.

#### Debugging options

#### FMSTR\_DISABLE

```
#define FMSTR_DISABLE [0|1]
```

**Value Type** boolean (0 or 1)

**Description** Define as non-zero to disable all FreeMASTER features, exclude the driver code from build, and compile all its API functions empty. This may be useful to remove FreeMASTER without modifying any application source code. Default value is 0 (false).

#### FMSTR\_DEBUG\_TX

```
#define FMSTR_DEBUG_TX [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to enable the driver to periodically transmit test frames out on the selected communication interface (SCI or CAN). With the debug transmission enabled, it is simpler to detect problems in the baudrate or other communication configuration settings.

The test frames are transmitted until the first valid command frame is received from the PC Host tool. The test frame is a valid error status frame, as defined by the protocol format. On the serial line, the test frame consists of three printable characters (+©W) which are easy to capture using the serial terminal tools.

This feature requires the FMSTR\_Poll() function to be called periodically. Default value is 0 (false).

**FMSTR\_APPLICATION\_STR**

```
#define FMSTR_APPLICATION_STR
```

**Value Type** String.

**Description** Name of the application visible in FreeMASTER host application.

**Memory access****FMSTR\_USE\_READMEM**

```
#define FMSTR_USE_READMEM [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the Memory Read command and enable FreeMASTER to have read access to memory and variables. The access can be further restricted by using a TSA feature.  
Default value is 1 (true).

**FMSTR\_USE\_WRITEMEM**

```
#define FMSTR_USE_WRITEMEM [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the Memory Write command.  
The default value is 1 (true).

**Oscilloscope options****FMSTR\_USE\_SCOPE**

```
#define FMSTR_USE_SCOPE [number]
```

**Value Type** Integer number.

**Description** Number of Oscilloscope instances to be supported. Set to 0 to disable the Oscilloscope feature.  
Default value is 0.

**FMSTR\_MAX\_SCOPE\_VARS**

```
#define FMSTR_MAX_SCOPE_VARS [number]
```

**Value Type** Integer number larger than 2.

**Description** Number of variables to be supported by each Oscilloscope instance.  
Default value is 8.

### Recorder options

#### FMSTR\_USE\_RECORDER

```
#define FMSTR_USE_RECORDER [number]
```

**Value Type** Integer number.

**Description** Number of Recorder instances to be supported. Set to 0 to disable the Recorder feature.  
Default value is 0.

#### FMSTR\_REC\_BUFF\_SIZE

```
#define FMSTR_REC_BUFF_SIZE [number]
```

**Value Type** Integer number larger than 2.

**Description** Defines the size of the memory buffer used by the Recorder instance #0.  
Default: not defined, user shall call 'FMSTR\_RecorderCreate()' API function to specify this parameter in run time.

#### FMSTR\_REC\_TIMEBASE

```
#define FMSTR_REC_TIMEBASE [time specification]
```

**Value Type** Number (nanoseconds time).

**Description** Defines the base sampling rate in nanoseconds (sampling speed) Recorder instance #0.

Use one of the following macros:

- FMSTR\_REC\_BASE\_SECONDS(x)
- FMSTR\_REC\_BASE\_MILLISEC(x)
- FMSTR\_REC\_BASE\_MICROSEC(x)
- FMSTR\_REC\_BASE\_NANOSEC(x)

Default: not defined, user shall call 'FMSTR\_RecorderCreate()' API function to specify this parameter in run time.

#### FMSTR\_REC\_FLOAT\_TRIG

```
#define FMSTR_REC_FLOAT_TRIG [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the floating-point triggering. Be aware that floating-point triggering may grow the code size by linking the floating-point standard library. Default value is 0 (false).

### Application Commands options

#### FMSTR\_USE\_APPCMD

```
#define FMSTR_USE_APPCMD [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Define as non-zero to implement the Application Commands feature. Default value is 0 (false).

#### FMSTR\_APPCMD\_BUFF\_SIZE

```
#define FMSTR_APPCMD_BUFF_SIZE [size]
```

**Value Type** Numeric buffer size in range 1..255

**Description** The size of the Application Command data buffer allocated by the driver. The buffer stores the (optional) parameters of the Application Command which waits to be processed.

#### FMSTR\_MAX\_APPCMD\_CALLS

```
#define FMSTR_MAX_APPCMD_CALLS [number]
```

**Value Type** Number in range 0..255

**Description** The number of different Application Commands that can be assigned a callback handler function using FMSTR\_RegisterAppCmdCall(). Default value is 0.

### TSA options

#### FMSTR\_USE\_TSA

```
#define FMSTR_USE_TSA [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable the FreeMASTER TSA feature to be used. With this option enabled, the TSA tables defined in the applications are made available to the FreeMASTER host tool. Default value is 0 (false).

#### FMSTR\_USE\_TSA\_SAFETY

```
#define FMSTR_USE_TSA_SAFETY [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable the memory access validation in the FreeMASTER driver. With this option, the host tool is not able to access the memory which is not described by at least one TSA descriptor. Also a write access is denied for objects defined as read-only in TSA tables. Default value is 0 (false).

#### FMSTR\_USE\_TSA\_INROM

```
#define FMSTR_USE_TSA_INROM [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Declare all TSA descriptors as *const*, which enables the linker to put the data into the flash memory. The actual result depends on linker settings or the linker commands used in the project. Default value is 0 (false).

#### FMSTR\_USE\_TSA\_DYNAMIC

```
#define FMSTR_USE_TSA_DYNAMIC [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable runtime-defined TSA entries to be added to the TSA table by the FMSTR\_SetUpTsaBuff() and FMSTR\_TsaAddVar() functions. Default value is 0 (false).

### Pipes options

#### FMSTR\_USE\_PIPES

```
#define FMSTR_USE_PIPES [0|1]
```

**Value Type** Boolean 0 or 1.

**Description** Enable the FreeMASTER Pipes feature to be used. Default value is 0 (false).

**FMSTR\_MAX\_PIPES\_COUNT**

```
#define FMSTR_MAX_PIPES_COUNT [number]
```

**Value Type** Number in range 1..63.

**Description** The number of simultaneous pipe connections to support. The default value is 1.

**Driver interrupt modes** To implement the communication, the FreeMASTER driver handles the Serial or CAN module's receive and transmit requests. Use the *freemaster\_cfg.h* configuration file to select whether the driver processes the communication automatically in the interrupt service routine handler or if it only polls the status of the module (typically during the application idle time).

This section describes each of the interrupt mode in more details.

**Completely Interrupt-Driven operation** Activated using:

```
#define FMSTR_LONG_INTR 1
```

In this mode, both the communication and the FreeMASTER protocol decoding is done in the *FMSTR\_SerialIsr*, *FMSTR\_CanIsr*, or other interrupt service routine. Because the protocol execution may be a lengthy task (especially with the TSA-Safety enabled) it is recommended to use this mode only if the interrupt prioritization scheme is possible in the application and the FreeMASTER interrupt is assigned to a lower (the lowest) priority.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR\_SerialIsr* or *FMSTR\_CanIsr* functions from that handler.

**Mixed Interrupt and Polling Modes** Activated using:

```
#define FMSTR_SHORT_INTR 1
```

In this mode, the communication processing time is split between the interrupt routine and the main application loop or task. The raw communication is handled by the *FMSTR\_SerialIsr*, *FMSTR\_CanIsr*, or other interrupt service routine, while the protocol decoding and execution is handled by the *FMSTR\_Poll* routine. Call *FMSTR\_Poll* during the idle time in the application main loop.

The interrupt processing in this mode is relatively fast and deterministic. Upon a serial-receive event, the received character is only placed into a FIFO-like queue and it is not further processed. Upon a CAN receive event, the received frame is stored into a receive buffer. When transmitting, the characters are fetched from the prepared transmit buffer.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR\_SerialIsr* or *FMSTR\_CanIsr* functions from that handler.

When the serial interface is used as the serial communication interface, ensure that the *FMSTR\_Poll* function is called at least once per *N* character time periods. *N* is the length of the FreeMASTER FIFO queue (*FMSTR\_COMM\_QUEUE\_SIZE*) and the character time is the time needed to transmit or receive a single byte over the SCI line.

## Completely Poll-driven

```
#define FMSTR_POLL_DRIVEN 1
```

In this mode, both the communication and the FreeMASTER protocol decoding are done in the *FMSTR\_Poll* routine. No interrupts are needed and the *FMSTR\_SerialIsr*, *FMSTR\_CanIsr*, and similar handlers compile to an empty code.

When using this mode, ensure that the *FMSTR\_Poll* function is called by the application at least once per the serial “character time” which is the time needed to transmit or receive a single character.

In the latter two modes (*FMSTR\_SHORT\_INTR* and *FMSTR\_POLL\_DRIVEN*), the protocol handling takes place in the *FMSTR\_Poll* routine. An application interrupt can occur in the middle of the Read Memory or Write Memory commands’ execution and corrupt the variable being accessed by the FreeMASTER driver. In these two modes, some issues or glitches may occur when using FreeMASTER to visualize or monitor volatile variables modified in interrupt servicing code.

The same issue may appear even in the full interrupt mode (*FMSTR\_LONG\_INTR*), if volatile variables are modified in the interrupt code with a priority higher than the priority of the communication interrupt.

**Data types** Simple portability was one of the main requirements when writing the FreeMASTER driver. This is why the driver code uses the privately-declared data types and the vast majority of the platform-dependent code is separated in the platform-dependent source files. The data types used in the driver API are all defined in the platform-specific header file.

To prevent name conflicts with the symbols used in the application, all data types, macros, and functions have the *FMSTR\_* prefix. The only global variables used in the driver are the transport and low-level API structures exported from the driver-implementation layer to upper layers. Other than that, all private variables are declared as static and named using the *fmstr\_* prefix.

**Communication interface initialization** The FreeMASTER driver does not perform neither the initialization nor the configuration of the peripheral module that it uses to communicate. It is the application startup code responsibility to configure the communication module before the FreeMASTER driver is initialized by the *FMSTR\_Init* call.

When the Serial communication module is used as the FreeMASTER communication interface, configure the UART receive and transmit pins, the serial communication baud rate, parity (no-parity), the character length (eight bits), and the number of stop bits (one) before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see *Driver interrupt modes*), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected serial peripheral module. Call the *FMSTR\_SerialIsr* function from the application handler.

When a CAN module is used as the FreeMASTER communication interface, configure the CAN receive and transmit pins and the CAN module bit rate before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see *Driver interrupt modes*), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected CAN peripheral module. Call the *FMSTR\_CanIsr* function from the application handler.

**Note:** It is not necessary to enable or unmask the serial nor the CAN interrupts before initializing the FreeMASTER driver. The driver enables or disables the interrupts and communication lines, as required during runtime.

**FreeMASTER Recorder calls** When using the FreeMASTER Recorder in the application (*FMSTR\_USE\_RECORDER* > 0), call the *FMSTR\_RecorderCreate* function early after *FMSTR\_Init* to set



up each recorder instance to be used in the application. Then call the `FMSTR_Recorder` function periodically in the code where the data recording should occur. A typical place to call the Recorder routine is at the timer or PWM interrupts, but it can be anywhere else. The example applications provided together with the driver code call the `FMSTR_Recorder` in the main application loop.

In applications where `FMSTR_Recorder` is called periodically with a constant period, specify the period in the Recorder configuration structure before calling `FMSTR_RecorderCreate`. This setting enables the PC Host FreeMASTER tool to display the X-axis of the Recorder graph properly scaled for the time domain.

**Driver usage** Start using or evaluating FreeMASTER by opening some of the example applications available in the driver setup package.

Follow these steps to enable the basic FreeMASTER connectivity in the application:

- Make sure that all `*c` files of the FreeMASTER driver from the `src/common/platforms/[your_platform]` folder are a part of the project. See [Driver files](#) for more details.
- Configure the FreeMASTER driver by creating or editing the `freemaster_cfg.h` file and by saving it into the application project directory. See [Driver configuration](#) for more details.
- Include the `freemaster.h` file into any application source file that makes the FreeMASTER API calls.
- Initialize the Serial or CAN modules. Set the baud rate, parity, and other parameters of the communication. Do not enable the communication interrupts in the interrupt mask registers.
- For the `FMSTR_LONG_INTR` and `FMSTR_SHORT_INTR` modes, install the application-specific interrupt routine and call the `FMSTR_SerialIsr` or `FMSTR_CanIsr` functions from this handler.
- Call the `FMSTR_Init` function early on in the application initialization code.
- Call the `FMSTR_RecorderCreate` functions for each Recorder instance to enable the Recorder feature.
- In the main application loop, call the `FMSTR_Poll` API function periodically when the application is idle.
- For the `FMSTR_SHORT_INTR` and `FMSTR_LONG_INTR` modes, enable the interrupts globally so that the interrupts can be handled by the CPU.

**Communication troubleshooting** The most common problem that causes communication issues is a wrong baud rate setting or a wrong pin multiplexer setting of the target MCU. When a communication between the PC Host running FreeMASTER and the target MCU cannot be established, try enabling the `FMSTR_DEBUG_TX` option in the `freemaster_cfg.h` file and call the `FMSTR_Poll` function periodically in the main application task loop.

With this feature enabled, the FreeMASTER driver periodically transmits a test frame through the Serial or CAN lines. Use a logic analyzer or an oscilloscope to monitor the signals at the communication pins of the CPU device to examine whether the bit rate and signal polarity are configured properly.

## Driver API

This section describes the driver Application Programmers' Interface (API) needed to initialize and use the FreeMASTER serial communication driver.

**Control API** There are three key functions to initialize and use the driver.

## FMSTR\_Init

### Prototype

```
FMSTR_BOOL FMSTR_Init(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_protocol.c*

**Description** This function initializes the internal variables of the FreeMASTER driver and enables the communication interface. This function does not change the configuration of the selected communication module. The hardware module must be initialized before the *FMSTR\_Init* function is called.

A call to this function must occur before calling any other FreeMASTER driver API functions.

## FMSTR\_Poll

### Prototype

```
void FMSTR_Poll(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_protocol.c*

**Description** In the poll-driven or short interrupt modes, this function handles the protocol decoding and execution (see *Driver interrupt modes*). In the poll-driven mode, this function also handles the communication interface with the PC. Typically, the *FMSTR\_Poll* function is called during the “idle” time in the main application task loop.

To prevent the receive data overflow (loss) on a serial interface, make sure that the *FMSTR\_Poll* function is called at least once per the time calculated as:

$N * T_{char}$

where:

- $N$  is equal to the length of the receive FIFO queue (configured by the *FMSTR\_COMM\_QUEUE\_SIZE* macro).  $N$  is 1 for the poll-driven mode.
- $T_{char}$  is the character time, which is the time needed to transmit or receive a single byte over the SCI line.

**Note:** In the long interrupt mode, this function typically compiles as an empty function and can still be called. It is worthwhile to call this function regardless of the interrupt mode used in the application. This approach enables a convenient switching between the different interrupt modes only by changing the configuration macros in the *freemaster\_cfg.h* file.

## FMSTR\_SerialIsr / FMSTR\_CanIsr

### Prototype

```
void FMSTR_SerialIsr(void);
void FMSTR_CanIsr(void);
```

- Declaration: *freemaster.h*
- Implementation: *hw-specific low-level driver C file*

**Description** This function contains the interrupt-processing code of the FreeMASTER driver. In long or short interrupt modes (see [Driver interrupt modes](#)), this function must be called from the application interrupt service routine registered for the communication interrupt vector. On platforms where the communication module uses multiple interrupt vectors, the application should register a handler for all vectors and call this function at each interrupt.

**Note:** In a poll-driven mode, this function is compiled as an empty function and does not have to be used.

## Recorder API

### FMSTR\_RecorderCreate

#### Prototype

```
FMSTR_BOOL FMSTR_RecorderCreate(FMSTR_INDEX recIndex, FMSTR_REC_BUFF* buffCfg);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_rec.c*

**Description** This function registers a recorder instance and enables it to be used by the PC Host tool. Call this function for all recorder instances from 0 to the maximum number defined by the FMSTR\_USE\_RECORDER configuration option (minus one). An exception to this requirement is the recorder of instance 0 which may be automatically configured by FMSTR\_Init when the *freemaster\_cfg.h* configuration file defines the *FMSTR\_REC\_BUFF\_SIZE* and *FMSTR\_REC\_TIMEBASE* options.

For more information, see [Configurable items](#).

### FMSTR\_Recorder

#### Prototype

```
void FMSTR_Recorder(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_rec.c*

**Description** This function takes a sample of the variables being recorded using the FreeMASTER Recorder instance *recIndex*. If the selected Recorder is not active when the *FMSTR\_Recorder* function is being called, the function returns immediately. When the Recorder is active, the values of the variables being recorded are copied into the recorder buffer and the trigger conditions are evaluated.

If a trigger condition is satisfied, the Recorder enters the post-trigger mode, where it counts down the follow-up samples (number of *FMSTR\_Recorder* function calls) and de-activates the Recorder when the required post-trigger samples are finished.

The *FMSTR\_Recorder* function is typically called in the timer or PWM interrupt service routines. This function can also be called in the application main loop (for testing purposes).

## FMSTR\_RecorderTrigger

### Prototype

```
void FMSTR_RecorderTrigger(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_rec.c*

**Description** This function forces the Recorder trigger condition to happen, which causes the Recorder to be automatically deactivated after the post-trigger samples are sampled. Use this function in the application code for programmatic control over the Recorder triggering. This can be useful when a more complex triggering conditions need to be used.

**Fast Recorder API** The Fast Recorder feature is not available in the FreeMASTER driver version 3. This feature was heavily dependent on the target platform and it was only available for the 56F8xxxx DSCs.

**TSA Tables** When the TSA is enabled in the FreeMASTER driver configuration file (by setting the *FMSTR\_USE\_TSA* macro to a non-zero value), it defines the so-called TSA tables in the application. This section describes the macros that must to be used to define the TSA tables.

There can be any number of TSA tables spread across the application source files. There must be always exactly one TSA Table List defined, which informs the FreeMASTER driver about the active TSA tables.

When there is at least one TSA table and one TSA Table List defined in the application, the TSA information automatically appears in the FreeMASTER symbols list. The symbols can then be used to create FreeMASTER variables for visualization or control.

**TSA table definition** The TSA table describes the static or global variables together with their address, size, type, and access-protection information. If the TSA-described variables are of a structure type, the TSA table may also describe this type and provide an access to the individual structure members of the variable.

The TSA table definition begins with the *FMSTR\_TSA\_TABLE\_BEGIN* macro with a *table\_id* identifying the table. The *table\_id* shall be a valid C-language symbol.

```
FMSTR_TSA_TABLE_BEGIN(table_id)
```

After this opening macro, the TSA descriptors are placed using these macros:

```
/* Adding variable descriptors */
FMSTR_TSA_RW_VAR(name, type) /* read/write variable entry */
FMSTR_TSA_RO_VAR(name, type) /* read-only variable entry */

/* Description of complex data types */
FMSTR_TSA_STRUCT(struct_name) /* structure or union type entry */
```

(continues on next page)

(continued from previous page)

```

FMSTR_TSA_MEMBER(struct_name, member_name, type) /* structure member entry */

/* Memory blocks */
FMSTR_TSA_RW_MEM(name, type, address, size) /* read/write memory block */
FMSTR_TSA_RO_MEM(name, type, address, size) /* read-only memory block */

```

The table is closed using the FMSTR\_TSA\_TABLE\_END macro:

```
FMSTR_TSA_TABLE_END()
```

**TSA descriptor parameters** The TSA descriptor macros accept these parameters:

- *name* — variable name. The variable must be defined before the TSA descriptor references it.
- *type* — variable or member type. Only one of the pre-defined type constants may be used (see below).
- *struct\_name* — structure type name. The type must be defined (typedef) before the TSA descriptor references it.
- *member\_name* — structure member name.

**Note:** The structure member descriptors (FMSTR\_TSA\_MEMBER) must immediately follow the parent structure descriptor (FMSTR\_TSA\_STRUCT) in the table.

**Note:** To write-protect the variables in the FreeMASTER driver (FMSTR\_TSA\_RO\_VAR), enable the TSA-Safety feature in the configuration file.

**TSA variable types** The table lists *type* identifiers which can be used in TSA descriptors:

| Constant                           | Description                                                    |
|------------------------------------|----------------------------------------------------------------|
| FMSTR_TSA_UINTn                    | Unsigned integer type of size <i>n</i> bits (n=8,16,32,64)     |
| FMSTR_TSA_SINTn                    | Signed integer type of size <i>n</i> bits (n=8,16,32,64)       |
| FMSTR_TSA_FRACn                    | Fractional number of size <i>n</i> bits (n=16,32,64).          |
| FMSTR_TSA_FRAC_Q( <i>m,n</i> )     | Signed fractional number in general Q form (m+n+1 total bits)  |
| FMSTR_TSA_FRAC_UQ( <i>m,n</i> )    | Unsigned fractional number in general UQ form (m+n total bits) |
| FMSTR_TSA_FLOAT                    | 4-byte standard IEEE floating-point type                       |
| FMSTR_TSA_DOUBLE                   | 8-byte standard IEEE floating-point type                       |
| FMSTR_TSA_POINTER                  | Generic pointer type defined (platform-specific 16 or 32 bit)  |
| FM-STR_TSA_USERTYPE( <i>name</i> ) | Structure or union type declared with FMSTR_TSA_STRUCT record  |

**TSA table list** There shall be exactly one TSA Table List in the application. The list contains one entry for each TSA table defined anywhere in the application.

The TSA Table List begins with the FMSTR\_TSA\_TABLE\_LIST\_BEGIN macro and continues with the TSA table entries for each table.

```

FMSTR_TSA_TABLE_LIST_BEGIN()

FMSTR_TSA_TABLE(table_id)
FMSTR_TSA_TABLE(table_id2)
FMSTR_TSA_TABLE(table_id3)
...

```

The list is closed with the FMSTR\_TSA\_TABLE\_LIST\_END macro:

```
FMSTR_TSA_TABLE_LIST_END()
```

**TSA Active Content entries** FreeMASTER v2.0 and higher supports TSA Active Content, enabling the TSA tables to describe the memory-mapped files, virtual directories, and URL hyperlinks. FreeMASTER can access such objects similarly to accessing the files and folders on the local hard drive.

With this set of TSA entries, the FreeMASTER pages can be embedded directly into the target MCU flash and accessed by FreeMASTER directly over the communication line. The HTML-coded pages rendered inside the FreeMASTER window can access the TSA Active Content resources using a special URL referencing the *fmstr:* protocol.

This example provides an overview of the supported TSA Active Content entries:

```
FMSTR_TSA_TABLE_BEGIN(files_and_links)

/* Directory entry applies to all subsequent MEMFILE entries */
FMSTR_TSA_DIRECTORY("/text_files") /* entering a new virtual directory */

/* The readme.txt file will be accessible at the fmstr://text_files/readme.txt URL */
FMSTR_TSA_MEMFILE("readme.txt", readme_txt, sizeof(readme_txt)) /* memory-mapped file */

/* Files can also be specified with a full path so the DIRECTORY entry does not apply */
FMSTR_TSA_MEMFILE("/index.htm", index, sizeof(index)) /* memory-mapped file */
FMSTR_TSA_MEMFILE("/prj/demo.pmp", demo_pmp, sizeof(demo_pmp)) /* memory-mapped file */

/* Hyperlinks can point to a local MEMFILE object or to the Internet */
FMSTR_TSA_HREF("Board's Built-in Welcome Page", "/index.htm")
FMSTR_TSA_HREF("FreeMASTER Home Page", "http://www.nxp.com/freemaster")

/* Project file links simplify opening the projects from any URLs */
FMSTR_TSA_PROJECT("Demonstration Project (embedded)", "/prj/demo.pmp")
FMSTR_TSA_PROJECT("Full Project (online)", "http://mycompany.com/prj/demo.pmp")

FMSTR_TSA_TABLE_END()
```

## TSA API

### FMSTR\_SetUpTsaBuff

#### Prototype

```
FMSTR_BOOL FMSTR_SetUpTsaBuff(FMSTR_ADDR buffAddr, FMSTR_SIZE buffSize);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_tsa.c*

#### Arguments

- *buffAddr* [in] - address of the memory buffer for the dynamic TSA table
- *buffSize* [in] - size of the memory buffer which determines the maximum number of TSA entries to be added in the runtime

**Description** This function must be used to assign the RAM memory buffer to the TSA subsystem when FMSTR\_USE\_TSA\_DYNAMIC is enabled. The memory buffer is then used to store the TSA entries added dynamically to the runtime TSA table using the FMSTR\_TsaAddVar function call. The runtime TSA table is processed by the FreeMASTER PC Host tool along with all static tables as soon as the communication port is open.

The size of the memory buffer determines the number of TSA entries that can be added dynamically. Depending on the MCU platform, one TSA entry takes either 8 or 16 bytes.

## FMSTR\_TsaAddVar

### Prototype

```
FMSTR_BOOL FMSTR_TsaAddVar(FMSTR_TSATBL_STRPTR tsaName, FMSTR_TSATBL_STRPTR ↵
↵ tsaType,
    FMSTR_TSATBL_VOIDPTR varAddr, FMSTR_SIZE32 varSize,
    FMSTR_SIZE flags);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_tsa.c*

### Arguments

- *tsaName* [in] - name of the object
- *tsaType* [in] - name of the object type
- *varAddr* [in] - address of the object
- *varSize* [in] - size of the object
- *flags* [in] - access flags; a combination of these values:
  - FMSTR\_TSA\_INFO\_RO\_VAR — read-only memory-mapped object (typically a variable)
  - FMSTR\_TSA\_INFO\_RW\_VAR — read/write memory-mapped object
  - FMSTR\_TSA\_INFO\_NON\_VAR — other entry, describing structure types, structure members, enumerations, and other types

**Description** This function can be called only when the dynamic TSA table is enabled by the FMSTR\_USE\_TSA\_DYNAMIC configuration option and when the FMSTR\_SetUpTsaBuff function call is made to assign the dynamic TSA table memory. This function adds an entry into the dynamic TSA table. It can be used to register a read-only or read/write memory object or describe an item of the user-defined type.

See [TSA table definition](#) for more details about the TSA table entries.

## Application Commands API

### FMSTR\_GetAppCmd

#### Prototype

```
FMSTR_APPCMD_CODE FMSTR_GetAppCmd(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*



**Description** This function can be used to detect if there is an Application Command waiting to be processed by the application. If no command is pending, this function returns the FMSTR\_APPCMDRESULT\_NOCMD constant. Otherwise, this function returns the code of the Application Command that must be processed. Use the FMSTR\_AppCmdAck call to acknowledge the Application Command after it is processed and to return the appropriate result code to the host.

The FMSTR\_GetAppCmd function does not report the commands for which a callback handler function exists. If the FMSTR\_GetAppCmd function is called when a callback-registered command is pending (and before it is actually processed by the callback function), this function returns FMSTR\_APPCMDRESULT\_NOCMD.

## FMSTR\_GetAppCmdData

### Prototype

```
FMSTR_APPCMD_PDATA FMSTR_GetAppCmdData(FMSTR_SIZE* dataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*

### Arguments

- *dataLen* [out] - pointer to the variable that receives the length of the data available in the buffer. It can be NULL when this information is not needed.

**Description** This function can be used to retrieve the Application Command data when the application determines that an Application Command is pending (see [FMSTR\\_GetAppCmd](#)).

There is just a single buffer to hold the Application Command data (the buffer length is FMSTR\_APPCMD\_BUFF\_SIZE bytes). If the data are to be used in the application after the command is processed by the FMSTR\_AppCmdAck call, copy the data out to a private buffer.

## FMSTR\_AppCmdAck

### Prototype

```
void FMSTR_AppCmdAck(FMSTR_APPCMD_RESULT resultCode);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*

### Arguments

- *resultCode* [in] - the result code which is to be returned to FreeMASTER

**Description** This function is used when the Application Command processing finishes in the application. The resultCode passed to this function is returned back to the host and the driver is re-initialized to expect the next Application Command.

After this function is called and before the next Application Command arrives, the return value of the FMSTR\_GetAppCmd function is FMSTR\_APPCMDRESULT\_NOCMD.

## FMSTR\_AppCmdSetResponseData



## Prototype

```
void FMSTR_AppCmdSetResponseData(FMSTR_ADDR resultDataAddr, FMSTR_SIZE resultDataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*

## Arguments

- *resultDataAddr* [in] - pointer to the data buffer that is to be copied to the Application Command data buffer
- *resultDataLen* [in] - length of the data to be copied. It must not exceed the FMSTR\_APPCMD\_BUFF\_SIZE value.

**Description** This function can be used before the Application Command processing finishes, when there are data to be returned back to the PC.

The response data buffer is copied into the Application Command data buffer, from where it is accessed when the host requires it. Do not use FMSTR\_GetAppCmdData and the data buffer after FMSTR\_AppCmdSetResponseData is called.

**Note:** The current version of FreeMASTER does not support the Application Command response data.

## FMSTR\_RegisterAppCmdCall

### Prototype

```
FMSTR_BOOL FMSTR_RegisterAppCmdCall(FMSTR_APPCMD_CODE appCmdCode, FMSTR_
↳PAPPCMDFUNC callbackFunc);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_appcmd.c*

## Arguments

- *appCmdCode* [in] - the Application Command code for which the callback is to be registered
- *callbackFunc* [in] - pointer to the callback function that is to be registered. Use NULL to unregister a callback registered previously with this Application Command.

**Return value** This function returns a non-zero value when the callback function was successfully registered or unregistered. It can return zero when trying to register a callback function for more than FMSTR\_MAX\_APPCMD\_CALLS different Application Commands.

**Description** This function can be used to register the given function as a callback handler for the Application Command. The Application Command is identified using single-byte code. The callback function is invoked automatically by the FreeMASTER driver when the protocol decoder obtains a request to get the application command result code.

The prototype of the callback function is

```
FMSTR_APPCMD_RESULT HandlerFunction(FMSTR_APPCMD_CODE nAppcmd,
FMSTR_APPCMD_PDATA pData, FMSTR_SIZE nDataLen);
```

Where:

- *nAppcmd* -Application Command code
- *pData* —points to the Application Command data received (if any)
- *nDataLen* —information about the Application Command data length

The return value of the callback function is used as the Application Command Result Code and returned to FreeMASTER.

**Note:** The FMSTR\_MAX\_APPCMD\_CALLS configuration macro defines how many different Application Commands may be handled by a callback function. When FMSTR\_MAX\_APPCMD\_CALLS is undefined or defined as zero, the FMSTR\_RegisterAppCmdCall function always fails.

## Pipes API

### FMSTR\_PipeOpen

#### Prototype

```
FMSTR_HPIPE FMSTR_PipeOpen(FMSTR_PIPE_PORT pipePort, FMSTR_PPIPEFUNC pipeCallback,  
    FMSTR_ADDR pipeRxBuff, FMSTR_PIPE_SIZE pipeRxSize,  
    FMSTR_ADDR pipeTxBuff, FMSTR_PIPE_SIZE pipeTxSize,  
    FMSTR_U8 type, const FMSTR_CHAR *name);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_pipes.c*

#### Arguments

- *pipePort* [in] - port number that identifies the pipe for the client
- *pipeCallback* [in] - pointer to the callback function that is called whenever a pipe data status changes
- *pipeRxBuff* [in] - address of the receive memory buffer
- *pipeRxSize* [in] - size of the receive memory buffer
- *pipeTxBuff* [in] - address of the transmit memory buffer
- *pipeTxSize* [in] - size of the transmit memory buffer
- *type* [in] - a combination of FMSTR\_PIPE\_MODE\_XXX and FMSTR\_PIPE\_SIZE\_XXX constants describing primary pipe data format and usage. This type helps FreeMASTER decide how to access the pipe by default. Optional, use 0 when undetermined.
- *name* [in] - user name of the pipe port. This name is visible to the FreeMASTER user when creating the graphical pipe interface.

**Description** This function initializes a new pipe and makes it ready to accept or send the data to the PC Host client. The receive memory buffer is used to store the received data before they are read out by the FMSTR\_PipeRead call. When this buffer gets full, the PC Host client denies the data transmission into this pipe until there is enough free space again. The transmit memory buffer is used to store the data transmitted by the application to the PC Host client using the FMSTR\_PipeWrite call. The transmit buffer can get full when the PC Host is disconnected or when it is slow in receiving and reading out the pipe data.

The function returns the pipe handle which must be stored and used in the subsequent calls to manage the pipe object.

The callback function (if specified) is called whenever new data are received through the pipe and available for reading. This callback is also called when the data waiting in the transmit buffer are successfully pushed to the PC Host and the transmit buffer free space increases. The prototype of the callback function provided by the user application must be as follows. The *PipeHandler* name is only a placeholder and must be defined by the application.

```
void PipeHandler(FMSTR_HPIPE pipeHandle);
```

## FMSTR\_PipeClose

### Prototype

```
void FMSTR_PipeClose(FMSTR_HPIPE pipeHandle);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_pipes.c*

### Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR\_PipeOpen function call

**Description** This function de-initializes the pipe object. No data can be received or sent on the pipe after this call.

## FMSTR\_PipeWrite

### Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeWrite(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,  
FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE writeGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_pipes.c*

### Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR\_PipeOpen function call
- *pipeData* [in] - address of the data to be written
- *pipeDataLen* [in] - length of the data to be written
- *writeGranularity* [in] - size of the minimum unit of data which is to be written

**Description** This function puts the user-specified data into the pipe's transmit memory buffer and schedules it for transmission. This function returns the number of bytes that were successfully written into the buffer. This number may be smaller than the number of the requested bytes if there is not enough free space in the transmit buffer.

The *writeGranularity* argument can be used to split the data into smaller chunks, each of the size given by the *writeGranularity* value. The FMSTR\_PipeWrite function writes as many data chunks as possible into the transmit buffer and does not attempt to write an incomplete chunk.

This feature can prove to be useful to avoid the intermediate caching when writing an array of integer values or other multi-byte data items. When making the `nGranularity` value equal to the `nLength` value, all data are considered as one chunk which is either written successfully as a whole or not at all. The `nGranularity` value of 0 or 1 disables the data-chunk approach.

## FMSTR\_PipeRead

### Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeRead(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,  
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE readGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster\_pipes.c*

### Arguments

- *pipeHandle* [in] - pipe handle returned from the `FMSTR_PipeOpen` function call
- *pipeData* [in] - address of the data buffer to be filled with the received data
- *pipeDataLen* [in] - length of the data to be read
- *readGranularity* [in] - size of the minimum unit of data which is to be read

**Description** This function copies the data received from the pipe from its receive buffer to the user buffer for further processing. The function returns the number of bytes that were successfully copied to the buffer. This number may be smaller than the number of the requested bytes if there is not enough data bytes available in the receive buffer.

The `readGranularity` argument can be used to copy the data in larger chunks in the same way as described in the `FMSTR_PipeWrite` function.

**API data types** This section describes the data types used in the FreeMASTER driver. The information provided here can be useful when modifying or porting the FreeMASTER Communication Driver to new NXP platforms.

**Note:** The licensing conditions prohibit use of FreeMASTER and the FreeMASTER Communication Driver with non-NXP MPU or MCU products.

**Public common types** The table below describes the public data types used in the FreeMASTER driver API calls. The data types are declared in the *freemaster.h* header file.

| Type name                                                                                                                                                                                                                 | Description                                                                                                                         |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>FM-STR_ADDR</i><br>For example, this type is defined as long integer on the 56F8xxx platform where the 24-bit addresses must be supported, but the C-pointer may be only 16 bits wide in some compiler configurations. | Data type used to hold the memory address. On most platforms, this is normally a C-pointer, but it may also be a pure integer type. |
| <i>FM-STR_SIZE</i><br>It is required that this type is unsigned and at least 16 bits wide integer.                                                                                                                        | Data type used to hold the memory block size.                                                                                       |
| <i>FM-STR_BOOL</i><br>This type is used only in zero/non-zero conditions in the driver code.                                                                                                                              | Data type used as a general boolean type.                                                                                           |
| <i>FM-STR_APPCM</i><br>Generally, this is an unsigned 8-bit value.                                                                                                                                                        | Data type used to hold the Application Command code.                                                                                |
| <i>FM-STR_APPCM</i><br>Generally, this is an unsigned 8-bit value.                                                                                                                                                        | Data type used to create the Application Command data buffer.                                                                       |
| <i>FM-STR_APPCM</i><br>Generally, this is an unsigned 8-bit value.                                                                                                                                                        | Data type used to hold the Application Command result code.                                                                         |

**Public TSA types** The table describes the TSA-specific public data types. These types are declared in the *freemaster\_tsa.h* header file, which is included in the user application indirectly by the *freemaster.h* file.

|                       |                                                                                                                                                               |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>FM-STR_TSA_TII</i> | Data type used to hold a descriptor index in the TSA table or a table index in the list of TSA tables.<br>By default, this is defined as <i>FM-STR_SIZE</i> . |
| <i>FM-STR_TSA_TS</i>  | Data type used to hold a memory block size, as used in the TSA descriptors.<br>By default, this is defined as <i>FM-STR_SIZE</i> .                            |

**Public Pipes types** The table describes the data types used by the FreeMASTER Pipes API:

|                       |                                                                                                                |
|-----------------------|----------------------------------------------------------------------------------------------------------------|
| <i>FM-STR_HPIPE</i>   | Pipe handle that identifies the open-pipe object.<br>Generally, this is a pointer to a void type.              |
| <i>FM-STR_PIPE_PC</i> | Integer type required to hold at least 7 bits of data.<br>Generally, this is an unsigned 8-bit or 16-bit type. |
| <i>FM-STR_PIPE_SI</i> | Integer type required to hold at least 16 bits of data.<br>This is used to store the data buffer sizes.        |
| <i>FM-STR_PPIPEF</i>  | Pointer to the pipe handler function.<br>See <a href="#">FM-STR_PipeOpen</a> for more details.                 |

**Internal types** The table describes the data types used internally by the FreeMASTER driver. The data types are declared in the platform-specific header file and they are not available in the application code.

|                                                                                                          |                                                                         |
|----------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|
| <i>FMSTR_U8</i>                                                                                          | The smallest memory entity.                                             |
| On the vast majority of platforms, this is an unsigned 8-bit integer.                                    |                                                                         |
| On the 56F8xx DSP platform, this is defined as an unsigned 16-bit integer.                               |                                                                         |
| <i>FM-STR_U16</i>                                                                                        | Unsigned 16-bit integer.                                                |
| <i>FM-STR_U32</i>                                                                                        | Unsigned 32-bit integer.                                                |
| <i>FMSTR_S8</i>                                                                                          | Signed 8-bit integer.                                                   |
| <i>FM-STR_S16</i>                                                                                        | Signed 16-bit integer.                                                  |
| <i>FM-STR_S32</i>                                                                                        | Signed 32-bit integer.                                                  |
| <i>FM-STR_FLOAT</i>                                                                                      | 4-byte standard IEEE floating-point type.                               |
| <i>FM-STR_FLAGS</i>                                                                                      | Data type forming a union with a structure of flag bit-fields.          |
| <i>FM-STR_SIZE8</i>                                                                                      | Data type holding a general size value, at least 8 bits wide.           |
| <i>FM-STR_INDEX</i>                                                                                      | General for-loop index. Must be signed, at least 16 bits wide.          |
| <i>FM-STR_BCHR</i>                                                                                       | A single character in the communication buffer.                         |
| Typically, this is an 8-bit unsigned integer, except for the DSP platforms where it is a 16-bit integer. |                                                                         |
| <i>FM-STR_BPTR</i>                                                                                       | A pointer to the communication buffer (an array of <i>FMSTR_BCHR</i> ). |

## Document references

### Links

- This document online: <https://mcuxpresso.nxp.com/mcuxsdk/latest/html/middleware/freemaster/doc/index.html>

- FreeMASTER tool home: [www.nxp.com/freemaster](http://www.nxp.com/freemaster)
- FreeMASTER community area: [community.nxp.com/community/freemaster](http://community.nxp.com/community/freemaster)
- FreeMASTER GitHub code repo: <https://github.com/nxp-mcuxpresso/mcux-freemaster>
- MCUXpresso SDK home: [www.nxp.com/mcuxpresso](http://www.nxp.com/mcuxpresso)
- MCUXpresso SDK builder: [mcuxpresso.nxp.com/en](http://mcuxpresso.nxp.com/en)

### Documents

- *FreeMASTER Usage Serial Driver Implementation* (document [AN4752](#))
- *Integrating FreeMASTER Time Debugging Tool With CodeWarrior For Microcontrollers v10.X Project* (document [AN4771](#))
- *Flash Driver Library For MC56F847xx And MC56F827xx DSC Family* (document [AN4860](#))

**Revision history** This Table summarizes the changes done to this document since the initial release.



| Revision | Date    | Description                                                                                                                                                                                                                                                                                                          |
|----------|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.0      | 03/2006 | Limited initial release                                                                                                                                                                                                                                                                                              |
| 2.0      | 09/2007 | Updated for FreeMASTER version. New Freescale document template used.                                                                                                                                                                                                                                                |
| 2.1      | 12/2007 | Added description of the new Fast Recorder feature and its API.                                                                                                                                                                                                                                                      |
| 2.2      | 04/2010 | Added support for MPC56xx platform, Added new API for use CAN interface.                                                                                                                                                                                                                                             |
| 2.3      | 04/2011 | Added support for Kxx Kinetis platform and MQX operating system.                                                                                                                                                                                                                                                     |
| 2.4      | 06/2011 | Serial driver update, adds support for USB CDC interface.                                                                                                                                                                                                                                                            |
| 2.5      | 08/2011 | Added Packet Driven BDM interface.                                                                                                                                                                                                                                                                                   |
| 2.7      | 12/2013 | Added FLEXCAN32 interface, byte access and isr callback configuration option.                                                                                                                                                                                                                                        |
| 2.8      | 06/2014 | Removed obsolete license text, see the software package content for up-to-date license.                                                                                                                                                                                                                              |
| 2.9      | 03/2015 | Update for driver version 1.8.2 and 1.9: FreeMASTER Pipes, TSA Active Content, LIN Transport Layer support, DEBUG-TX communication troubleshooting, Kinetis SDK support.                                                                                                                                             |
| 3.0      | 08/2016 | Update for driver version 2.0: Added support for MPC56xx, MPC57xx, KEAxx and S32Kxx platforms. New NXP document template as well as new license agreement used. added MCAN interface. Folders structure at the installation destination was rearranged.                                                              |
| 4.0      | 04/2019 | Update for driver released as part of FreeMASTER v3.0 and MCUXpresso SDK 2.6. Updated to match new V4 serial communication protocol and new configuration options. This version of the document removes substantial portion of outdated information related to S08, S12, ColdFire, Power and other legacy platforms. |
| 4.1      | 04/2020 | Minor update for FreeMASTER driver included in MCUXpresso SDK 2.8.                                                                                                                                                                                                                                                   |
| 4.2      | 09/2020 | Added example applications description and information about the MCUXpresso Config Tools. Fixed the pipe-related API description.                                                                                                                                                                                    |
| 4.3      | 10/2024 | Added description of Network and Segger J-Link RTT interface configuration. Accompanying the MCUXpresso SDK version 24.12.00.                                                                                                                                                                                        |
| 4.4      | 04/2025 | Added Zephyr-specific information. Accompanying the MCUXpresso SDK version 25.06.00.                                                                                                                                                                                                                                 |

## 3.3 MultiCore

### 3.3.1 Multicore SDK

Multicore Software Development Kit (MCSDK) is a Software Development Kit that provides comprehensive software support for NXP dual/multicore devices. The MCSDK is combined with the MCUXpresso SDK to make the software framework for easy development of multicore applications.

## Multicore SDK (MCSDK) Release Notes

**Overview** These are the release notes for the NXP Multicore Software Development Kit (MCSDK) version 25.12.00.

This software package contains components for efficient work with multicore devices as well as for the multiprocessor communication.

### What is new

- eRPC [CHANGELOG](#)
- RMPmsg-Lite [CHANGELOG](#)
- MCMgr [CHANGELOG](#)
- Supported evaluation boards (multicore examples):
  - LPCXpresso55S69
  - FRDM-K32L3A6
  - MIMXRT1170-EVKB
  - MIMXRT1160-EVK
  - MIMXRT1180-EVK
  - MCX-N5XX-EVK
  - MCX-N9XX-EVK
  - FRDM-MCXN947
  - MIMXRT700-EVK
  - KW47-EVK
  - KW47-LOC
  - FRDM-MCXW72
  - MCX-W72-EVK
  - FRDM-IMXRT1186
- Supported evaluation boards (multiprocessor examples):
  - LPCXpresso55S36
  - FRDM-K22F
  - FRDM-K32L2B
  - MIMXRT685-EVK
  - MIMXRT1170-EVKB
  - MIMXRT1180
  - FRDM-MCXN236
  - FRDM-MCXC242
  - FRDM-MCXC444
  - MCX-N9XX-EVK
  - FRDM-MCXN947
  - MIMXRT700-EVK
  - FRDM-IMXRT1186

**Development tools** The Multicore SDK (MCSDK) was compiled and tested with development tools referred in: [Development tools](#)

**Release contents** This table describes the release contents. Not all MCUXpresso SDK packages contain the whole set of these components.

| Deliverable                              | Location                                                      |
|------------------------------------------|---------------------------------------------------------------|
| Multicore SDK location<br><MCSDK_dir>    | <MCUXpressoSDK_install_dir>/middleware/multicore/             |
| Documentation                            | <MCSDK_dir>/mcuxsdk-doc/                                      |
| Embedded Remote Procedure Call component | <MCSDK_dir>/erpc/                                             |
| Multicore Manager component              | <MCSDK_dir>/mcmgr/                                            |
| RPMsg-Lite                               | <MCSDK_dir>/rpmsg_lite/                                       |
| Multicore demo applications              | <MCUXpressoSDK_install_dir>/examples/multicore_examples/      |
| Multiprocessor demo applications         | <MCUXpressoSDK_install_dir>/examples/multiprocessor_examples/ |

**Multicore SDK release overview** Together, the Multicore SDK (MCSDK) and the MCUXpresso SDK (SDK) form a framework for the development of software for NXP multicore devices. The MCSDK release consists of the following elementary software components for multicore:

- Embedded Remote Procedure Call (eRPC)
- Multicore Manager (MCMGR) - included just in SDK for multicore devices
- Remote Processor Messaging - Lite (RPMsg-Lite) - included just in SDK for multicore devices

The MCSDK is also accompanied with documentation and several multicore and multiprocessor demo applications.

**Demo applications** The multicore demo applications demonstrate the usage of the MCSDK software components on supported multicore development boards.

The following multicore demo applications are located together with other MCUXpresso SDK examples in

the <MCUXpressoSDK\_install\_dir>/examples/multicore\_examples subdirectories.

- erpc\_matrix\_multiply\_mu
- erpc\_matrix\_multiply\_mu\_rtos
- erpc\_matrix\_multiply\_rpmsg
- erpc\_matrix\_multiply\_rpmsg\_rtos
- erpc\_two\_way\_rpc\_rpmsg\_rtos
- freertos\_message\_buffers
- hello\_world
- multicore\_manager
- rpmsg\_lite\_pingpong
- rpmsg\_lite\_pingpong\_rtos
- rpmsg\_lite\_pingpong\_dsp
- rpmsg\_lite\_pingpong\_tzm

The eRPC multicore component can be leveraged for inter-processor communication and remote procedure calls between SoCs / development boards.

The following multiprocessor demo applications are located together with other MCUXpresso SDK examples in

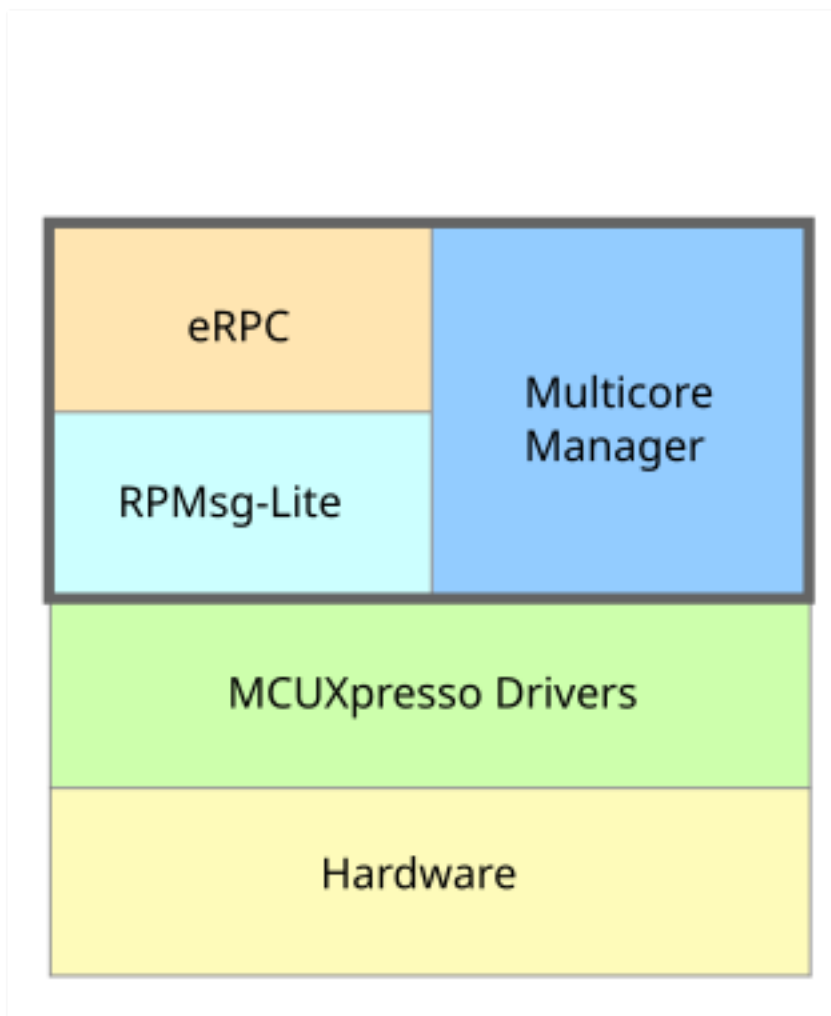
the <MCUXpressoSDK\_install\_dir>/examples/multiprocessor\_examples subdirectories.

- erpc\_client\_matrix\_multiply\_spi
- erpc\_server\_matrix\_multiply\_spi
- erpc\_client\_matrix\_multiply\_uart
- erpc\_server\_matrix\_multiply\_uart
- erpc\_server\_dac\_adc
- erpc\_remote\_control

### Getting Started with Multicore SDK (MCSDK)

**Overview** Multicore Software Development Kit (MCSDK) is a Software Development Kit that provides comprehensive software support for NXP dual/multicore devices. The MCSDK is combined with the MCUXpresso SDK to make the software framework for easy development of multicore applications.

The following figure highlights the layers and main software components of the MCSDK.

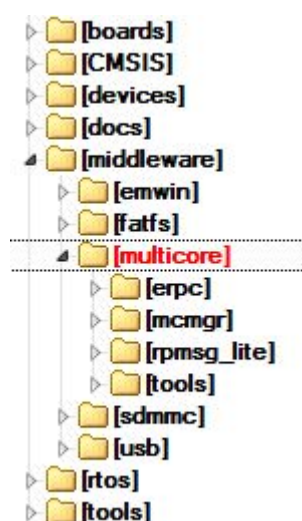


All the MCSDK-related files are located in `<MCUXpressoSDK_install_dir>/middleware/multicore` folder.

For supported toolchain versions, see the *Multicore SDK v25.12.00 Release Notes* (document MCS-DKRN). For the latest version of this and other MCSDK documents, visit [www.nxp.com](http://www.nxp.com).

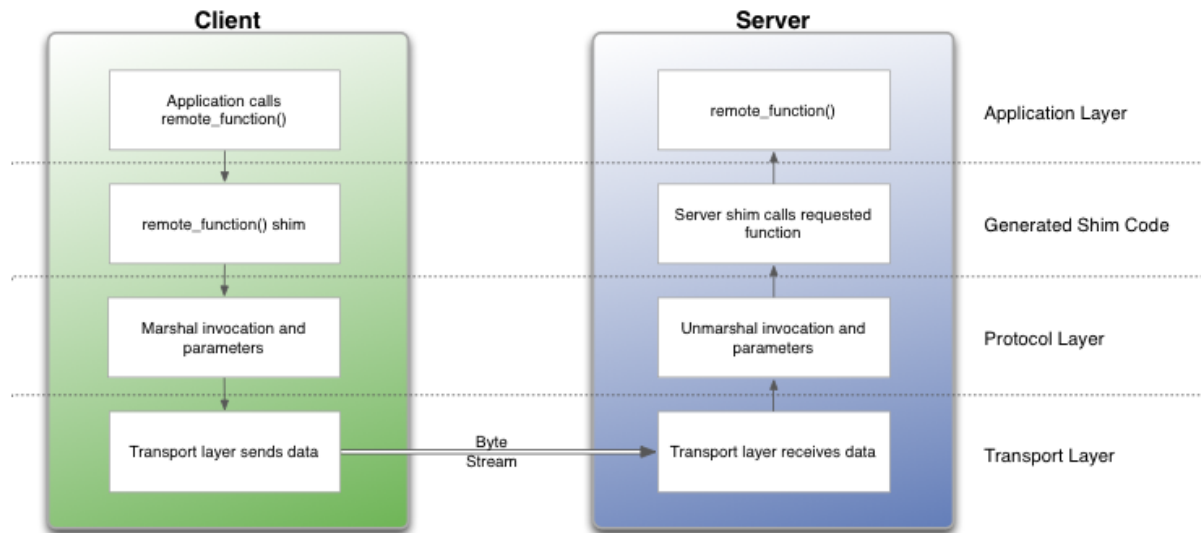
**Multicore SDK (MCSDK) components** The MCSDK consists of the following software components:

- **Embedded Remote Procedure Call (eRPC):** This component is a combination of a library and code generator tool that implements a transparent function call interface to remote services (running on a different core).
- **Multicore Manager (MCMGR):** This library maintains information about all cores and starts up secondary/auxiliary cores.
- **Remote Processor Messaging - Lite (RPMsg-Lite):** Inter-Processor Communication library.



**Embedded Remote Procedure Call (eRPC)** The Embedded Remote Procedure Call (eRPC) is the RPC system created by NXP. The RPC is a mechanism used to invoke a software routine on a remote system via a simple local function call.

When a remote function is called by the client, the function's parameters and an identifier for the called routine are marshaled (or serialized) into a stream of bytes. This byte stream is transported to the server through a communications channel (IPC, TPC/IP, UART, and so on). The server unmarshals the parameters, determines which function was invoked, and calls it. If the function returns a value, it is marshaled and sent back to the client.



RPC implementations typically use a combination of a tool (erpcgen) and IDL (interface definition language) file to generate source code to handle the details of marshaling a function's parameters and building the data stream.

#### Main eRPC features:

- Scalable from BareMetal to Linux OS - configurable memory and threading policies.
- Focus on embedded systems - intrinsic support for C, modular, and lightweight implementation.
- Abstracted transport interface - RPMsg is the primary transport for multicore, UART, or SPI-based solutions can be used for multichip.

The eRPC library is located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/erpc` folder. For detailed information about the eRPC, see the documentation available in the `<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/doc` folder.

**Multicore Manager (MCMGR)** The Multicore Manager (MCMGR) software library provides a number of services for multicore systems.

The main MCMGR features:

- Maintains information about all cores in system.
- Secondary/auxiliary cores startup and shutdown.
- Remote core monitoring and event handling.

The MCMGR library is located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr` folder. For detailed information about the MCMGR library, see the documentation available in the `<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/doc` folder.

**Remote Processor Messaging Lite (RPMsg-Lite)** RPMsg-Lite is a lightweight implementation of the RPMsg protocol. The RPMsg protocol defines a standardized binary interface used to communicate between multiple cores in a heterogeneous multicore system. Compared to the legacy OpenAMP implementation, RPMsg-Lite offers a code size reduction, API simplification, and improved modularity.

The main RPMsg protocol features:

- Shared memory interprocessor communication.
- Virtio-based messaging bus.
- Application-defined messages sent between endpoints.

- Portable to different environments/platforms.
- Available in upstream Linux OS.

The RPMsg-Lite library is located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg-lite` folder. For detailed information about the RPMsg-Lite, see the RPMsg-Lite User's Guide located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/doc` folder.

**MCSDK demo applications** Multicore and multiprocessor example applications are stored together with other MCUXpresso SDK examples, in the dedicated multicore subfolder.

| Location                        | Folder                                                                                                    |
|---------------------------------|-----------------------------------------------------------------------------------------------------------|
| Multicore example projects      | <code>&lt;MCUXpressoSDK_install_dir&gt;/examples/multicore_examples/&lt;application_name&gt;/</code>      |
| Multiprocessor example projects | <code>&lt;MCUXpressoSDK_install_dir&gt;/examples/multiprocessor_examples/&lt;application_name&gt;/</code> |

See the *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) and *Getting Started with MCUXpresso SDK for XXX Derivatives* documents for more information about the MCUXpresso SDK example folder structure and the location of individual files that form the example application projects. These documents also contain information about building, running, and debugging multicore demo applications in individual supported IDEs. Each example application also contains a readme file that describes the operation of the example and required setup steps.

**Inter-Processor Communication (IPC) levels** The MCSDK provides several mechanisms for Inter-Processor Communication (IPC). Particular ways and levels of IPC are described in this chapter.

### IPC using low-level drivers

The NXP multicore SoCs are equipped with peripheral modules dedicated for data exchange between individual cores. They deal with the Mailbox peripheral for LPC parts and the Messaging Unit (MU) peripheral for Kinetis and i.MX parts. The common attribute of both modules is the ability to provide a means of IPC, allowing multiple CPUs to share resources and communicate with each other in a simple manner.

The most lightweight method of IPC uses the MCUXpresso SDK low-level drivers for these peripherals. Using the Mailbox/MU driver API functions, it is possible to pass a value from core to core via the dedicated registers (could be a scalar or a pointer to shared memory) and also to trigger inter-core interrupts for notifications.

For details about individual driver API functions, see the MCUXpresso SDK API Reference Manual of the specific multicore device. The MCUXpresso SDK is accompanied with the RPMsg-Lite documentation that shows how to use this API in multicore applications.

### Messaging mechanism

On top of Mailbox/MU drivers, a messaging system can be implemented, allowing messages to send between multiple endpoints created on each of the CPUs. The RPMsg-Lite library of the MCSDK provides this ability and serves as the preferred MCUXpresso SDK messaging library. It implements ring buffers in shared memory for messages exchange without the need of a locking mechanism.

The RPMsg-Lite provides the abstraction layer and can be easily ported to different multicore platforms and environments (Operating Systems). The advantages of such a messaging system are ease of use (there is no need to study behavior of the used underlying hardware) and smooth application code portability between platforms due to unified messaging API.



However, this costs several kB of code and data memory. The MCUXpresso SDK is accompanied by the RPMsg-Lite documentation and several multicore examples. You can also obtain the latest RPMsg-Lite code from the GitHub account [github.com/nxp-mcuxpresso/rpmsg-lite](https://github.com/nxp-mcuxpresso/rpmsg-lite).

### **Remote procedure calls**

To facilitate the IPC even more and to allow the remote functions invocation, the remote procedure call mechanism can be implemented. The eRPC of the MCSDK serves for these purposes and allows the ability to invoke a software routine on a remote system via a simple local function call. Utilizing different transport layers, it is possible to communicate between individual cores of multicore SoCs (via RPMsg-Lite) or between separate processors (via SPI, UART, or TCP/IP). The eRPC is mostly applicable to the MPU parts with enough of memory resources like i.MX parts.

The eRPC library allows you to export existing C functions without having to change their prototypes (in most cases). It is accompanied by the code generator tool that generates the shim code for serialization and invocation based on the IDL file with definitions of data types and remote interfaces (API).

If the communicating peer is running as a Linux OS user-space application, the generated code can be either in C/C++ or Python.

Using the eRPC simplifies the access to services implemented on individual cores. This way, the following types of applications running on dedicated cores can be easily interfaced:

- Communication stacks (USB, Thread, Bluetooth Low Energy, Zigbee)
- Sensor aggregation/fusion applications
- Encryption algorithms
- Virtual peripherals

The eRPC is publicly available from the following GitHub account: [github.com/EmbeddedRPC/erpc](https://github.com/EmbeddedRPC/erpc). Also, the MCUXpresso SDK is accompanied by the eRPC code and several multicore and multiprocessor eRPC examples.

The mentioned IPC levels demonstrate the scalability of the Multicore SDK library. Based on application needs, different IPC techniques can be used. It depends on the complexity, required speed, memory resources, system design, and so on. The MCSDK brings users the possibility for quick and easy development of multicore and multiprocessor applications.

### **Changelog Multicore SDK**

All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog](#), and this project adheres to [Semantic Versioning](#).

#### **[25.12.00]**

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.0.2
  - RPMsg-Lite v5.3.0

#### **[25.09.00]**

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0



- eRPC generator (erpcgen) v1.14.0
- Multicore Manager (MCMgr) v5.0.1
- RPSMsg-Lite v5.2.1

#### [25.06.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.0.0
  - RPSMsg-Lite v5.2.0

#### [25.03.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.13.0
  - eRPC generator (erpcgen) v1.13.0
  - Multicore Manager (MCMgr) v4.1.7
  - RPSMsg-Lite v5.1.4

#### [24.12.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.13.0
  - eRPC generator (erpcgen) v1.13.0
  - Multicore Manager (MCMgr) v4.1.6
  - RPSMsg-Lite v5.1.3

#### [2.16.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.13.0
  - eRPC generator (erpcgen) v1.13.0
  - Multicore Manager (MCMgr) v4.1.5
  - RPSMsg-Lite v5.1.2

#### [2.15.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.12.0
  - eRPC generator (erpcgen) v1.12.0
  - Multicore Manager (MCMgr) v4.1.5
  - RPSMsg-Lite v5.1.1

#### [2.14.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.11.0
  - eRPC generator (erpcgen) v1.11.0
  - Multicore Manager (MCMgr) v4.1.4
  - RPSMsg-Lite v5.1.0

#### [2.13.0\_imxrt1180a0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.10.0
  - eRPC generator (erpcgen) v1.10.0
  - Multicore Manager (MCMgr) v4.1.3
  - RPSMsg-Lite v5.0.0

#### [2.13.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.10.0
  - eRPC generator (erpcgen) v1.10.0
  - Multicore Manager (MCMgr) v4.1.3
  - RPSMsg-Lite v5.0.0

#### [2.12.0\_imx93]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.1
  - eRPC generator (erpcgen) v1.9.1
  - Multicore Manager (MCMgr) v4.1.2
  - RPSMsg-Lite v4.0.1

#### [2.12.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.1
  - eRPC generator (erpcgen) v1.9.1
  - Multicore Manager (MCMgr) v4.1.2
  - RPSMsg-Lite v4.0.0

**[2.11.1]**

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.0
  - eRPC generator (erpcgen) v1.9.0
  - Multicore Manager (MCMgr) v4.1.1
  - RPSMsg-Lite v3.2.1

**[2.11.0]**

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.0
  - eRPC generator (erpcgen) v1.9.0
  - Multicore Manager (MCMgr) v4.1.1
  - RPSMsg-Lite v3.2.0

**[2.10.0]**

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.8.1
  - eRPC generator (erpcgen) v1.8.1
  - Multicore Manager (MCMgr) v4.1.1
  - RPSMsg-Lite v3.1.2

**[2.9.0]**

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.8.0
  - eRPC generator (erpcgen) v1.8.0
  - Multicore Manager (MCMgr) v4.1.1
  - RPSMsg-Lite v3.1.1

**[2.8.0]**

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.4
  - eRPC generator (erpcgen) v1.7.4
  - Multicore Manager (MCMgr) v4.1.0
  - RPSMsg-Lite v3.1.0

#### [2.7.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.3
  - eRPC generator (erpcgen) v1.7.3
  - Multicore Manager (MCMgr) v4.1.0
  - RPSMsg-Lite v3.0.0

#### [2.6.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.2
  - eRPC generator (erpcgen) v1.7.2
  - Multicore Manager (MCMgr) v4.0.3
  - RPSMsg-Lite v2.2.0

#### [2.5.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.1
  - eRPC generator (erpcgen) v1.7.1
  - Multicore Manager (MCMgr) v4.0.2
  - RPSMsg-Lite v2.0.2

#### [2.4.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.0
  - eRPC generator (erpcgen) v1.7.0
  - Multicore Manager (MCMgr) v4.0.1
  - RPSMsg-Lite v2.0.1

#### [2.3.1]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.6.0
  - eRPC generator (erpcgen) v1.6.0
  - Multicore Manager (MCMgr) v4.0.0
  - RPSMsg-Lite v1.2.0

**[2.3.0]**

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.5.0
  - eRPC generator (erpcgen) v1.5.0
  - Multicore Manager (MCMgr) v3.0.0
  - RPSMsg-Lite v1.2.0

**[2.2.0]**

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.4.0
  - eRPC generator (erpcgen) v1.4.0
  - Multicore Manager (MCMgr) v2.0.1
  - RPSMsg-Lite v1.1.0

**[2.1.0]**

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.3.0
  - eRPC generator (erpcgen) v1.3.0

**[2.0.0]**

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.2.0
  - eRPC generator (erpcgen) v1.2.0
  - Multicore Manager (MCMgr) v2.0.0
  - RPSMsg-Lite v1.0.0

**[1.1.0]**

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.1.0
  - Multicore Manager (MCMgr) v1.1.0
  - Open-AMP / RPSMsg based on SHA1 ID 44b5f3c0a6458f3cf80 rev01

**[1.0.0]**

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.0.0
  - Multicore Manager (MCMgr) v1.0.0
  - Open-AMP / RPSMsg based on SHA1 ID 44b5f3c0a6458f3cf80 rev00

## Multicore SDK Components

### RPMSG-Lite

#### MCUXpresso SDK : mcuxsdk-middleware-rpmsg-lite

**Overview** This repository is for MCUXpresso SDK RPMSG-Lite middleware delivery and it contains RPMSG-Lite component officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository [mcuxsdk](#) for the complete delivery of MCUXpresso SDK to be able to build and run RPMSG-Lite examples that are based on mcux-sdk-middleware-rpmsg-lite component.

**Documentation** Overall details can be reviewed here: [MCUXpresso SDK Online Documentation](#)

Visit [RPMSG-Lite - Documentation](#) to review details on the contents in this sub-repo.

For Further API documentation, please look at [doxygen documentation](#)

**Setup** Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest [Getting Started with SDK - Detailed Installation Instructions](#)

**Contribution** We welcome and encourage the community to submit patches directly to the rpmsg-lite project placed on github. Contributing can be managed via pull-requests. Before a pull-request is created the code should be tested and properly formatted.

---

**RPMSG-Lite** This documentation describes the RPMsg-Lite component, which is a lightweight implementation of the Remote Processor Messaging (RPMsg) protocol. The RPMsg protocol defines a standardized binary interface used to communicate between multiple cores in a heterogeneous multicore system.

Compared to the RPMsg implementation of the Open Asymmetric Multi Processing (OpenAMP) framework (<https://github.com/OpenAMP/open-amp>), the RPMsg-Lite offers a code size reduction, API simplification, and improved modularity. On smaller Cortex-M0+ based systems, it is recommended to use RPMsg-Lite.

The RPMsg-Lite is an open-source component developed by NXP Semiconductors and released under the BSD-compatible license.

For overview please read RPMSG-Lite VirtIO Overview.

For RPMSG-Lite Design Considerations please read RPMSG-Lite Design Considerations.

**Motivation to create RPMsg-Lite** There are multiple reasons why RPMsg-Lite was developed. One reason is the need for the small footprint of the RPMsg protocol-compatible communication component, another reason is the simplification of extensive API of OpenAMP RPMsg implementation.

RPMsg protocol was not documented, and its only definition was given by the Linux Kernel and legacy OpenAMP implementations. This has changed with [1] which is a standardization protocol allowing multiple different implementations to coexist and still be mutually compatible.

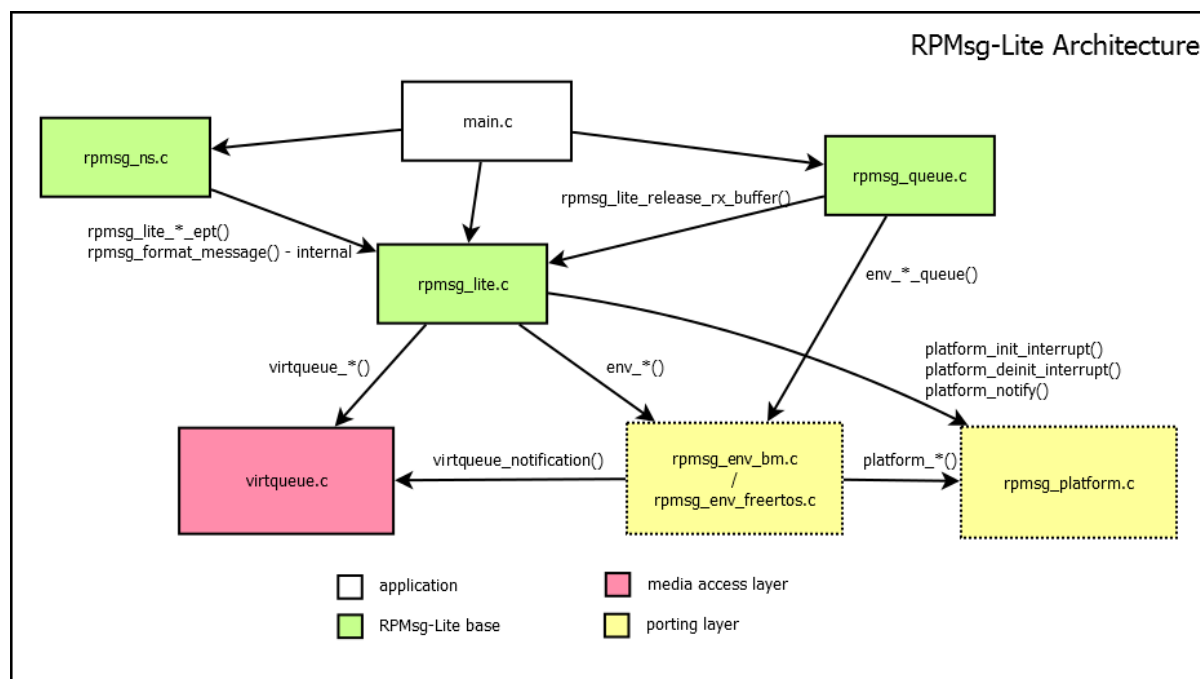
Small MCU-based systems often do not implement dynamic memory allocation. The creation of static API in RPMsg-Lite enables another reduction of resource usage. Not only does the dynamic allocation adds another 5 KB of code size, but also communication is slower and less deterministic, which is a property introduced by dynamic memory. The following table shows some rough comparison data between the OpenAMP RPMsg implementation and new RPMsg-Lite implementation:

| Component / Configuration                    | Flash [B] | RAM [B]       |
|----------------------------------------------|-----------|---------------|
| OpenAMP RPMsg / Release (reference)          | 5547      | 456 + dynamic |
| RPMsg-Lite / Dynamic API, Release            | 3462      | 56 + dynamic  |
| Relative Difference [%]                      | ~62.4%    | ~12.3%        |
| RPMsg-Lite / Static API (no malloc), Release | 2926      | 352           |
| Relative Difference [%]                      | ~52.7%    | ~77.2%        |

**Implementation** The implementation of RPMsg-Lite can be divided into three sub-components, from which two are optional. The core component is situated in `rpmsg_lite.c`. Two optional components are used to implement a blocking receive API (in `rpmsg_queue.c`) and dynamic “named” endpoint creation and deletion announcement service (in `rpmsg_ns.c`).

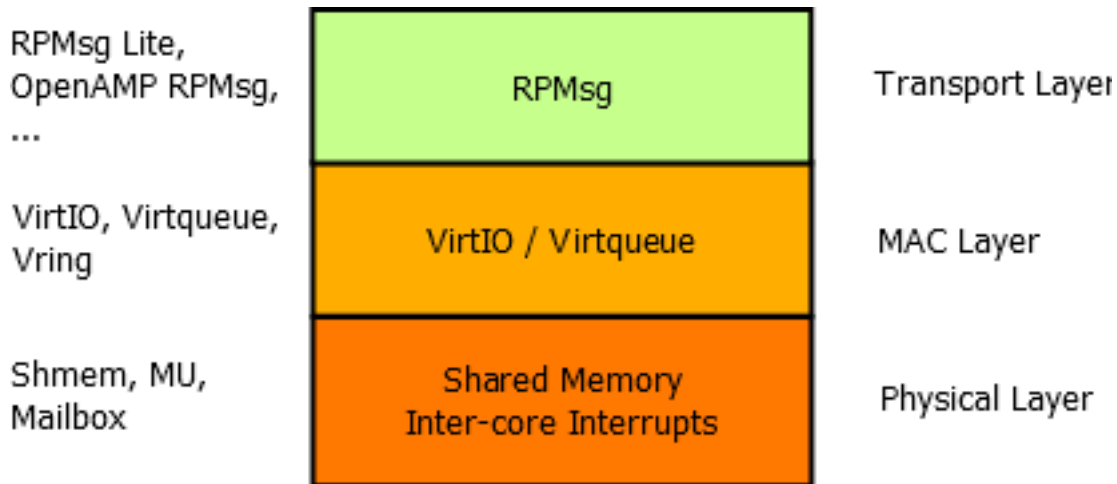
The actual “media access” layer is implemented in `virtqueue.c`, which is one of the few files shared with the OpenAMP implementation. This layer mainly defines the shared memory model, and internally defines used components such as `vring` or `virtqueue`.

The porting layer is split into two sub-layers: the environment layer and the platform layer. The first sublayer is to be implemented separately for each environment. (The bare metal environment already exists and is implemented in `rpmsg_env_bm.c`, and the FreeRTOS environment is implemented in `rpmsg_env_freertos.c` etc.) Only the source file, which matches the used environment, is included in the target application project. The second sublayer is implemented in `rpmsg_platform.c` and defines low-level functions for interrupt enabling, disabling, and triggering mainly. The situation is described in the following figure:



**RPMsg-Lite core sub-component** This subcomponent implements a blocking send API and callback-based receive API. The RPMsg protocol is part of the transport layer. This is realized by using so-called endpoints. Each endpoint can be assigned a different receive callback function.

However, it is important to notice that the callback is executed in an interrupt environment in current design. Therefore, certain actions like memory allocation are discouraged to execute in the callback. The following figure shows the role of RPMsg in an ISO/OSI-like layered model:



**Queue sub-component (optional)** This subcomponent is optional and requires implementation of the `env_*_queue()` functions in the environment porting layer. It uses a blocking receive API, which is common in RTOS-environments. It supports both copy and nocopy blocking receive functions.

**Name Service sub-component (optional)** This subcomponent is a minimum implementation of the name service which is present in the Linux Kernel implementation of RPMsg. It allows the communicating node both to send announcements about “named” endpoint (in other words, channel) creation or deletion and to receive these announcement taking any user-defined action in an application callback. The endpoint address used to receive name service announcements is arbitrarily fixed to be 53 (0x35).

**Usage** The application should put the `/rpmmsg_lite/lib/include` directory to the include path and in the application, include either the `rpmmsg_lite.h` header file, or optionally also include the `rpmmsg_queue.h` and/or `rpmmsg_ns.h` files. Both porting sublayers should be provided for you by NXP, but if you plan to use your own RTOS, all you need to do is to implement your own environment layer (in other words, `rpmmsg_env_myrtos.c`) and to include it in the project build.

The initialization of the stack is done by calling the `rpmmsg_lite_master_init()` on the master side and the `rpmmsg_lite_remote_init()` on the remote side. This initialization function must be called prior to any RPMsg-Lite API call. After the init, it is wise to create a communication endpoint, otherwise communication is not possible. This can be done by calling the `rpmmsg_lite_create_ept()` function. It optionally accepts a last argument, where an internal context of the endpoint is created, just in case the `RL_USE_STATIC_API` option is set to 1. If not, the stack internally calls `env_alloc()` to allocate dynamic memory for it. In case a callback-based receiving is to be used, an ISR-callback is registered to each new endpoint with user-defined callback data pointer. If a blocking receive is desired (in case of RTOS environment), the `rpmmsg_queue_create()` function must be called before calling `rpmmsg_lite_create_ept()`. The queue handle is passed to the endpoint creation function as a callback data argument and the callback function is set to `rpmmsg_queue_rx_cb()`. Then, it is possible to use `rpmmsg_queue_receive()` function to listen on a queue object for incoming messages. The `rpmmsg_lite_send()` function is used to send messages to the other side.

The RPMsg-Lite also implements no-copy mechanisms for both sending and receiving operations. These methods require specifics that have to be considered when used in an application.



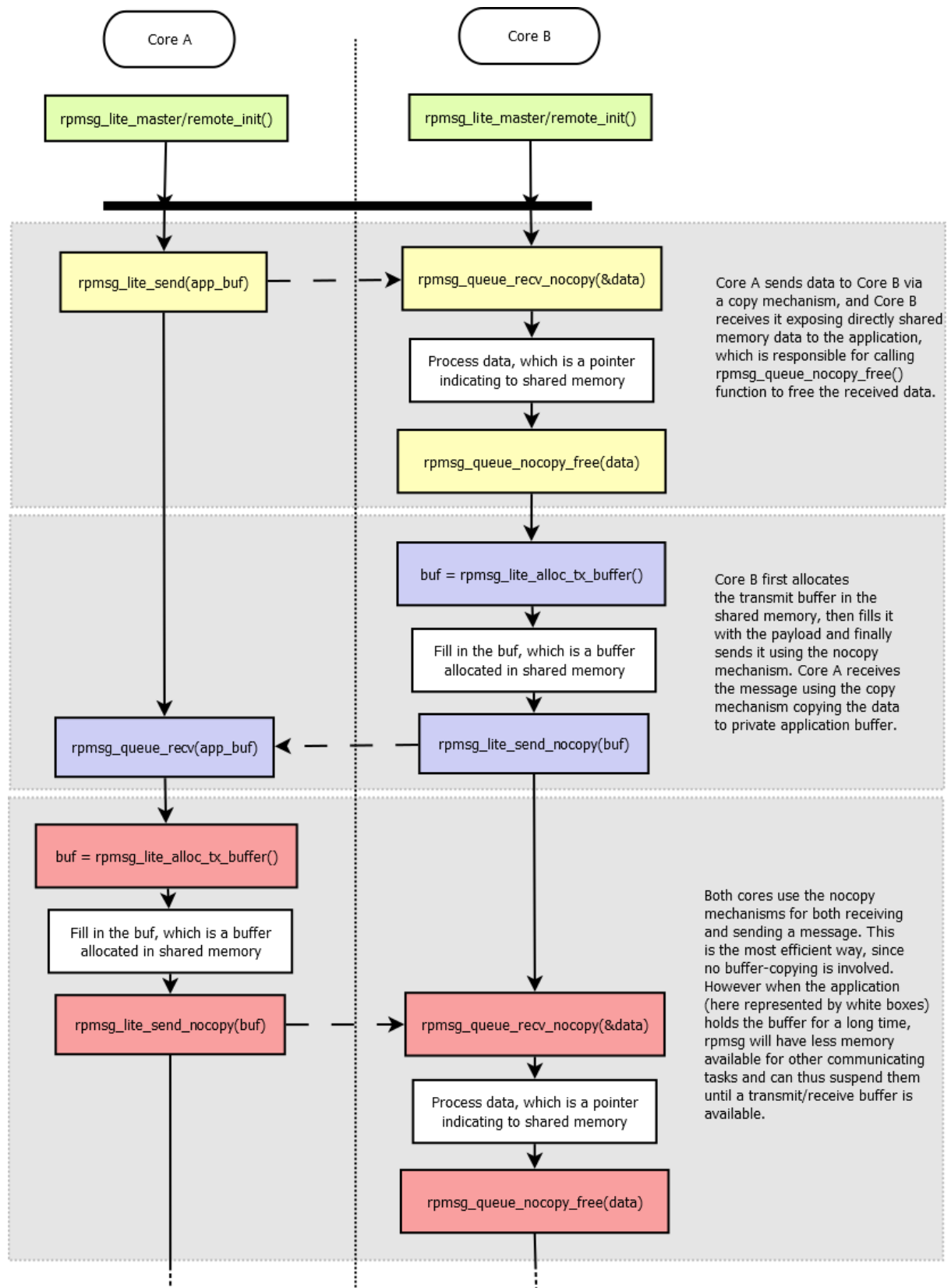
no-copy-send mechanism: This mechanism allows sending messages without the cost for copying data from the application buffer to the RPMsg/virtio buffer in the shared memory. The sequence of no-copy sending steps to be performed is as follows:

- Call the `rpmmsg_lite_alloc_tx_buffer()` function to get the virtio buffer and provide the buffer pointer to the application.
- Fill the data to be sent into the pre-allocated virtio buffer. Ensure that the filled data does not exceed the buffer size (provided as the `rpmmsg_lite_alloc_tx_buffer()` size output parameter).
- Call the `rpmmsg_lite_send_nocopy()` function to send the message to the destination endpoint. Consider the cache functionality and the virtio buffer alignment. See the `rpmmsg_lite_send_nocopy()` function description below.

no-copy-receive mechanism: This mechanism allows reading messages without the cost for copying data from the virtio buffer in the shared memory to the application buffer. The sequence of no-copy receiving steps to be performed is as follows:

- Call the `rpmmsg_queue_rcv_nocopy()` function to get the virtio buffer pointer to the received data.
- Read received data directly from the shared memory.
- Call the `rpmmsg_queue_nocopy_free()` function to release the virtio buffer and to make it available for the next data transfer.

The user is responsible for destroying any RPMsg-Lite objects he has created in case of deinitialization. In order to do this, the function `rpmmsg_queue_destroy()` is used to destroy a queue, `rpmmsg_lite_destroy_ept()` is used to destroy an endpoint and finally, `rpmmsg_lite_deinit()` is used to deinitialize the RPMsg-Lite intercore communication stack. Deinitialize all endpoints using a queue before deinitializing the queue. Otherwise, you are actively invalidating the used queue handle, which is not allowed. RPMsg-Lite does not check this internally, since its main aim is to be lightweight.



**Examples** RPMsg\_Lite multicore examples are part of NXP MCUXpressoSDK packages. Visit <https://mcuxpresso.nxp.com> to configure, build and download these packages. To get the board list with multicore support (RPMsg\_Lite included) use filtering based on Middleware and search for 'multicore' string. Once the selected package with the multicore middleware is downloaded,

see

<MCUXpressoSDK\_install\_dir>/boards/<board\_name>/multicore\_examples for RPMsg\_Lite multicore examples with 'rpmsg\_lite\_' name prefix.

Another way of getting NXP MCUXpressoSDK RPMsg\_Lite multicore examples is using the [mcuxsdk-manifests](#) Github repo. Follow the description how to use the West tool to clone and update the mcuxsdk-manifests repo in [readme section](#). Once done the armgcc rpmsg\_lite examples can be found in

mcuxsdk/examples/\_<board\_name>/multicore\_examples

You can use the evkmimxrt1170 as the board\_name for instance. Similar to MCUXpressoSDK packages the RPMsg\_Lite examples use the 'rpmsg\_lite\_' name prefix.

## Notes

**Environment layers implementation** Several environment layers are provided in lib/rpmsg\_lite/porting/environment folder. Not all of them are fully tested however. Here is the list of environment layers that passed testing:

- rpmsg\_env\_bm.c
- rpmsg\_env\_freertos.c
- rpmsg\_env\_xos.c
- rpmsg\_env\_threadx.c

The rest of environment layers has been created and used in some experimental projects, it has been running well at the time of creation but due to the lack of unit testing there is no guarantee it is still fully functional.

**Shared memory configuration** It is important to correctly initialize/configure the shared memory for data exchange in the application. The shared memory must be accessible from both the master and the remote core and it needs to be configured as Non-Cacheable memory. Dedicated shared memory section in linker file is also a good practise, it is recommended to use linker files from MCUXpressoSDK packages for NXP devices based applications. It needs to be ensured no other application part/component is unintentionally accessing this part of memory.

**Configuration options** The RPMsg-Lite can be configured at the compile time. The default configuration is defined in the rpmsg\_default\_config.h header file. This configuration can be customized by the user by including rpmsg\_config.h file with custom settings. The following table summarizes all possible RPMsg-Lite configuration options.

| Config-<br>uration<br>option | De-<br>fault<br>value | Usage                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|------------------------------|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RL_MS_PE (1)                 |                       | Delay in milliseconds used in non-blocking API functions for polling.                                                                                                                                                                                                                                                                                                                                                                             |
| RL_BUFEE (496)               |                       | Size of the buffer payload, it must be more than 1 byte, and has to be word align (including rpmsg header size 16 bytes), if not it will be aligned up                                                                                                                                                                                                                                                                                            |
| RL_BUFEE (2)                 |                       | Number of the buffers, it must be power of two (2, 4, ...)                                                                                                                                                                                                                                                                                                                                                                                        |
| RL_API_H (1)                 |                       | Zero-copy API functions enabled/disabled.                                                                                                                                                                                                                                                                                                                                                                                                         |
| RL_USE_S' (0)                |                       | Static API functions (no dynamic allocation) enabled/disabled.                                                                                                                                                                                                                                                                                                                                                                                    |
| RL_USE_D (0)                 |                       | Memory cache management of shared memory. Use in case of data cache is enabled for shared memory.                                                                                                                                                                                                                                                                                                                                                 |
| RL_CLEAF (0)                 |                       | Clearing used buffers before returning back to the pool of free buffers enabled/disabled.                                                                                                                                                                                                                                                                                                                                                         |
| RL_USE_M (0)                 |                       | When enabled IPC interrupts are managed by the Multicore Manager (IPC interrupts router), when disabled RPMsg-Lite manages IPC interrupts by itself.                                                                                                                                                                                                                                                                                              |
| RL_USE_E (0)                 |                       | When enabled the environment layer uses its own context. Required for some environments (QNX). The default value is 0 (no context, saves some RAM).                                                                                                                                                                                                                                                                                               |
| RL_DEBU (0)                  |                       | When enabled buffer pointers passed to <code>rpmsg_lite_send_nocopy()</code> and <code>rpmsg_lite_release_rx_buffer()</code> functions (enabled by <code>RL_API_HAS_ZEROCOPY</code> config) are checked to avoid passing invalid buffer pointer. The default value is 0 (disabled). Do not use in RPMsg-Lite to Linux configuration.                                                                                                              |
| RL_ALLOV (0)                 |                       | When enabled the opposite side is notified each time received buffers are consumed and put into the queue of available buffers. Enable this option in RPMsg-Lite to Linux configuration to allow unblocking of the Linux blocking send. The default value is 0 (RPMsg-Lite to RPMsg-Lite communication).                                                                                                                                          |
| RL_ALLOV (0)                 |                       | It allows to define custom shared memory configuration and replacing the shared memory related global settings from <code>rpmsg_config.h</code> . This is useful when multiple instances are running in parallel but different shared memory arrangement (vring size & alignment, buffers size & count) is required. The default value is 0 (all RPMsg_Lite instances use the same shared memory arrangement as defined by common config macros). |
| RL_ASSER                     | see<br>rpmsg          | Assert implementation.                                                                                                                                                                                                                                                                                                                                                                                                                            |

**How to format rpmsg-lite code** To format code, use the application developed by Google, named *clang-format*. This tool is part of the [llvm](#) project. Currently, the clang-format 10.0.0 version is used for rpmsg-lite. The set of style settings used for clang-format is defined in the `.clang-format` file, placed in a root of the rpmsg-lite directory where Python script `run_clang_format.py` can be executed. This script executes the application named *clang-format.exe*. You need to have the path of this application in the OS's environment path, or you need to change the script.

## References

[1] M. Novak, M. Cingel, **Lockless Shared Memory Based Multicore Communication Protocol**  
Copyright © 2016 Freescale Semiconductor, Inc. Copyright © 2016-2025 NXP

**Changelog RPMSG-Lite** All notable changes to this project will be documented in this file. The format is based on [Keep a Changelog](#), and this project adheres to [Semantic Versioning](#).

**[v5.3.0]****Added**

- RT700 porting layer added support to send rpmsg messages between CM33\_0 <-> Hifi1 and CM33\_1 <-> Hifi4 cores.
- Add new platform macro `RL_PLATFORM_MAX_ISR_COUNT` this will set number of IRQ count per platform. This macro is then used in environment layers to set `isr_table` size where irq handles are registered. It size should match the bit length of `VQ_ID` so all combinations can fit into table.
- Unit tests updated to improve code coverage, new unit tests added covering static allocations in rtos environment layers.

**Fixed**

- `virtio.h` removed `typedef uint8_t boolean` and in its place use standard C99 `bool` type to avoid potential type conflicts.
- `env_acquire_sync_lock()` and `env_release_sync_lock()` synchronization primitives removed
- Kconfig consolidation, when `RL_ALLOW_CUSTOM_SHMEM_CONFIG` enabled the `platform_get_custom_shmem_config()` function needs to be implemented in platform layer to provide custom shared memory configuration for RPMsg-Lite instance.

**v5.2.1****Added**

- Doc added RPMSG-Lite VirtIO Overview
- Doc added RPMSG-Lite Design Considerations
- Added `frdmimxrt1186` unit testing

**Changed**

- Remove limitation that `RL_BUFFER_SIZE` needs to be power of 2. It just has to be more than 16 bytes, e.g. 16 bytes of rpmsg header and payload size at least 1 byte and word aligned, if not it will be aligned up.

**Fixed**

- Fixed CERT-C INT31-C violation in `platform_notify` function in `rpmsg_platform.c` for `imxrt700_m33`, `imxrt700_hifi4`, `imxrt700_hifi1` platforms

**v5.2.0****Added**

- Add MCXL20 porting layer and unit testing
- New utility macro `RL_CALCULATE_BUFFER_COUNT_DOWN_SAFE` to safely determine maximum buffer count within shared memory while preventing integer underflow.
- RT700 platform add support for MCMGR in DSPs

## Changed

- Change `rpmsg_platform.c` to support new MCMGR API
- Improved input validation in initialization functions to properly handle insufficient memory size conditions.
- Refactored repeated buffer count calculation pattern for better code maintainability.
- To make sure that remote has already registered IRQ there is required App level IPC mechanism to notify master about it

## Fixed

- Fixed `env_wait_for_link_up` function to handle timeout in link state checks for baremetal and qnx environment, `RL_BLOCK` mode can be used to wait indefinitely.
- Fixed CERT-C INT31-C violation by adding compile-time check to ensure `RL_PLATFORM_HIGHEST_LINK_ID` remains within safe range for 16-bit casting in virtqueue ID creation.
- Fixed CERT-C INT30-C violations by adding protection against unsigned integer underflow in shared memory calculations, specifically in `shmem_length - (uint32_t)RL_VRING_OVERHEAD` and `shmem_length - 2U * shmem_config.vring_size` expressions.
- Fixed CERT INT31-C violation in `platform_interrupt_disable()` and similar functions by replacing unsafe cast from `uint32_t` to `int32_t` with a return of 0 constant.
- Fixed unsigned integer underflow in `rpmsg_lite_alloc_tx_buffer()` where subtracting header size from buffer size could wrap around if buffer was too small, potentially leading to incorrect buffer sizing.
- Fixed CERT-C INT31-C violation in `rpmsg_lite.c` where `size` parameter was cast from `uint32_t` to `uint16_t` without proper validation.
  - Applied consistent masking approach to both `size` and `flags` parameters: `(uint16_t)(value & 0xFFFFU)`.
  - This fix prevents potential data loss when `size` values exceed 65535.
- Fixed CERT INT31-C violation in `env_memset` functions by explicitly converting `int32_t` values to unsigned char using bit masking. This prevents potential data loss or misinterpretation when passing values outside the unsigned char range (0-255) to the standard `memset()` function.
- Fixed CERT-C INT31-C violations in RPMsg-Lite environment porting: Added validation checks for signed-to-unsigned integer conversions to prevent data loss and misinterpretation.
  - `rpmsg_env_freertos.c`: Added validation before converting `int32_t` to `UBaseType_t`.
  - `rpmsg_env_qnx.c`: Fixed format string and added validation before assigning to `mqstat` fields.
  - `rpmsg_env_threadx.c`: Added validation to prevent integer overflow and negative values.
  - `rpmsg_env_xos.c`: Added range checking before casting to `uint16_t`.
  - `rpmsg_env_zephyr.c`: Added validation before passing values to `k_msgq_init`.
- Fixed a CERT INT31-C compliance issue in `env_get_current_queue_size()` function where an unsigned queue count was cast to a signed `int32_t` without proper validation, which could lead to lost or misinterpreted data if queue size exceeded `INT32_MAX`.
- Fixed CERT INT31-C violation in `rpmsg_platform.c` where `memcmp()` return value (signed int) was compared with unsigned constant without proper type handling.

- Fixed CERT INT31-C violation in `rpmsg_platform.c` where casting from `uint32_t` to `uint16_t` could potentially result in data loss. Changed length variable type from `uint16_t` to `uint32_t` to properly handle memory address differences without truncation.
- Fixed potential integer overflow in `env_sleep_msec()` function in ThreadX environment implementation by rearranging calculation order in the sleep duration formula.
- Fixed CERT-C INT31-C violation in RPMsg-Lite where bitwise NOT operations on integer constants were performed in signed integer context before being cast to unsigned. This could potentially lead to misinterpreted data on `imx943` platform.
- Added `RL_MAX_BUFFER_COUNT` (32768U) and `RL_MAX_VRING_ALIGN` (65536U) limit to ensure alignment values cannot contribute to integer overflow
- Fixed CERT INT31-C violation in `vring_need_event()`, added cast to `uint16_t` for each operand.

#### v5.1.4 - 27-Mar-2025

##### Added

- Add KW43B43 porting layer

##### Changed

- Doxygen bump to version 1.9.6

#### v5.1.3 - 13-Jan-2025

##### Added

- Memory cache management of shared memory. Enable with `#define RL_USE_DCACHE (1)` in `rpmsg_config.h` in case of data cache is used.
- Cmake/Kconfig support added.
- Porting layers for `imx95`, `imxrt700`, `mcmxw71x`, `mcmxw72x`, `kw47b42` added.

#### v5.1.2 - 08-Jul-2024

##### Changed

- Zephyr-related changes.
- Minor Misra corrections.

#### v5.1.1 - 19-Jan-2024

##### Added

- Test suite provided.
- Zephyr support added.

### Changed

- Minor changes in platform and env. layers, minor test code updates.

### v5.1.0 - 02-Aug-2023

### Added

- RPMsg-Lite: Added aarch64 support.

### Changed

- RPMsg-Lite: Increased the queue size to (2 \* RL\_BUFFER\_COUNT) to cover zero copy cases.
- Code formatting using LLVM16.

### Fixed

- Resolved issues in ThreadX env. layer implementation.

### v5.0.0 - 19-Jan-2023

### Added

- Timeout parameter added to `rpmmsg_lite_wait_for_link_up` API function.

### Changed

- Improved debug check buffers implementation - instead of checking the pointer fits into shared memory check the presence in the VirtIO ring descriptors list.
- VRING\_SIZE is set based on number of used buffers now (as calculated in `vring_init`) - updated for all platforms that are not communicating to Linux rpmmsg counterpart.

### Fixed

- Fixed wrong RL\_VRING\_OVERHEAD macro comment in platform.h files
- Misra corrections.

### v4.0.0 - 20-Jun-2022

### Added

- Added support for custom shared memory arrangement per the RPMsg\_Lite instance.
- Introduced new `rpmmsg_lite_wait_for_link_up()` API function - this allows to avoid using busy loops in rtos environments, GitHub PR [#21](#).

### Changed

- Adjusted `rpmmsg_lite_is_link_up()` to return RL\_TRUE/RL\_FALSE.



**v3.2.0 - 17-Jan-2022****Added**

- Added support for i.MX8 MP multicore platform.

**Changed**

- Improved static allocations - allow OS-specific objects being allocated statically, GitHub PR [#14](#).
- Aligned rpmsg\_env\_xos.c and some platform layers to latest static allocation support.

**Fixed**

- Minor Misra and typo corrections, GitHub PR [#19](#), [#20](#).

**v3.1.2 - 16-Jul-2021****Added**

- Addressed MISRA 21.6 rule violation in rpmsg\_env.h (use SDK's PRINTF in MCUXpressoSDK examples, otherwise stdio printf is used).
- Added environment layers for XOS.
- Added support for i.MX RT500, i.MX RT1160 and i.MX RT1170 multicore platforms.

**Fixed**

- Fixed incorrect description of the rpmsg\_lite\_get\_endpoint\_from\_addr function.

**Changed**

- Updated RL\_BUFFER\_COUNT documentation (issue [#10](#)).
- Updated imxrt600\_hifi4 platform layer.

**v3.1.1 - 15-Jan-2021****Added**

- Introduced RL\_ALLOW\_CONSUMED\_BUFFERS\_NOTIFICATION config option to allow opposite side notification sending each time received buffers are consumed and put into the queue of available buffers.
- Added environment layers for Threadx.
- Added support for i.MX8QM multicore platform.

**Changed**

- Several MISRA C-2012 violations addressed.

**v3.1.0 - 22-Jul-2020**

### Added

- Added support for several new multicore platforms.

### Fixed

- MISRA C-2012 violations fixed (7.4).
- Fixed missing lock in `rpmsg_lite_rx_callback()` for QNX env.
- Correction of `rpmsg_lite_instance` structure members description.
- Address -Waddress-of-packed-member warnings in GCC9.

### Changed

- Clang update to v10.0.0, code re-formatted.

## v3.0.0 - 20-Dec-2019

### Added

- Added support for several new multicore platforms.

### Fixed

- MISRA C-2012 violations fixed, incl. data types consolidation.
- Code formatted.

## v2.2.0 - 20-Mar-2019

### Added

- Added configuration macro `RL_DEBUG_CHECK_BUFFERS`.
- Several MISRA violations fixed.
- Added environment layers for QNX and Zephyr.
- Allow environment context required for some environment (controlled by the `RL_USE_ENVIRONMENT_CONTEXT` configuration macro).
- Data types consolidation.

## v1.1.0 - 28-Apr-2017

### Added

- Supporting i.MX6SX and i.MX7D MPU platforms.
- Supporting LPC5411x MCU platform.
- Baremetal and FreeRTOS support.
- Support of copy and zero-copy transfer.
- Support of static API (without dynamic allocations).

## Multicore Manager

### MCUXpresso SDK : mcuxsdk-middleware-mcmgr (Multicore Manager)

**Overview** This repository is for MCUXpresso SDK Multicore Manager middleware delivery and it contains Multicore Manager component officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository [mcuxsdk](#) for the complete delivery of MCUXpresso SDK to be able to build and run Multicore Manager examples that are based on mcux-sdk-middleware-mcmgr component.

**Documentation** Overall details can be reviewed here: [MCUXpresso SDK Online Documentation](#)

Visit [Multicore Manager - Documentation](#) to review details on the contents in this sub-repo.

For Further API documentation, please look at [doxygen documentation](#)

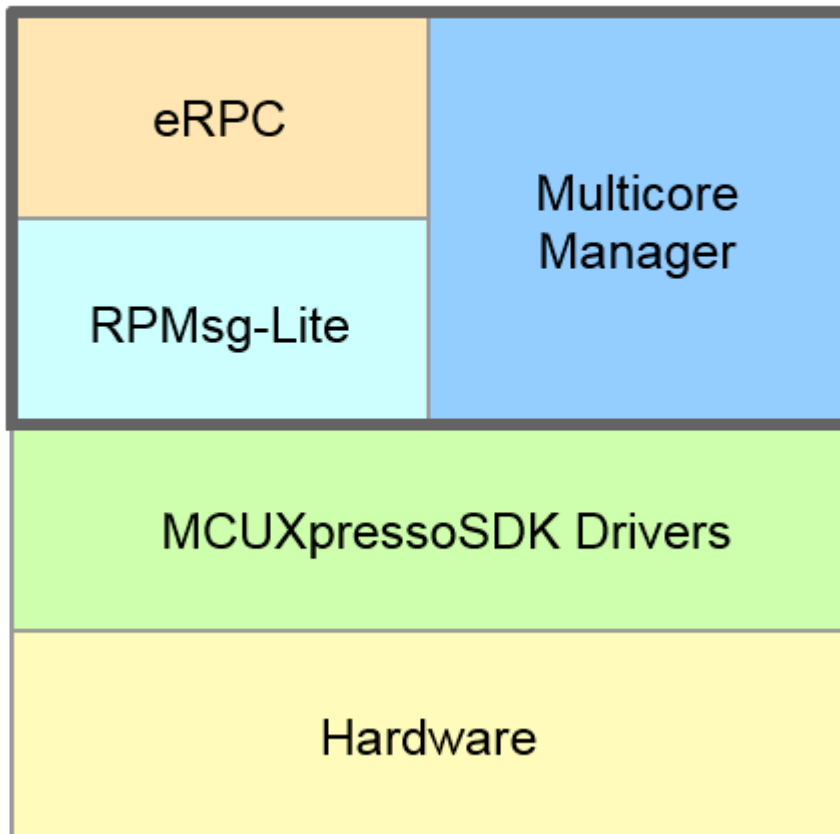
**Setup** Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest [Getting Started with SDK - Detailed Installation Instructions](#)

**Contribution** We welcome and encourage the community to submit patches directly to the mcmgr project placed on github. Contributing can be managed via pull-requests. Before a pull-request is created the code should be tested and properly formatted.

---

**Multicore Manager (MCMGR)** The Multicore Manager (MCMGR) software library provides a number of services for multicore systems. This library is distributed as a part of the Multicore SDK (MCSDK). Together, the MCSDK and the MCUXpresso SDK (SDK) form a framework for development of software for NXP multicore devices.

The MCMGR component is located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr` directory.



The Multicore Manager provides the following major functions:

- Maintains information about all cores in system.
- Secondary/auxiliary core(s) startup and shutdown.
- Remote core monitoring and event handling.

**Usage of the MCMGR software component** The main use case of MCMGR is the secondary/auxiliary core start. This functionality is performed by the public API function.

Example of MCMGR usage to start secondary core:

```
#include "mcmgr.h"

void main()
{
    /* Initialize MCMGR - low level multicore management library.
       Call this function as close to the reset entry as possible,
       (into the startup sequence) to allow CoreUp event triggering. */
    MCMGR_EarlyInit();

    /* Initialize MCMGR, install generic event handlers */
    MCMGR_Init();
}
```

(continues on next page)

(continued from previous page)

```

/* Boot secondary core application from the CORE1_BOOT_ADDRESS, pass "1" as startup data,
↳ starting synchronously. */
MCMGR_StartCore(kMCMGR_Core1, CORE1_BOOT_ADDRESS, 1, kMCMGR_Start_Synchronous);
.
.
.
/* Stop secondary core execution. */
MCMGR_StopCore(kMCMGR_Core1);
}

```

Some platforms allow stopping and re-starting the secondary core application again, using the MCMGR\_StopCore / MCMGR\_StartCore API calls. It is necessary to ensure the initially loaded image is not corrupted before re-starting, especially if it deals with the RAM target. Cache coherence has to be considered/ensured as well.

It could also happen that the secondary core application stops running correctly and the primary core application does not know about that situation. Therefore, it is beneficial to implement a mechanism for core health monitoring. The *test\_heartbeat* unit test can serve as an example how to ensure that: secondary core could periodically send heartbeat signals to the primary core using MCMGR\_TriggerEvent() API to indicate that it is alive and functioning properly.

Another important MCMGR feature is the ability for remote core monitoring and handling of events such as reset, exception, and application events. Application-specific callback functions for events are registered by the MCMGR\_RegisterEvent() API. Triggering these events is done using the MCMGR\_TriggerEvent() API. mcmgr\_event\_type\_t enums all possible event types.

An example of MCMGR usage for remote core monitoring and event handling. Code for the primary side:

```

#include "mcmgr.h"

#define APP_RPMSG_READY_EVENT_DATA (1)
#define APP_NUMBER_OF_CORES (2)
#define APP_SECONDARY_CORE kMCMGR_Core1

/* Callback function registered via the MCMGR_RegisterEvent() and triggered by MCMGR_TriggerEvent()
↳ called on the secondary core side */
void RPSMsgRemoteReadyEventHandler(mcmgr_core_t coreNum, uint16_t eventData, void *context)
{
    uint16_t *data = &((uint16_t *)context)[coreNum];

    *data = eventData;
}

void main()
{
    uint16_t RPSMsgRemoteReadyEventData[NUMBER_OF_CORES] = {0};

    /* Initialize MCMGR - low level multicore management library.
       Call this function as close to the reset entry as possible,
       (into the startup sequence) to allow CoreUp event triggering. */
    MCMGR_EarlyInit();

    /* Initialize MCMGR, install generic event handlers */
    MCMGR_Init();

    /* Register the application event before starting the secondary core */
    MCMGR_RegisterEvent(kMCMGR_RemoteApplicationEvent, RPSMsgRemoteReadyEventHandler, (void
↳ *)RPSMsgRemoteReadyEventData);

```

(continues on next page)

(continued from previous page)

```
/* Boot secondary core application from the CORE1_BOOT_ADDRESS, pass rpmsg_lite_base address,
↳as startup data, starting synchronously. */
MCMGR_StartCore(APP_SECONDARY_CORE, CORE1_BOOT_ADDRESS, (uint32_t)rpmsg_lite_
↳base, kMCMGR_Start_Synchronous);

/* Wait until the secondary core application signals the rpmsg remote has been initialized and is ready to
↳communicate. */
while(APP_RPMSG_READY_EVENT_DATA != RPSMsgRemoteReadyEventData[APP_SECONDARY_
↳CORE]) {};
.
.
.
}
```

Code for the secondary side:

```
#include "mcmgr.h"

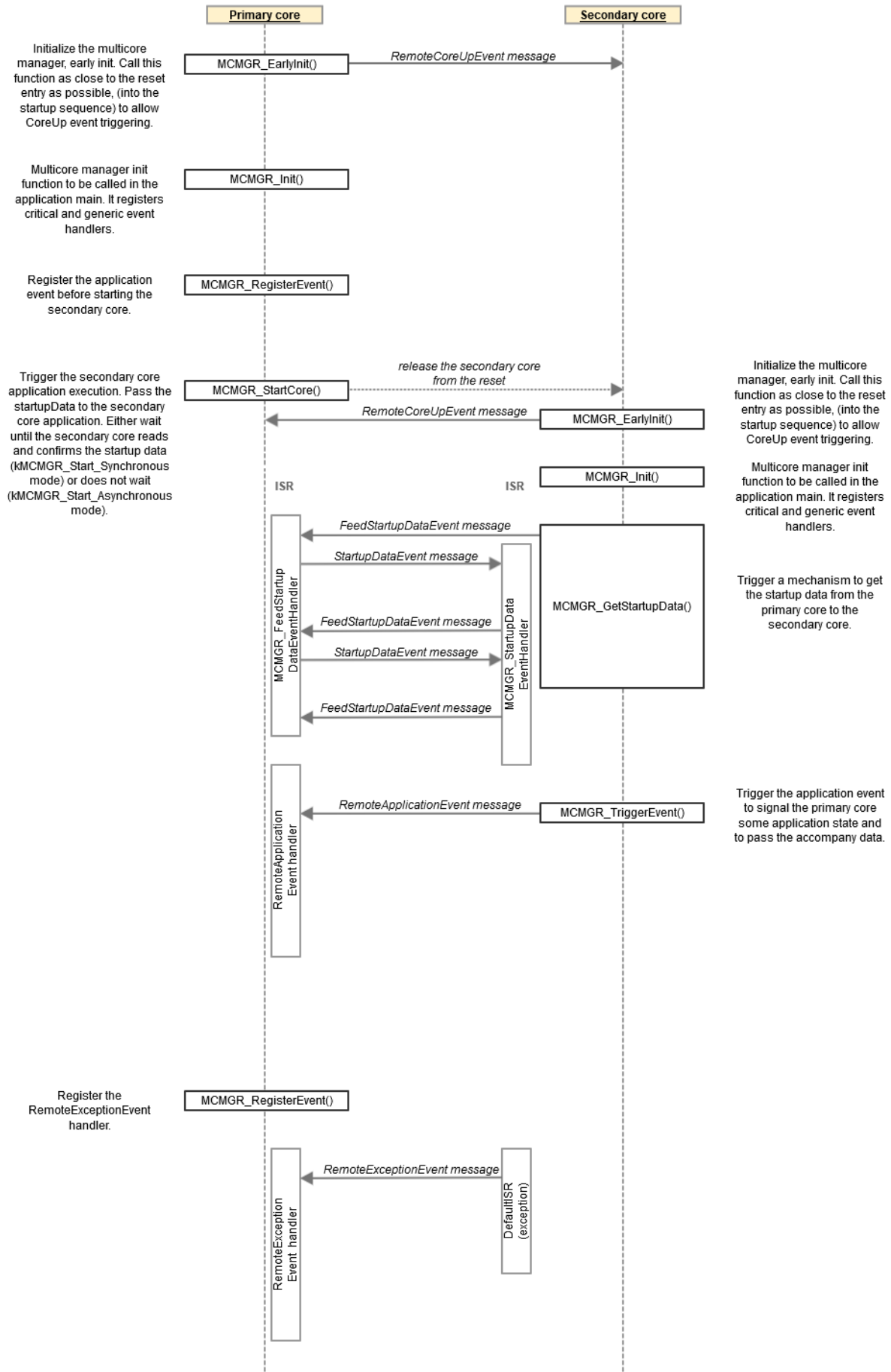
#define APP_RPMSG_READY_EVENT_DATA (1)

void main()
{
    /* Initialize MCMGR - low level multicore management library.
       Call this function as close to the reset entry as possible,
       (into the startup sequence) to allow CoreUp event triggering. */
    MCMGR_EarlyInit();

    /* Initialize MCMGR, install generic event handlers */
    MCMGR_Init();
    .
    .
    .

    /* Signal the to other core that we are ready by triggering the event and passing the APP_RPMSG_
    ↳READY_EVENT_DATA */
    MCMGR_TriggerEvent(kMCMGR_Core0, kMCMGR_RemoteApplicationEvent, APP_RPMSG_
    ↳READY_EVENT_DATA);
    .
    .
    .
}
```

**MCMGR Data Exchange Diagram** The following picture shows how the handshakes are supposed to work between the two cores in the MCMGR software.



**Changelog Multicore Manager** All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog](#), and this project adheres to [Semantic Versioning](#).

## [v5.0.2]

### Added

- Added gcov options and configs to support mcmgr code coverage
- Added new test\_weak\_mu\_isr testcase for devices with MU peripheral
- Added new test\_heartbeat testcase showing heartbeat mechanism between primary and secondary cores using the MCMGR

## v5.0.1

### Added

- Added frdmimxrt1186 unit testing

### Changed

- [KW43] Rename core#1 reset control register

### Fixed

- Added CX flag into CMakeLists.txt to allow c++ build compatibility.
- Fix path to mcmgr headers directory in doxyfile

## v5.0.0

### Added

- Added MCMGR\_BUSY\_POLL\_COUNT macro to prevent infinite polling loops in MCMGR operations.
- Implemented timeout mechanism for all polling loops in MCMGR code.
- Added support to handle more then two cores. Breaking API change by adding parameter coreNum specifying core number in functions bellow.
  - MCMGR\_GetStartupData(uint32\_t \*startupData, mcmgr\_core\_t coreNum)
  - MCMGR\_TriggerEvent(mcmgr\_event\_type\_t type, uint16\_t eventData, mcmgr\_core\_t coreNum)
  - MCMGR\_TriggerEventForce(mcmgr\_event\_type\_t type, uint16\_t eventData, mcmgr\_core\_t coreNum)
  - typedef void (\*mcmgr\_event\_callback\_t)(uint16\_t data, void \*context, mcmgr\_core\_t coreNum);



When registering the event with function `MCMGR_RegisterEvent()` user now needs to provide `callbackData` pointer to array of elements per every core in system (see README.md for example). In case of systems with only two cores the `coreNum` in callback can be ignored as events can arrive only from one core. Please see Porting guide for more details: Porting-GuideTo\_v5.md

- Updated all porting files to support new MCMGR API.
- Added new platform specific include file `mcmgr_platform.h`. It will contain common platform specific macros that can be then used in `mcmgr` and application. e.g. platform core count `MCMGR_CORECOUNT` 4.
- Move all header files to new `inc` directory.
- Added new platform-specific include files `inc/platform/<platform_name>/mcmgr_platform.h`.

#### Added

- Add MCXL20 porting layer and unit testing

#### v4.1.7

#### Fixed

- `mcmgr_stop_core_internal()` function now returns `kStatus_MCMGR_NotImplemented` status code instead of `kStatus_MCMGR_Success` when device does not support stop of secondary core. Ports affected: kw32w1, kw45b41, kw45b42, mcxw716, mcxw727.

#### [v4.1.6]

#### Added

- Multicore Manager moved to standalone repository.
- Add porting layers for imxrt700, mcmxw727, kw47b42.
- New `MCMGR_ProcessDeferredRxIsr()` API added.

#### [v4.1.5]

#### Added

- Add notification into `MCMGR_EarlyInit` and `mcmgr_early_init_internal` functions to avoid using uninitialized data in their implementations.

#### [v4.1.4]

#### Fixed

- Avoid calling tx isr callbacks when respective Messaging Unit Transmit Interrupt Enable flag is not set in the CR/TCR register.
- Messaging Unit RX and status registers are cleared after the initialization.

#### [v4.1.3]

##### Added

- Add porting layers for imxrt1180.

##### Fixed

- mu\_isr() updated to avoid calling tx isr callbacks when respective Transmit Interrupt Enable flag is not set in the CR/TCR register.
- mcmgr\_mu\_internal.c code adaptation to new supported SoCs.

#### [v4.1.2]

##### Fixed

- Update mcmgr\_stop\_core\_internal() implementations to set core state to kMCMGR\_ResetCoreState.

#### [v4.1.0]

##### Fixed

- Code adjustments to address MISRA C-2012 Rules

#### [v4.0.3]

##### Fixed

- Documentation updated to describe handshaking in a graphic form.
- Minor code adjustments based on static analysis tool findings

#### [v4.0.2]

##### Fixed

- Align porting layers to the updated MCUXpressoSDK feature files.

#### [v4.0.1]

##### Fixed

- Code formatting, removed unused code

#### [v4.0.0]

**Added**

- Add new MCMGR\_TriggerEventForce() API.

**[v3.0.0]****Removed**

- Removed MCMGR\_LoadApp(), MCMGR\_MapAddress() and MCMGR\_SignalReady()

**Modified**

- Modified MCMGR\_GetStartupData()

**Added**

- Added MCMGR\_EarlyInit(), MCMGR\_RegisterEvent() and MCMGR\_TriggerEvent()
- Added the ability for remote core monitoring and event handling

**[v2.0.1]****Fixed**

- Updated to be Misra compliant.

**[v2.0.0]****Added**

- Support for lpcxpresso54114 board.

**[v1.1.0]****Fixed**

- Ported to KSDK 2.0.0.

**[v1.0.0]****Added**

- Initial release.

**eRPC****MCUXpresso SDK : mcuxsdk-middleware-erpc**

**Overview** This repository is for MCUXpresso SDK eRPC middleware delivery and it contains eRPC component officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository [mcuxsdk](#) for the complete delivery of MCUXpresso SDK to be able to build and run eRPC examples that are based on mcux-sdk-middleware-erpc component.

**Documentation** Overall details can be reviewed here: [MCUXpresso SDK Online Documentation](#)

Visit [eRPC - Documentation](#) to review details on the contents in this sub-repo.

**Setup** Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest [Getting Started with SDK - Detailed Installation Instructions](#)

**Contribution** We welcome and encourage the community to submit patches directly to the eRPC project placed on github. Contributing can be managed via pull-requests. Before a pull-request is created the code should be tested and properly formatted.

---

### eRPC

- [MCUXpresso SDK : mcuxsdk-middleware-erpc](#)
  - [Overview](#)
  - [Documentation](#)
  - [Setup](#)
  - [Contribution](#)
- [eRPC](#)
  - [About](#)
  - [Releases](#)
    - \* [Edge releases](#)
  - [Documentation](#)
  - [Examples](#)
  - [References](#)
  - [Directories](#)
  - [Building and installing](#)
    - \* [Requirements](#)
      - [Windows](#)
      - [Mac OS X](#)
    - \* [Building](#)
      - [CMake and KConfig](#)
      - [Make](#)

*\* Installing for Python*

- *Known issues and limitations*
- *Code providing*

## About

eRPC (Embedded RPC) is an open source Remote Procedure Call (RPC) system for multichip embedded systems and heterogeneous multicore SoCs.

Unlike other modern RPC systems, such as the excellent [Apache Thrift](#), eRPC distinguishes itself by being designed for tightly coupled systems, using plain C for remote functions, and having a small code size (<5kB). It is not intended for high performance distributed systems over a network.

eRPC does not force upon you any particular API style. It allows you to export existing C functions, without having to change their prototypes. (There are limits, of course.) And although the internal infrastructure is written in C++, most users will be able to use only the simple C setup APIs shown in the examples below.

A code generator tool called `erpcgen` is included. It accepts input IDL files, having an `.erpc` extension, that have definitions of your data types and remote interfaces, and generates the shim code that handles serialization and invocation. `erpcgen` can generate either C/C++ or Python code.

Example `.erpc` file:

```
// Define a data type.
enum LEDName { kRed, kGreen, kBlue }

// An interface is a logical grouping of functions.
interface IO {
    // Simple function declaration with an empty reply.
    set_led(LEDName whichLed, bool onOrOff) -> void
}
```

Client side usage:

```
void example_client(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_client_t client_manager;

    /* Init eRPC client infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);
    message_buffer_factory = erpc_mbf_dynamic_init();
    client_manager = erpc_client_init(transport, message_buffer_factory);

    /* init eRPC client IO service */
    initIO_client(client_manager);

    // Now we can call the remote function to turn on the green LED.
    set_led(kGreen, true);

    /* deinit objects */
    deinitIO_client();
    erpc_client_deinit(client_manager);
    erpc_mbf_dynamic_deinit(message_buffer_factory);
}
```

(continues on next page)

(continued from previous page)

```

    erpc_transport_tcp_deinit(transport);
}

void example_client(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_client_t client_manager;

    /* Init eRPC client infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);
    message_buffer_factory = erpc_mbf_dynamic_init();
    client_manager = erpc_client_init(transport, message_buffer_factory);

    /* scope for client service */
    {
        /* init eRPC client IO service */
        IO_client client(client_manager);

        // Now we can call the remote function to turn on the green LED.
        client.set_led(kGreen, true);
    }

    /* deinit objects */
    erpc_client_deinit(client_manager);
    erpc_mbf_dynamic_deinit(message_buffer_factory);
    erpc_transport_tcp_deinit(transport);
}

```

**Server side usage:**

```

// Implement the remote function.
void set_led(LEDName whichLed, bool onOrOff) {
    // implementation goes here
}

void example_server(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_server_t server;
    erpc_service_t service = create_IO_service();

    /* Init eRPC server infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);
    message_buffer_factory = erpc_mbf_dynamic_init();
    server = erpc_server_init(transport, message_buffer_factory);

    /* add custom service implementation to the server */
    erpc_add_service_to_server(server, service);

    // Run the server.
    erpc_server_run();

    /* deinit objects */
    destroy_IO_service(service);
    erpc_server_deinit(server);
    erpc_mbf_dynamic_deinit(message_buffer_factory);
    erpc_transport_tcp_deinit(transport);
}

```

```

// Implement the remote function.
class IO : public IO_interface

```

(continues on next page)

(continued from previous page)

```

{
    /* eRPC call definition */
    void set_led(LEDName whichLed, bool onOrOff) override {
        // implementation goes here
    }
}

void example_server(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_server_t server;
    IO IOImpl;
    IO_service io(&IOImpl);

    /* Init eRPC server infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);
    message_buffer_factory = erpc_mbf_dynamic_init();
    server = erpc_server_init(transport, message_buffer_factory);

    /* add custom service implementation to the server */
    erpc_add_service_to_server(server, &io);

    /* poll for requests */
    erpc_status_t err = server.run();

    /* deinit objects */
    erpc_server_deinit(server);
    erpc_mbf_dynamic_deinit(message_buffer_factory);
    erpc_transport_tcp_deinit(transport);
}

```

A number of transports are supported, and new transport classes are easy to write.

Supported transports can be found in *erpc/erpc\_c/transport* folder. E.g:

- CMSIS UART
- NXP Kinetis SPI and DSPI
- POSIX and Windows serial port
- TCP/IP (mostly for testing)
- NXP RPMsg-Lite / RPMsg TTY
- SPIdev Linux
- USB CDC
- NXP Messaging Unit

eRPC is available with an unrestrictive BSD 3-clause license. See the [LICENSE file](#) for the full license text.

## Releases [eRPC releases](#)

**Edge releases** Edge releases can be found on [eRPC CircleCI](#) webpage. Choose build of interest, then platform target and choose ARTIFACTS tab. Here you can find binary application from chosen build.

**Documentation** Documentation is in the wiki section.

eRPC Infrastructure documentation

**Examples** Example IDL is available in the *examples/* folder.

Plenty of eRPC multicore and multiprocessor examples can be also found in NXP MCUXpressoSDK packages. Visit <https://mcuxpresso.nxp.com> to configure, build and download these packages.

To get the board list with multicore support (eRPC included) use filtering based on Middleware and search for 'multicore' string. Once the selected package with the multicore middleware is downloaded, see

<MCUXpressoSDK\_install\_dir>/boards/<board\_name>/multicore\_examples for eRPC multicore examples (RPMsg\_Lite or Messaging Unit transports used) or

<MCUXpressoSDK\_install\_dir>/boards/<board\_name>/multiprocessor\_examples for eRPC multiprocessor examples (UART or SPI transports used).

eRPC examples use the 'erpc\_' name prefix.

Another way of getting NXP MCUXpressoSDK eRPC multicore and multiprocessor examples is using the [mcux-sdk](#) Github repo. Follow the description how to use the West tool to clone and update the mcuxsdk repo in [readme Overview section](#). Once done the armgcc eRPC examples can be found in

mcuxsdk/examples/<board\_name>/multicore\_examples or in

mcuxsdk/examples/<board\_name>/multiprocessor\_examples folders.

You can use the evkmimxrt1170 as the board\_name for instance. Similar to MCUXpressoSDK packages the eRPC examples use the 'erpc\_' name prefix.

**References** This section provides links to interesting erpc-based projects, articles, blogs or guides:

- [erpc \(EmbeddedRPC\) getting started notes](#)
- [ERPC Linux Local Environment Construction and Use](#)
- [The New Wio Terminal eRPC Firmware](#)

**Directories** *doc* - Documentation.

*doxygen* - Configuration and support files for running Doxygen over the eRPC C++ infrastructure and erpcgen code.

*erpc\_c* - Holds C/C++ infrastructure for eRPC. This is the code you will include in your application.

*erpc\_python* - Holds Python version of the eRPC infrastructure.

*erpcgen* - Holds source code for erpcgen and makefiles or project files to build erpcgen on Windows, Linux, and OS X.

*erpcsniffer* - Holds source code for erpcsniffer application.

*examples* - Several example IDL files.

*mk* - Contains common makefiles for building eRPC components.

*test* - Client/server tests. These tests verify the entire communications path from client to server and back.

*utilities* - Holds utilities which bring additional benefit to eRPC apps developers.



**Building and installing** These build instructions apply to host PCs and embedded Linux. For bare metal or RTOS embedded environments, you should copy the *erpc\_c* directory into your application sources.

### CMake and KConfig build:

It builds a static library of the eRPC C/C++ infrastructure, the `erpcgen` executable, and optionally the unit tests and examples.

CMake is compatible with gcc and clang. On Windows local MingGW downloaded by *script* can be used.

## Make build:

It builds a static library of the eRPC C/C++ infrastructure, the `erpcgen` executable, and optionally the unit tests.

The makefiles are compatible with gcc or clang on Linux, OS X, and Cygwin. A Windows build of erpcgen using Visual Studio is also available in the *erpcgen/VisualStudio\_v14* directory. There is also an Xcode project file in the *erpcgen* directory, which can be used to build erpcgen for OS X.

**Requirements** erPC now support building **erpcgen**, **erpc\_lib**, **tests** and **C examples** using CMake.

### Requirements when using CMake:

- **CMake** (minimal version 3.20.0)
- Generator - **Make**, **Ninja**, ...
- **C/C++ compiler** - **GCC**, **CLANG**, ...
- **Binson** - <https://www.gnu.org/software/bison/>
- **Flex** - <https://github.com/westes/flex/>

### Requirements when using Make:

- **Make**
- **C/C++ compiler - GCC, CLANG, ...**
- **Binson** - <https://www.gnu.org/software/bison/>
- **Flex** - <https://github.com/westes/flex/>

**Windows** Related steps to build **erpcgen** using **Visual Studio** are described in `erpcgen/VisualStudio_v14/readme_erpcgen.txt`.

To install MinGW, Bison, Flex locally on Windows:

```
./install_dependencies.ps1
* ^^^
#### Linux
^^^ bash
./install_dependencies.sh
```

Mandatory for case, when build for different architecture is needed

- gcc-multilib, g++-multilib

## Mac OS X

```
./install_dependencies.sh
```

## Building

**CMake and KConfig** eRPC use CMake and KConfig to configure and build eRPC related targets. KConfig can be edited by *prj.conf* or *menuconfig* when building.

Generate project, config and build. In *erpc/* execute:

```
cmake -B ./build # in erpc/build generate cmake project
cmake --build ./build --target menuconfig # Build menuconfig and configure erpcgen, erpc_lib, tests and
↳examples
cmake --build ./build # Build all selected target from prj.conf/menuconfig
```

**\*\*CMake will use the system's default compilers and generator**

If you want to use Windows and locally installed MinGW, use *CMake preset* :

```
cmake --preset mingw64 # Generate project in ./build using mingw64's make and compilers
cmake --build ./build --target menuconfig # Build menuconfig and configure erpcgen, erpc_lib, tests and
↳examples
cmake --build ./build # Build all selected target from prj.conf/menuconfig
```

**Make** To build the library and erpcgen, run from the repo root directory:

```
make
```

To install the library, erpcgen, and include files, run:

```
make install
```

You may need to `sudo` the `make install`.

By default this will install into `/usr/local`. If you want to install elsewhere, set the `PREFIX` environment variable. Example for installing into `/opt`:

```
make install PREFIX=/opt
```

List of top level Makefile targets:

- `erpc`: build the `liberpc.a` static library
- `erpcgen`: build the `erpcgen` tool
- `erpcsniffer`: build the sniffer tool
- `test`: build the unit tests under the *test* directory
- `all`: build all of the above
- `install`: install `liberpc.a`, `erpcgen`, and include files

eRPC code is validated with respect to the C++ 11 standard.

**Installing for Python** To install the Python infrastructure for eRPC see instructions in the *erpc python readme*.

### Known issues and limitations

- Static allocations controlled by the `ERPC_ALLOCATION_POLICY` config macro are not fully supported yet, i.e. not all erpc objects can be allocated statically now. It deals with the ongoing process and the full static allocations support will be added in the future.

**Code providing** Repository on Github contains two main branches: **main** and **develop**. Code is developed on **develop** branch. Release version is created via merging **develop** branch into **main** branch.

---

Copyright 2014-2016 Freescale Semiconductor, Inc.

Copyright 2016-2025 NXP

### eRPC Getting Started

**Overview** This *Getting Started User Guide* shows software developers how to use Remote Procedure Calls (RPC) in embedded multicore microcontrollers (eRPC).

The eRPC documentation is located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/doc` folder.

**Create an eRPC application** This section describes a generic way to create a client/server eRPC application:

1. **Design the eRPC application:** Decide which data types are sent between applications, and define functions that send/receive this data.
2. **Create the IDL file:** The IDL file contains information about data types and functions used in an eRPC application, and is written in the IDL language.
3. **Use the eRPC generator tool:** This tool takes an IDL file and generates the shim code for the client and the server-side applications.
4. **Create an eRPC application:**
  1. Create two projects, where one project is for the client side (primary core) and the other project is for the server side (secondary core).
  2. Add generated files for the client application to the client project, and add generated files for the server application to the server project.
  3. Add infrastructure files.
  4. Add user code for client and server applications.
  5. Set the client and server project options.
5. **Run the eRPC application:** Run both the server and the client applications. Make sure that the server has been run before the client request was sent.

A specific example follows in the next section.

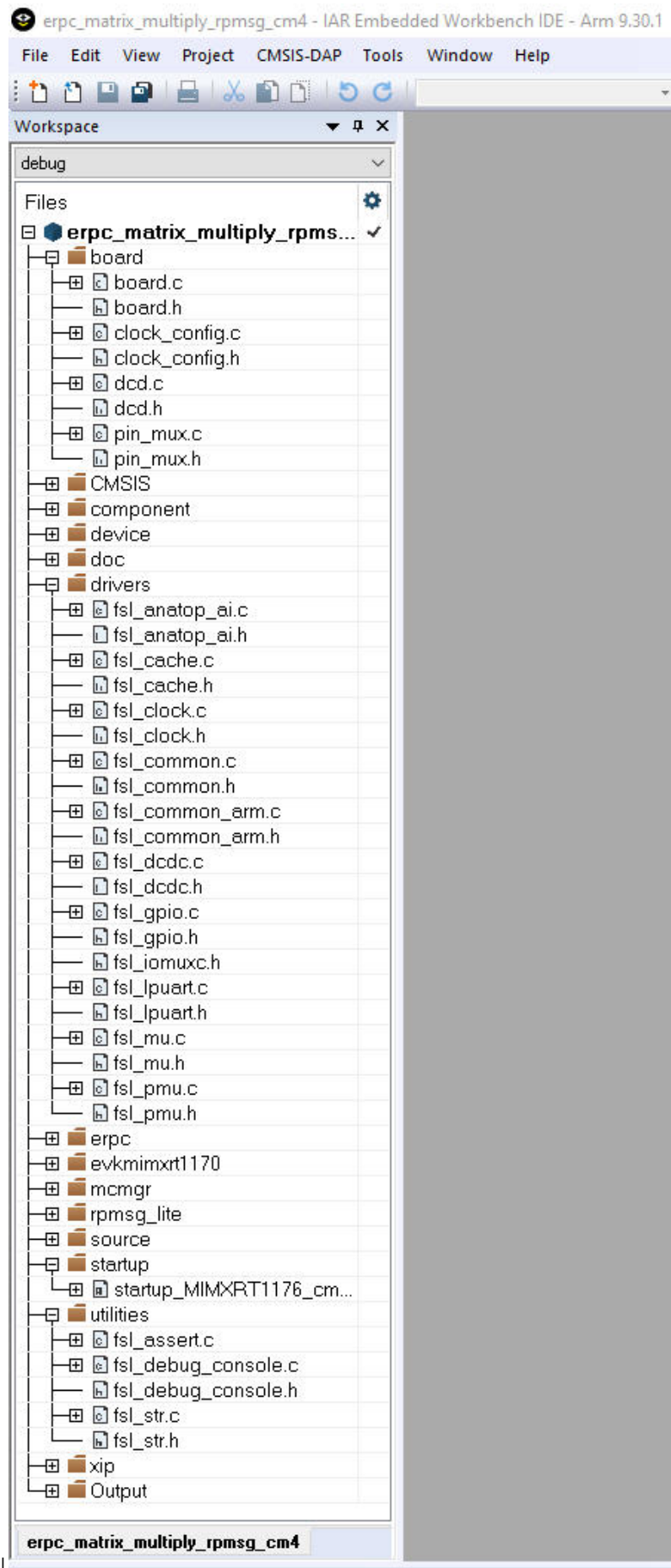
**Multicore server application** The “Matrix multiply” eRPC server project is located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4/iar`

The project files for the eRPC server have the `_cm4` suffix.

**Server project basic source files** The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in:

- `<MCUXpressoSDK_install_dir>/devices/<device>`
- `<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/`



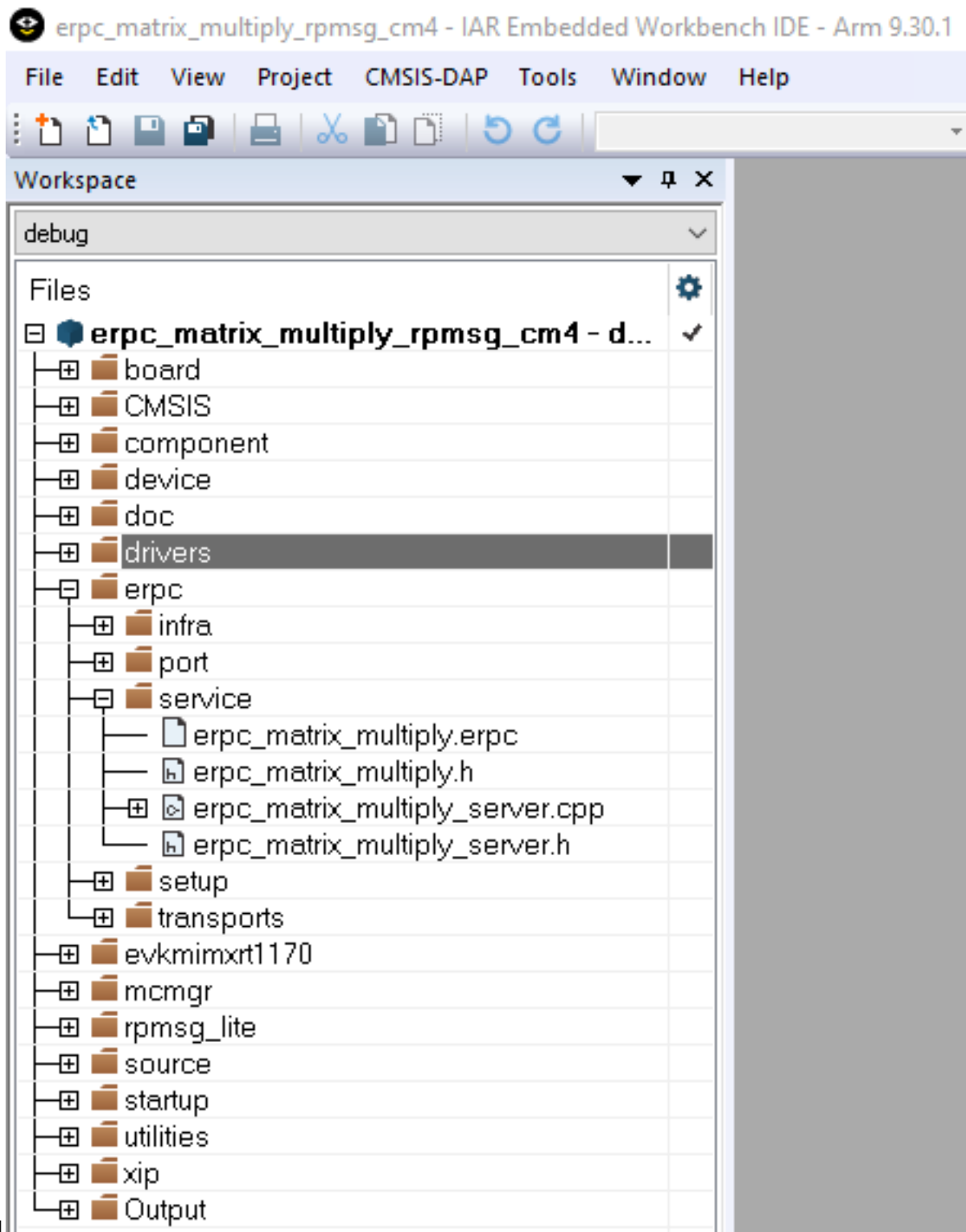
|

**Parent topic:** Multicore server application**Server related generated files** The server-related generated files are:

- erpc\_\_matric\_\_multiply.h
- erpc\_\_matrix\_\_multiply\_\_server.h
- erpc\_\_matrix\_\_multiply\_\_server.cpp

The server-related generated files contain the shim code for functions and data types declared in the IDL file. These files also contain functions for the identification of client requested functions, data deserialization, calling requested function's implementations, and data serialization and return, if requested by the client. These shim code files can be found in the following folder:

<MCUXpressoSDK\_install\_dir>/boards/evkmimxrt1170/multicore\_examples/erpc\_common/erpc\_matrix\_multiply/



**Parent topic:** Multicore server application

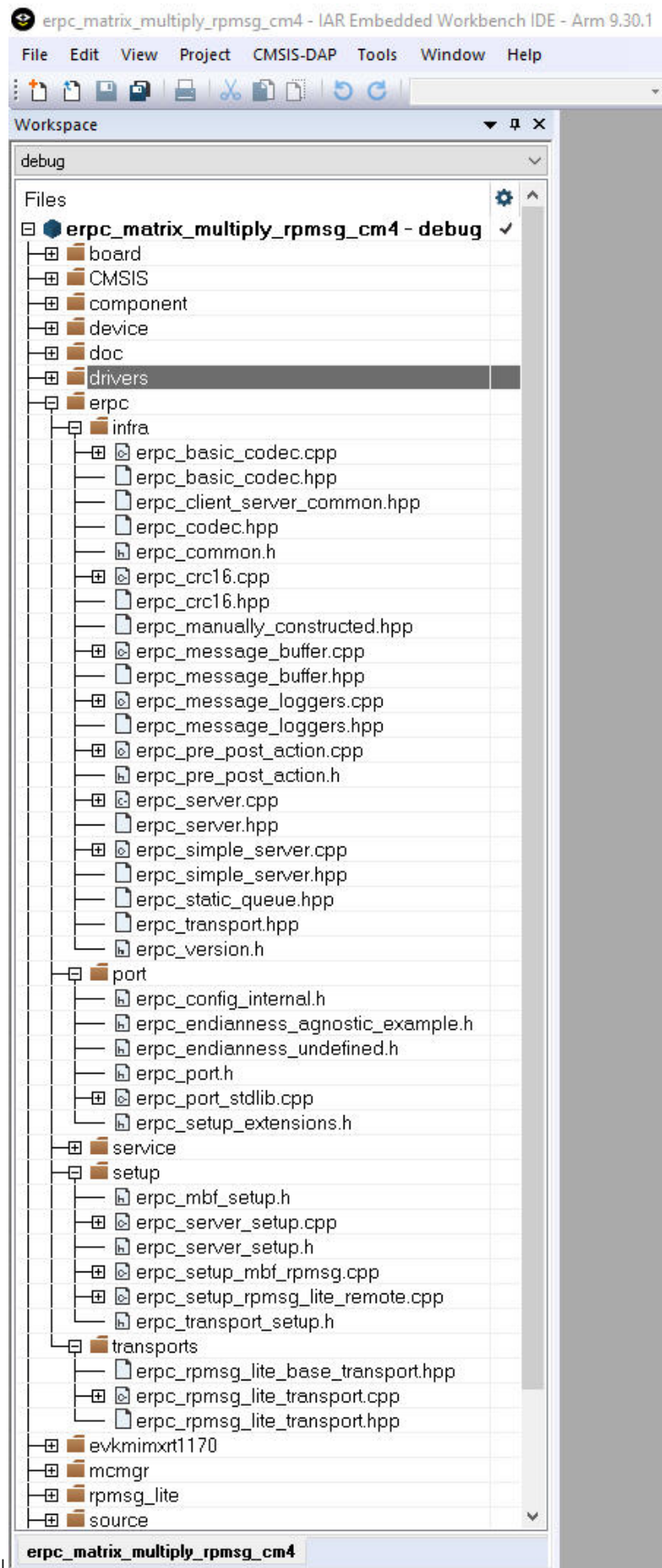
**Server infrastructure files** The eRPC infrastructure files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c`

The **erpc\_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.
  - Four files, `erpc_server.hpp`, `erpc_server.cpp`, `erpc_simple_server.hpp`, and `erpc_simple_server.cpp`, are used for running the eRPC server on the server-side applications. The simple server is currently the only implementation of the server, and its role is to catch client requests, identify and call requested functions, and send data back when requested.
  - Three files (`erpc_codec.hpp`, `erpc_basic_codec.hpp`, and `erpc_basic_codec.cpp`) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
  - The `erpc_common.hpp` file is used for common eRPC definitions, typedefs, and enums.
  - The `erpc_manually_constructed.hpp` file is used for allocating static storage for the used objects.
  - Message buffer files are used for storing serialized data: `erpc_message_buffer.h` and `erpc_message_buffer.cpp`.
  - The `erpc_transport.h` file defines the abstract interface for transport layer.
- The **port** subfolder contains the eRPC porting layer to adapt to different environments.
  - `erpc_port.h` file contains definition of `erpc_malloc()` and `erpc_free()` functions.
  - `erpc_port_stdlib.cpp` file ensures adaptation to `stdlib`.
  - `erpc_config_internal.h` internal erpc configuration file.
- The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.
  - The `erpc_server_setup.h` and `erpc_server_setup.cpp` files need to be added into the “Matrix multiply” example project to demonstrate the use of C-wrapped functions in this example.
  - The `erpc_transport_setup.h` and `erpc_setup_rpmsg_lite_remote.cpp` files need to be added into the project in order to allow the C-wrapped function for transport layer setup.
  - The `erpc_mbf_setup.h` and `erpc_setup_mbf_rpmsg.cpp` files need to be added into the project in order to allow message buffer factory usage.
- The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions in the setup folder.
  - RPMsg-Lite is used as the transport layer for the communication between cores, `erpc_rpmsg_lite_base_transport.hpp`, `erpc_rpmsg_lite_transport.hpp`, and `erpc_rpmsg_lite_transport.cpp` files need to be added into the server project.





|

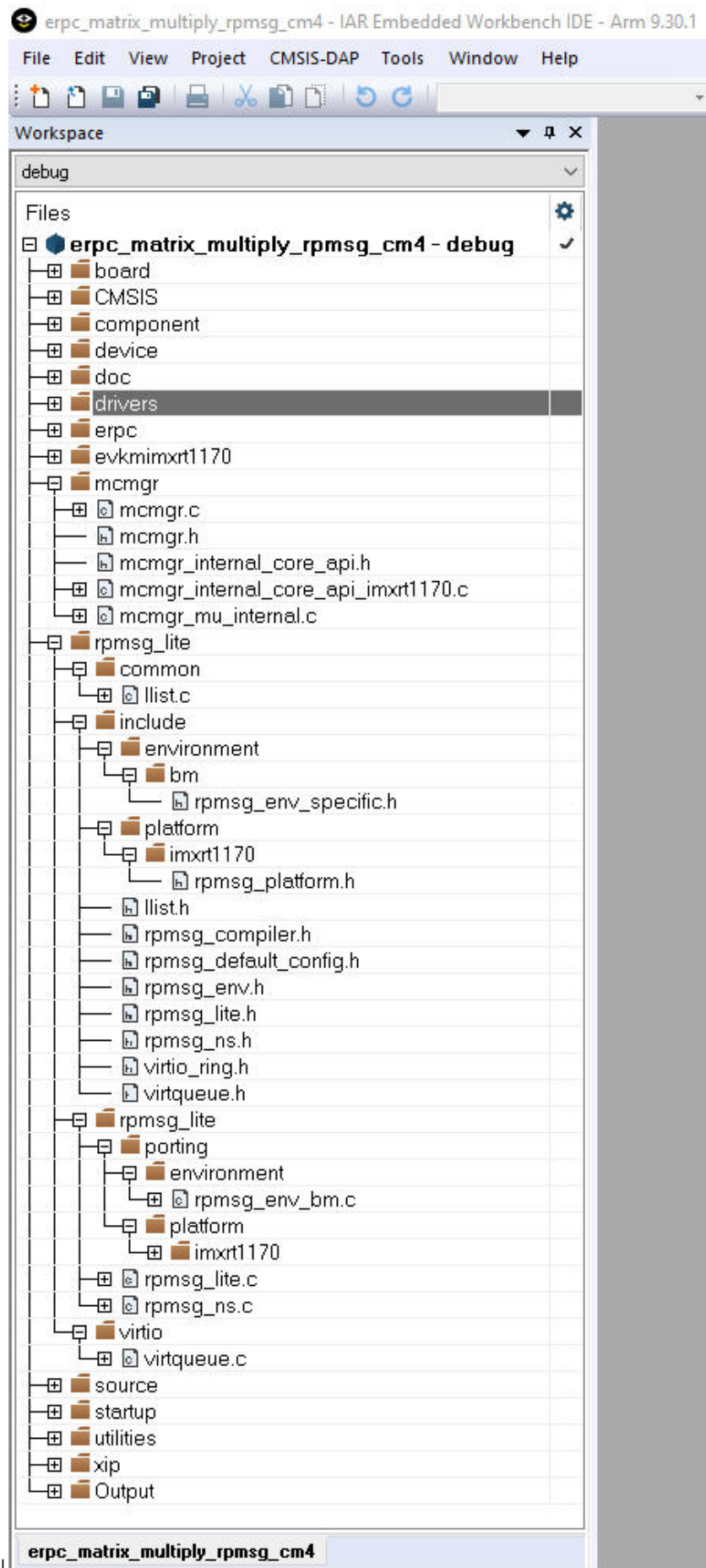
**Parent topic:** Multicore server application

**Server multicore infrastructure files** Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/`

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/`



|

**Parent topic:** Multicore server application

**Server user code** The server's user code is stored in the `main_core1.c` file, located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4`

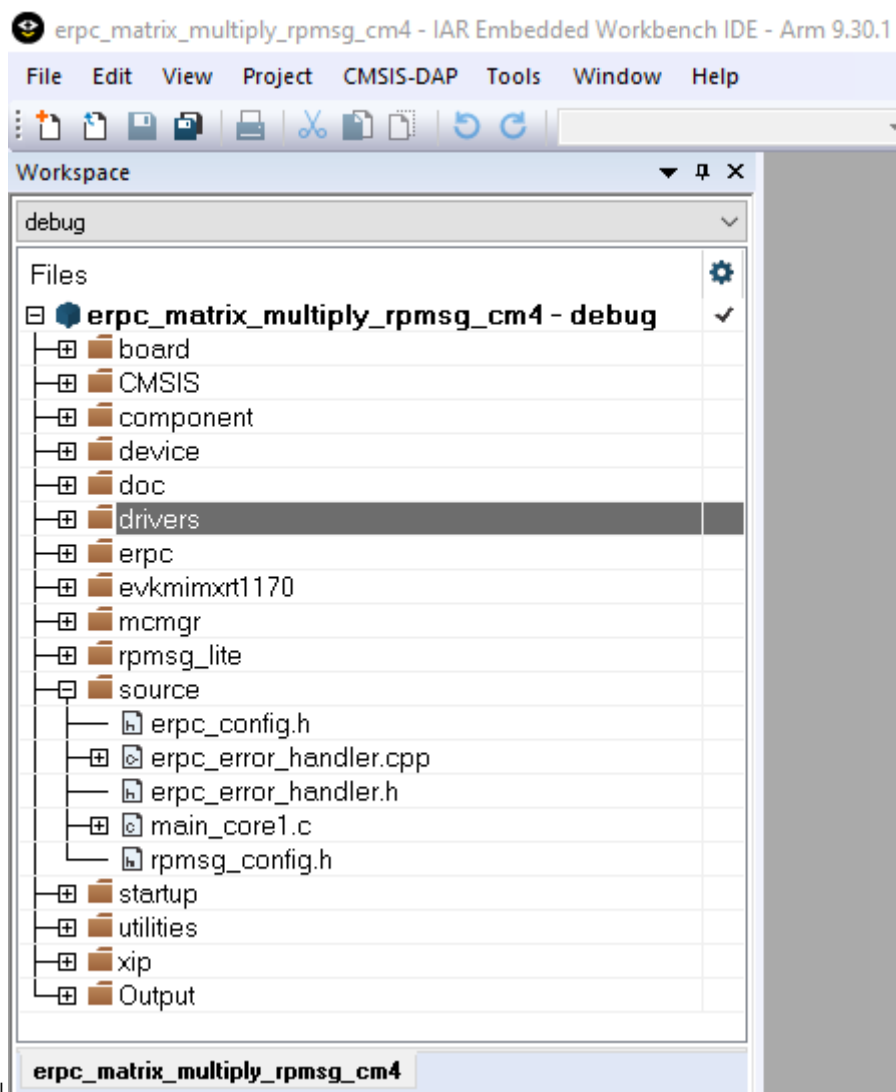
The `main_core1.c` file contains two functions:

- The **main()** function contains the code for the target board and eRPC server initialization. After the initialization, the matrix multiply service is added and the eRPC server waits for client's requests in the while loop.
- The **erpcMatrixMultiply()** function is the user implementation of the eRPC function defined in the IDL file.
- There is the possibility to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in the `erpc_error_handler.h` and `erpc_error_handler.cpp` files.

The eRPC-relevant code is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(const Matrix *matrix1, const Matrix *matrix2, Matrix *result_matrix)
{
    ...
}
int main()
{
    ...
    /* RPSMsg-Lite transport layer initialization */
    erpc_transport_t transport;
    transport = erpc_transport_rpmsg_lite_remote_init(src, dst, (void*)startupData,
    ERPC_TRANSPORT_RPMSG_LITE_LINK_ID, SignalReady, NULL);
    ...
    /* MessageBufferFactory initialization */
    erpc_mbf_t message_buffer_factory;
    message_buffer_factory = erpc_mbf_rpmsg_init(transport);
    ...
    /* eRPC server side initialization */
    erpc_server_t server;
    server = erpc_server_init(transport, message_buffer_factory);
    ...
    /* Adding the service to the server */
    erpc_service_t service = create_MatrixMultiplyService_service();
    erpc_add_service_to_server(server, service);
    ...
    while (1)
    {
        /* Process eRPC requests */
        erpc_status_t status = erpc_server_poll(server);
        /* handle error status */
        if (status != kErpcStatus_Success)
        {
            /* print error description */
            erpc_error_handler(status, 0);
            ...
        }
        ...
    }
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg\_config.h) and eRPC (erpc\_config.h), located in the `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg` folder.



**Parent topic:**Multicore server application

**Parent topic:**[Create an eRPC application](#)

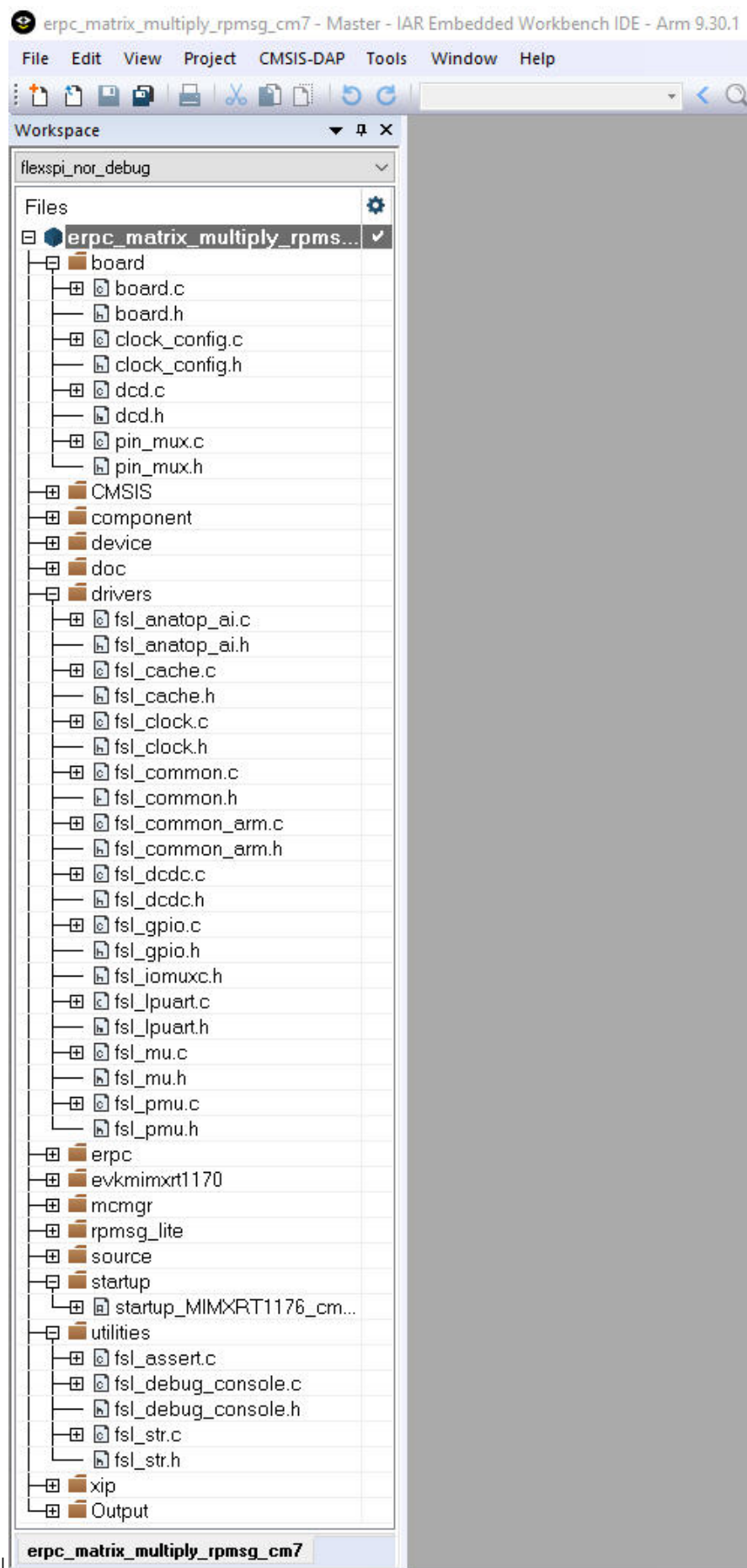
**Multicore client application** The “Matrix multiply” eRPC client project is located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm7/iar`

Project files for the eRPC client have the `_cm7` suffix.

**Client project basic source files** The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in the following folders:

- `<MCUXpressoSDK_install_dir>/devices/<device>`
- `<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/`



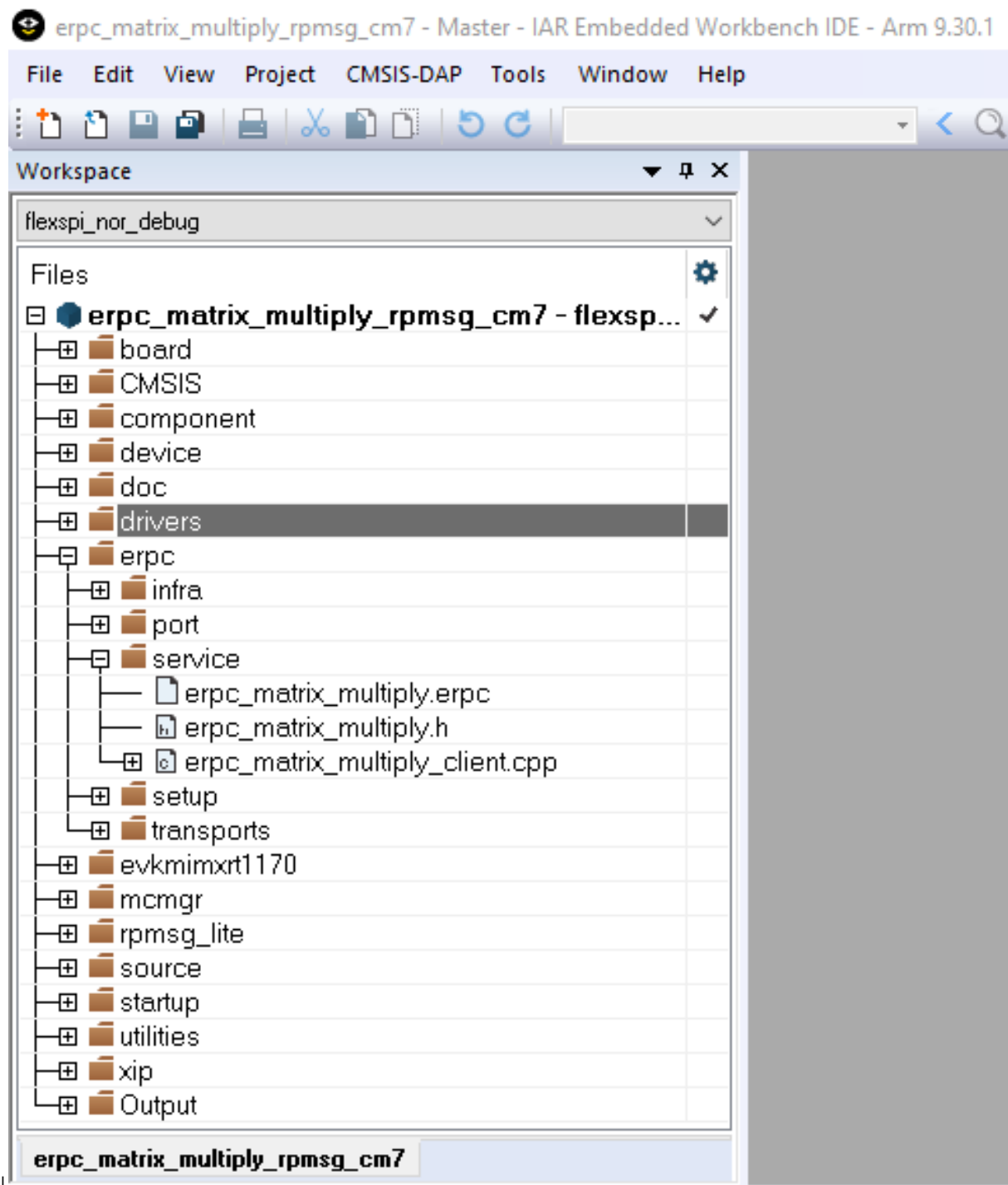
|

**Parent topic:**Multicore client application

**Client-related generated files** The client-related generated files are:

- `erpc__matric__multiply.h`
- `erpc__matrix__multiply__client.cpp`

These files contain the shim code for the functions and data types declared in the IDL file. These functions also call methods for codec initialization, data serialization, performing eRPC requests, and de-serializing outputs into expected data structures (if return values are expected). These shim code files can be found in the `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/` folder.



**Parent topic:** Multicore client application

**Client infrastructure files** The eRPC infrastructure files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c`

The **erpc\_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.



- Two files, `erpc_client_manager.h` and `erpc_client_manager.cpp`, are used for managing the client-side application. The main purpose of the client files is to create, perform, and release eRPC requests.
- Three files (`erpc_codec.hpp`, `erpc_basic_codec.hpp`, and `erpc_basic_codec.cpp`) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
- `erpc_common.h` file is used for common eRPC definitions, typedefs, and enums.
- `erpc_manually_constructed.hpp` file is used for allocating static storage for the used objects.
- Message buffer files are used for storing serialized data: `erpc_message_buffer.hpp` and `erpc_message_buffer.cpp`.
- `erpc_transport.hpp` file defines the abstract interface for transport layer.

The **port** subfolder contains the eRPC porting layer to adapt to different environments.

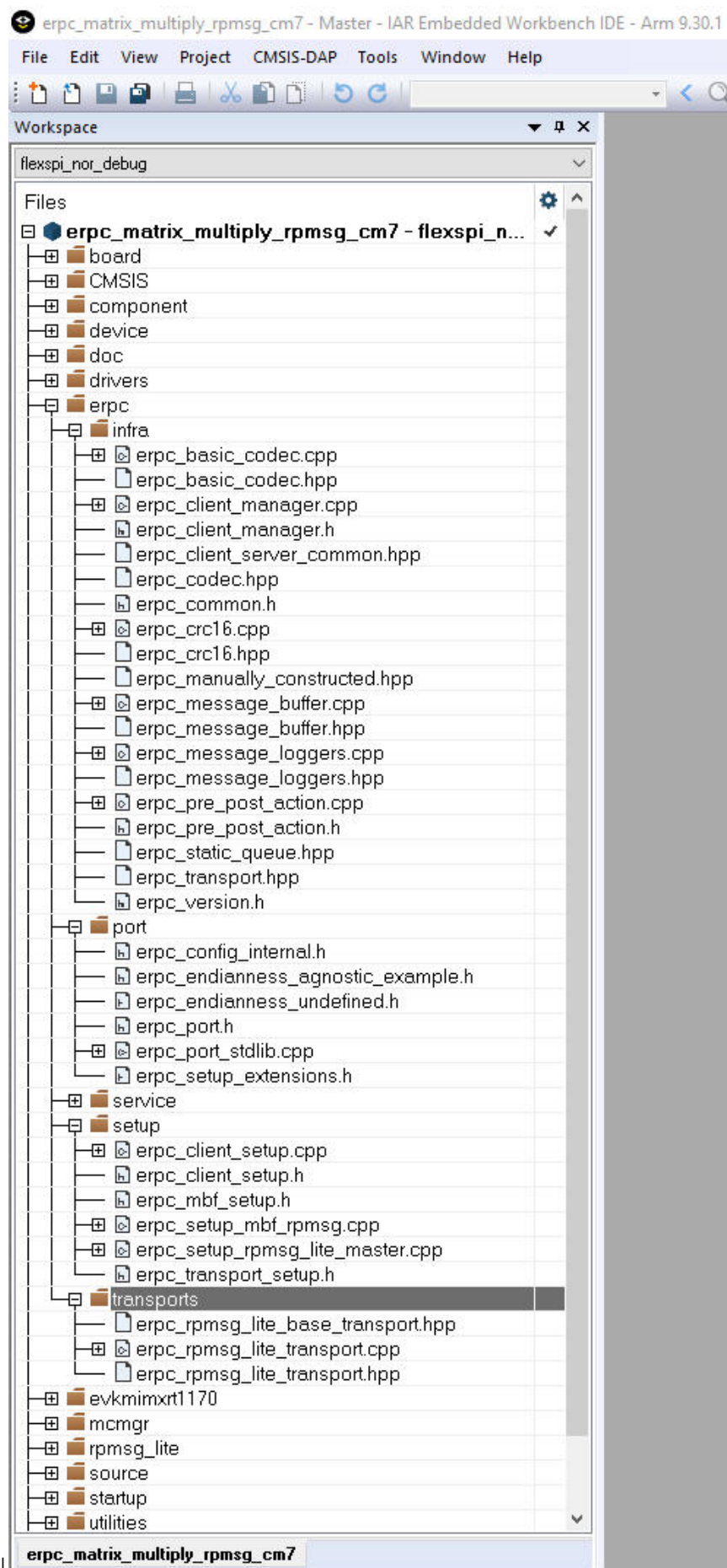
- `erpc_port.h` file contains definition of `erpc_malloc()` and `erpc_free()` functions.
- `erpc_port_stdlib.cpp` file ensures adaptation to `stdlib`.
- `erpc_config_internal.h` internal eRPC configuration file.

The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.

- `erpc_client_setup.h` and `erpc_client_setup.cpp` files needs to be added into the “Matrix multiply” example project to demonstrate the use of C-wrapped functions in this example.
- `erpc_transport_setup.h` and `erpc_setup_rpmsg_lite_master.cpp` files needs to be added into the project in order to allow C-wrapped function for transport layer setup.
- `erpc_mbf_setup.h` and `erpc_setup_mbf_rpmsg.cpp` files needs to be added into the project in order to allow message buffer factory usage.

The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions, in the setup folder.

- RPMsg-Lite is used as the transport layer for the communication between cores, `erpc_rpmsg_lite_base_transport.hpp`, `erpc_rpmsg_lite_transport.hpp`, and `erpc_rpmsg_lite_transport.cpp` files needs to be added into the client project.



|

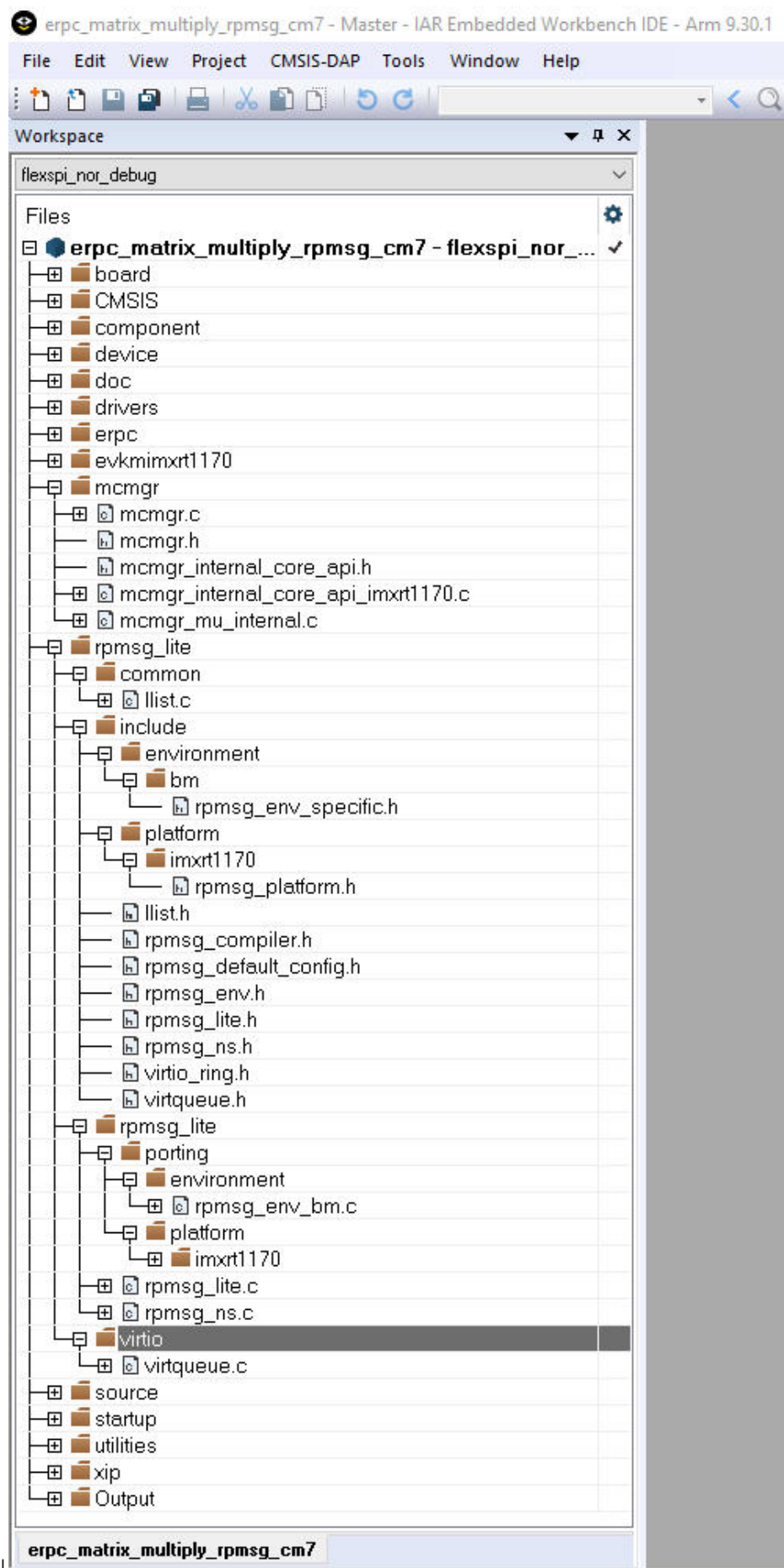
**Parent topic:**Multicore client application

**Client multicore infrastructure files** Because of the RPSMsg-Lite (transport layer), it is also necessary to include RPSMsg-Lite related files, which are in the following folder:

*<MCUXpressoSDK\_install\_dir>/middleware/multicore/rpsmsg\_lite/*

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

*<MCUXpressoSDK\_install\_dir>/middleware/multicore/mcmgr/*



**Parent topic:**Multicore client application

**Client user code** The client's user code is stored in the main\_core0.c file, located in the following folder:

<MCUXpressoSDK\_install\_dir>/boards/evkmimxrt1170/multicore\_example/erpc\_matrix\_multiply\_rpmsg/cm7

The main\_core0.c file contains the code for target board and eRPC initialization.

- After initialization, the secondary core is released from reset.
- When the secondary core is ready, the primary core initializes two matrix variables.
- The erpcMatrixMultiply eRPC function is called to issue the eRPC request and get the result.

It is possible to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in erpc\_error\_handler.h and erpc\_error\_handler.cpp files.

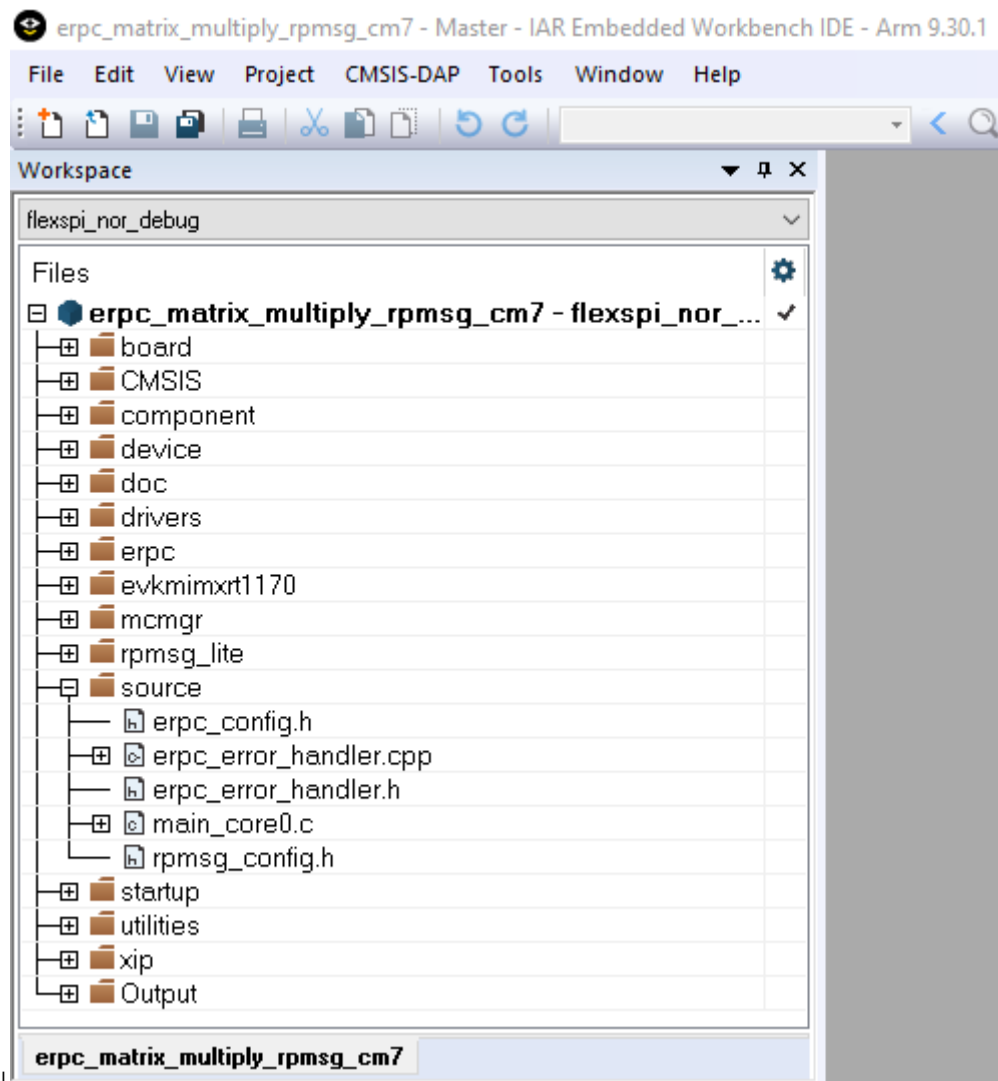
The matrix multiplication can be issued repeatedly, when pressing a software board button.

The eRPC-relevant code is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* RPMsg-Lite transport layer initialization */
erpc_transport_t transport;
transport = erpc_transport_rpmsg_lite_master_init(src, dst,
ERPC_TRANSPORT_RPMSG_LITE_LINK_ID);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_rpmsg_init(transport);
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport, message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
    /* Invoke the erpcMatrixMultiply function */
    erpcMatrixMultiply(matrix1, matrix2, result_matrix);
    ...
    /* Check if some error occurred in eRPC */
    if (g_erpc_error_occurred)
    {
        /* Exit program loop */
        break;
    }
    ...
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg\_config.h) and eRPC (erpc\_config.h), located in the following folder:

<MCUXpressoSDK\_install\_dir>/boards/evkmimxrt1170/multicore\_examples/erpc\_matrix\_multiply\_rpmsg



Parent topic: Multicore client application

Parent topic: [Create an eRPC application](#)

**Multiprocessor server application** The “Matrix multiply” eRPC server project for multiprocessor applications is located in the `<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>` folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires server-related generated files (server shim code), server infrastructure files, and the server user code. There is no need for server multicore infrastructure files (MCMGR and RPMs-Lite). The RPMs-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|      |                                                                         |
|------|-------------------------------------------------------------------------|
| SPI  | <eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_slave.cpp          |
|      | <eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.hpp |
|      | <eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.cpp |
| UART | <eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp            |

<eRPC base directory>/erpc\_c/transport/erpc\_uart\_cmsis\_transport.hpp

<eRPC base directory>/erpc\_c/transport/erpc\_uart\_cmsis\_transport.cpp

|

**Server user code** The server's user code is stored in the main\_server.c file, located in the <MCUXpressoSDK\_install\_dir>/boards/<board\_name>/multiprocessor\_examples/erpc\_server\_matrix\_multiply\_<transport\_layer>/ folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(Matrix matrix1, Matrix matrix2, Matrix result_matrix)
{
    ...
}
int main()
{
    ...
    /* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver
    ↪operations */
    erpc_transport_t transport;
    transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
    ...
    /* MessageBufferFactory initialization */
    erpc_mbf_t message_buffer_factory;
    message_buffer_factory = erpc_mbf_dynamic_init();
    ...
    /* eRPC server side initialization */
    erpc_server_t server;
    server = erpc_server_init(transport, message_buffer_factory);
    ...
    /* Adding the service to the server */
    erpc_service_t service = create_MatrixMultiplyService_service();
    erpc_add_service_to_server(server, service);
    ...
    while (1)
    {
        /* Process eRPC requests */
        erpc_status_t status = erpc_server_poll(server)
        /* handle error status */
        if (status != kErpcStatus_Success)
        {
            /* print error description */
            erpc_error_handler(status, 0);
            ...
        }
        ...
    }
}
```

**Parent topic:**Multiprocessor server application

**Multiprocessor client application** The “Matrix multiply” eRPC client project for multiprocessor applications is located in the <MCUXpressoSDK\_install\_dir>/boards/<board\_name>/multiprocessor\_examples/erpc\_client\_matrix\_multiply\_<transport\_layer>/iar/ folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires client-related generated files (server shim code),

client infrastructure files, and the client user code. There is no need for client multicore infrastructure files (MCMGR and RPSMsg-Lite). The RPSMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|                               |                                                                 |
|-------------------------------|-----------------------------------------------------------------|
| SPI                           | <eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_master.cpp |
| <eRPC base directory>/erpc_c/ | transports/ erpc_(d)spi_master_transport.hpp                    |
| <eRPC base directory>/erpc_c/ | transports/ erpc_(d)spi_master_transport.cpp                    |
| UART                          | <eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp    |
| <eRPC base directory>/erpc_c/ | transports/erpc_uart_cmsis_transport.hpp                        |
| <eRPC base directory>/erpc_c/ | transports/erpc_uart_cmsis_transport.cpp                        |
|                               |                                                                 |

**Client user code** The client's user code is stored in the `main_client.c` file, located in the `<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<transport_layer>/` folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver
↳operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_dynamic_init();
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport,message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
    /* Invoke the erpcMatrixMultiply function */
    erpcMatrixMultiply(matrix1, matrix2, result_matrix);
    ...
    /* Check if some error occurred in eRPC */
    if (g_erpc_error_occurred)
    {
        /* Exit program loop */
        break;
    }
    ...
}
```

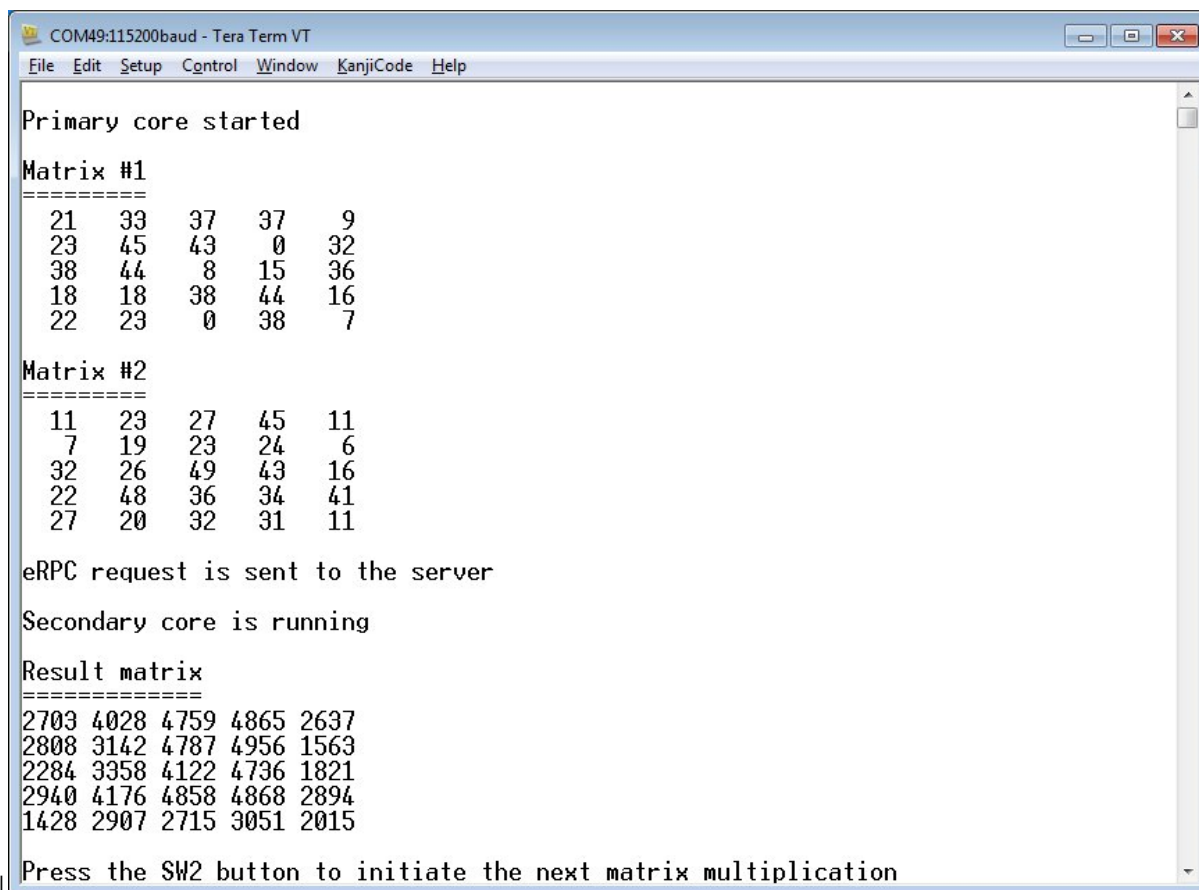
**Parent topic:**Multiprocessor client application

**Parent topic:**Multiprocessor server application



Parent topic:[Create an eRPC application](#)

**Running the eRPC application** Follow the instructions in *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) (located in the <MCUXpressoSDK\_install\_dir>/docs folder), to load both the primary and the secondary core images into the on-chip memory, and then effectively debug the dual-core application. After the application is running, the serial console should look like:



```

COM49:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help

Primary core started

Matrix #1
=====
 21  33  37  37   9
 23  45  43   0  32
 38  44   8  15  36
 18  18  38  44  16
 22  23   0  38   7

Matrix #2
=====
 11  23  27  45  11
  7  19  23  24   6
 32  26  49  43  16
 22  48  36  34  41
 27  20  32  31  11

eRPC request is sent to the server

Secondary core is running

Result matrix
=====
2703 4028 4759 4865 2637
2808 3142 4787 4956 1563
2284 3358 4122 4736 1821
2940 4176 4858 4868 2894
1428 2907 2715 3051 2015

Press the SW2 button to initiate the next matrix multiplication

```

For multiprocessor applications that are running between PC and the target evaluation board or between two boards, follow the instructions in the accompanied example readme files that provide details about the proper board setup and the PC side setup (Python).

Parent topic:[Create an eRPC application](#)

Parent topic:[eRPC example](#)

**eRPC example** This section shows how to create an example eRPC application called “Matrix multiply”, which implements one eRPC function (matrix multiply) with two function parameters (two matrices). The client-side application calls this eRPC function, and the server side performs the multiplication of received matrices. The server side then returns the result.

For example, use the NXP MIMXRT1170-EVK board as the target dual-core platform, and the IAR Embedded Workbench for ARM (EWARM) as the target IDE for developing the eRPC example.

- The primary core (CM7) runs the eRPC client.
- The secondary core (CM4) runs the eRPC server.
- RPMsg-Lite (Remote Processor Messaging Lite) is used as the eRPC transport layer.

The “Matrix multiply” application can be also run in the multi-processor setup. In other words, the eRPC client running on one SoC communicates with the eRPC server that runs on another SoC, utilizing different transport channels. It is possible to run the board-to-PC example (PC as the eRPC server and a board as the eRPC client, and vice versa) and also the board-to-board example. These multiprocessor examples are prepared for selected boards only.

| Multicore application source and project files | `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore/`  
| Multiprocessor application source and project files | `<MCUXpressoSDK_install_dir>/boards/<board_name>/multi`  
`<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<tr`  
| | eRPC source files | `<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/` | | RPLite  
source files | `<MCUXpressoSDK_install_dir>/middleware/multicore/rplite/`

**Designing the eRPC application** The matrix multiply application is based on calling single eRPC function that takes 2 two-dimensional arrays as input and returns matrix multiplication results as another 2 two-dimensional array. The IDL file syntax supports arrays with the dimension length set by the number only (in the current eRPC implementation). Because of this, a variable is declared in the IDL dedicated to store information about matrix dimension length, and to allow easy maintenance of the user and server code.

For a simple use of the two-dimensional array, the alias name (new type definition) for this data type has been declared in the IDL. Declaring this alias name ensures that the same data type can be used across the client and server applications.

**Parent topic:** [eRPC example](#)

**Creating the IDL file** The created IDL file is located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/`

The created IDL file contains the following code:

```
program erpc_matrix_multiply
/*! This const defines the matrix size. The value has to be the same as the
Matrix array dimension. Do not forget to re-generate the erpc code once the
matrix size is changed in the erpc file */
const int32 matrix_size = 5;
/*! This is the matrix array type. The dimension has to be the same as the
matrix size const. Do not forget to re-generate the erpc code once the
matrix size is changed in the erpc file */
type Matrix = int32[matrix_size][matrix_size];
interface MatrixMultiplyService {
erpcMatrixMultiply(in Matrix matrix1, in Matrix matrix2, out Matrix result_matrix) ->
void
}
```

Details:

- The IDL file starts with the program name (*erpc\_matrix\_multiply*), and this program name is used in the naming of all generated outputs.
- The declaration and definition of the constant variable named *matrix\_size* follows next. The *matrix\_size* variable is used for passing information about the length of matrix dimensions to the client/server user code.
- The alias name for the two-dimensional array type (*Matrix*) is declared.
- The interface group *MatrixMultiplyService* is located at the end of the IDL file. This interface group contains only one function declaration *erpcMatrixMultiply*.
- As shown above, the function’s declaration contains three parameters of *Matrix* type: *matrix1* and *matrix2* are input parameters, while *result\_matrix* is the output parameter. Additionally, the returned data type is declared as *void*.

When writing the IDL file, the following order of items is recommended:

1. Program name at the top of the IDL file.
2. New data types and constants declarations.
3. Declarations of interfaces and functions at the end of the IDL file.

**Parent topic:** [eRPC example](#)

**Using the eRPC generator tool** | Windows OS | `<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x64`  
 | Linux OS | `<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x86`  
 | Mac OS | `<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Mac`

The files for the “Matrix multiply” example are pre-generated and already a part of the application projects. The following section describes how they have been created.

- The easiest way to create the shim code is to copy the erpcgen application to the same folder where the IDL file (\*.erpc) is located; then run the following command:

```
erpcgen <IDL_file>.erpc
```

- In the “Matrix multiply” example, the command should look like:

```
erpcgen erpc_matrix_multiply.erpc
```

Additionally, another method to create the shim code is to execute the eRPC application using input commands:

- “-?”/”—help” – Shows supported commands.
- “-o <filePath>”/”—output<filePath>” – Sets the output directory.

For example,

```
<path_to_erpcgen>/erpcgen -o <path_to_output>  
<path_to_IDL>/<IDL_file_name>.erpc
```

For the “Matrix multiply” example, when the command is executed from the default erpcgen location, it looks like:

```
erpcgen -o  
../../../../../boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service  
../../../../../boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/erpc_matrix_mu
```

In both cases, the following four files are generated into the `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service` folder:

- erpc\_matrix\_multiply.h
- erpc\_matrix\_multiply\_client.cpp
- erpc\_matrix\_multiply\_server.h
- erpc\_matrix\_multiply\_server.cpp

For multiprocessor examples, the eRPC file and pre-generated files can be found in the `<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_common/erpc_matrix_multiply/service` folder.

**For Linux OS users:**

- Do not forget to set the permissions for the eRPC generator application.
- Run the application as `./erpcgen...` instead of as `erpcgen ....`

Parent topic: [eRPC example](#)

**Create an eRPC application** This section describes a generic way to create a client/server eRPC application:

1. **Design the eRPC application:** Decide which data types are sent between applications, and define functions that send/receive this data.
2. **Create the IDL file:** The IDL file contains information about data types and functions used in an eRPC application, and is written in the IDL language.
3. **Use the eRPC generator tool:** This tool takes an IDL file and generates the shim code for the client and the server-side applications.
4. **Create an eRPC application:**
  1. Create two projects, where one project is for the client side (primary core) and the other project is for the server side (secondary core).
  2. Add generated files for the client application to the client project, and add generated files for the server application to the server project.
  3. Add infrastructure files.
  4. Add user code for client and server applications.
  5. Set the client and server project options.
5. **Run the eRPC application:** Run both the server and the client applications. Make sure that the server has been run before the client request was sent.

A specific example follows in the next section.

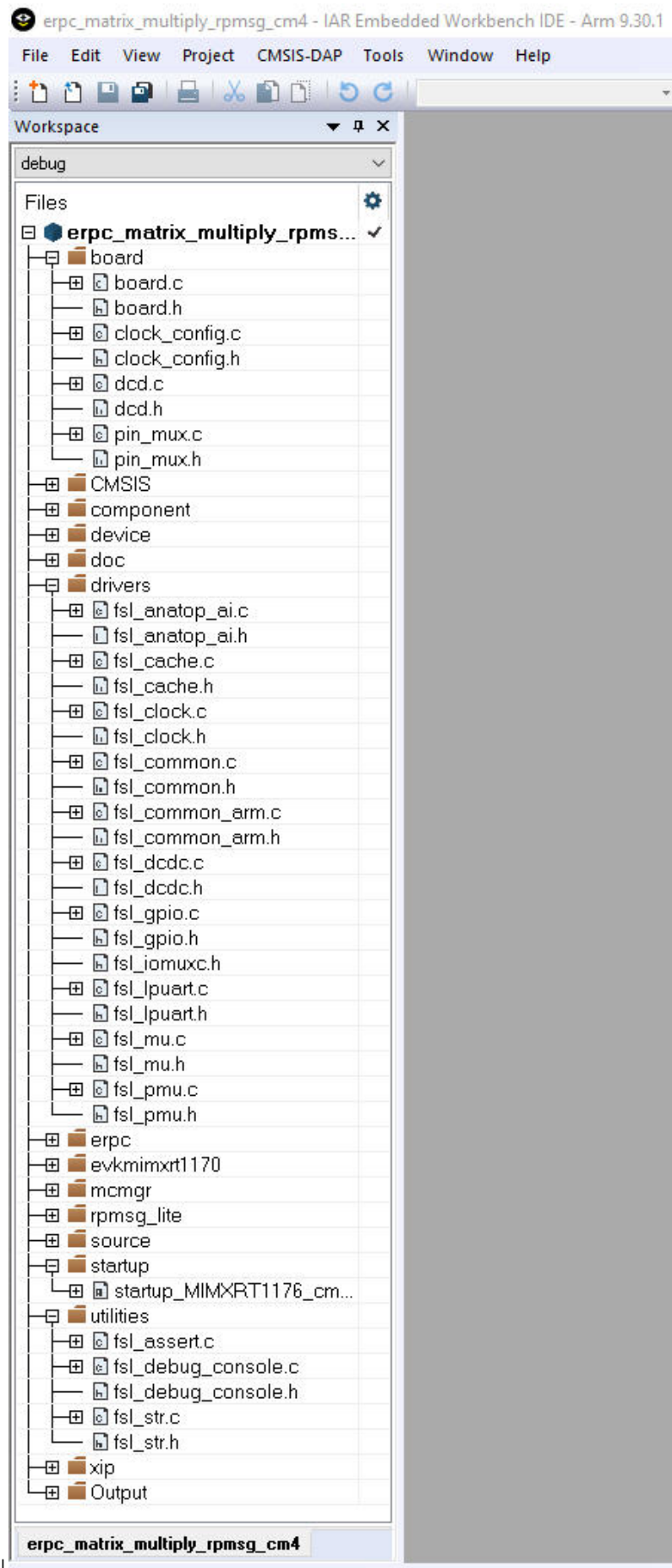
**Multicore server application** The “Matrix multiply” eRPC server project is located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmmsg/cm4/iar`

The project files for the eRPC server have the `_cm4` suffix.

**Server project basic source files** The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in:

- `<MCUXpressoSDK_install_dir>/devices/<device>`
- `<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/`



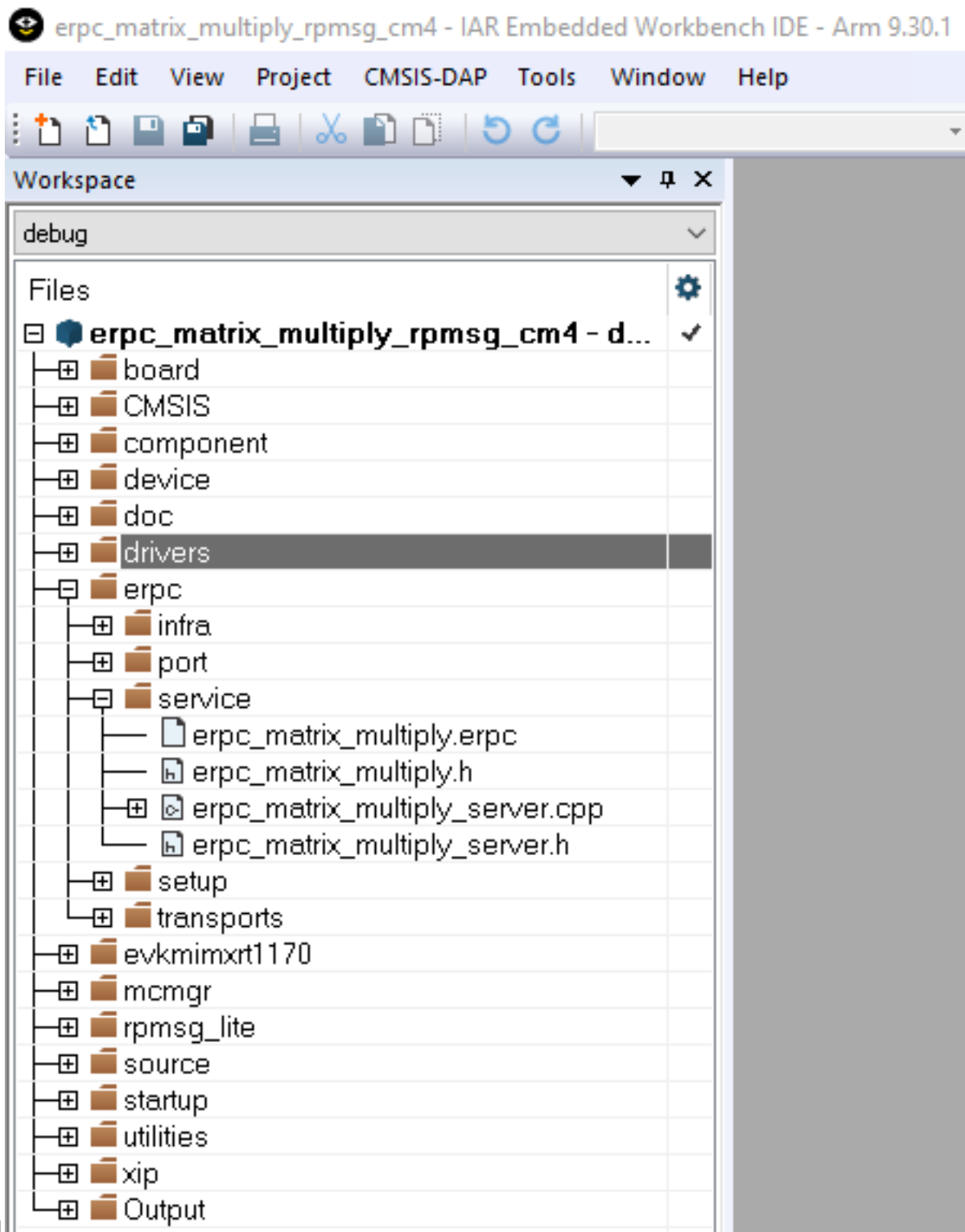
|

**Parent topic:** Multicore server application**Server related generated files** The server-related generated files are:

- erpc\_\_matric\_\_multiply.h
- erpc\_\_matrix\_\_multiply\_\_server.h
- erpc\_\_matrix\_\_multiply\_\_server.cpp

The server-related generated files contain the shim code for functions and data types declared in the IDL file. These files also contain functions for the identification of client requested functions, data deserialization, calling requested function's implementations, and data serialization and return, if requested by the client. These shim code files can be found in the following folder:

<MCUXpressoSDK\_install\_dir>/boards/evkmimxrt1170/multicore\_examples/erpc\_common/erpc\_matrix\_multiply/



**Parent topic:** Multicore server application

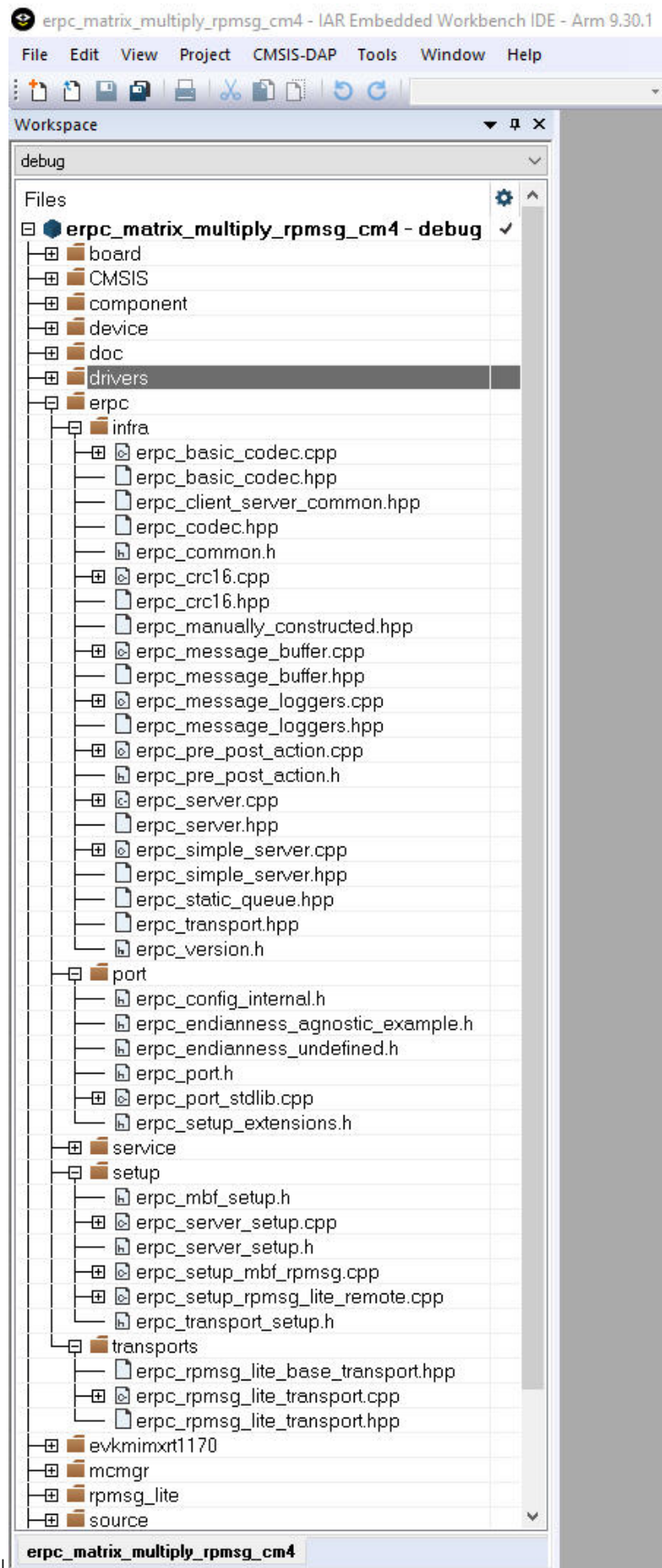
**Server infrastructure files** The eRPC infrastructure files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c`

The **erpc\_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.
  - Four files, `erpc_server.hpp`, `erpc_server.cpp`, `erpc_simple_server.hpp`, and `erpc_simple_server.cpp`, are used for running the eRPC server on the server-side applications. The simple server is currently the only implementation of the server, and its role is to catch client requests, identify and call requested functions, and send data back when requested.
  - Three files (`erpc_codec.hpp`, `erpc_basic_codec.hpp`, and `erpc_basic_codec.cpp`) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
  - The `erpc_common.hpp` file is used for common eRPC definitions, typedefs, and enums.
  - The `erpc_manually_constructed.hpp` file is used for allocating static storage for the used objects.
  - Message buffer files are used for storing serialized data: `erpc_message_buffer.h` and `erpc_message_buffer.cpp`.
  - The `erpc_transport.h` file defines the abstract interface for transport layer.
- The **port** subfolder contains the eRPC porting layer to adapt to different environments.
  - `erpc_port.h` file contains definition of `erpc_malloc()` and `erpc_free()` functions.
  - `erpc_port_stdlib.cpp` file ensures adaptation to `stdlib`.
  - `erpc_config_internal.h` internal erpc configuration file.
- The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.
  - The `erpc_server_setup.h` and `erpc_server_setup.cpp` files need to be added into the “Matrix multiply” example project to demonstrate the use of C-wrapped functions in this example.
  - The `erpc_transport_setup.h` and `erpc_setup_rpmsg_lite_remote.cpp` files need to be added into the project in order to allow the C-wrapped function for transport layer setup.
  - The `erpc_mbf_setup.h` and `erpc_setup_mbf_rpmsg.cpp` files need to be added into the project in order to allow message buffer factory usage.
- The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions in the setup folder.
  - RPMsg-Lite is used as the transport layer for the communication between cores, `erpc_rpmsg_lite_base_transport.hpp`, `erpc_rpmsg_lite_transport.hpp`, and `erpc_rpmsg_lite_transport.cpp` files need to be added into the server project.





|

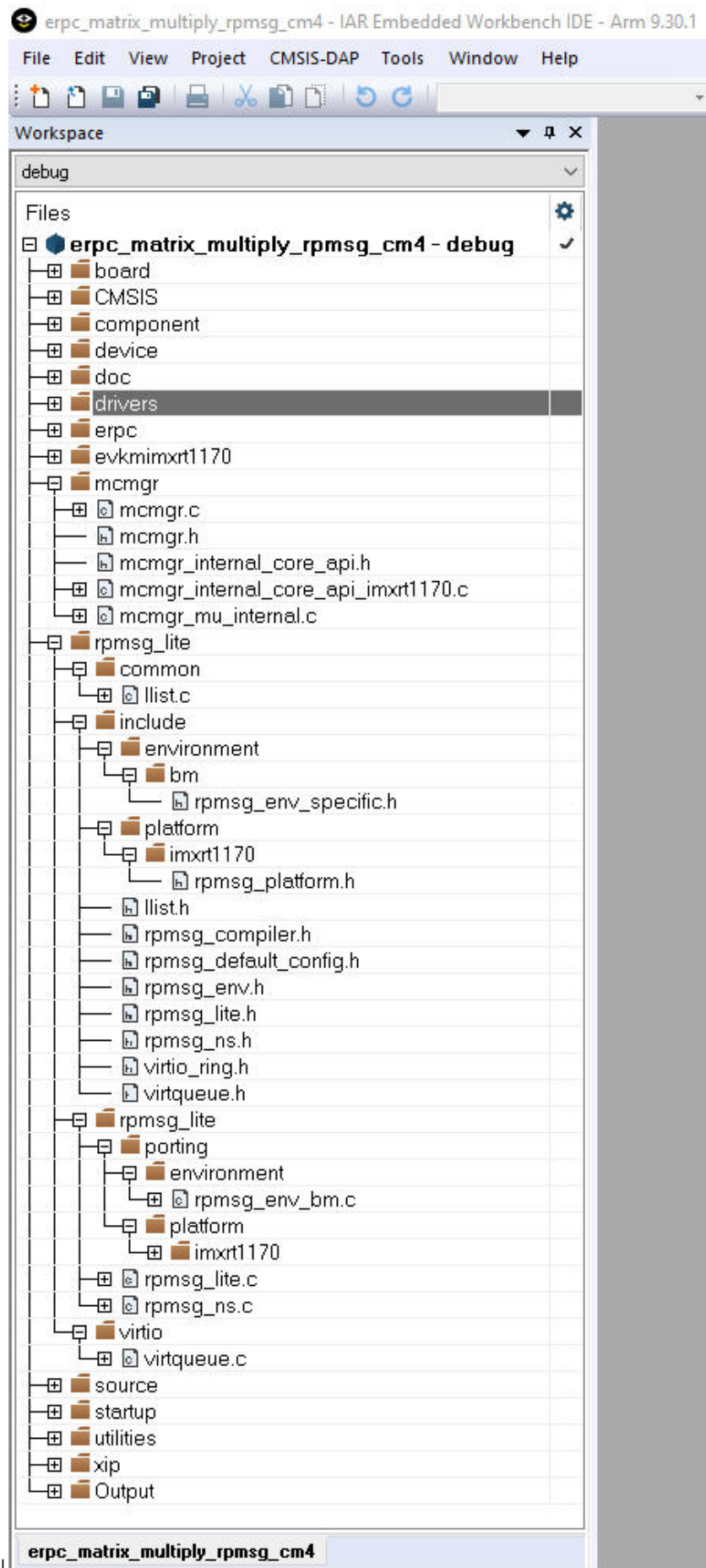
**Parent topic:** Multicore server application

**Server multicore infrastructure files** Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/`

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/`



|

**Parent topic:** Multicore server application

**Server user code** The server's user code is stored in the `main_core1.c` file, located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4`

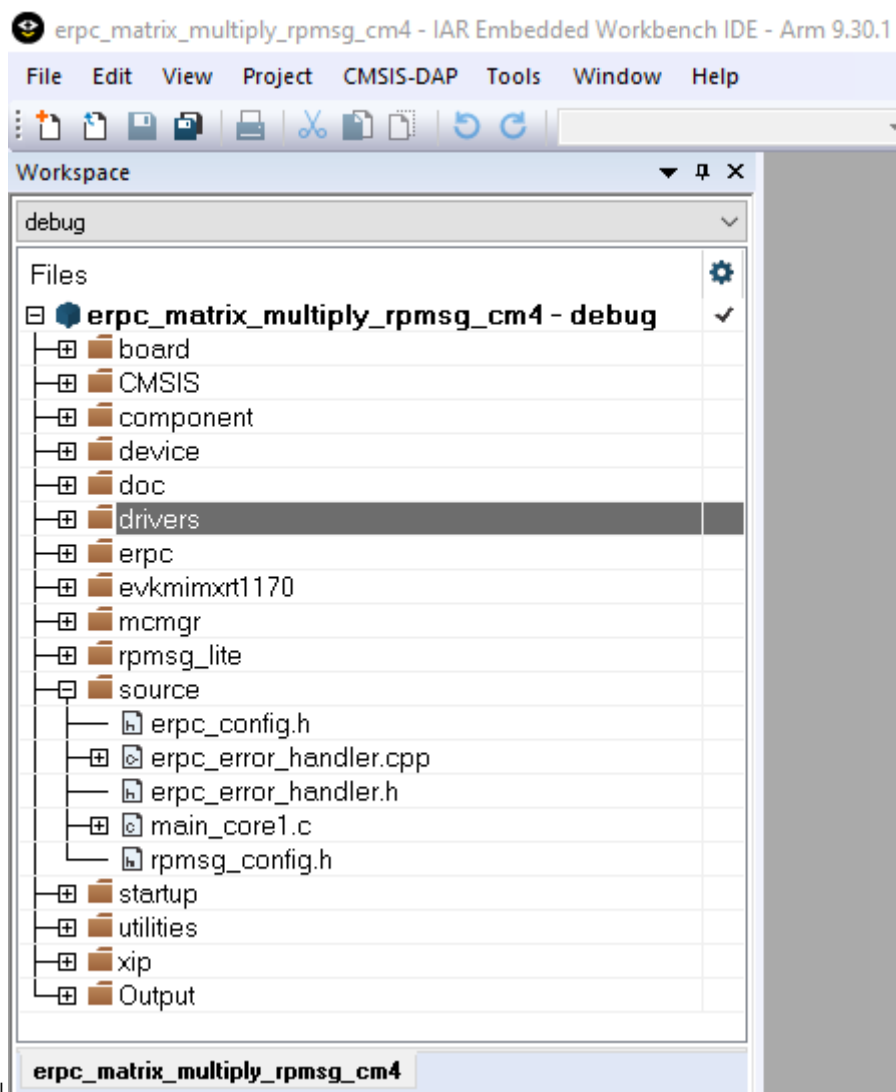
The `main_core1.c` file contains two functions:

- The **main()** function contains the code for the target board and eRPC server initialization. After the initialization, the matrix multiply service is added and the eRPC server waits for client's requests in the while loop.
- The **erpcMatrixMultiply()** function is the user implementation of the eRPC function defined in the IDL file.
- There is the possibility to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in the `erpc_error_handler.h` and `erpc_error_handler.cpp` files.

The eRPC-relevant code is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(const Matrix *matrix1, const Matrix *matrix2, Matrix *result_matrix)
{
    ...
}
int main()
{
    ...
    /* RPSMsg-Lite transport layer initialization */
    erpc_transport_t transport;
    transport = erpc_transport_rpmsg_lite_remote_init(src, dst, (void*)startupData,
    ERPC_TRANSPORT_RPMSG_LITE_LINK_ID, SignalReady, NULL);
    ...
    /* MessageBufferFactory initialization */
    erpc_mbf_t message_buffer_factory;
    message_buffer_factory = erpc_mbf_rpmsg_init(transport);
    ...
    /* eRPC server side initialization */
    erpc_server_t server;
    server = erpc_server_init(transport, message_buffer_factory);
    ...
    /* Adding the service to the server */
    erpc_service_t service = create_MatrixMultiplyService_service();
    erpc_add_service_to_server(server, service);
    ...
    while (1)
    {
        /* Process eRPC requests */
        erpc_status_t status = erpc_server_poll(server);
        /* handle error status */
        if (status != kErpcStatus_Success)
        {
            /* print error description */
            erpc_error_handler(status, 0);
            ...
        }
        ...
    }
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg\_config.h) and eRPC (erpc\_config.h), located in the `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg` folder.



**Parent topic:**Multicore server application

**Parent topic:**[Create an eRPC application](#)

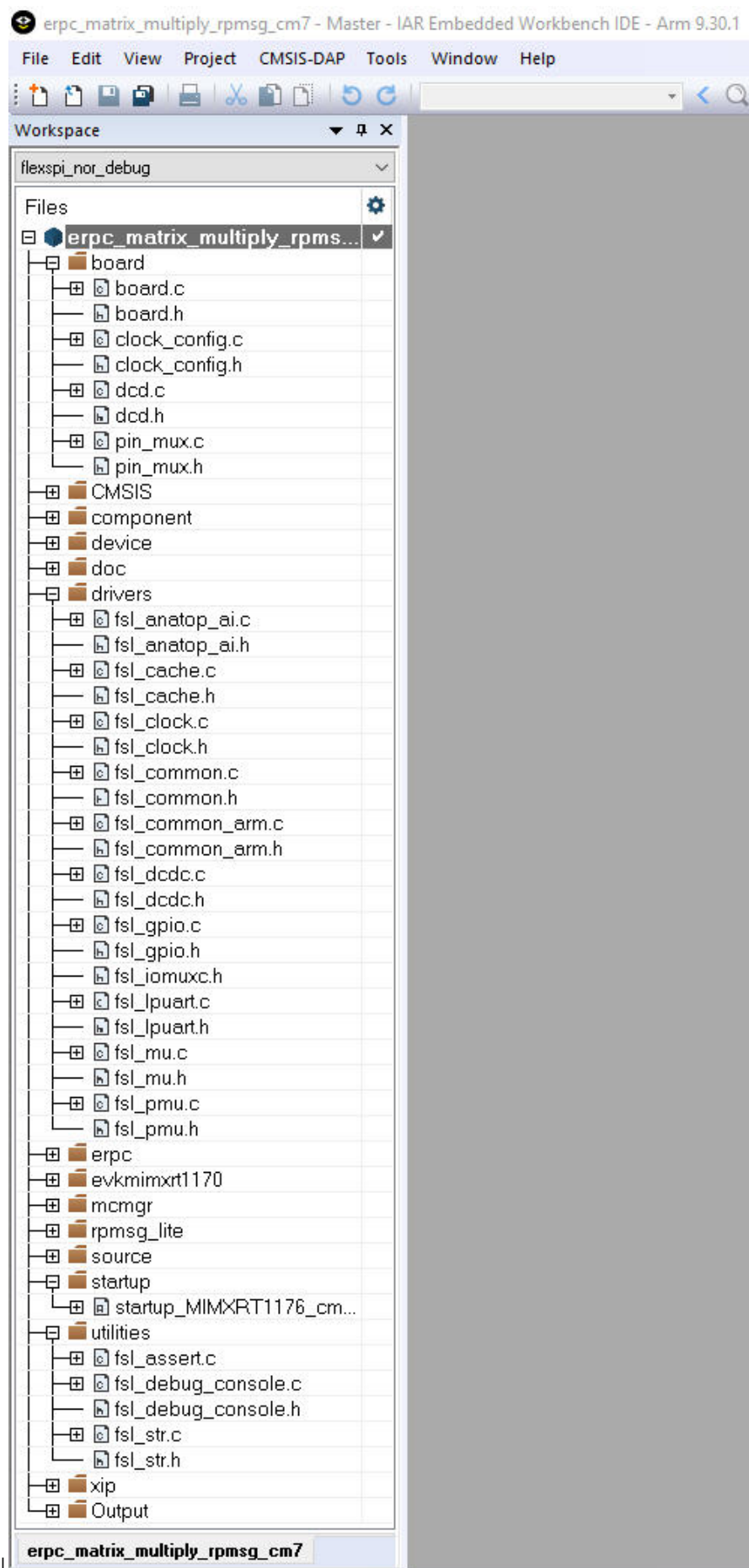
**Multicore client application** The “Matrix multiply” eRPC client project is located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm7/iar`

Project files for the eRPC client have the `_cm7` suffix.

**Client project basic source files** The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in the following folders:

- `<MCUXpressoSDK_install_dir>/devices/<device>`
- `<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/`



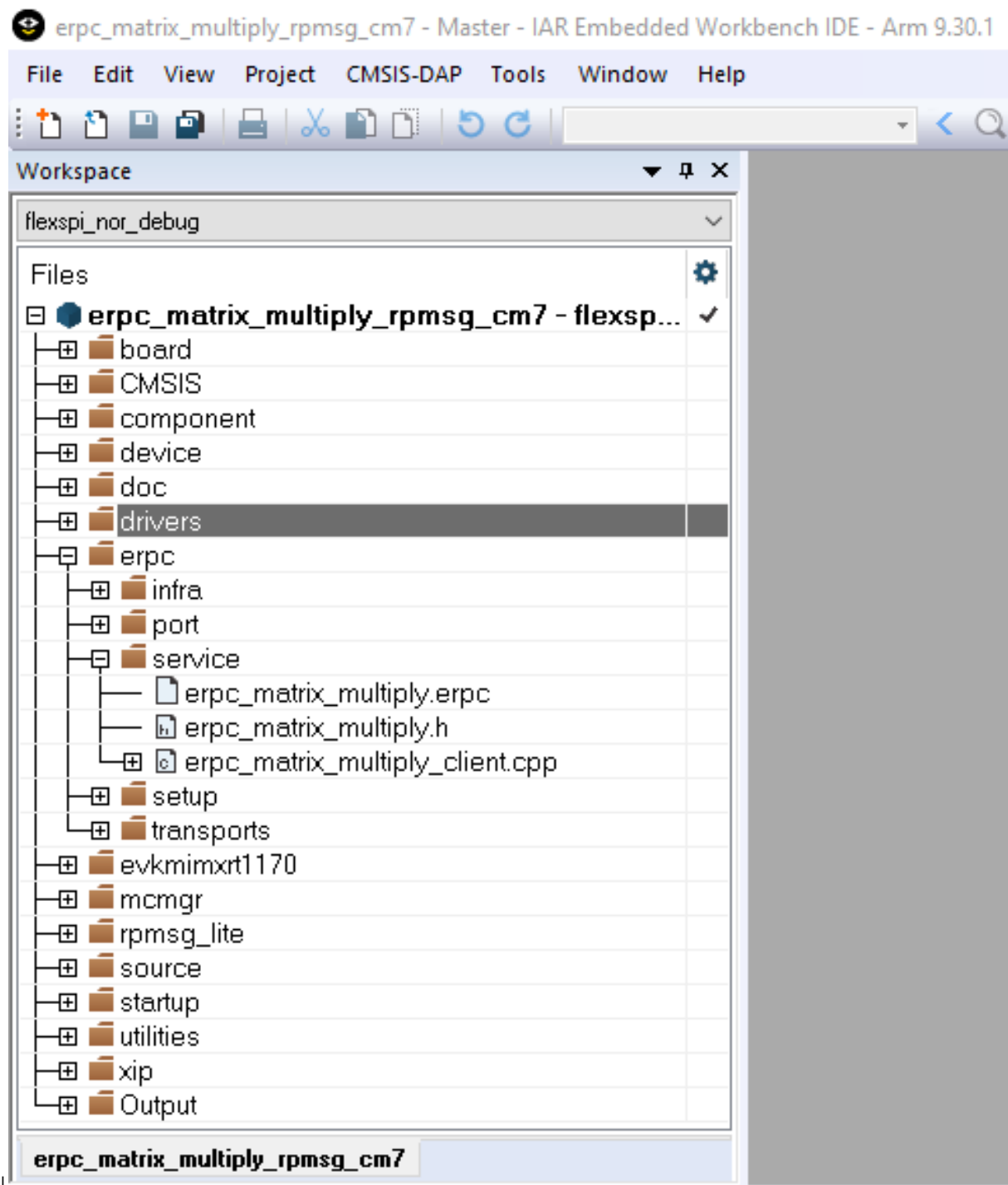
|

**Parent topic:** Multicore client application

**Client-related generated files** The client-related generated files are:

- `erpc__matric__multiply.h`
- `erpc__matrix__multiply__client.cpp`

These files contain the shim code for the functions and data types declared in the IDL file. These functions also call methods for codec initialization, data serialization, performing eRPC requests, and de-serializing outputs into expected data structures (if return values are expected). These shim code files can be found in the `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/` folder.



**Parent topic:** Multicore client application

**Client infrastructure files** The eRPC infrastructure files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c`

The **erpc\_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.



- Two files, `erpc_client_manager.h` and `erpc_client_manager.cpp`, are used for managing the client-side application. The main purpose of the client files is to create, perform, and release eRPC requests.
- Three files (`erpc_codec.hpp`, `erpc_basic_codec.hpp`, and `erpc_basic_codec.cpp`) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
- `erpc_common.h` file is used for common eRPC definitions, typedefs, and enums.
- `erpc_manually_constructed.hpp` file is used for allocating static storage for the used objects.
- Message buffer files are used for storing serialized data: `erpc_message_buffer.hpp` and `erpc_message_buffer.cpp`.
- `erpc_transport.hpp` file defines the abstract interface for transport layer.

The **port** subfolder contains the eRPC porting layer to adapt to different environments.

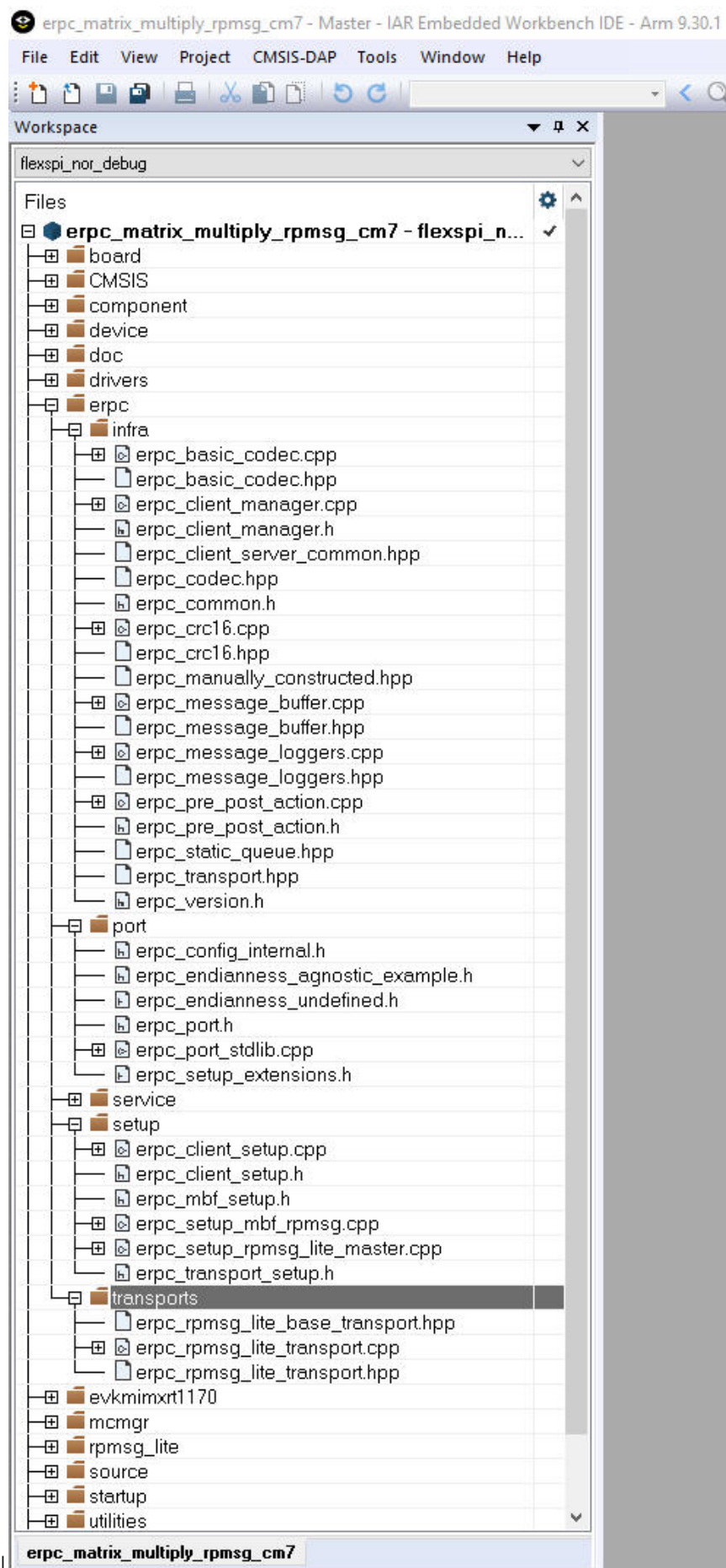
- `erpc_port.h` file contains definition of `erpc_malloc()` and `erpc_free()` functions.
- `erpc_port_stdlib.cpp` file ensures adaptation to `stdlib`.
- `erpc_config_internal.h` internal eRPC configuration file.

The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.

- `erpc_client_setup.h` and `erpc_client_setup.cpp` files needs to be added into the “Matrix multiply” example project to demonstrate the use of C-wrapped functions in this example.
- `erpc_transport_setup.h` and `erpc_setup_rpmsg_lite_master.cpp` files needs to be added into the project in order to allow C-wrapped function for transport layer setup.
- `erpc_mbf_setup.h` and `erpc_setup_mbf_rpmsg.cpp` files needs to be added into the project in order to allow message buffer factory usage.

The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions, in the setup folder.

- RPMsg-Lite is used as the transport layer for the communication between cores, `erpc_rpmsg_lite_base_transport.hpp`, `erpc_rpmsg_lite_transport.hpp`, and `erpc_rpmsg_lite_transport.cpp` files needs to be added into the client project.



|

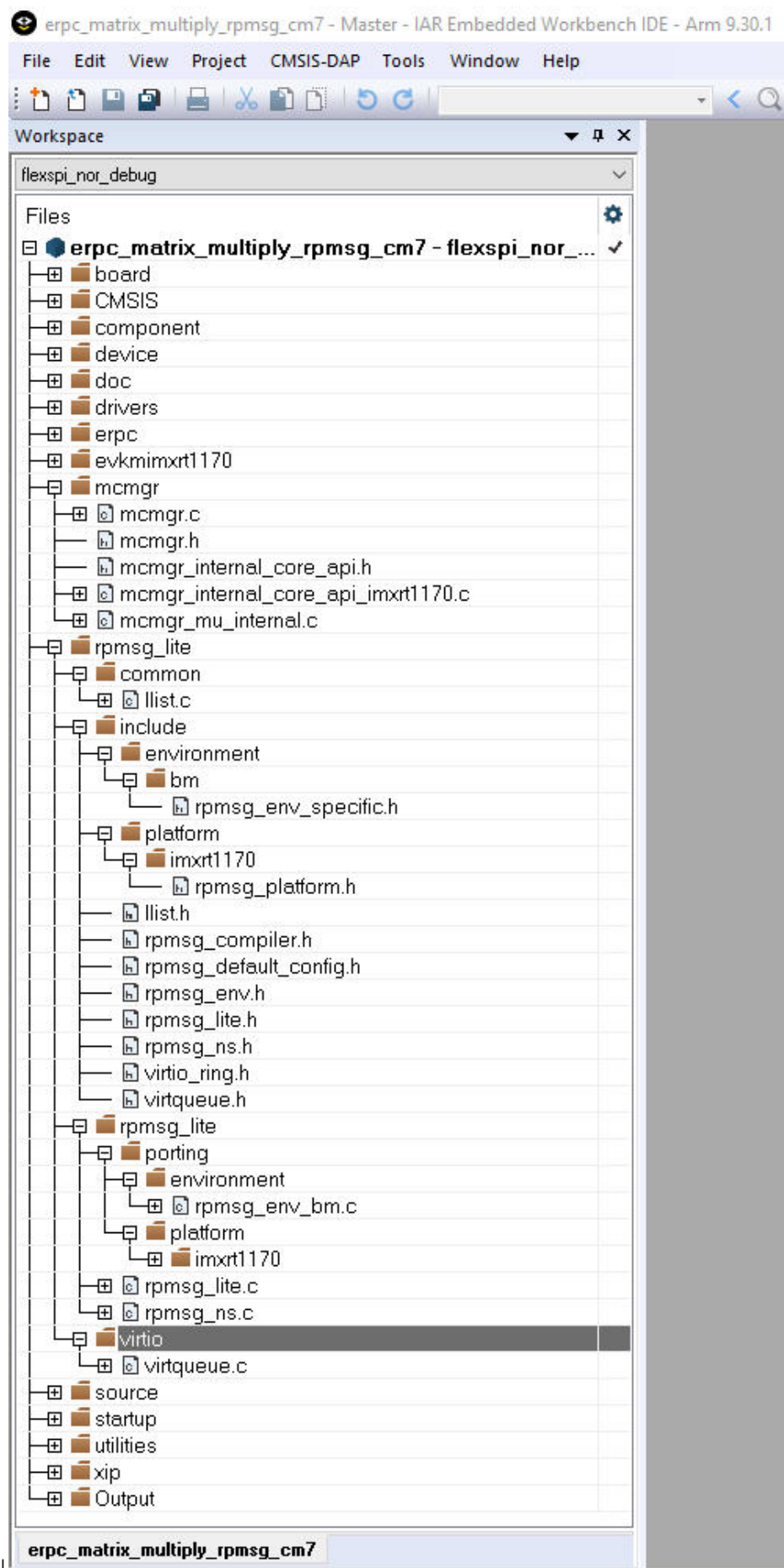
**Parent topic:** Multicore client application

**Client multicore infrastructure files** Because of the RPSMsg-Lite (transport layer), it is also necessary to include RPSMsg-Lite related files, which are in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/rpsmsg_lite/`

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/`



**Parent topic:**Multicore client application

**Client user code** The client's user code is stored in the main\_core0.c file, located in the following folder:

<MCUXpressoSDK\_install\_dir>/boards/evkmimxrt1170/multicore\_example/erpc\_matrix\_multiply\_rpmsg/cm7

The main\_core0.c file contains the code for target board and eRPC initialization.

- After initialization, the secondary core is released from reset.
- When the secondary core is ready, the primary core initializes two matrix variables.
- The erpcMatrixMultiply eRPC function is called to issue the eRPC request and get the result.

It is possible to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in erpc\_error\_handler.h and erpc\_error\_handler.cpp files.

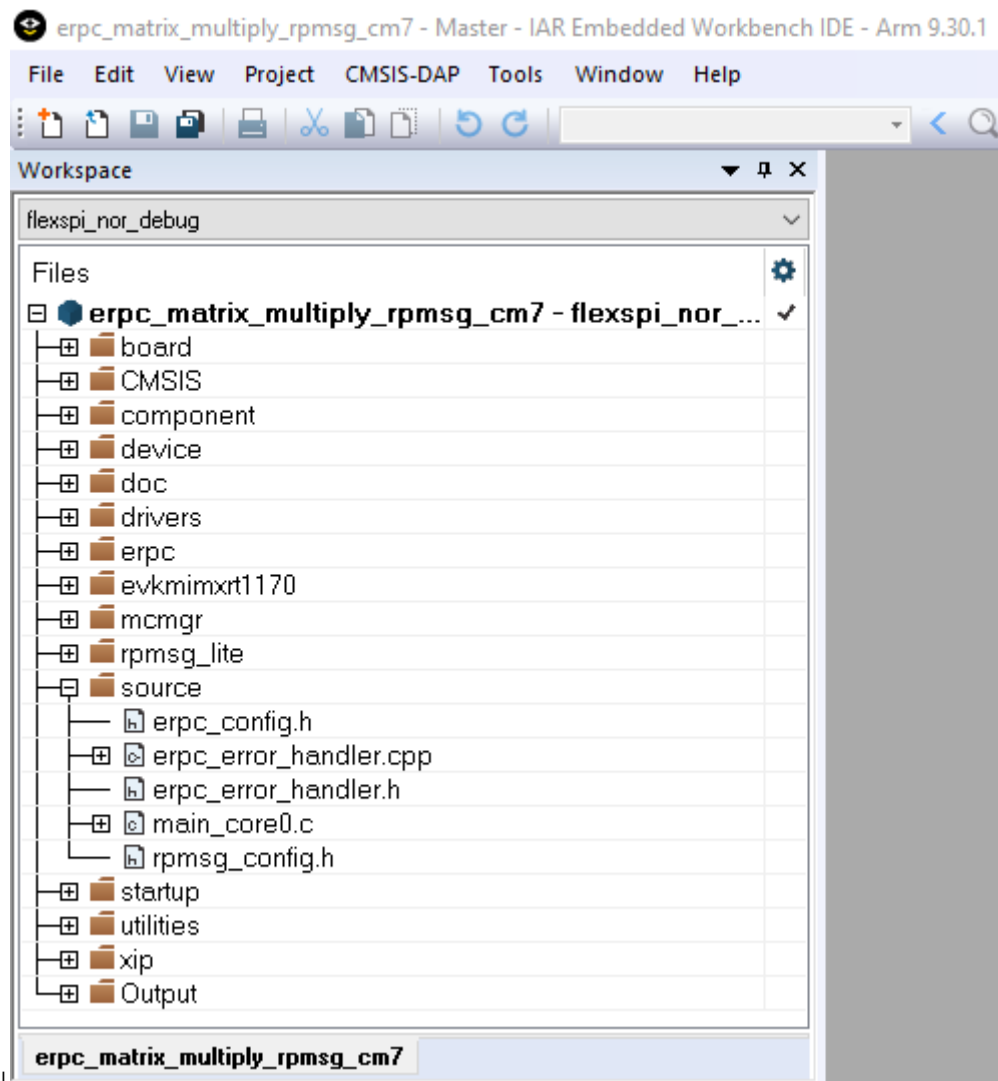
The matrix multiplication can be issued repeatedly, when pressing a software board button.

The eRPC-relevant code is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* RPMsg-Lite transport layer initialization */
erpc_transport_t transport;
transport = erpc_transport_rpmsg_lite_master_init(src, dst,
ERPC_TRANSPORT_RPMSG_LITE_LINK_ID);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_rpmsg_init(transport);
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport, message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
    /* Invoke the erpcMatrixMultiply function */
    erpcMatrixMultiply(matrix1, matrix2, result_matrix);
    ...
    /* Check if some error occurred in eRPC */
    if (g_erpc_error_occurred)
    {
        /* Exit program loop */
        break;
    }
    ...
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg\_config.h) and eRPC (erpc\_config.h), located in the following folder:

<MCUXpressoSDK\_install\_dir>/boards/evkmimxrt1170/multicore\_examples/erpc\_matrix\_multiply\_rpmsg



**Parent topic:**Multicore client application

**Parent topic:**[Create an eRPC application](#)

**Multiprocessor server application** The “Matrix multiply” eRPC server project for multiprocessor applications is located in the `<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>` folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires server-related generated files (server shim code), server infrastructure files, and the server user code. There is no need for server multicore infrastructure files (MCMGR and RPMsg-Lite). The RPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|      |                                                                         |
|------|-------------------------------------------------------------------------|
| SPI  | <eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_slave.cpp          |
|      | <eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.hpp |
|      | <eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.cpp |
| UART | <eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp            |

<eRPC base directory>/erpc\_c/transport/erpc\_uart\_cmsis\_transport.hpp

<eRPC base directory>/erpc\_c/transport/erpc\_uart\_cmsis\_transport.cpp

|

**Server user code** The server's user code is stored in the main\_server.c file, located in the <MCUXpressoSDK\_install\_dir>/boards/<board\_name>/multiprocessor\_examples/erpc\_server\_matrix\_multiply\_<transport\_layer>/ folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(Matrix matrix1, Matrix matrix2, Matrix result_matrix)
{
    ...
}
int main()
{
    ...
    /* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver
    ↪operations */
    erpc_transport_t transport;
    transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
    ...
    /* MessageBufferFactory initialization */
    erpc_mbf_t message_buffer_factory;
    message_buffer_factory = erpc_mbf_dynamic_init();
    ...
    /* eRPC server side initialization */
    erpc_server_t server;
    server = erpc_server_init(transport, message_buffer_factory);
    ...
    /* Adding the service to the server */
    erpc_service_t service = create_MatrixMultiplyService_service();
    erpc_add_service_to_server(server, service);
    ...
    while (1)
    {
        /* Process eRPC requests */
        erpc_status_t status = erpc_server_poll(server)
        /* handle error status */
        if (status != kErpcStatus_Success)
        {
            /* print error description */
            erpc_error_handler(status, 0);
            ...
        }
        ...
    }
}
```

**Parent topic:**Multiprocessor server application

**Multiprocessor client application** The “Matrix multiply” eRPC client project for multiprocessor applications is located in the <MCUXpressoSDK\_install\_dir>/boards/<board\_name>/multiprocessor\_examples/erpc\_client\_matrix\_multiply\_<transport\_layer>/iar/ folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires client-related generated files (server shim code),

client infrastructure files, and the client user code. There is no need for client multicore infrastructure files (MCMGR and RMPMsg-Lite). The RMPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|                               |                                                                 |
|-------------------------------|-----------------------------------------------------------------|
| SPI                           | <eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_master.cpp |
| <eRPC base directory>/erpc_c/ | transports/ erpc_(d)spi_master_transport.hpp                    |
| <eRPC base directory>/erpc_c/ | transports/ erpc_(d)spi_master_transport.cpp                    |
| UART                          | <eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp    |
| <eRPC base directory>/erpc_c/ | transports/erpc_uart_cmsis_transport.hpp                        |
| <eRPC base directory>/erpc_c/ | transports/erpc_uart_cmsis_transport.cpp                        |
|                               |                                                                 |

**Client user code** The client's user code is stored in the `main_client.c` file, located in the `<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<transport_layer>/` folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver
↳operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_dynamic_init();
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport,message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
    /* Invoke the erpcMatrixMultiply function */
    erpcMatrixMultiply(matrix1, matrix2, result_matrix);
    ...
    /* Check if some error occurred in eRPC */
    if (g_erpc_error_occurred)
    {
        /* Exit program loop */
        break;
    }
    ...
}
```

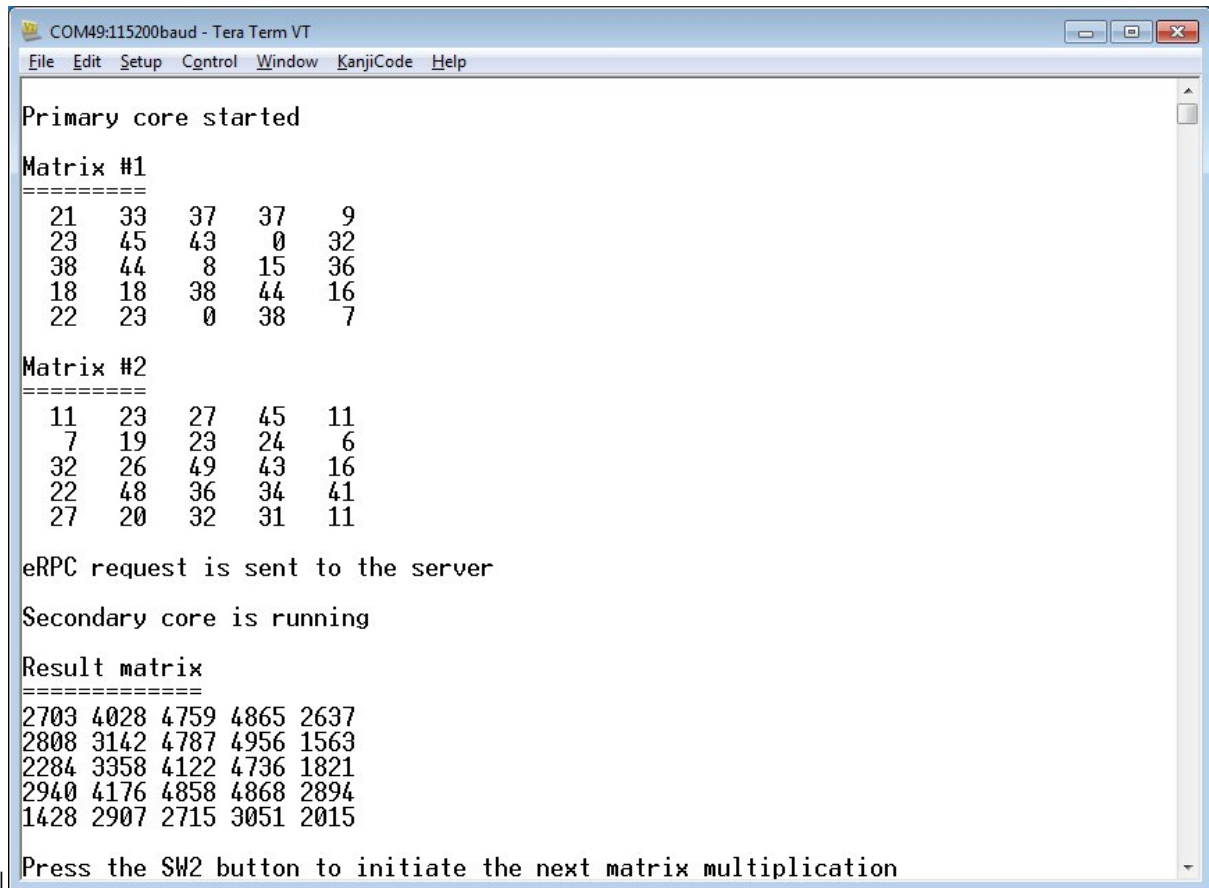
**Parent topic:**Multiprocessor client application

**Parent topic:**Multiprocessor server application



Parent topic:[Create an eRPC application](#)

**Running the eRPC application** Follow the instructions in *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) (located in the <MCUXpressoSDK\_install\_dir>/docs folder), to load both the primary and the secondary core images into the on-chip memory, and then effectively debug the dual-core application. After the application is running, the serial console should look like:



```

COM49:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help

Primary core started

Matrix #1
=====
 21  33  37  37   9
 23  45  43   0  32
 38  44   8  15  36
 18  18  38  44  16
 22  23   0  38   7

Matrix #2
=====
 11  23  27  45  11
  7  19  23  24   6
 32  26  49  43  16
 22  48  36  34  41
 27  20  32  31  11

eRPC request is sent to the server

Secondary core is running

Result matrix
=====
2703 4028 4759 4865 2637
2808 3142 4787 4956 1563
2284 3358 4122 4736 1821
2940 4176 4858 4868 2894
1428 2907 2715 3051 2015

Press the SW2 button to initiate the next matrix multiplication

```

For multiprocessor applications that are running between PC and the target evaluation board or between two boards, follow the instructions in the accompanied example readme files that provide details about the proper board setup and the PC side setup (Python).

Parent topic:[Create an eRPC application](#)

Parent topic:[eRPC example](#)

**Other uses for an eRPC implementation** The eRPC implementation is generic, and its use is not limited to just embedded applications. When creating an eRPC application outside the embedded world, the same principles apply. For example, this manual can be used to create an eRPC application for a PC running the Linux operating system. Based on the used type of transport medium, existing transport layers can be used, or new transport layers can be implemented.

For more information and erpc updates see the [github.com/EmbeddedRPC](https://github.com/EmbeddedRPC).

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

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

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

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

**Changelog eRPC** All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog](#), and this project adheres to [Semantic Versioning](#).

## Unreleased

### Added

### Fixed

- Python code of the eRPC infrastructure was updated to match the proper python code style, add type annotations and improve readability.

## 1.14.0

### Added

- Added Cmake/Kconfig support.
- Made java code jdk11 compliant, GitHub PR #432.
- Added imxrt1186 support into mu transport layer.
- erpcgen: Added assert for listType before usage, GitHub PR #406.

### Fixed

- eRPC: Sources reformatted.
- erpc: Fixed typo in semaphore get (mutex -> semaphore), and write it can fail in case of timeout, GitHub PR #446.
- erpc: Free the arbitrated client token from client manager, GitHub PR #444.

- erpc: Fixed Makefile, install the erpc\_simple\_server header, GitHub PR #447.
- erpc\_python: Fixed possible AttributeError and OSError on calling TCPTransport.close(), GitHub PR #438.
- Examples and tests consolidated.

### 1.13.0

#### Added

- erpc: Add BSD-3 license to endianness agnostic files, GitHub PR #417.
- eRPC: Add new Zephyr-related transports (zephyr\_uart, zephyr\_mbox).
- eRPC: Add new Zephyr-related examples.

#### Fixed

- eRPC,erpcgen: Fixing/improving markdown files, GitHub PR #395.
- eRPC: Fix Python client TCPTransports not being able to close, GitHub PR #390.
- eRPC,erpcgen: Align switch brackets, GitHub PR #396.
- erpc: Fix zephyr uart transport, GitHub PR #410.
- erpc: UART ZEPHYR Transport stop to work after a few transactions when using USB-CDC resolved, GitHub PR #420.

#### Removed

- eRPC,erpcgen: Remove cstbool library, GitHub PR #403.

### 1.12.0

#### Added

- eRPC: Add dynamic/static option for transport init, GitHub PR #361.
- eRPC,erpcgen: Winsock2 support, GitHub PR #365.
- eRPC,erpcgen: Feature/support multiple clients, GitHub PR #271.
- eRPC,erpcgen: Feature/buffer head - Framed transport header data stored in Message-Buffer, GitHub PR #378.
- eRPC,erpcgen: Add experimental Java support.

#### Fixed

- eRPC: Fix receive error value for spidev, GitHub PR #363.
- eRPC: UartTransport::init adaptation to changed driver.
- eRPC: Fix typo in assert, GitHub PR #371.
- eRPC,erpcgen: Move enums to enum classes, GitHub PR #379.
- eRPC: Fixed rpmsg tty transport to work with serial transport, GitHub PR #373.

### 1.11.0

#### Fixed

- eRPC: Makefiles update, GitHub PR #301.
- eRPC: Resolving warnings in Python, GitHub PR #325.
- eRPC: Python3.8 is not ready for usage of typing.Any type, GitHub PR #325.
- eRPC: Improved codec function to use reference instead of address, GitHub PR #324.
- eRPC: Fix NULL check for pending client creation, GitHub PR #341.
- eRPC: Replace sprintf with snprintf, GitHub PR #343.
- eRPC: Use MU\_SendMsg blocking call in MU transport.
- eRPC: New LPSPI and LPI2C transport layers.
- eRPC: Freeing static objects, GitHub PR #353.
- eRPC: Fixed casting in deinit functions, GitHub PR #354.
- eRPC: Align LIBUSB\_SIO.GetNumPorts API use with libusb\_sio python module v. 2.1.11.
- erpcgen: Renamed temp variable to more generic one, GitHub PR #321.
- erpcgen: Add check that string read is not more than max length, GitHub PR #328.
- erpcgen: Move to g++ in pytest, GitHub PR #335.
- erpcgen: Use build=release for make, GitHub PR #334.
- erpcgen: Removed boost dependency, GitHub PR #346.
- erpcgen: Mingw support, GitHub PR #344.
- erpcgen: VS build update, GitHub PR #347.
- erpcgen: Modified name for common types macro scope, GitHub PR #337.
- erpcgen: Fixed memcpy for template, GitHub PR #352.
- eRPC,erpcgen: Change default build target to release + adding artefacts, GitHub PR #334.
- eRPC,erpcgen: Remove redundant includes, GitHub PR #338.
- eRPC,erpcgen: Many minor code improvements, GitHub PR #323.

### 1.10.0

#### Fixed

- eRPC: MU transport layer switched to blocking MU\_SendMsg() API use.

### 1.10.0

#### Added

- eRPC: Add TCP\_NODELAY option to python, GitHub PR #298.

## Fixed

- eRPC: MUPort adaptation to new supported SoCs.
- eRPC: Simplifying CI with installing dependencies using shell script, GitHub PR #267.
- eRPC: Using event for waiting for sock connection in TCP python server, formatting python code, C specific includes, GitHub PR #269.
- eRPC: Endianness agnostic update, GitHub PR #276.
- eRPC: Assertion added for functions which are returning status on freeing memory, GitHub PR #277.
- eRPC: Fixed closing arbitrator server in unit tests, GitHub PR #293.
- eRPC: Makefile updated to reflect the correct header names, GitHub PR #295.
- eRPC: Compare value length to used length() in reading data from message buffer, GitHub PR #297.
- eRPC: Replace EXPECT\_TRUE with EXPECT\_EQ in unit tests, GitHub PR #318.
- eRPC: Adapt rpmsg\_lite based transports to changed rpmsg\_lite\_wait\_for\_link\_up() API parameters.
- eRPC, erpcgen: Better distinguish which file can and cannot be linked by C linker, GitHub PR #266.
- eRPC, erpcgen: Stop checking if pointer is NULL before sending it to the erpc\_free function, GitHub PR #275.
- eRPC, erpcgen: Changed api to count with more interfaces, GitHub PR #304.
- erpcgen: Check before reading from heap the buffer boundaries, GitHub PR #287.
- erpcgen: Several fixes for tests and CI, GitHub PR #289.
- erpcgen: Refactoring erpcgen code, GitHub PR #302.
- erpcgen: Fixed assigning const value to enum, GitHub PR #309.
- erpcgen: Enable runTesttest\_enumErrorCode\_allDirection, serialize enums as int32 instead of uint32.

### 1.9.1

## Fixed

- eRPC: Construct the USB CDC transport, rather than a client, GitHub PR #220.
- eRPC: Fix premature import of package, causing failure when attempting installation of Python library in a clean environment, GitHub PR #38, #226.
- eRPC: Improve python detection in make, GitHub PR #225.
- eRPC: Fix several warnings with deprecated call in pytest, GitHub PR #227.
- eRPC: Fix freeing union members when only default need be freed, GitHub PR #228.
- eRPC: Fix making test under Linux, GitHub PR #229.
- eRPC: Assert costumizing, GitHub PR #148.
- eRPC: Fix corrupt clientList bug in TransportArbitrator, GitHub PR #199.
- eRPC: Fix build issue when invoking g++ with -Wno-error=free-nonheap-object, GitHub PR #233.
- eRPC: Fix inout cases, GitHub PR #237.

- eRPC: Remove ERPC\_PRE\_POST\_ACTION dependency on return type, GitHub PR #238.
- eRPC: Adding NULL to ptr when codec function failed, fixing memcpy when fail is present during deserialization, GitHub PR #253.
- eRPC: MessageBuffer usage improvement, GitHub PR #258.
- eRPC: Get rid of serial and enum34 dependency (enum34 is in python3 since 3.4 (from 2014)), GitHub PR #247.
- eRPC: Several MISRA violations addressed.
- eRPC: Fix timeout for Freertos semaphore, GitHub PR #251.
- eRPC: Use of rpmsg\_lite\_wait\_for\_link\_up() in rpmsg\_lite based transports, GitHub PR #223.
- eRPC: Fix codec nullptr dereferencing, GitHub PR #264.
- erpcgen: Fix two syntax errors in erpcgen Python output related to non-encapsulated unions, improved test for union, GitHub PR #206, #224.
- erpcgen: Fix serialization of list/binary types, GitHub PR #240.
- erpcgen: Fix empty list parsing, GitHub PR #72.
- erpcgen: Fix templates for malloc errors, GitHub PR #110.
- erpcgen: Get rid of encapsulated union declarations in global scale, improve enum usage in unions, GitHub PR #249, #250.
- erpcgen: Fix compile error:UniqueIdChecker.cpp:156:104:'sort' was not declared, GitHub PR #265.

## 1.9.0

### Added

- eRPC: Allow used LIBUSB\_SIO device index being specified from the Python command line argument.

### Fixed

- eRPC: Improving template usage, GitHub PR #153.
- eRPC: run\_clang\_format.py cleanup, GitHub PR #177.
- eRPC: Build TCP transport setup code into liberpc, GitHub PR #179.
- eRPC: Fix multiple definitions of g\_client error, GitHub PR #180.
- eRPC: Fix memset past end of buffer in erpc\_setup\_mbf\_static.cpp, GitHub PR #184.
- eRPC: Fix deprecated error with newer pytest version, GitHub PR #203.
- eRPC, erpcgen: Static allocation support and usage of rpmsg static FreeRTOSs related API, GitHub PR #168, #169.
- erpcgen: Remove redundant module imports in erpcgen, GitHub PR #196.

## 1.8.1

### Added

- eRPC: New i2c\_slave\_transport transport introduced.

### Fixed

- eRPC: Fix misra erpc c, GitHub PR #158.
- eRPC: Allow conditional compilation of message\_loggers and pre\_post\_action.
- eRPC: (D)SPI slave transports updated to avoid busy loops in rtos environments.
- erpcgen: Re-implement EnumMember::hasValue(), GitHub PR #159.
- erpcgen: Fixing several misra issues in shim code, erpcgen and unit tests updated, GitHub PR #156.
- erpcgen: Fix bison file, GitHub PR #156.

### 1.8.0

### Added

- eRPC: Support win32 thread, GitHub PR #108.
- eRPC: Add mbed support for malloc() and free(), GitHub PR #92.
- eRPC: Introduced pre and post callbacks for eRPC call, GitHub PR #131.
- eRPC: Introduced new USB CDC transport.
- eRPC: Introduced new Linux spidev-based transport.
- eRPC: Added formatting extension for VSC, GitHub PR #134.
- erpcgen: Introduce ustring type for unsigned char and force cast to char\*, GitHub PR #125.

### Fixed

- eRPC: Update makefile.
- eRPC: Fixed warnings and error with using MessageLoggers, GitHub PR #127.
- eRPC: Extend error msg for python server service handle function, GitHub PR #132.
- eRPC: Update CMSIS UART transport layer to avoid busy loops in rtos environments, introduce semaphores.
- eRPC: SPI transport update to allow usage without handshaking GPIO.
- eRPC: Native \_WIN32 erpc serial transport and threading.
- eRPC: Arbitrator deadlock fix, TCP transport updated, TCP setup functions introduced, GitHub PR #121.
- eRPC: Update of matrix\_multiply.py example: Add -serial and -baud argument, GitHub PR #137.
- eRPC: Update of .clang-format, GitHub PR #140.
- eRPC: Update of erpc\_framed\_transport.cpp: return error if received message has zero length, GitHub PR #141.
- eRPC, erpcgen: Fixed error messages produced by -Wall -Wextra -Wshadow -pedantic-errors compiler flags, GitHub PR #136, #139.
- eRPC, erpcgen: Core re-formatted using Clang version 10.
- erpcgen: Enable deallocation in server shim code when callback/function pointer used as out parameter in IDL.
- erpcgen: Removed '\$' character from generated symbol name in '\_\$union' suffix, GitHub PR #103.

- erpcgen: Resolved mismatch between C++ and Python for callback index type, GitHub PR #111.
- erpcgen: Python generator improvements, GitHub PR #100, #118.
- erpcgen: Fixed error messages produced by -Wall -Wextra -Wshadow -pedantic-errors compiler flags, GitHub PR #136.

#### 1.7.4

##### Added

- eRPC: Support MU transport unit testing.
- eRPC: Adding mbed os support.

##### Fixed

- eRPC: Unit test code updated to handle service add and remove operations.
- eRPC: Several MISRA issues in rpmsg-based transports addressed.
- eRPC: Fixed Linux/TCP acceptance tests in release target.
- eRPC: Minor documentation updates, code formatting.
- erpcgen: Whitespace removed from C common header template.

#### 1.7.3

##### Fixed

- eRPC: Improved the test\_callbacks logic to be more understandable and to allow requested callback execution on the server side.
- eRPC: TransportArbitrator::prepareClientReceive modified to avoid incorrect return value type.
- eRPC: The ClientManager and the ArbitratedClientManager updated to avoid performing client requests when the previous serialization phase fails.
- erpcgen: Generate the shim code for destroy of statically allocated services.

#### 1.7.2

##### Added

- eRPC: Add missing doxygen comments for transports.

##### Fixed

- eRPC: Improved support of const types.
- eRPC: Fixed Mac build.
- eRPC: Fixed serializing python list.
- eRPC: Documentation update.



### 1.7.1

#### Fixed

- eRPC: Fixed semaphore in static message buffer factory.
- erpcgen: Fixed MU received error flag.
- erpcgen: Fixed tcp transport.

### 1.7.0

#### Added

- eRPC: List names are based on their types. Names are more deterministic.
- eRPC: Service objects are as a default created as global static objects.
- eRPC: Added missing doxygen comments.
- eRPC: Added support for 64bit numbers.
- eRPC: Added support of program language specific annotations.

#### Fixed

- eRPC: Improved code size of generated code.
- eRPC: Generating crc value is optional.
- eRPC: Fixed CMSIS Uart driver. Removed dependency on KSDK.
- eRPC: Forbid users use reserved words.
- eRPC: Removed outByref for function parameters.
- eRPC: Optimized code style of callback functions.

### 1.6.0

#### Added

- eRPC: Added @nullable support for scalar types.

#### Fixed

- eRPC: Improved code size of generated code.
- eRPC: Improved eRPC nested calls.
- eRPC: Improved eRPC list length variable serialization.

### 1.5.0

### Added

- eRPC: Added support for unions type non-wrapped by structure.
- eRPC: Added callbacks support.
- eRPC: Added support @external annotation for functions.
- eRPC: Added support @name annotation.
- eRPC: Added Messaging Unit transport layer.
- eRPC: Added RPMSG Lite RTOS TTY transport layer.
- eRPC: Added version verification and IDL version verification between eRPC code and eRPC generated shim code.
- eRPC: Added support of shared memory pointer.
- eRPC: Added annotation to forbid generating const keyword for function parameters.
- eRPC: Added python matrix multiply example.
- eRPC: Added nested call support.
- eRPC: Added struct member “byref” option support.
- eRPC: Added support of forward declarations of structures
- eRPC: Added Python RPMsg Multiendpoint kernel module support
- eRPC: Added eRPC sniffer tool

### 1.4.0

### Added

- eRPC: New RPMsg-Lite Zero Copy (RPMsgZC) transport layer.

### Fixed

- eRPC: win\_flex\_bison.zip for windows updated.
- eRPC: Use one codec (instead of inCodec outCodec).

### [1.3.0]

### Added

- eRPC: New annotation types introduced (@length, @max\_length, ...).
- eRPC: Support for running both erpc client and erpc server on one side.
- eRPC: New transport layers for (LP)UART, (D)SPI.
- eRPC: Error handling support.

### [1.2.0]

### Added

- eRPC source directory organization changed.
- Many eRPC improvements.

**[1.1.0]**

**Added**

- Multicore SDK 1.1.0 ported to KSDK 2.0.0.

**[1.0.0]**

**Added**

- Initial Release



# 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**