MCUXpresso SDK Documentation

Release 25.06.00

NXP
Jun 26, 2025

Table of contents

1 EVK-MIMX8MP **3**
- 1.1 Overview . . . 3
- 1.2 Getting Started with MCUXpresso SDK Package . . . 3
  - 1.2.1 Getting Started with Package . . . 3
- 1.3 Getting Started with MCUXpresso SDK GitHub . . . 30
  - 1.3.1 Getting Started with MCUXpresso SDK Repository . . . 30
- 1.4 Release Notes . . . 43
  - 1.4.1 MCUXpresso SDK Release Notes . . . 43
- 1.5 ChangeLog . . . 46
  - 1.5.1 MCUXpresso SDK Changelog . . . 46
- 1.6 Driver API Reference Manual . . . 101
- 1.7 Middleware Documentation . . . 101
  - 1.7.1 Multicore . . . 101
  - 1.7.2 FreeMASTER . . . 102
  - 1.7.3 FreeRTOS . . . 102

2 MIMX8ML8 **103**
- 2.1 AIPSTZ: AHB to IP Bridge . . . 103
- 2.2 ASRC: Asynchronous sample rate converter . . . 105
- 2.3 ASRC Driver . . . 105
- 2.4 ASRC SDMA Driver . . . 119
- 2.5 Audiomix . . . 123
- 2.6 CACHE: ARMV7-M7 CACHE Memory Controller . . . 129
- 2.7 Clock . . . 132
- 2.8 EDMA: Enhanced Direct Memory Access (eDMA) Controller Driver . . . 170
- 2.9 ECSPI: Enhanced Configurable Serial Peripheral Interface Driver . . . 189
- 2.10 ECSPI Driver . . . 189
- 2.11 ECSPI SDMA Driver . . . 202
- 2.12 ENET: Ethernet MAC Driver . . . 205
- 2.13 EQOS-TSN: Ethernet QoS with TSN Driver . . . 232
- 2.14 Enet_qos_qos . . . 232
- 2.15 FlexCAN: Flex Controller Area Network Driver . . . 272
- 2.16 FlexCAN Driver . . . 272
- 2.17 FlexCAN eDMA Driver . . . 294
- 2.18 FLEXSPI: Flexible Serial Peripheral Interface Driver . . . 296
- 2.19 FLEXSPI eDMA Driver . . . 313
- 2.20 GPC: General Power Controller Driver . . . 316
- 2.21 GPIO: General-Purpose Input/Output Driver . . . 320
- 2.22 GPT: General Purpose Timer . . . 324
- 2.23 I2C: Inter-Integrated Circuit Driver . . . 331
- 2.24 I2C Driver . . . 331
- 2.25 Iomuxc_driver . . . 343
- 2.26 IRQSTEER: Interrupt Request Steering Driver . . . 369
- 2.27 ISI: Image Sensing Interface . . . 374
- 2.28 Common Driver . . . 389
- 2.29 LCDIF: LCD interface . . . 401

2.30 MU: Messaging Unit . . . 413
2.31 OCOTP: On Chip One-Time Programmable controller. . . . 421
2.32 PDM: Microphone Interface . . . 424
2.33 PDM Driver . . . 424
2.34 PDM EDMA Driver . . . 443
2.35 PDM SDMA Driver . . . 447
2.36 PWM: Pulse Width Modulation Driver . . . 449
2.37 RDC: Resource Domain Controller . . . 456
2.38 RDC_SEMA42: Hardware Semaphores Driver . . . 462
2.39 SAI: Serial Audio Interface . . . 464
2.40 SAI Driver . . . 464
2.41 SAI EDMA Driver . . . 487
2.42 SAI SDMA Driver . . . 494
2.43 SDMA: Smart Direct Memory Access (SDMA) Controller Driver . . . 497
2.44 SEMA4: Hardware Semaphores Driver . . . 515
2.45 SNVS: Secure Non-Volatile Storage . . . 518
2.46 Secure Non-Volatile Storage High-Power . . . 518
2.47 Secure Non-Volatile Storage Low-Power . . . 527
2.48 SRC: System Reset Controller Driver . . . 533
2.49 TMU: Thermal Management Unit Driver . . . 536
2.50 UART: Universal Asynchronous Receiver/Transmitter Driver . . . 541
2.51 UART Driver . . . 541
2.52 UART FreeRTOS Driver . . . 556
2.53 UART SDMA Driver . . . 556
2.54 USDHC: Ultra Secured Digital Host Controller Driver . . . 559
2.55 WDOG: Watchdog Timer Driver . . . 586

3 Middleware **591**
3.1 Motor Control . . . 591
3.1.1 FreeMASTER . . . 591

4 RTOS **629**
4.1 FreeRTOS . . . 629
4.1.1 FreeRTOS kernel . . . 629
4.1.2 FreeRTOS drivers . . . 629
4.1.3 backoffalgorithm . . . 629
4.1.4 corehttp . . . 629
4.1.5 corejson . . . 629
4.1.6 coremqtt . . . 630
4.1.7 coremqtt-agent . . . 630
4.1.8 corepkcs11 . . . 630
4.1.9 freertos-plus-tcp . . . 630

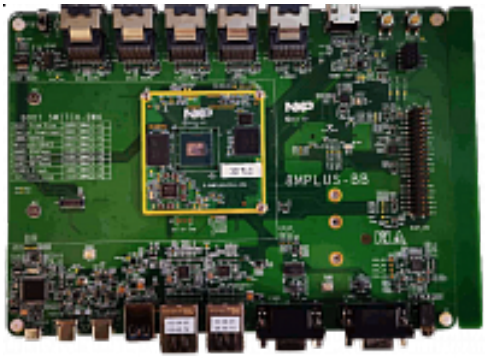
This documentation contains information specific to the evkmimx8mp board.

Chapter 1

EVK-MIMX8MP

1.1 Overview

The i.MX 8M Plus family of boards provides a powerful and flexible development system for NXP's Cortex-M7 MCUs.



MCU device and part on board is shown below:

- Device: MIMX8ML8
- PartNumber: MIMX8ML8DVNLZ

1.2 Getting Started with MCUXpresso SDK Package

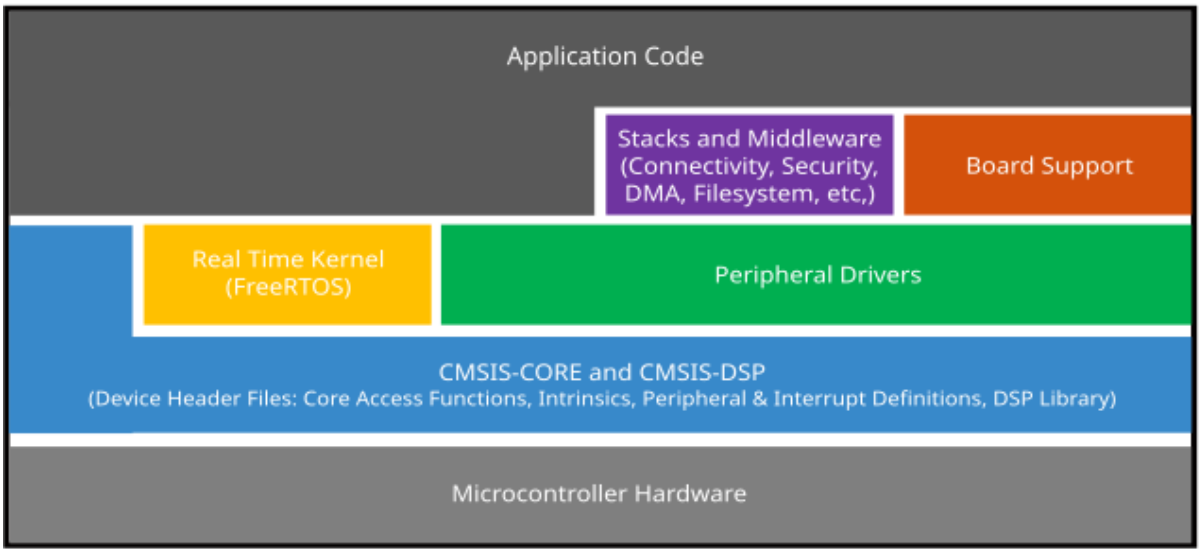
1.2.1 Getting Started with Package

Overview

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

For supported toolchain versions, see *MCUXpresso SDK Release Notes for EVK-MIMX8MP* (document MCUXSDKIMX8MPRN).

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

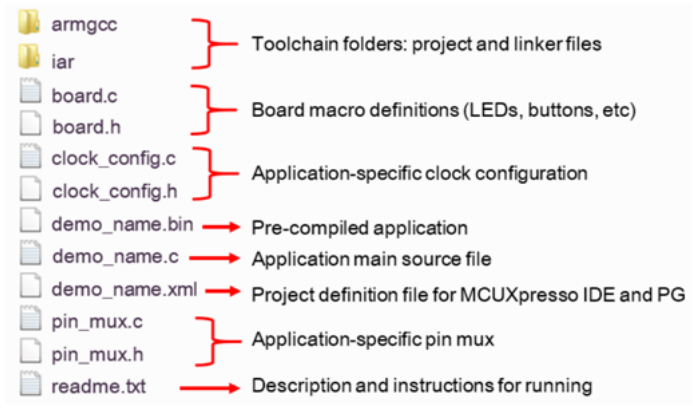
MCUXpresso SDK board support provides example applications for NXP development and evaluation boards for Arm Cortex-M cores. Board support packages are found inside of the top level boards folder, and each supported board has its own folder (MCUXpresso SDK package can support multiple boards). Within 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 FreeRTOSTM 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/<devices_name>/project: Project template used in CMSIS PACK new project creation

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

Parent topic:*MCUXpresso SDK board support folders*

Toolchain introduction

The MCUXpresso SDK release for i.MX 8M Plus 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 8M Plus 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.

Compiler/Debugger	Supported host OS	Debug probe	Tool website
ArmGCC/J-Link GDB server	Windows OS/Linux OS	J-Link Plus	developer.arm.com/open-source/gnu-toolchain/gnu-rm

www.segger.com

| |IAR/J-Link|Windows OS|J-Link Plus|www.iar.com

www.segger.com

|

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

Note:

- To support i.MX 8M Plus, the patch for IAR and Segger J-Link should be installed. The patch named iar_segger_support_patch_imx8mp.zip can be used with MCUXpresso SDK. See the *readme.txt* in the patch for additional information about patch installation.
- For the security reason, only after the U-Boot is boot up, the JTAG can be allowed to connect to M core.

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 8MP 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 Before using IAR, get the IAR and Segger J-Link patch, iar_segger_support_patch_imx8p.zip. Install the i.MX8MP support patch following the guides in readme.txt located in the archive.

Do the following steps to build the hello_world example application.

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

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

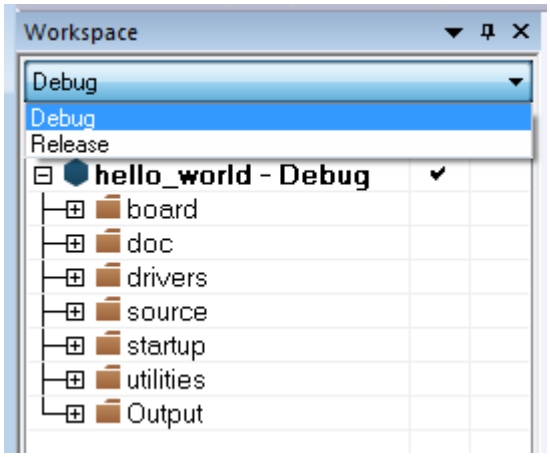
   Using the i.MX 8MP EVK hardware platform as an example, the hello_world workspace is located in:

   ```
   <install_dir>/boards/evkmimx8mp/demo_apps/hello_world/iar/hello_world.eww
   ```

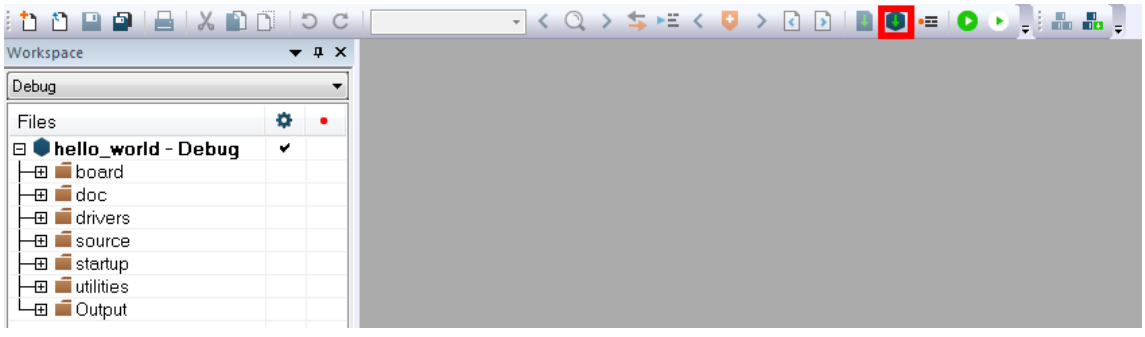
   Other example applications may have additional folders in their path.

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

   For this example, select **hello_world** – **debug**.



3. To build the demo application, click **Make**, highlighted in red in Figure 2.

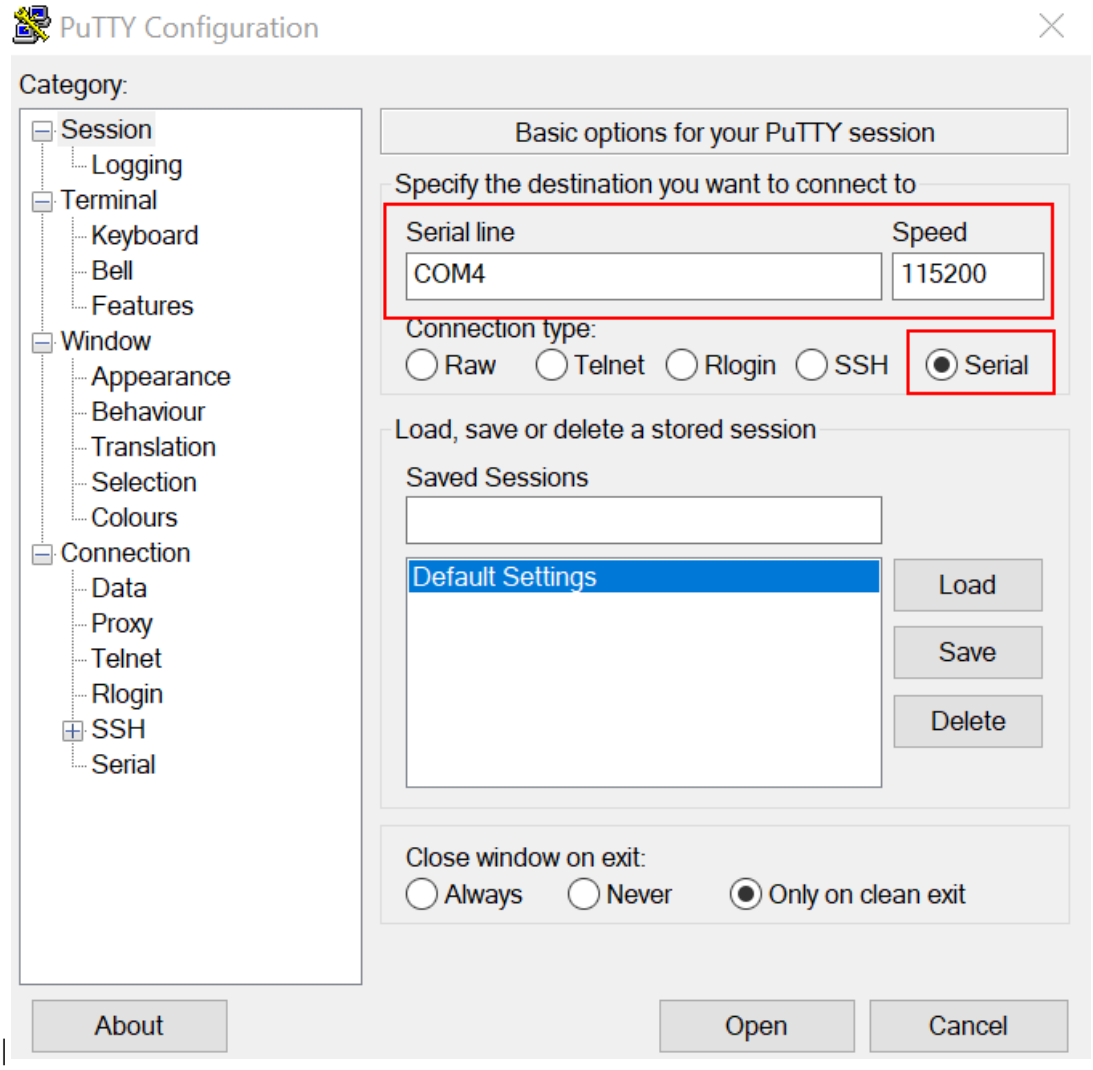


4. The build completes without errors.

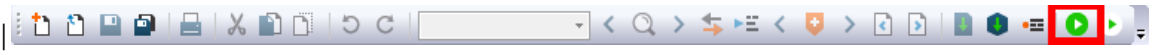
Parent topic:*Run a demo application using IAR*

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

1. This board supports the J-Link PLUS debug probe. Before using it, install SEGGER J-Link software, which can be downloaded from http://www.segger.com/downloads/jlink/.
2. Connect the development platform to your PC via USB cable between the USB-UART MICRO USB connector and the PC USB connector, then connect 12 V power supply and J-Link Plus to the device.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (to determine the COM port number, see *How to determine COM port*). Configure the terminal with these settings:
   1. 115200 baud rate
   2. No parity
   3. 8 data bits
   4. 1 stop bit



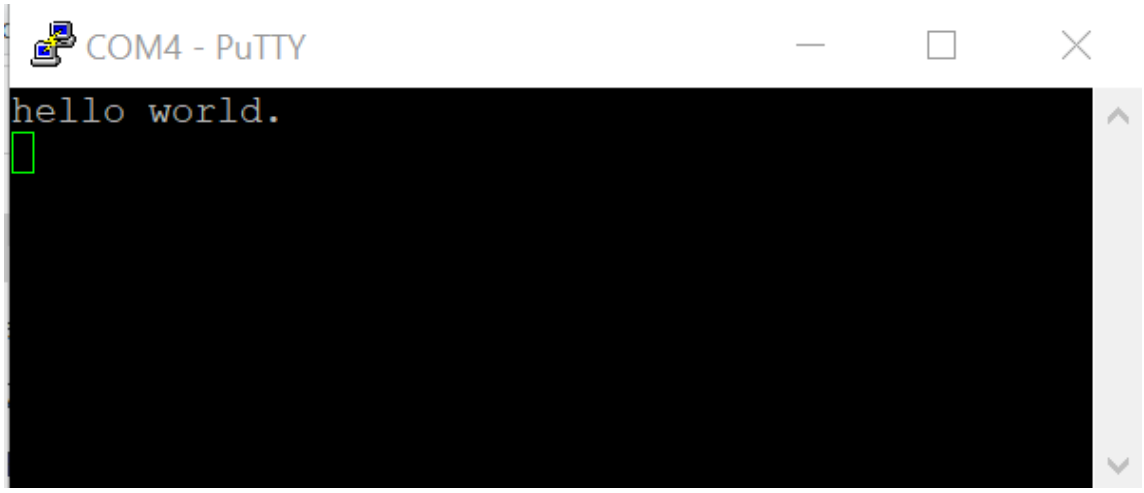
4. In IAR, click **Download and Debug** to download the application to the target.



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

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

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



Note: For downloading the DDR target application, insert one TF card with U-Boot code. This requires both on IAR and GCC.

Parent topic:*Run a demo application using IAR*

Run a demo using Arm GCC

This section describes the steps to configure the command line Arm GCC tools to build, run, and debug demo applications and necessary driver libraries provided in the MCUXpresso SDK. The hello_world demo application targeted for i.MX 8M Plus is used as an example, though these steps can be applied to any board, demo or example application in the MCUXpresso SDK.

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

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

Install GCC Arm embedded tool chain Download and run the installer from launchpad.net/gcc-arm-embedded. This is the actual toolset (in other words, compiler, linker, and so on). The GCC toolchain should correspond to the latest supported version, as described in the *MCUXpresso SDK Release Notes* (document MCUXSDKRN).

Note: See *Host setup* for Linux OS before compiling the application.

Parent topic: Set up toolchain

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

```
$ export ARMGCC_DIR=/work/platforms/tmp/gcc-arm-none-eabi-9-2019-q4-major
```

```
$ export PATH= $PATH:/work/platforms/tmp/gcc-arm-none-eabi-9-2019-q4-major
```

Parent topic: Set up toolchain

Parent topic: Linux OS host

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

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

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

   For this example, the exact path is: <install_dir>/boards/evkmimx8mp/demo_apps/hello_world/armgcc

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

```
$ ./build_debug.sh
-- TOOLCHAIN_DIR: /work/platforms/tmp/gcc-arm-none-eabi-9-2019-q4-major
-- BUILD_TYPE: debug
-- TOOLCHAIN_DIR: /work/platforms/tmp/gcc-arm-none-eabi-9-2019-q4-major
-- BUILD_TYPE: debug
-- The ASM compiler identification is GNU
-- Found assembler: /work/platforms/tmp/gcc-arm-none-eabi-8-2019-q3-update/bin/arm-none-eabi-gcc
-- Configuring done
-- Generating done
-- Build files have been written to:
/work/platforms/tmp/nxp/SDK\_2.8.1\_EVK\_MIMX8MP/boards/evkmimx8mp/demo\_apps/hello\
↪_world/armgcc
Scanning dependencies of target hello_world.elf
\[ 6%\] Building C object CMakeFiles/hello\_world.elf.dir/work/platforms/tmp/nxp/SDK\_2.8.1\_
↪EVK\_MIMX8MP/boards/evkmimx8mp/demo\_apps/hello\_world/hello\_world.c.obj
 < -- skipping lines -- >
[100%] Linking C executable debug/hello_world.elf
[100%] Built target hello_world.elf
```

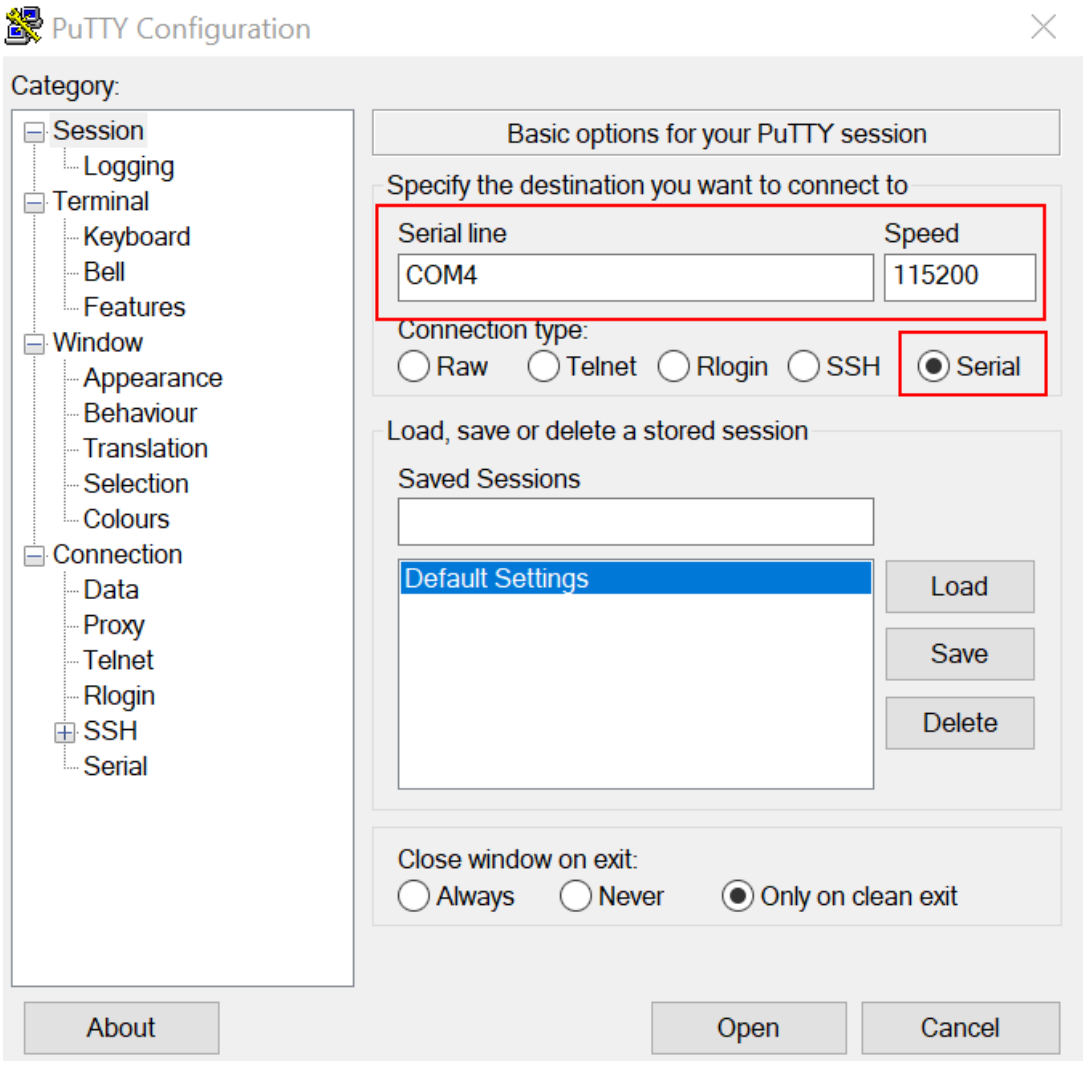
Parent topic: Linux OS host

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

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

1. Connect the development platform to your PC via USB cable between the USB-UART connector and the PC USB connector. If using a standalone J-Link debug pod, also connect it to the SWD/JTAG connector of the board.
2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see *How to determine COM port*). Configure the terminal with these settings:
   1. 115200 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the board.h file)

2. No parity
3. 8 data bits
4. 1 stop bit



3. Open the J-Link GDB Server application. Assuming the J-Link software is installed, the application can be launched from a new terminal for the MIMX8ML_M7 device:

```
$ JLinkGDBServer -if JTAG -device
SEGGER J-Link GDB Server  Command Line Version
JLinkARM.dll
Command line: -if JTAG -device MIMX8ML8\_M7
-----GDB Server start settings-----
GDBInit file: none
GDB Server Listening port: 2331
SWO raw output listening port: 2332
Terminal I/O port: 2333
Accept remote connection: yes
< -- Skipping lines -- >
Target connection timeout: 0 ms
------J-Link related settings------
J-Link Host interface: USB
J-Link script: none
J-Link settings file: none
```

(continues on next page)

(continued from previous page)

```
------Target related settings------
Target device:
Target interface: JTAG
Target interface speed: 1000 kHz
Target endian: little
Connecting to J-Link...
J-Link is connected.
Firmware: J-Link V10 compiled Feb 2 2020 18:12:40
Hardware: V10.10
S/N: 600109545
Feature(s): RDI, FlashBP, FlashDL, JFlash, GDB
Checking target voltage...
Target voltage: 1.82 V
Listening on TCP/IP port 2331
Connecting to target...
J-Link found 1 JTAG device, Total IRLen = 4
JTAG ID: 0x5BA00477 \(Cortex-M7\)
Connected to target
Waiting for GDB connection...
```

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

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

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

   For this example, the path is:

   <install_dir\>/boards/evkmimx8mp/demo_apps/hello_world/armgcc/debug

5. Start the GDB client:

```
$ arm-none-eabi-gdb hello_world.elf
GNU gdb (GNU Tools for Arm Embedded Processors 9-2019-q4-major) 8.3.0.20190709-git
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hello_world.elf...
(gdb)
```

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

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

```
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
```

(continues on next page)

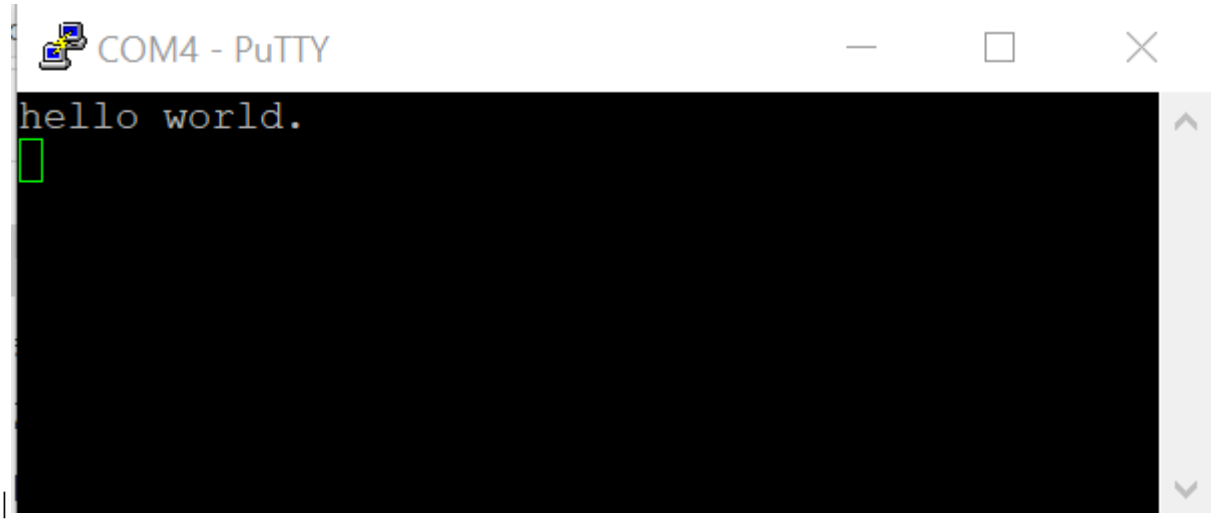
(continued from previous page)

```
0x00000008 in \_\_isr\_vector \(\)
(gdb) monitor reset
Resetting target
(gdb) monitor halt
(gdb) load
Loading section .interrupts, size 0x240 lma 0x0
Loading section .text, size 0x3ab8 lma 0x240
Loading section .ARM, size size 0x8 lma 0x3cf8
Loading section .init\_array, size 0x4 lma 0x3d00
Loading section .fini\_array, size 0x4 lma 0x3d04
Loading section .data, size 0x64 lma 0x3d08
Start address 0x2f4, load size 15724
Transfer rate: 264 KB/sec, 2620 bytes/write.
\(gdb\)
```

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

```
(gdb) monitor go
```

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



Parent topic: Linux OS host

Parent topic:*Run a demo using Arm GCC*

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

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

Install GCC Arm Embedded tool chain Download and run the installer from GNU Arm Embedded Toolchain. This is the actual toolset (in other words, compiler, linker, and so on). The

GCC toolchain should correspond to the latest supported version, as described in *MCUXpresso SDK Release Notes*.

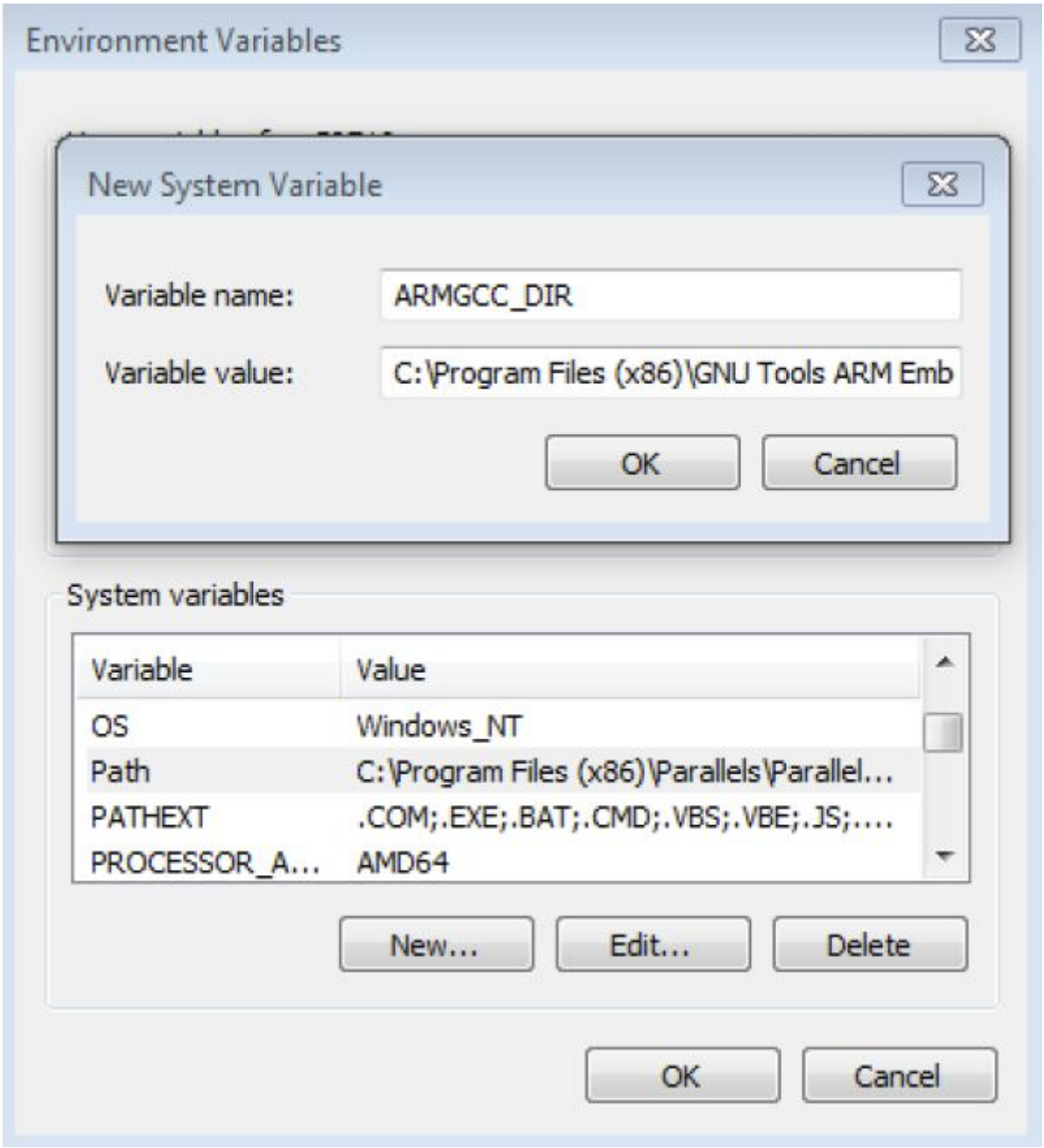
Note: See *Host setup* for Windows OS before compiling the application.

Parent topic: Set up toolchain

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

```
C:\Program Files (x86)\GNU Tools Arm Embedded\9 2019-q4-major
```

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



|

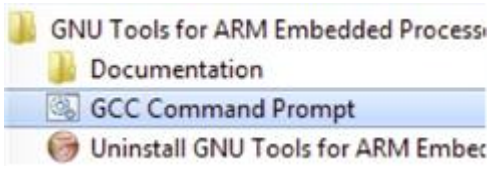
|

Parent topic: Set up toolchain

Parent topic: Windows OS host

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

1. Open a GCC Arm Embedded tool chain command window. To launch the window, from the Windows operating system Start menu, go to **Programs** -> **GNU Tools Arm Embedded <version>** and select **GCC Command Prompt**.



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

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

For this example, the exact path is:

```
<install_dir>/boards/evkmimx8mp/demo_apps/hello_world/armgcc
```

3. Type **build_debug.bat** on the command line or double click on the build_debug.bat file in Windows Explorer to perform the build. The output is as shown in the following figure.

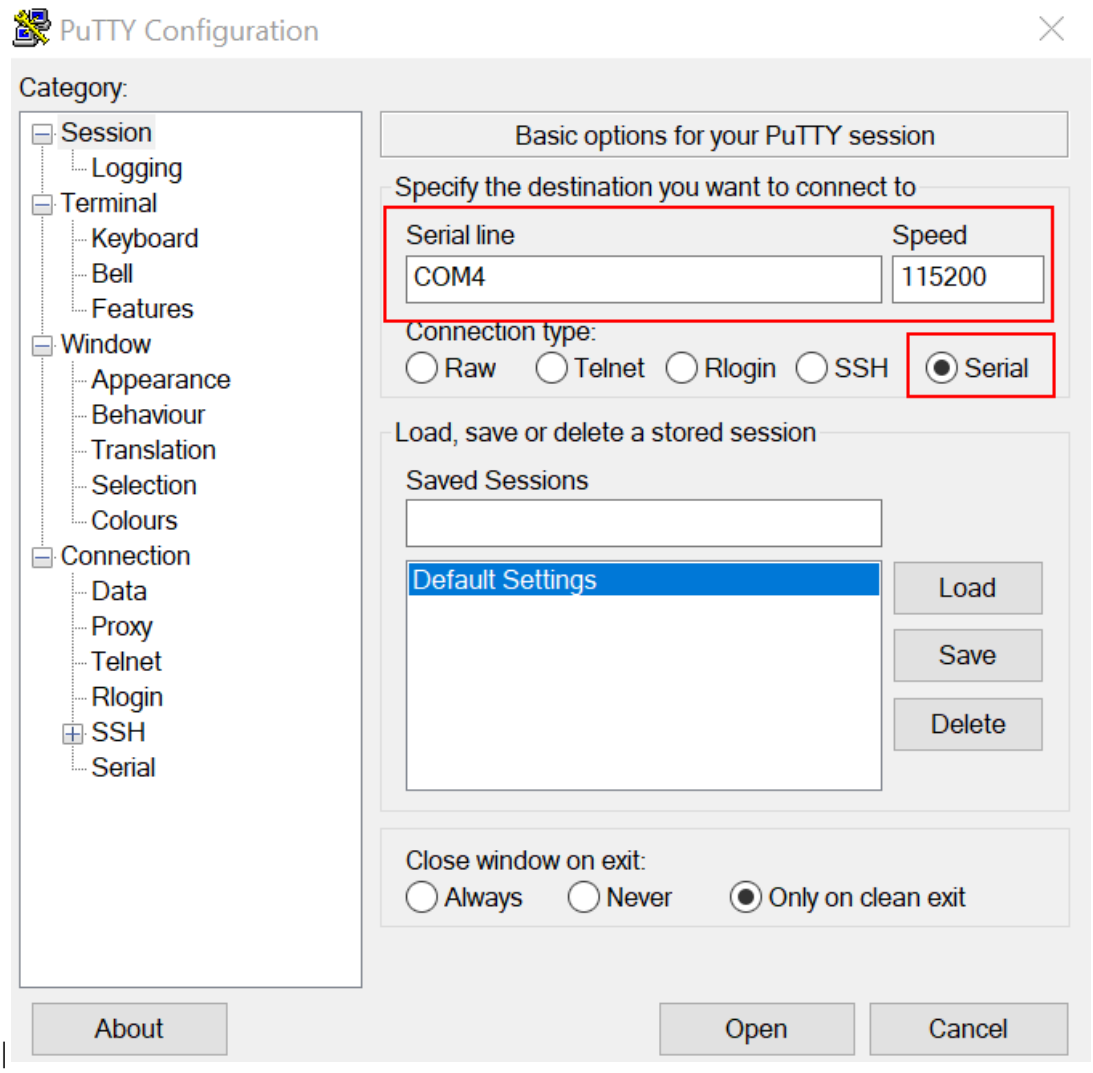
Parent topic: Windows OS host

Run an example application This section describes steps to run a demo application using J-Link GDB Server application. To perform this exercise, the following step must be done:

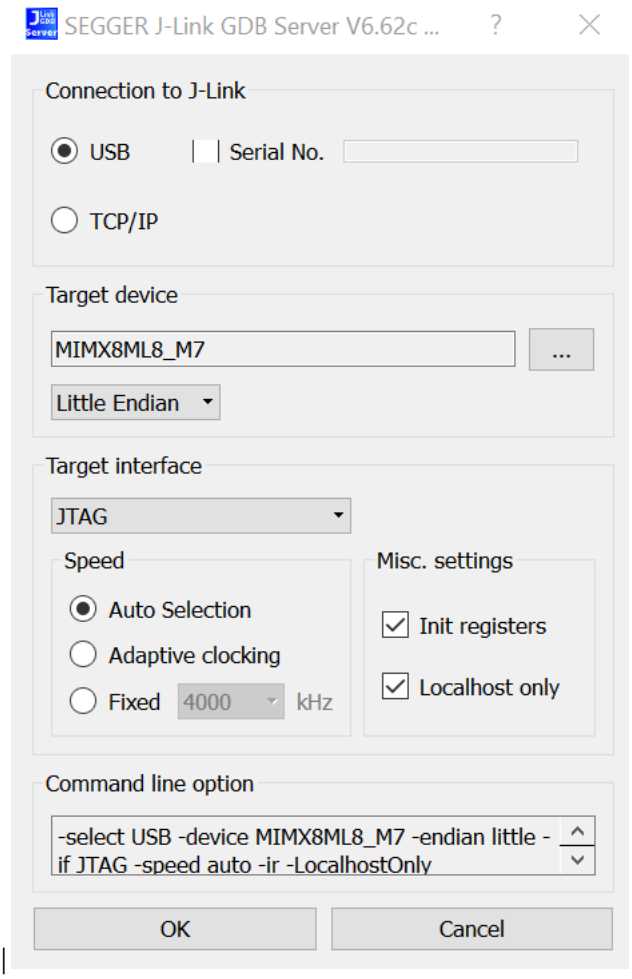
- You have a standalone J-Link pod that is connected to the debug interface of your board.

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

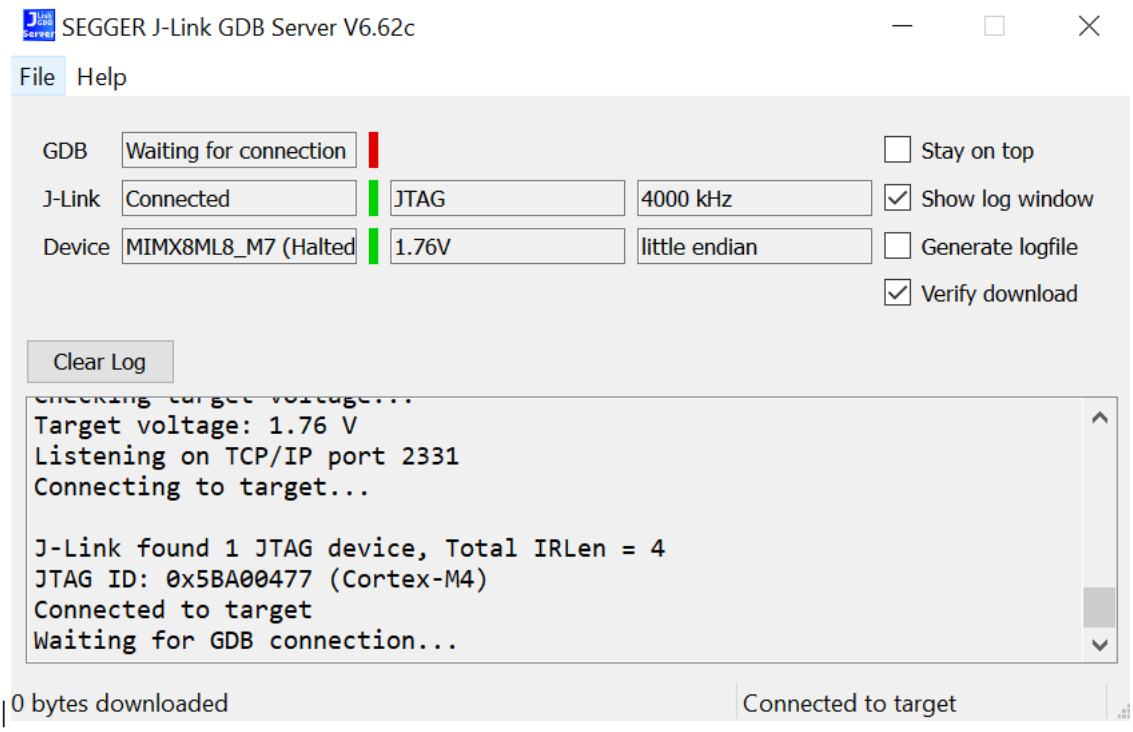
1. Connect the development platform to your PC via USB cable between the USB-UART connector and the PC USB connector. If using a standalone J-Link debug pod, also connect it to the SWD/JTAG connector of the board.
2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see *How to determine COM port*). Configure the terminal with these settings:
    1. 115200 baud rate
    2. No parity
    3. 8 data bits
    4. 1 stop bit



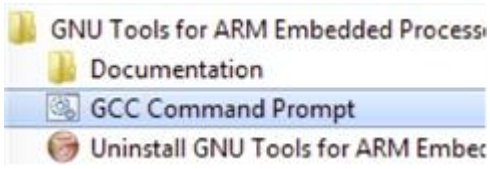
3. Open the J-Link GDB Server application. Assuming the J-Link software is installed, the application can be launched by going to the Windows operating system **Start** menu and selecting **Programs** -> **SEGGER** -> **J-Link <version> J-Link GDB Server**.
4. Modify the settings as shown in Figure 2. The target device selection chosen for this example is MIMX8ML8_M7 .



5. After GDB server is running, the screen should resemble Figure 3:



6. If not already running, open a GCC Arm Embedded tool chain command window. To launch the window, from the Windows operating system **Start** menu, go to **Programs** -> **GNU Tools Arm Embedded <version>** and select **GCC Command Prompt**.



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

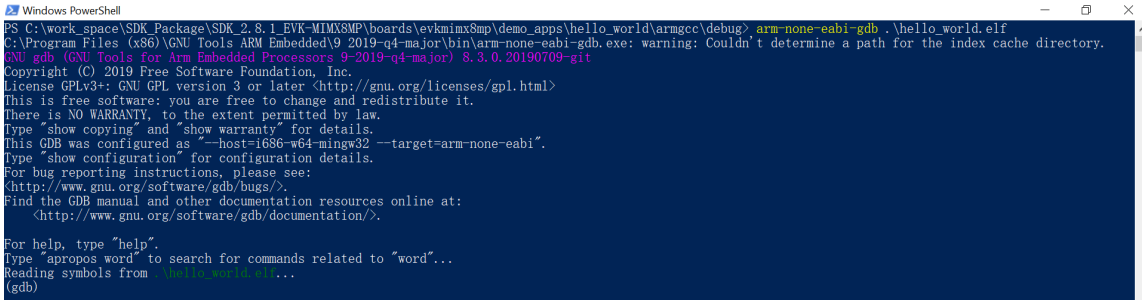
```
<install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc/debug
```

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

For this example, the path is:

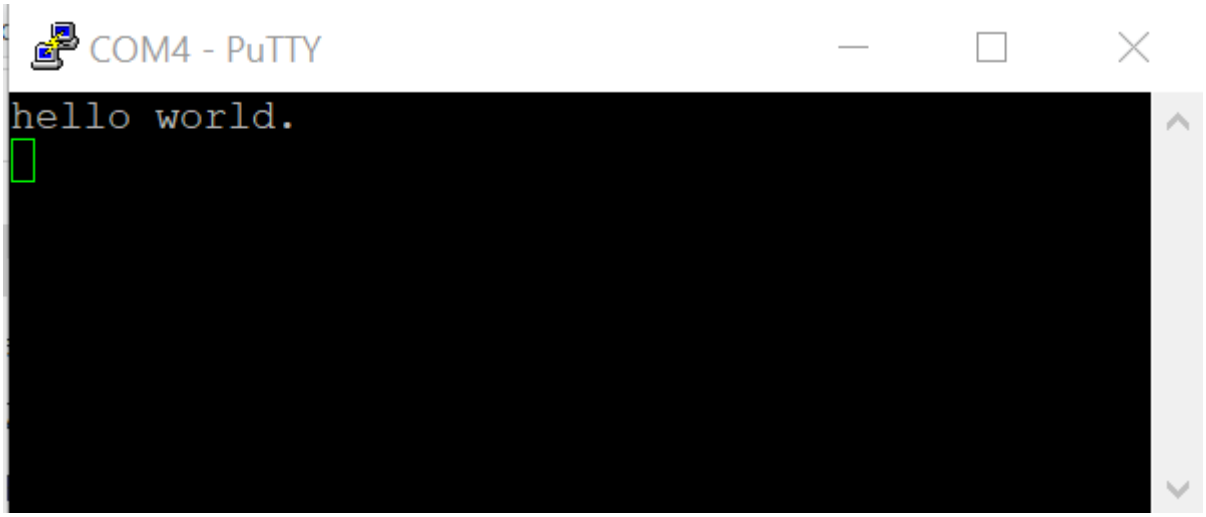
```
<install_dir>/boards/evkmimx8mp/demo_apps/hello_world/armgcc/debug
```

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



9. Run these commands:
    1. target remote localhost:2331
    2. monitor reset
    3. monitor halt
    4. load
10. The application is now downloaded and halted at the reset vector. Execute the monitor go command to start the demo application.

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



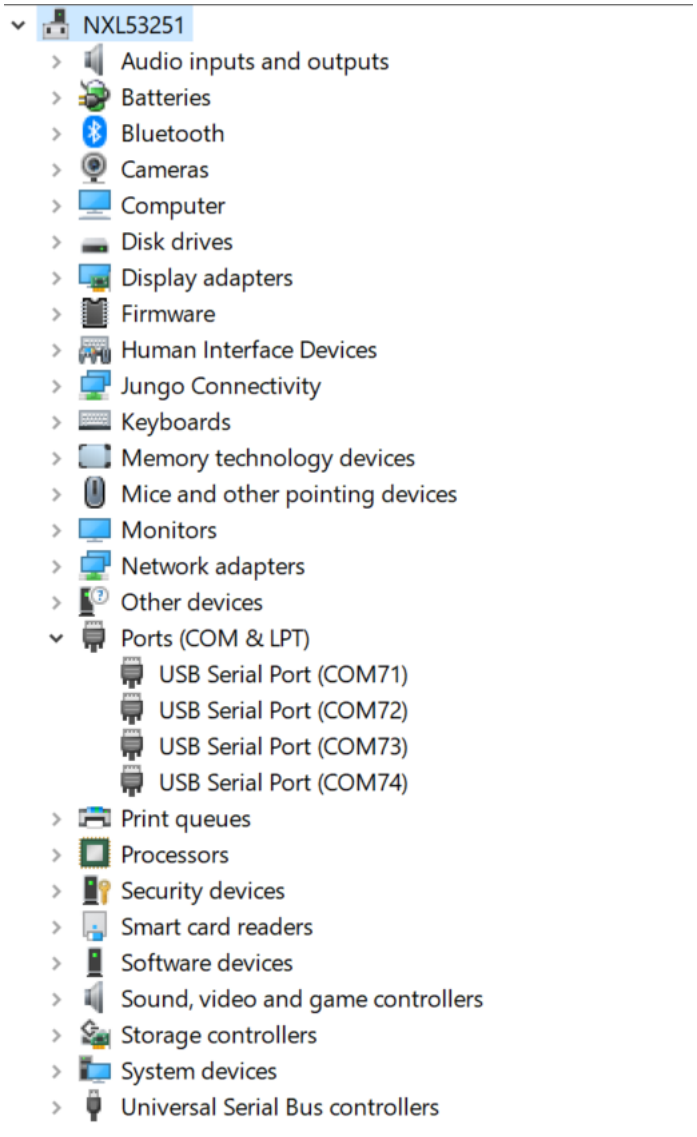
Parent topic: Windows OS host

Parent topic:*Run a demo using Arm GCC*

Running an application by U-Boot

This section describes the steps to write a bootable SDK bin file to TCM, DRAM and QSPI/XIP 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-A53 and COM74 is for the Cortex-M7. The port number is allocated randomly, so opening both is beneficial for development. On Ubuntu OS, find the TTY device with name /dev/ttyUSB* to determine your debug port. Similar to Windows OS, opening both is beneficial for development.



3. Build the application (for example, hello_world) to get the bin file (hello_world.bin).
4. Prepare an SD card with the prebuilt U-Boot image and copy bin file (hello_world.bin) into the SD card. Then, insert the SD card to the target board. Make sure to use the default boot SD slot and check the dipswitch configuration.
5. Open your preferred serial terminals for the serial devices, setting the speed to 115200 bps, 8 data bits, 1 stop bit (115200, 8N1), no parity, then power on the board.
6. Power on the board and hit any key to stop autoboot in the terminals, then enter to U-Boot command line mode. You can then write the image and run it from TCM, DRAM and QSPI/XIP with the following commands:
    1. If the hello_world.bin is made from the debug/release target, which means the binary file will run at TCM, use the following commands to boot:
        - fatload mmc 1:1 0x48000000 hello_world.bin;cp.b 0x48000000 0x7e0000 20000;
        - bootaux 0x7e0000
    2. If the hello_world.bin is made from the ddr_debug/ddr_release target, which means the binary file runs at DRAM, use the following commands:
        - fatload mmc 1:1 0x80000000 hello_world.bin

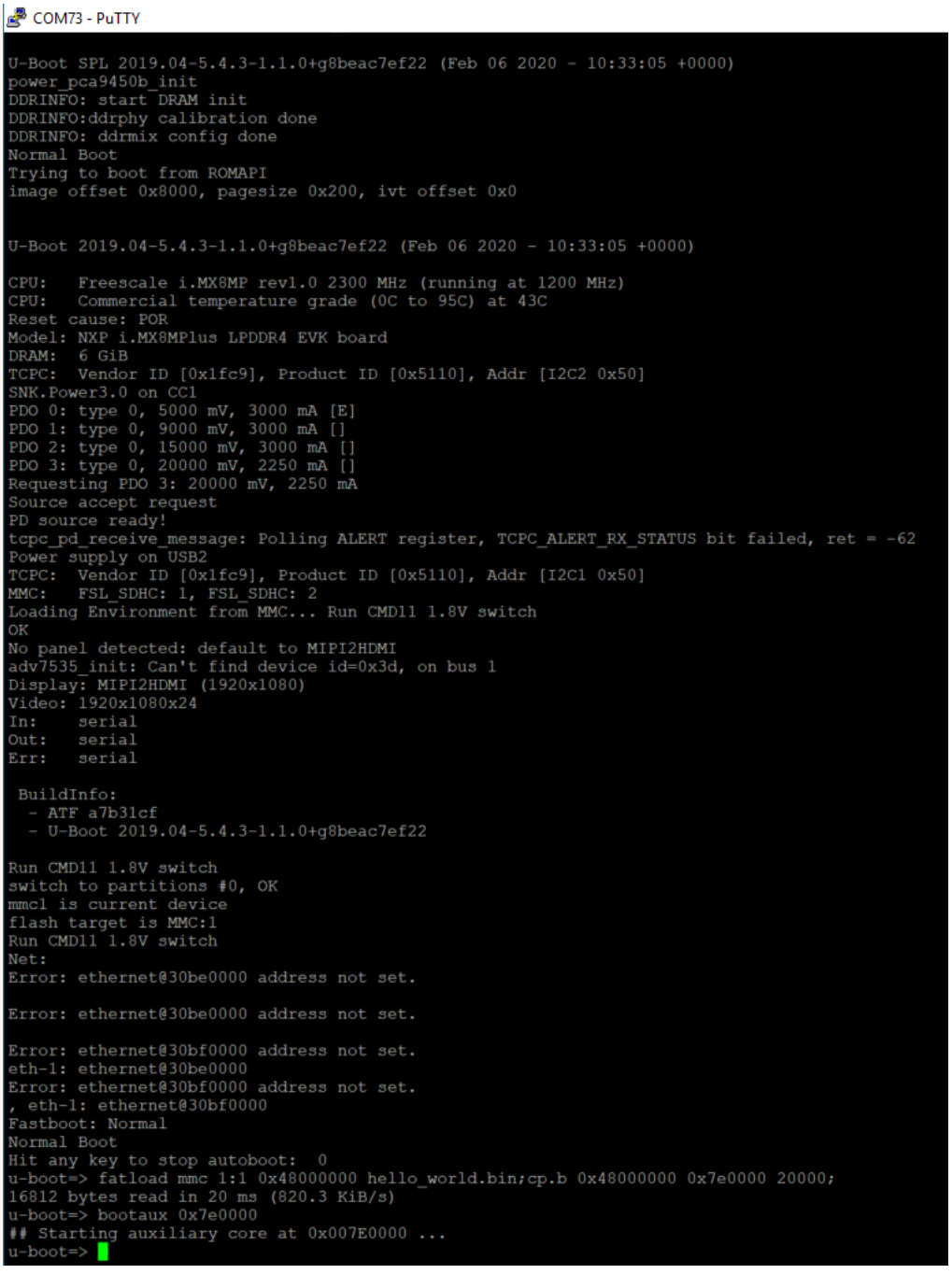
- dcache flush
- bootaux 0x80000000

3. If the hello_world.bin is made from the flash_debug/flash_release target, which means the binary file runs at QSPI/XIP, use the following commands:
   - sf probe
   - sf read 0x80000000 0 4
   - fatload mmc 1:1 0x48000000 flash.bin
   - dcache flush
   - sf erase 0 0x100000
   - sf write 0x48000000 0 0x100000
   - bootaux 0x8000000 **Note:** If the Linux OS kernel runs together with M7, make sure the correct dtb file is used. This dtb file reserves resources used by M7 and avoids the Linux kernel from configuring them. Use the following command in U-Boot before running the kernel:

```
setenv fdtfile 'imx8mp-evk-rpmsg.dtb'
```

Note: For the SCH RevB4 board, use imx8mp-evk-revb4-rpmsg.dtb for multicore examples.

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



|

|

|

|

|

7. Open another terminal application on the PC, such as PuTTY and connect to the debug COM port (to determine the COM port number, see *How to determine COM port*). Configure the terminal with these settings:

- 115200
- No parity
- 8 data bits
- 1 stop bit

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

Run a flash target demo by UUU

This section describes the steps to use the UUU to build and run example applications provided in the MCUXpresso SDK. The hello_world demo application targeted for the i.MX 8M Plus EVK 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.

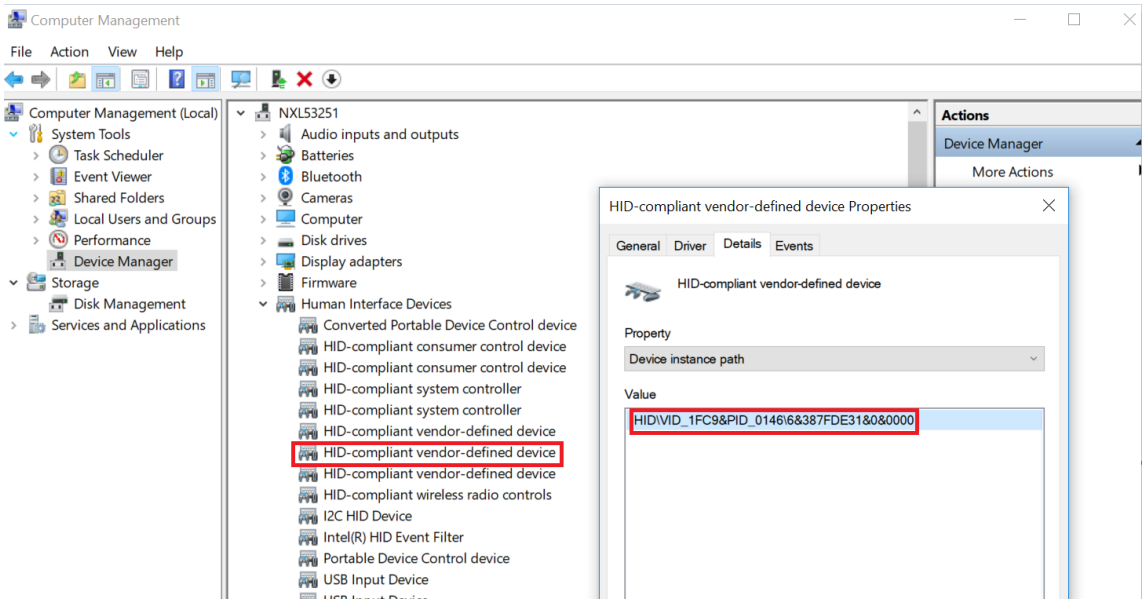
Download the MfgTool The Universal Upgrade Utility (UUU) is an upgraded version of MfgTool. It is a command line tool that aims at installing the bootloader to various storage including SD, QSPI, and so on, for i.MX series devices with ease.

The tool can be downloaded from github. Use version 1.3.134 or higher for full support for the M7 image. 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 [SW0001[1-4]].
2. Connect the development platform to your PC via USB cable between the SERIAL port and the PC USB connector. The SERIAL port is J301 on the base board.
3. The PC recognizes the i.MX 8M Plus device as (VID:PID)=(1FC9:0146).



Parent topic: Set up environment

Parent topic:*Run a flash target demo by 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.

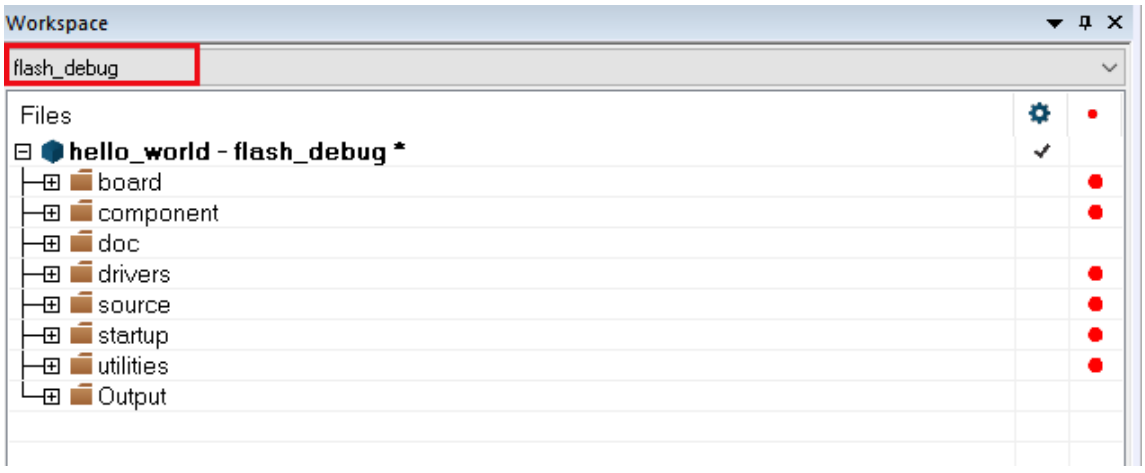
1. If not already done, open the desired demo application workspace. Most example application workspace files can be located using the following path:

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

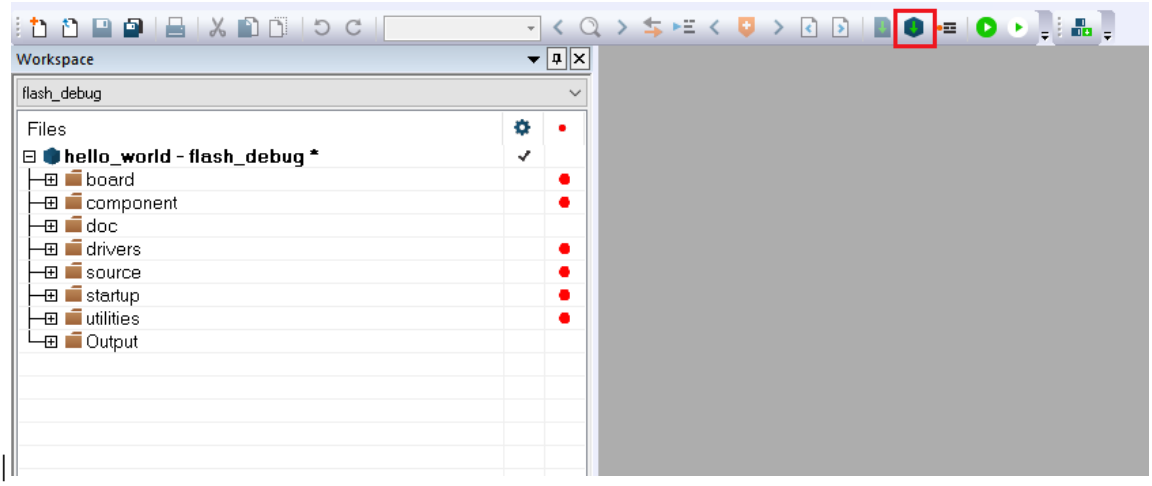
   Using the i.MX 8M Plus EVK board as an example, the workspace is located in:

   ```
   <install_dir>/boards/evkmimx8mp/demo_apps/hello_world/iar/hello_world.eww
   ```

2. Select the desired build target from the drop-down. For this example, select **hello_world – flash_debug**.



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



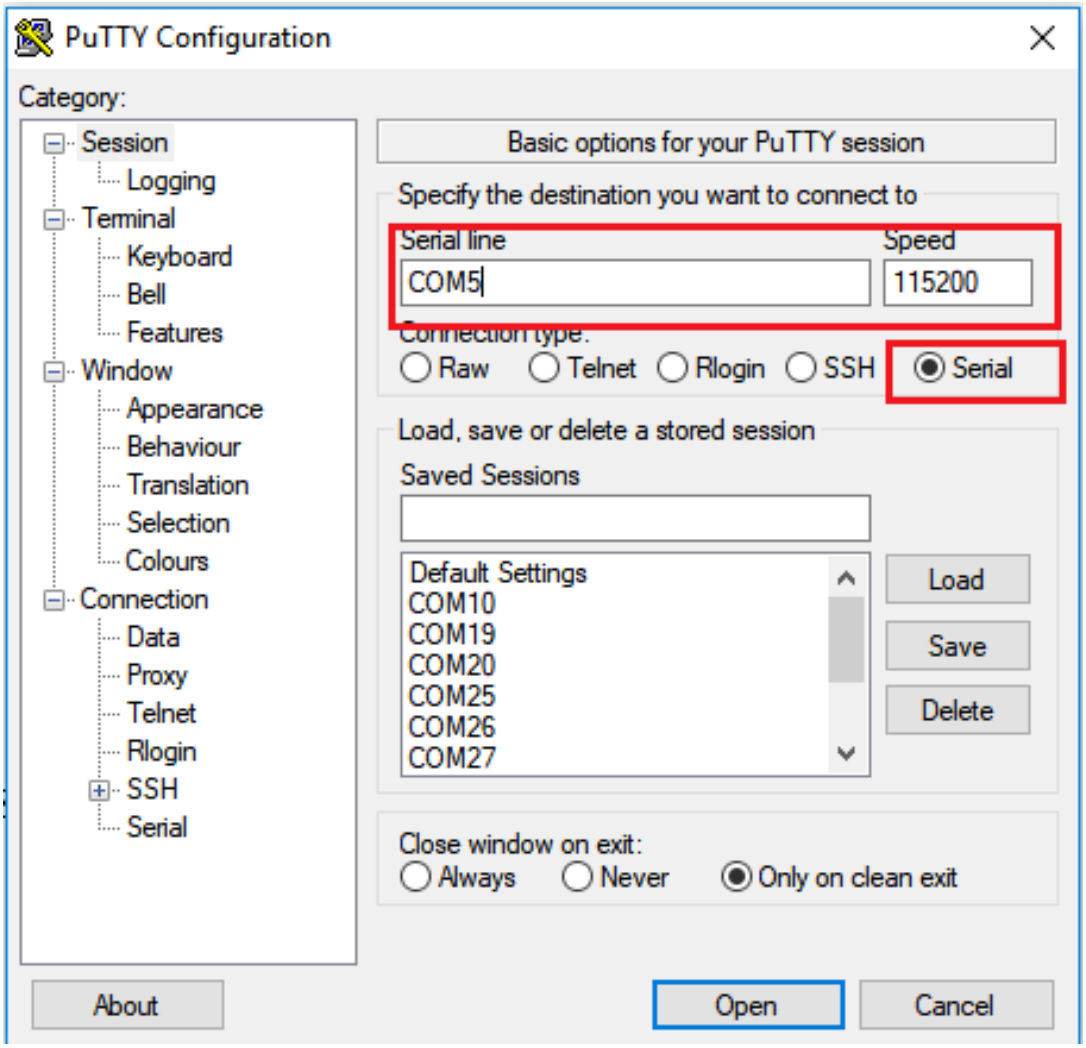
4. The build completes without errors.
5. Copy it to the UUU tool directory.

Parent topic:*Run a flash target demo by 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 J23 USB DEBUG connector and the PC. It provides console output while using UUU.
2. Connect the J6 USB Type-C 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 |



|

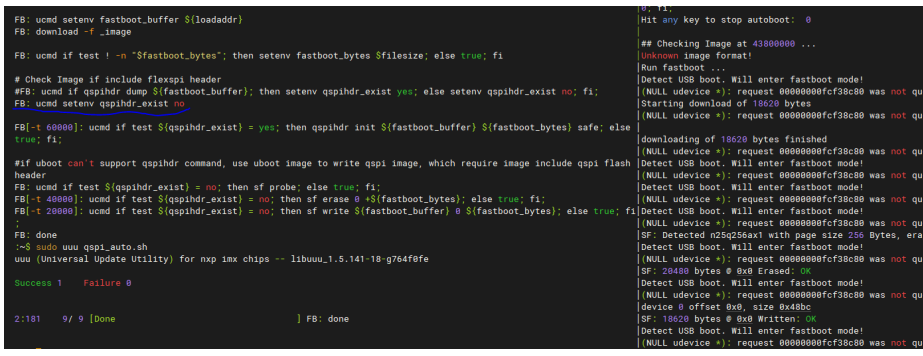
4. Get the fspi version U-Boot image named **imx-boot-imx8mpevk-fspi.bin-flash_evk_flexspi** from the linux release package.
5. In the command line, execute uuu to get script for qspi: uuu -bshow qspi > qspi_auto.sh.

   Then edit qspi_auto.sh, repleace this line:

   FB: ucmd if qspihdr dump ${fastboot_buffer}; then setenv qspihdr_exist yes; else setenv qspihdr_exist no; fi;

   with the below line:

   FB: ucmd setenv qspihdr_exist no |



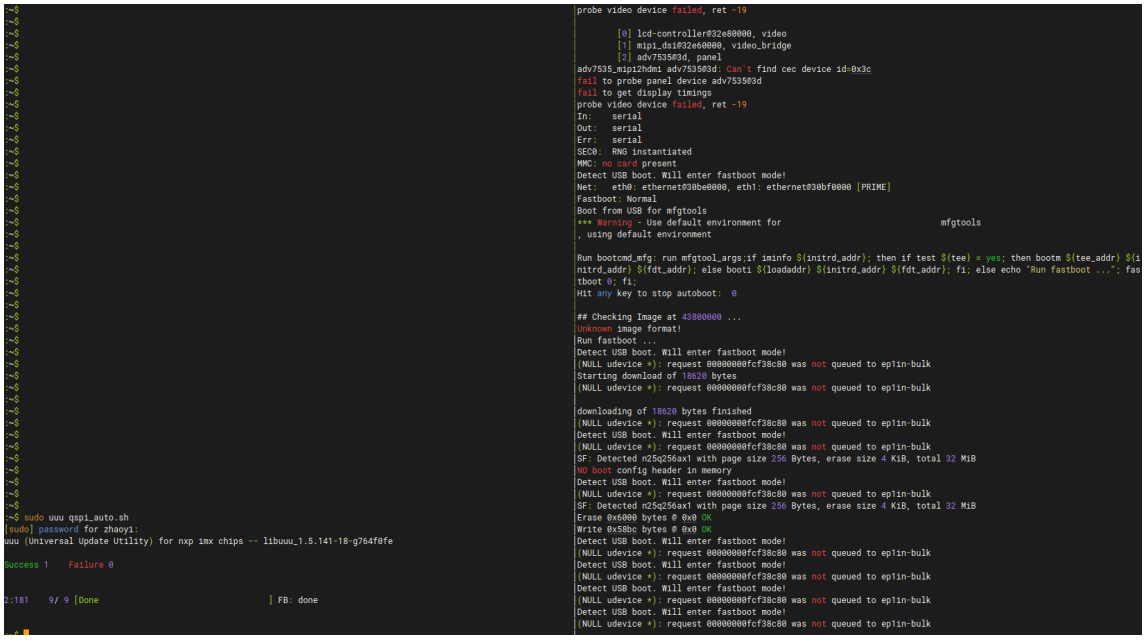
6. rename file imx-boot-imx8mpevk-fspi.bin-flash_evk_flexspi with _flexspi.bin,

   rename hello_world.bin with _image

7. In the command line, execute uuu with the qspi_auto.sh:

   uuu qspi_auto.sh

   The UUU puts the platform into fast boot mode and automatically flashes the target bin to QSPI. The command line and fast boot console is as shown (run_an_example_application.md#COMMANDLINSEFASTBOOT).



8. Then, power off the board and change the boot mode to eMMC/SDHC3 [SW0010[1-4]], and power on the board again.
9. Use following command in U-Boot to kickoff m7:

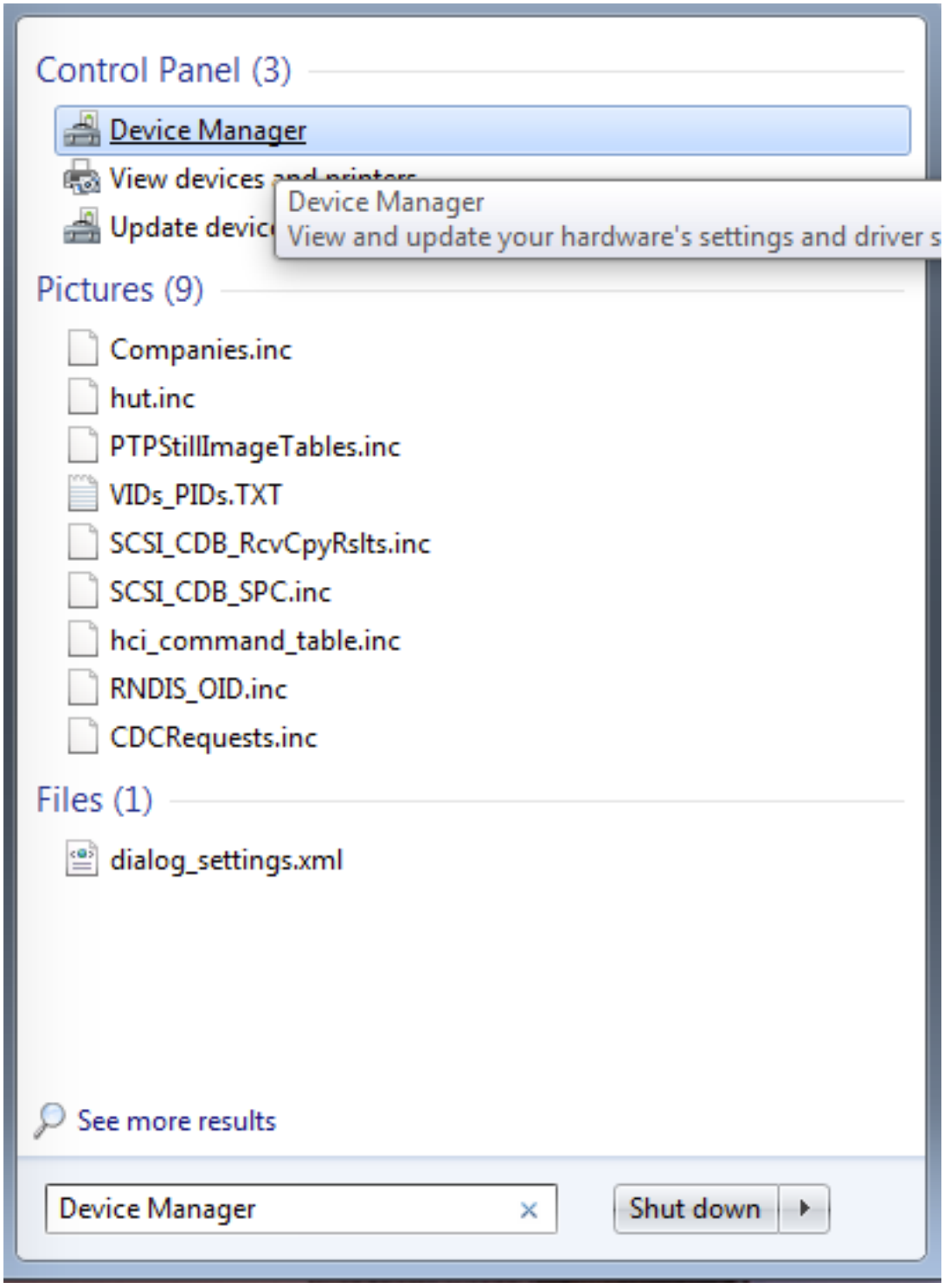
```
sf probe
sf read ${loadaddr} 0 4
bootaux 0x8000000
```

Parent topic:*Run a flash target demo by 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

|

Host setup

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

Linux:

- Cmake

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

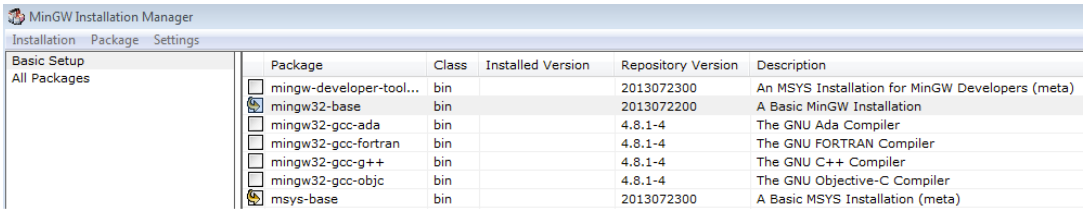
Windows:

- MinGW

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

  1. Download the latest MinGW mingw-get-setup installer from sourceforge.net/projects/mingw/files/Installer/.
  2. Run the installer. The recommended installation path is C:\MinGW, however, you may install to any location.

     Note: The installation path cannot contain any spaces.
  3. Ensure that **mingw32-base** and **msys-base** are selected under **Basic Setup**.



|

4. Click **Apply Changes** in the **Installation** menu and follow the remaining instructions to complete the installation.

   |

|

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

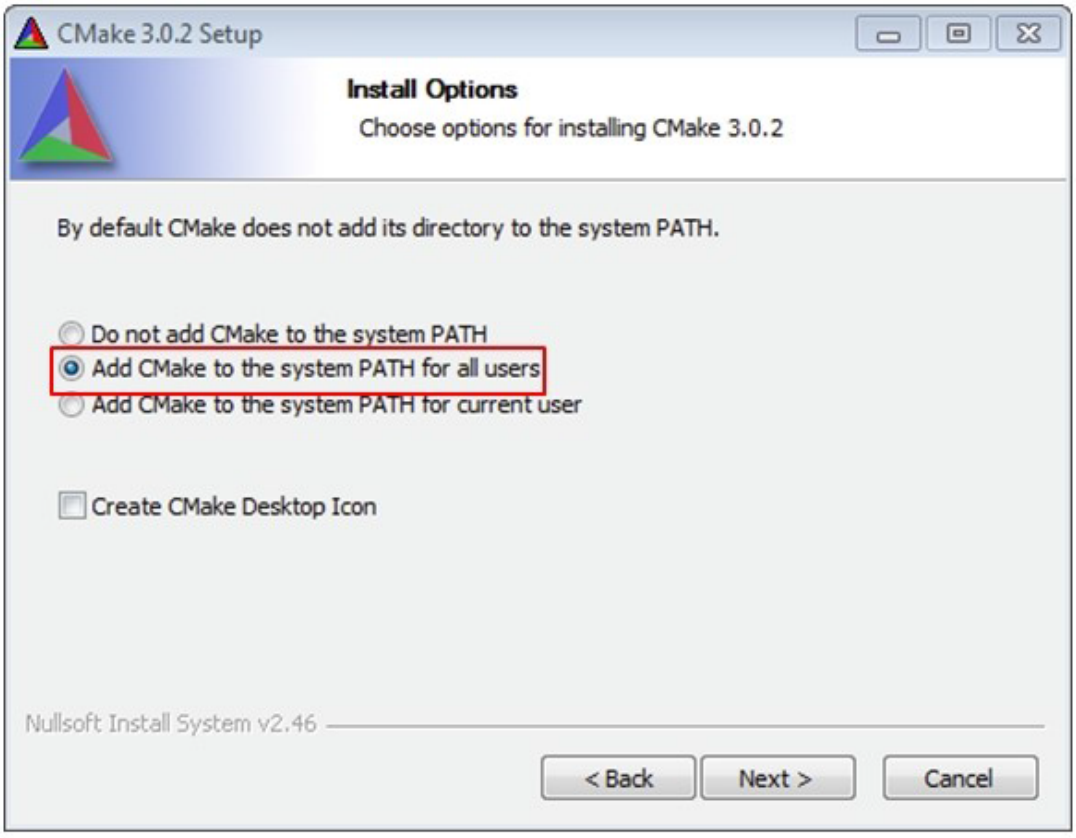
   Assuming the default installation path, `C:\MinGW`, an example is as shown in [Figure 3](host_setup.md#ADDINGPATH). If the path is not set correctly, the toolchain does not work.

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

   |

|

- Cmake
  1. Download CMake 3.0.x from www.cmake.org/cmake/resources/software.html.
  2. Install CMake, ensuring that the option **Add CMake to system PATH** is selected when installing. The user chooses to select whether it is installed into the PATH for all users or just the current user. In this example, it is installed for all users.



3. Follow the remaining instructions of the installer.
4. You may need to reboot your system for the PATH changes to take effect.

1.3 Getting Started with MCUXpresso SDK GitHub

1.3.1 Getting Started with MCUXpresso SDK Repository

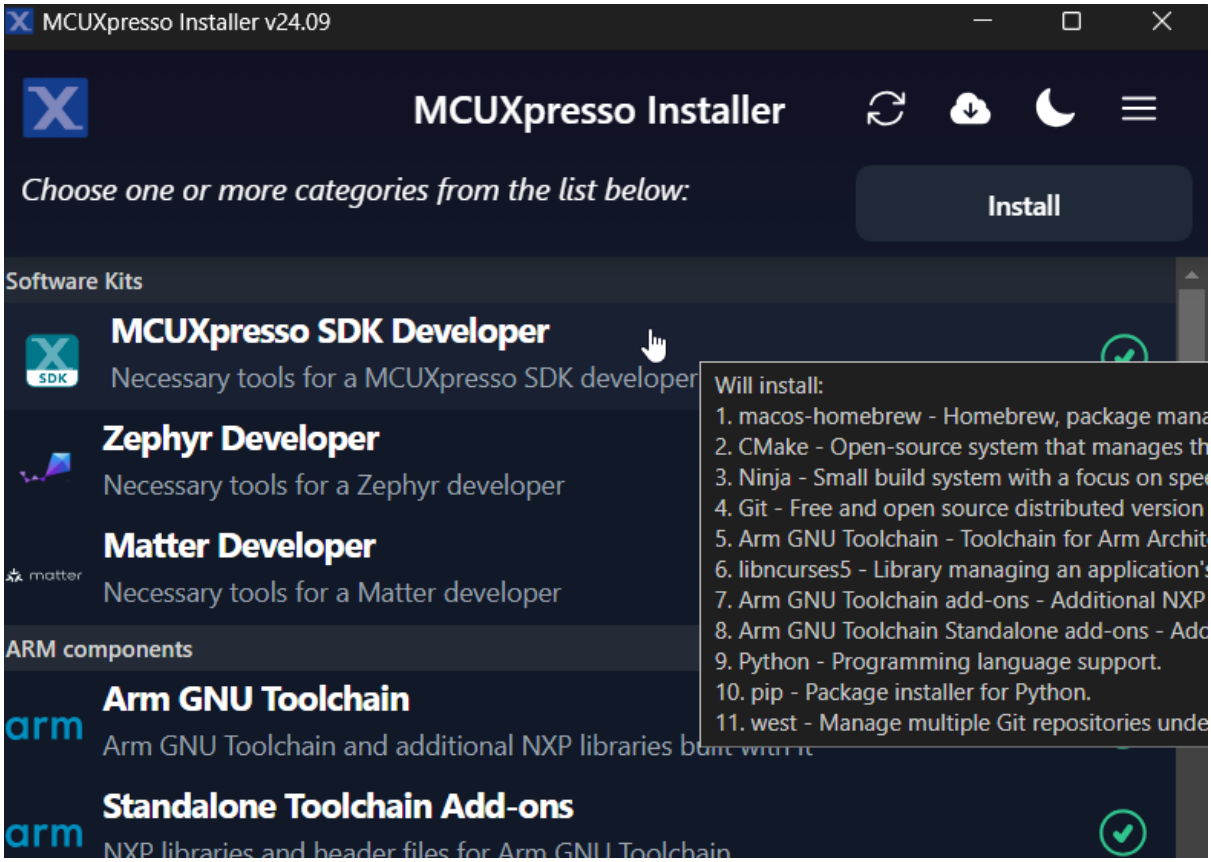
Installation

NOTE

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

Install Prerequisites with MCUXpresso Installer The MCUXpresso Installer offers a quick and easy way to install the basic tools needed. The MCUXpresso Installer can be obtained from https://github.com/nxp-mcuxpresso/vscode-for-mcux/wiki/Dependency-Installation. The MCUXpresso Installer is an automated installation process, simply select MCUXpresso SDK Developer

from the menu and click install. If you prefer to install the basic tools manually, refer to the next section.



Alternative: Manual Installation

Basic tools

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

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

Then configure your username and email using the commands:

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

Python Install python 3.10 or latest. Follow the Python Download guide.

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

West Please use the west version equal or greater than 1.2.0

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

Build And Configuration System

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

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

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

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

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

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

For Linux, you can use your system package manager or you can directly download the ninja binary to work.

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

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

Please make sure *python* environment is setup ready then you can use the Kconfig.

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

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

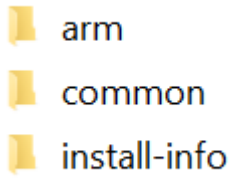
Toolchain MCUXpresso SDK supports all mainstream toolchains for embedded development. You can install your used or interested toolchains following the guides.

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

Get MCUXpresso SDK Repo

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

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

# Update the west projects
cd mcuxpresso-sdk
west update

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

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

1. Navigate to the core directory:

   ```
   cd mcuxsdk
   ```

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

   ```
   python -m venv .venv

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

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

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

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

3. Install the required Python packages:

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

Explore Contents

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

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

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

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

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

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

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

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

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

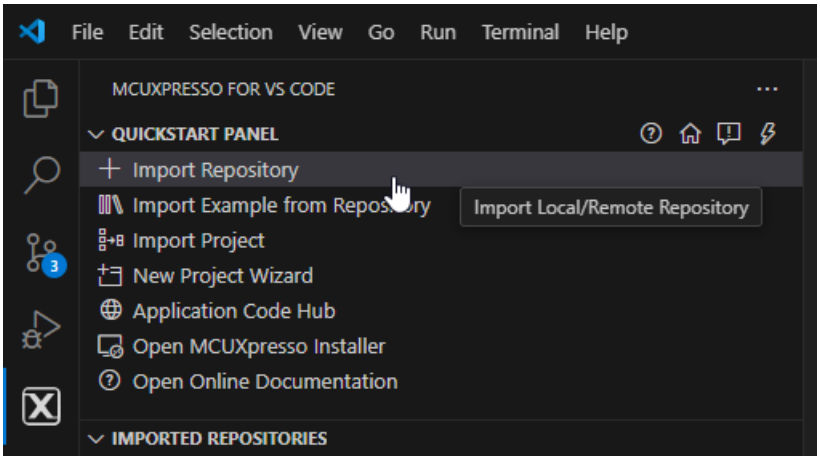
Run a demo using MCUXpresso for VS Code

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

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

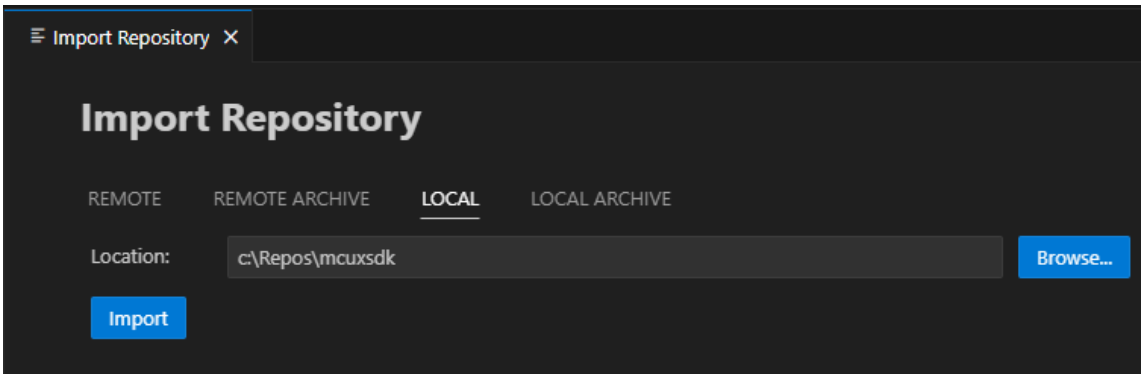
To build an example application:

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

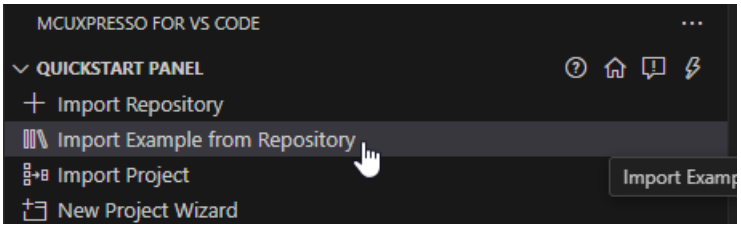


Note: You can import the SDK in several ways. Refer to MCUXpresso for VS Code Wiki for details.

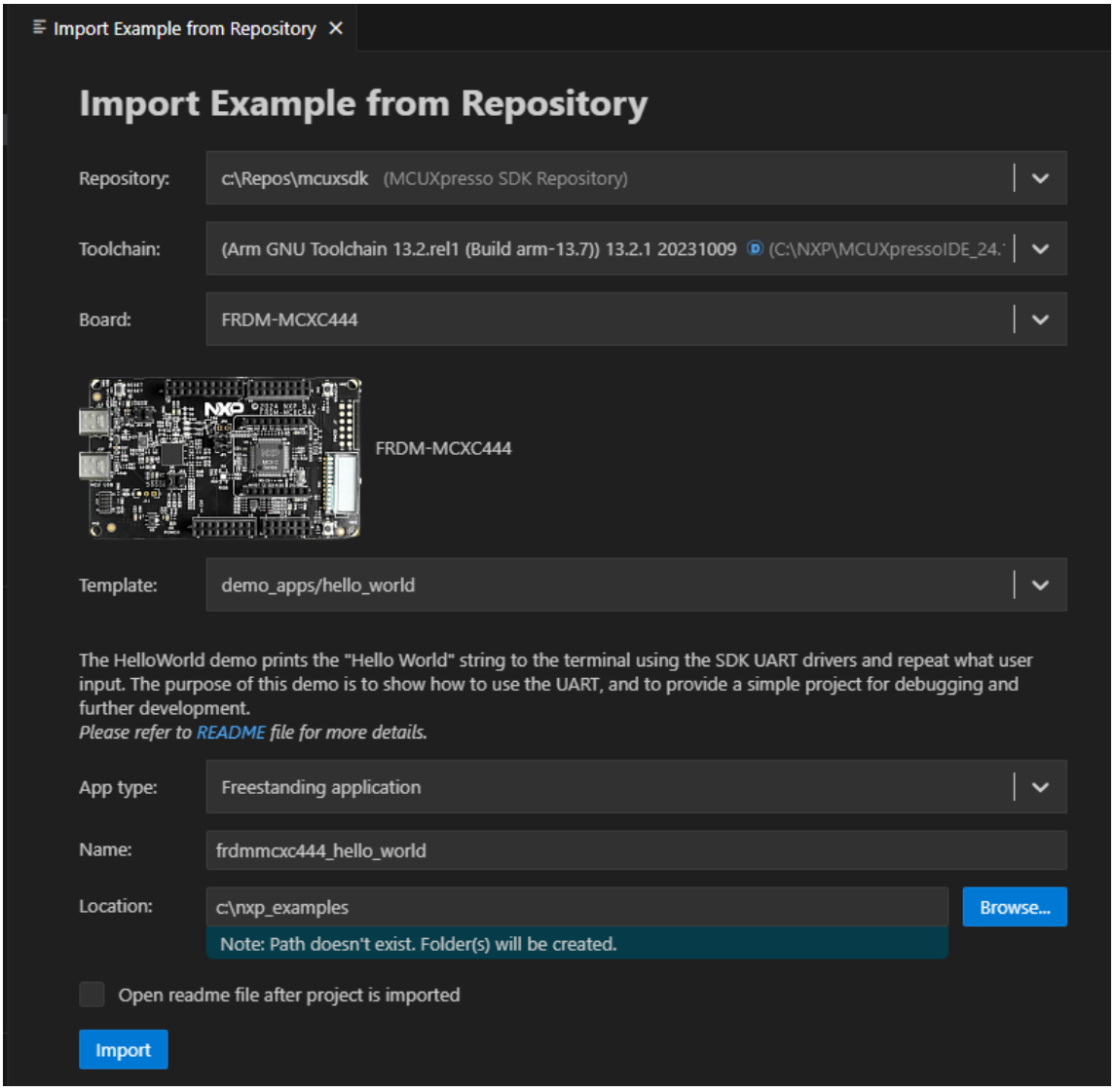
Select **Local** if you've already obtained the SDK as seen in *Get MCUXpresso SDK Repo*. Select your location and click **Import**.



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

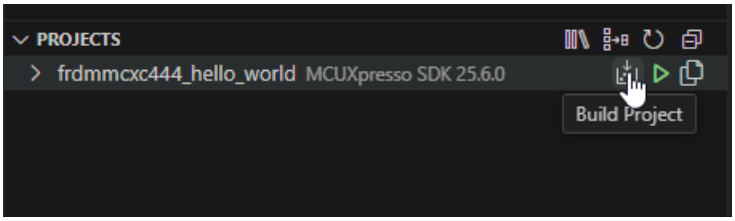


In the dropdown menu, select the MCUXpresso SDK, the Arm GNU Toolchain, your board, template, and application type. Click **Import**.

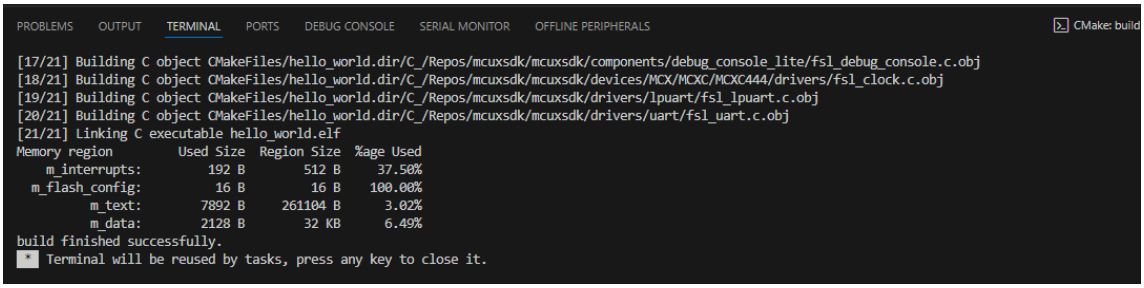


Note: The MCUXpresso SDK projects can be imported as **Repository applications** or **Free-standing applications**. The difference between the two is the import location. Projects imported as Repository examples will be located inside the MCUXpresso SDK, whereas Free-standing examples can be imported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

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

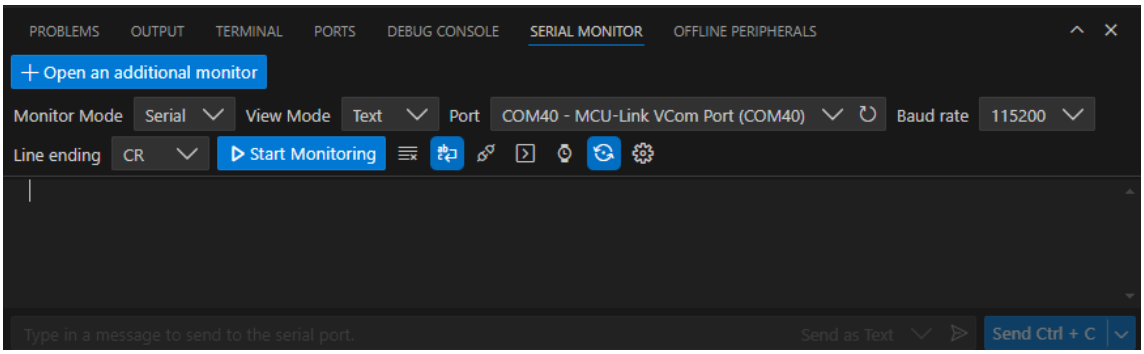


The integrated terminal will open at the bottom and will display the build output.

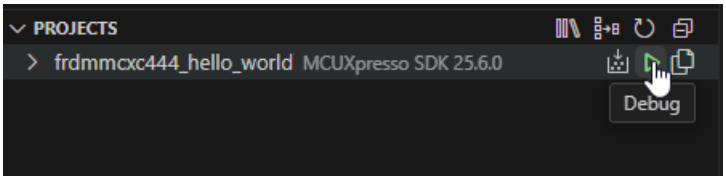


Run an example application **Note:** for full details on MCUXpresso for VS Code debug probe support, see MCUXpresso for VS Code Wiki.

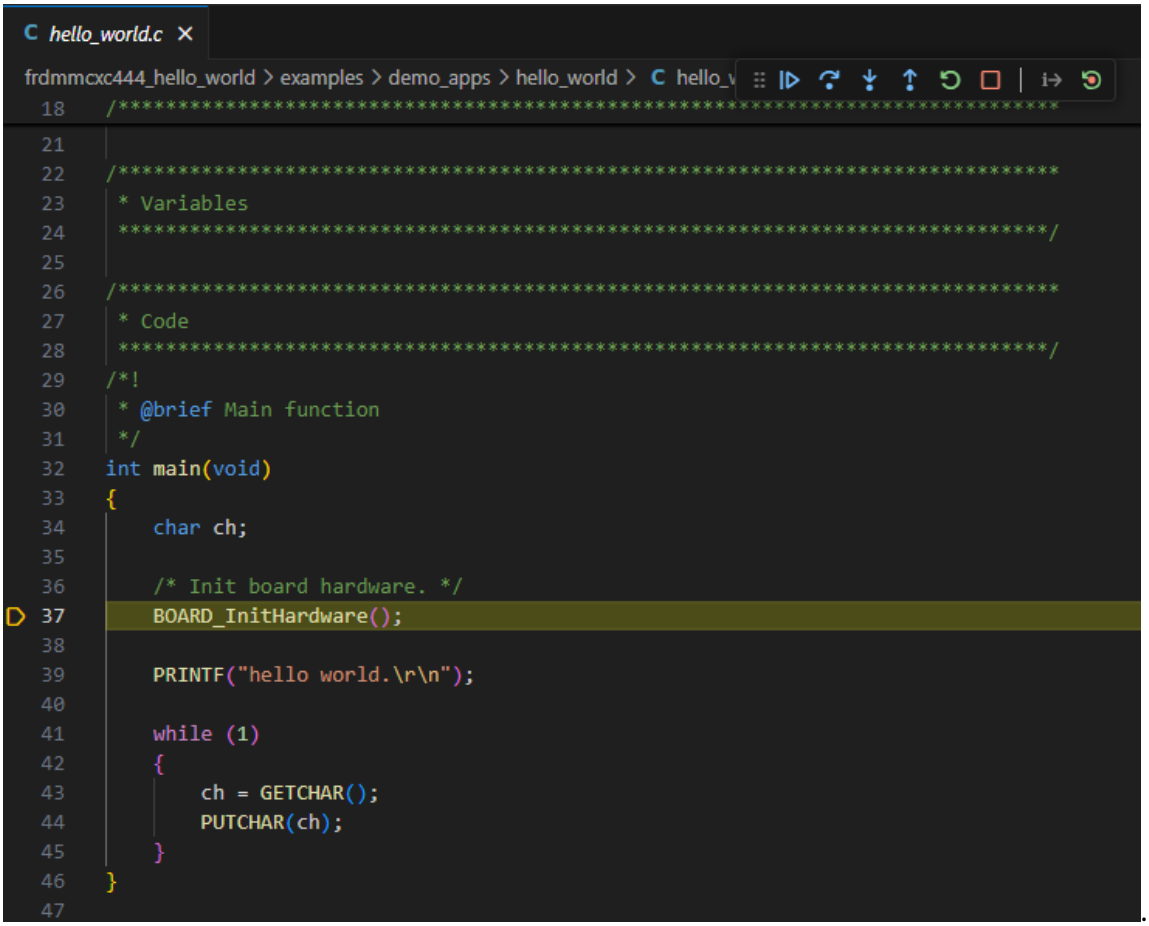
1. Open the **Serial Monitor** from the VS Code's integrated terminal. Select the VCom Port for your device and set the baud rate to 115200.



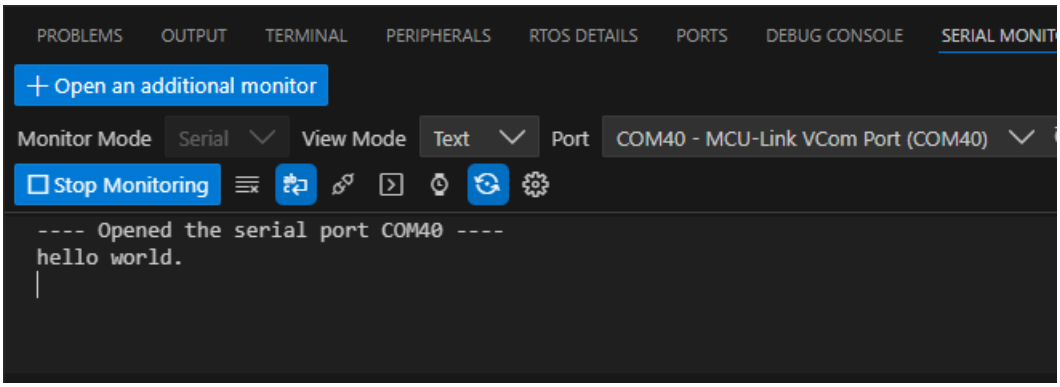
2. Navigate to the **PROJECTS** view and click the play button to initiate a debug session.



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



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



Running a demo using ARMGCC CLI/IAR/MDK

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

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

INFO: [  1][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b␣
↪evk9mimx8ulp -Dcore_id=cm33]
INFO: [  2][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b␣
↪evkbimxrt1050]
INFO: [  3][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b␣
```

(continues on next page)

(continued from previous page)

```
↪evkbmimxrt1060]
INFO: [  4][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b␣
↪evkbmimxrt1170 -Dcore_id=cm4]
INFO: [  5][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b␣
↪evkbmimxrt1170 -Dcore_id=cm7]
INFO: [  6][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b␣
↪evkcmimxrt1060]
INFO: [  7][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b␣
↪evkmcimx7ulp]
...
```

The supported toolchains and build targets for an example are decided by the example-self example.yml and board example.yml, please refer Example Toolchains and Targets for more details.

Build the project Use west build -h to see help information for west build command. Compared to zephyr's west build, MCUXpresso SDK's west build command provides following additional options for mcux examples:

- --toolchain: specify the toolchain for this build, default armgcc.
- --config: value for CMAKE_BUILD_TYPE. If not provided, build system will get all the example supported build targets and use the first debug target as the default one. Please refer Example Toolchains and Targets for more details about example supported build targets.

Here are some typical usages for generating a SDK example:

```
# Generate example with default settings, default used device is the mainset MK22F51212
west build -b frdmk22f examples/demo_apps/hello_world

# Just print cmake commands, do not execute it
west build -b frdmk22f examples/demo_apps/hello_world --dry-run

# Generate example with other toolchain like iar, default armgcc
west build -b frdmk22f examples/demo_apps/hello_world --toolchain iar

# Generate example with other config type
west build -b frdmk22f examples/demo_apps/hello_world --config release

# Generate example with other devices with --device
west build -b frdmk22f examples/demo_apps/hello_world --device MK22F12810 --config release
```

For multicore devices, you shall specify the corresponding core id by passing the command line argument -Dcore_id. For example

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

For shield, please use the --shield to specify the shield to run, like

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

Sysbuild(System build) To support multicore project building, we ported Sysbuild from Zephyr. It supports combine multiple projects for compilation. You can build all projects by adding --sysbuild for main application. For example:

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

For more details, please refer to System build.

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

1. Run cmake configuration

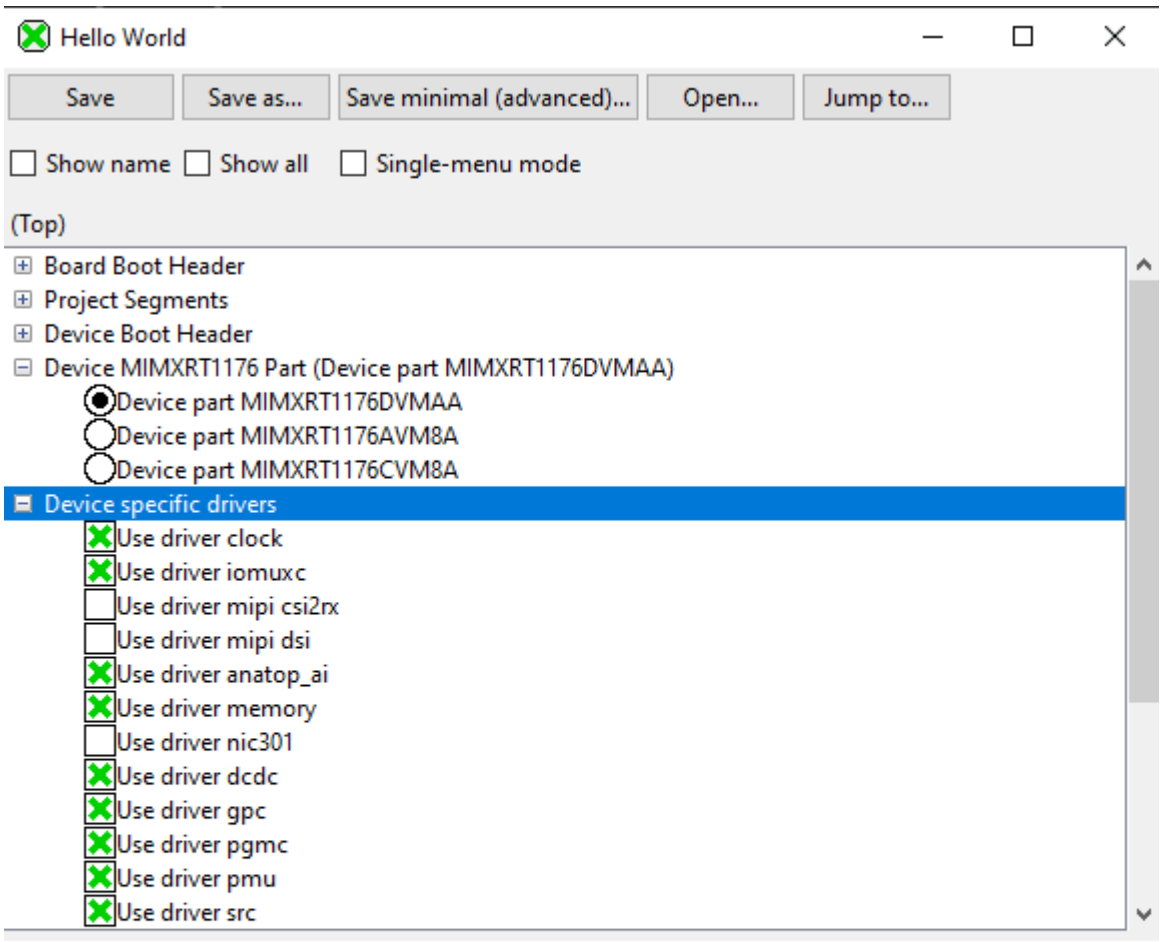
```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

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

2. Run guiconfig target

```
west build -t guiconfig
```

Then you will get the Kconfig GUI launched, like




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

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

menu "Device specific drivers"
```

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

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

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

Flash the hello_world example:

```
west flash -r linkserver
```

Debug Start a gdb interface by following command:

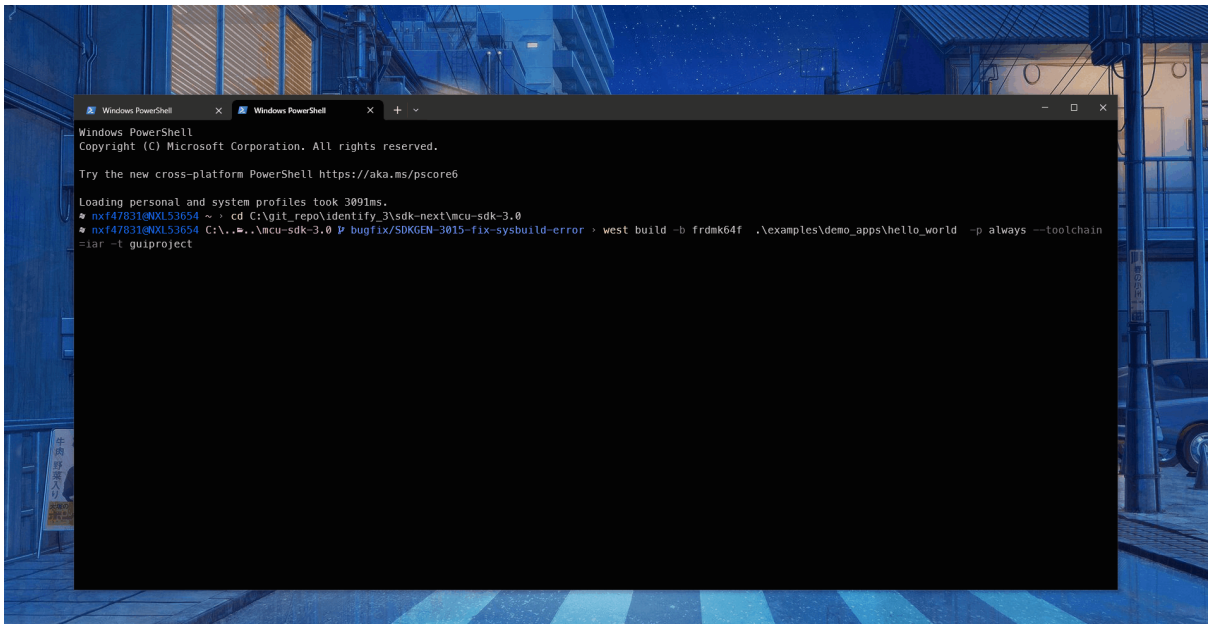
```
west debug -r linkserver
```

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

This is the cmd to generate the evkbmimxrt1170 hello_world IAR IDE project files.

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

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



Note, please follow the *Installation* to setup the environment especially make sure that *ruby* has been installed.

1.4 Release Notes

1.4.1 MCUXpresso SDK Release Notes

Overview

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

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

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

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

MCUXpresso SDK

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

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

Development tools

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

- IAR Embedded Workbench for Arm, version is 9.60.4
- MCUXpresso for VS Code v25.06
- GCC Arm Embedded Toolchain 14.2.x

Supported development systems

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

Development boards	MCU devices
EVK-MIMX8M	MIMX8ML3CVNKZ, MIMX8ML3DVNLZ, MIMX8ML4CVNKZ, MIMX8ML4DVNLZ, MIMX8ML5CVNKZ, MIMX8ML5DVNLZ, MIMX8ML6CVNKZ, MIMX8ML6DVNLZ, MIMX8ML8CVNKZ, **MIMX8ML8DVNLZ**, MIMX8ML2CVNKZ, MIMX8ML2DVNLZ

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.

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.

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

TinyCBOR Concise Binary Object Representation (CBOR) Library

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

Multicore Multicore Software Development Kit

llhttp HTTP parser llhttp

FreeMASTER FreeMASTER communication driver for 32-bit platforms.

Release contents

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

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

Known issues

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

Cannot add SDK components into FreeRTOS projects

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

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

AIPSTZ

[2.0.1]

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

[2.0.0]

- Initial version.

---

eASRC

[2.0.7]

- 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.

[2.0.6]

- Improvements
  - Updated use of definitions in the function ASRC_SetSlotConfig and ASRC_EnableContextSlot.

[2.0.5]

- Improvements
  - Fixed the typo and multichannel configuration issues in asrc driver.
  - Updated the asrc firmware table for better THD performance.

[2.0.4]

- Improvements
  - Corrected the data width support by IEC60958,
  - Updated the default stage1 result to float to avoid convert result distortion.

[2.0.3]

- Bug Fixes
  - Corrected the btis shift index used in function ASRC_TransferBlocking.
  - Ensured the resampler and prefilter are not in bypass mode when the convertion require the function.

[2.0.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.4, 14.3, 14.4, 16.3, 16.1, 16.8, 17.7.

[2.0.1]

- Bug Fixes
  - Fixed the context id hard code issue in the function ASRC_TransferInCreateHandleSDMA/ASRC_TransferOutCreateHandleSDMA.
- Improvements
  - Added support for the data size bigger than 64K in sdma driver.

[2.0.0]

- Initial version.

ASRC_SDMA

[2.0.4]

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

[2.0.3]

- Refer asrc driver change log 2.0.0 to 2.0.3

CACHE ARMv7-M7

[2.0.4]

- Bug Fixes
  - Fixed doxygen issue.

[2.0.3]

- Improvements
  - Deleted redundancy code about calculating cache clean/invalidate size and address aligns.

[2.0.2]

- Bug Fixes
  - Fixed violation of MISRA C-2012 Rule 10.1, 10.3 and 10.4.

[2.0.1]

- Bug Fixes
  - Fixed cache size issue in L2CACHE_GetDefaultConfig API.

[2.0.0]

- Initial version.

COMMON

[2.6.0]

- Bug Fixes
  - Fix CERT-C violations.

[2.5.0]

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

[2.4.3]

- Improvements
  - Enable 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.

ECSPI

[2.3.3]

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

[2.3.2]

- Improvements
  - Changed ECSPI_DUMMYDATA to 0x00.

[2.3.1]

- Bug Fixes
  - Fixed ECSPI_GetInstance potential issue that return wrong instance number.

[2.3.0]

- Bug Fixes
  - Fixed burst length issue,the burst length range shall range from 1-4096 bits, so the width shall be uint8_t rather than uint16_t.

[2.2.0]

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

[2.1.1]

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

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

EDMA (DMA3)

[2.5.0]

- Improvements
  - Add API EDMA_GetTransferSize for EDMA_PrepareTransferConfig to reduce HIS CCM value.

[2.4.0]

- Improvements
  - Added peripheral to peripheral support in DMA3 driver.
- Bug Fixes
  - Fixed the ERQ bit reading error issue.

[2.3.2]

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

[2.3.1]

- Bug Fixes
  - Added clear TCD_CITER_ELINKNO and TCD_BITER_ELINKNO registers in EDMA_AbortTransfer to make sure the TCD registers in a correct state for next calling of EDMA_SubmitTransfer.

[2.3.0]

- Improvements
  - Added feature FSL_FEATURE_EDMA_HAS_NO_SBR_ATTR_BIT to separate DMA without ATTR bitfield.

[2.2.7]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.8, 5.6.

[2.2.6]

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

[2.2.5]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.4.

[2.2.4]

- Bug Fixes
  - Fix the issue that EDMA_AbortTransfer not reset edma handle(tcdUsed, tail, header) and fix EDMA_InstallTCDMemory(handle->header = 1)

[2.2.3]

- Improvements
  - Added feature FSL_FEATURE_EDMA_MODULE_CHANNEL_IRQ_ENTRY_SUPPORT_PARAMETER to support driver IRQ handler with parameters.
  - Added feature FSL_FEATURE_EDMA_HAS_COMMON_CLOCK_GATE to improve clock gate control in dma driver.

[2.2.2]

- Bug Fixes
  - Fixed the issue of EDMA_SubmitTransfer return busy when calling EDMA_EnableChannelInterrupts before submit transfer.
  - Fixed violations of MISRA C-2012 rule 10.4, 10.1, 9.2, 10.4, 10.6, 14.4, 10.7, 14.3, 11.6.

[2.2.1]

- Improvements
  - Removed channel MUX reset from EDMA_ResetChannel, since channel mux should be constant while channel is alive.

[2.2.0]

- Improvements
  - Added new API EDMA_SetChannelMux to support channel mux feature.
  - Added new API EDMA_PrepareTransferConfig to expose paramters source offset and destination offset.
  - Exposed EDMA_InstallTCD function to application.
  - Added source/destination address alignment check.

[2.1.1]

- Improvements
  - Added 8bytes transfer width feature support in driver.

[2.1.0]

- Bug Fixes
  - Added const type for parameter configuration in EDMA_SubmitTransfer and EDMA_HandleTransferConfig API.
  - Added configurations for srcAddr and destAddr in EDMA_PrepareTransfer API.

[2.0.2]

- Improvements
  - Updated eDMA driver to support MP_CR bit GMRC.
  - Updated eDMA instance name for i.MX 8QM.
  - Used instance number as factor to calculate channel number for different instance instead of hard code.

[2.0.1]

- New Features
  - Added control macro to enable/disable the CLOCK code in current driver.
  - Added s_EDMAEnabledChannel to record enabled channel to merge all the channel IRQ handler into driver IRQ handler.
  - Added feature macro for bits EMI and EBW in MP_CSR.
- Improvements
  - Removed all the separated channel IRQ handler in DMA driver.

[2.0.0]

- Initial version.

ENET

[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_DriverIRQHan
    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 caculates 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 featrues 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 caculates 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 relaim 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 relaim 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 interupt 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

[2.6.5]

- 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_GetRxFrame, 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_SetSMI.

[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 convertion 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

[2.14.1]

- Bug Fixes
  - Fixed register IMASK2-4 IFLAG2-4 HR_TIME_STAMPn 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

[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. Thses 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 are unassociated with interrupt in macro FLEXCAN_MEMORY_ERROR_INT_FLAG.
  - Remove flags which will are 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_SubHandlerForEhancedRxFifo() 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_SubHandlerForDataTransfered() 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 enablePretendedeNetworking 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 PRESDIV.
  - 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 comming.

[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 evkmimxrt1064.
  - 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 folllow 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 transfering 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 comming.

[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.

GPC

[2.2.0]

- Improvements
  - Optimized the exited APIs to support extended IRQs and slots.
- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules.

[2.1.0]

- Improvements
  - Unified the register or bit fields' name which contains specific cortex M core information.
- Bug Fixes
  - Fixed Coverity Out-of-bounds issue.

[2.0.1]

- Improvements
  - Added parameters to enable/disable the WFI and DSM mask.

[2.0.0]

- Initial version.

GPIO

[2.0.6]

- Bug Fixes
  - Fixed compile warning: 'GPIO_GetInstance' defined but not used when macro FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL is defined.

[2.0.5]

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

[2.0.4]

- Improvements
  - Updated the GPIO_PinWrite to use atomic operation if possible.
- Bug Fixes
  - Fixed GPIO_PortToggle bug with platforms don't have register DR_TOGGLE.

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

GPT

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.4, 10.6, 10.8, 17.7.

[2.0.0]

- Initial version.

I2C

[2.0.7]

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

[2.0.6]

- Bug Fixes
  - Fixed the bug that, in I2C_MasterStop after the stop command is issued, the IBB flag should be cleared rather than set.
  - Fixed the bug that to clear kI2C_ArbitrationLostFlag and kI2C_IntPendingFlag, their bits should be written '0' rather than '1'.

[2.0.5]

- Bug Fixes
  - Fixed Coverity issue of unchecked return value in I2C_RTOS_Transfer.
  - Fixed MISRA issues.
    - Fixed rules 10.1, 10.3, 10.4, 11.9, 14.4, 15.7, 16.4, 17.7.
- Improvements
  - Updated the I2C_WAIT_TIMEOUT macro to unified name I2C_RETRY_TIMES.

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

MU

[2.2.0]

- New Features
  - Added API MU_GetRxStatusFlags.

[2.1.3]

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

[2.1.2]

- Bug Fixes
  - Fixed issue that MU_GetInstance() is defined but never used.

[2.1.1]

- Bug Fixes
  - Fixed general interrupt comment typo.

[2.1.0]

- Improvements
  - Added new enum mu_msg_reg_index_t.

[2.0.7]

- Bug Fixes
  - Fixed MU_GetInterruptsPending bug that can not get general interrupt status.

[2.0.6]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

- Improvements
  - Added support for MIMX8MQx.

[2.0.1]

- Improvements
  - Added support for MCIMX7Ux_M4.

[2.0.0]

- Initial version.

PDM

[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 hadware 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 bit-field 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

PDM_SDMA

[2.7.1]

- 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.

[2.7.0]

- Improvements
  - Added new api PDM_TransferTerminateReceiveSDMA to reset all PDM SDMA internal state machine.

[2.6.0]

- Bug Fixes
  - Fixed burst length overflow issue when using multi fifo.
  - Added alignment check between transfer size and the fifo width.

[2.5.0]

- Refer PDM driver change log 2.1.0 to 2.3.1

PWM

[2.0.1]

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

[2.0.0]

- Initial version.

RDC

[2.2.0]

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

[2.1.1]

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

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

RDC_SEMA42

[2.0.4]

- Improvements
  - Changed to implement RDC_SEMAPHORE_Lock base on RDC_SEMAPHORE_TryLock.

[2.0.3]

- Improvements:
  - Supported the RDC_SEMAPHORE_Type structure whose gate registers are defined as an array.

[2.0.2]

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

[2.0.1]

- Improvements:
  - Added support for the platforms that don't have dedicated RDC_SEMA42 clock gate.

[2.0.0]

- Initial version.

SAI

[2.4.7]

- 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
  - 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

SAI_SDMA

[2.6.0]

- Improvements
  - Removed deprecated API (SAI_TransferTxSetFormatSDMA,SAI_TransferRxSetFormatSDMA) and replaced them with new API (SAI_TransferRxSetConfigSDMA,SAI_TransferRxSetConfigSDMA)

[2.5.3]

- Bug Fixes
  - Fixed the internal state of bd pool can not be reset in SAI_TransferTerminateSendSDMA.

[2.5.2]

- 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

[2.5.1]

- Bug Fixes
  - Fixed the incorrect internal queue state after abort the transfer in callback.

[2.5.0]

- Bug Fixes
  - Fixed the incorrect driver queue state after calling abort function.
- Improvements
  - Added new api SAI_TransferTerminateSendSDMA/SAI_TransferTerminateReceiveSDMA.

[2.4.1]

- Bug Fixes
  - Fixed the handler array s_sdmaPrivateHandle overflow issue.

[2.4.0]

- Bug Fixes
  - Fixed burst length overflow issue when using multi fifo.
  - Added alignment check between transfer size and the fifo width.
  - Fixed violations of MISRA C-2012 rule 10.6.

[2.3.1]

- Refer SAI driver change log 2.1.0 to 2.3.1

SDMA

[2.4.2]

- Bug Fixes
  - Add global variable s_ramScriptLoaded to fix compatibility issues.

[2.4.1]

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

[2.4.0]

- Improvements
  - Added load ram script automatically according to the peripheral type.
  - Updated sdma ram script to version I.MX7D_4_6.
  - Added api SDMA_GetRamScriptVersion to support to SDMA ram script version.
- Bug Fixes
  - Removed the SDMA_StartChannelEvents from SDMA_StartTransfer as sdma scheduler will handle it automatically

[2.3.6]

- Bug Fixes
  - Cleared the SDMAARM_CONFIG_CSM initial value before write value 0 using OR.

[2.3.5]

- Bug Fixes
  - Added transfer size validation to aovid overflow.

[2.3.4]

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

[2.3.3]

- Improvements
  - Improved sdma driver comments and parameters validation.

[2.3.2]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.6.

[2.3.1]

- Improvements
  - Removed clear all channel interrupt status in SDMA_HandleIRQ to avoid the possibility of lossing interrupt.

[2.3.0]

- Improvements
  - Added peripheral-to-peripheral support in SDMA driver.
  - Added 24bit data width support in sdma driver.
- Bug Fixes
  - Fixed Coverity issue: left shift may overflow issue.
  - Fixed MISRA2004 issue: the operand of underlying type 'unsigned char' or 'unsigned short' caused the result cast to the underlying type.
  - Fixed violations of MISRA C-2012 rule 10.3, 11.9, 10.4, 17.7, 20.7, 14.4, 11.6, 12.2, 16.4.

[2.2.1]

- Bug Fixes
  - Fixed MISRA 2004 issue in sdma driver.

[2.2.0]

- Improvements
  - Added fsl_sdma_script.h to define the sdma script address and firmware.
  - Updated the format of generic register R7 to align with newest firmware.

[2.1.1]

- Improvements
  - Added SDMA_SetDoneConfig to support hardware/software done configuration.
  - Marked SDMC_EnableSwDone as deprecated.
- Bug Fixes
  - Fixed logical dead code issue in function SDMA_SetDoneConfig.

[2.1.0]

- Improvements
  - Added SDMA_SetMultiFifoConfig API to support multi fifo feature.
  - Added SDMA_EnableSwDone API to support software done feature.
  - Added SDMA_LoadScript API to support load script to SDMA program memory.
  - Added SDMA_DumpScript API to support dump script from SDMA program memory.
  - Added SDMA3 IRQ handler.

[2.0.0]

- Initial version.

SEMA4

[2.2.0]

- New Features
  - Added SEMA4_BUSY_POLL_COUNT parameter to prevent infinite polling loops in SEMA4 operations.
  - Added timeout mechanism to all polling loops in SEMA4 driver code.
- Improvements
  - Updated SEMA4_Lock function to return status_t instead of void for better error handling.
  - Enhanced documentation to clarify timeout behavior and return values.

[2.1.0]

- Improvements
  - Changed mask parameter type in SEMA4_EnableGateNotifyInterrupt() and SEMA4_DisableGateNotifyInterrupt() functions to avoid casting from unsigned long to unsigned short in the code when modifying the 16bits CPINE register.

[2.0.3]

- Improvements
  - Changed to implement SEMA4_Lock base on SEMA4_TryLock.

[2.0.2]

- Improvements:
  - Supported the SEMA4_Type structure whose gate registers are defined as an array.

[2.0.1]

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

[2.0.0]

- Initial version.

TMU

[2.0.0]

- Initial version.
- This module was first developed on i.MX 8MP to support multi-site temperature monitor.

UART

[2.3.2]

- Improvements
  - Make driver aarch64 compatible

[2.3.1]

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

[2.3.0]

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

[2.2.1]

- Bug Fixes
  - Fixed MISRA 2012 rule 10.4 violation.

[2.2.0]

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

[2.1.1]

- Bug Fixes
  - Fixed MISRA 8.5 violation.

[2.1.0]

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

[2.0.2]

- Improvements
  - Added check for transmission complete in UART_WriteBlocking, UART_TransferHandleIRQ and UART_SendSDMACallback to ensure all the data would be sent out to bus.
  - Modified UART_ReadBlocking so that if more than one receiver errors occur, all status flags will be cleared and the most severe error status will be returned.
- Bug Fixes
  - Fixed MISRA issues.
    - Fixed rules 10.1, 10.3, 10.4, 10.6, 10.7, 10.8, 11.9, 14.4.

[2.0.1]

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

[2.0.0]

- Initial version.

UART_SDMA

[2.3.0]

- Refer UART driver change log 2.0.0 to 2.1.1, 2.3.0.

USDHC

[2.8.5]

- Improvements
  - Enable the driver to be AARCH64 compatible.

[2.8.4]

- Improvements
  - Add feature macro FSL_FEATURE_USDHC_HAS_NO_VS18.

[2.8.3]

- Improvements
  - Improved api USDHC_EnableAutoTuningForCmdAndData to adapt to new bit field name for USDHC_VEND_SPEC2 register.

[2.8.2]

- Improvements
  - Added feature macro FSL_FEATURE_USDHC_HAS_NO_VOLTAGE_SELECT.

[2.8.1]

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

[2.8.0]

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

[2.7.0]

- Improvements
  - Added api USDHC_TransferScatterGatherADMANonBlocking to support scatter gather transfer.
  - Added feature FSL_FEATURE_USDHC_REGISTER_HOST_CTRL_CAP_HAS_NO_RETUNING_TIME_COUNT for re-tuning time counter field in HOST_CTRL_CAP register.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 11.9, 10.1, 10.3, 10.4, 8.4.

[2.6.0]

- Improvements
  - Added api USDHC_SetStandardTuningCounter to support adjust tuning counter of Standard tuning.

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.0]

- Improvements
  - Added USDHC_SetDataConfig API to support manual tuning.
  - Removed the limitaion that source clock must be bigger than the target in function USDHC_SetSdClock by using source clock frequency as target directly.
  - Added peripheral reset in USDHC_Init function.
  - Added tuning reset support in function USDHC_Reset function.

[2.2.8]

- Bug Fixes
  - Fixed out-of bounds write in function USDHC_ReceiveCommandResponse.

[2.2.7]

- Improvements
  - Added API USDHC_GetEnabledInterruptStatusFlags and used in USDHC_TransferHandleIRQ.
  - Removed useless member interruptFlags in usdhc_handle_t.

[2.2.6]

- Improvements
  - Added address align check for ADMA descriptor table address.
  - Changed USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY to (65536-4096) to make sure the data address is 4KB align for a transfer which need more than one ADMA1 descriptor.

[2.2.5]

- Bug Fixes
  - Fixed MDK 66-D warning.

[2.2.4]

- Bug Fixes
  - Fixed issue that real clock frequency wss mismatched with target clock frequency, which was caused by an incorrect prescaler calculation.
- New Features
  - Added control macro to enable/disable the CLOCK code in current driver.

[2.2.3]

- Bug Fixes
  - Fixed issue where AMDA did not disable with DMAEN clear.
- Improvements
  - Improved set clock function to check the output frequency range.
  - Dynamic set SDCLKFS during DDR enable or disable.

[2.2.2]

- Improvements
  - Improved read transfer cache maintain operation, combined clean, and invalidated them into one function.

[2.2.1]

- Bug Fixes
  - Disabled the invalidate cache operation for tuning.

[2.2.0]

- Improvements
  - Improved USDHC to support MMC boot feature.

[2.1.3]

- Bug Fixes
  - Fixed MISRA issue.

[2.1.2]

- Bug Fixes
  - Fixed Coverity issue.
  - Added base address and userData parameter for all callback functions.

[2.1.1]

- Improvements
  - Added cache maintain operation.
  - Added timeout status check for the DATA transfer which ignore error.
  - Added feature macro for SDR50/SDR104 mode.
  - Removed useless IRQ handler from different platforms.

[2.1.0]

- Improvements
  - Integrated tuning into transfer function.
  - Added strobe DLL feature.
  - Added enableAutoCommand23 in data structure.
  - Removed enable card clock function because the controller would handle the clock on/off.

[2.0.0]

- Initial version.

WDOG

[2.2.0]

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

[2.1.1]

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

[2.1.0]

- New Features
  - Added new API "WDOG_TriggerSystemSoftwareReset()" to allow users to reset the system by software.
  - Added new API "WDOG_TriggerSoftwareSignal()" to allow users to trigger a WDOG_B signal by software.
  - Removed the parameter "softwareAssertion" and "softwareResetSignal" out of the wdog_config_t structure.
  - Added new parameter "enableTimeOutAssert" to the wdog_config_t structure. With this parameter enabled, when the WDOG timeout occurs, a WDOG_B signal will be asserted. This signal can be routed to external pin of the chip. Note that WDOG_B signal remains asserted until a power-on reset (POR) occurs.

[2.0.1]

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

[2.0.0]

- Initial version.

1.6 Driver API Reference Manual

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

MIMX8ML8

1.7 Middleware Documentation

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

1.7.1 Multicore

multicore

1.7.2 FreeMASTER

freemaster

1.7.3 FreeRTOS

FreeRTOS

Chapter 2

MIMX8ML8

2.1 AIPSTZ: AHB to IP Bridge

void AIPSTZ_SetMasterPriviledgeLevel(AIPSTZ_Type *base, *aipstz_master_t* master, uint32_t privilegeConfig)

Configure the privilege level for master.

Parameters

- base – AIPSTZ peripheral base pointer
- master – Masters for AIPSTZ.
- privilegeConfig – Configuration is ORed from aipstz_master_privilege_level_t.

void AIPSTZ_SetPeripheralAccessControl(AIPSTZ_Type *base, *aipstz_peripheral_t* peripheral, uint32_t accessControl)

Configure the access for peripheral.

Parameters

- base – AIPSTZ peripheral base pointer
- peripheral – Peripheral for AIPSTZ.
- accessControl – Configuration is ORed from aipstz_peripheral_access_control_t.

FSL_AIPSTZ_DRIVER_VERSION

Version 2.0.1

enum _aipstz_master_privilege_level

List of AIPSTZ privilege configuration.

Values:

enumerator kAIPSTZ_MasterBufferedWriteEnable

Write accesses from this master are allowed to be buffered.

enumerator kAIPSTZ_MasterTrustedForReadEnable

This master is trusted for read accesses.

enumerator kAIPSTZ_MasterTrustedForWriteEnable

This master is trusted for write accesses.

enumerator kAIPSTZ_MasterForceUserModeEnable
Accesses from this master are forced to user-mode.

enum _aipstz_master
List of AIPSTZ masters. Organized by width for the 8-15 bits and shift for lower 8 bits.

Values:

enumerator kAIPSTZ_Master0

enumerator kAIPSTZ_Master1

enumerator kAIPSTZ_Master2

enumerator kAIPSTZ_Master3

enumerator kAIPSTZ_Master5

enum _aipstz_peripheral_access_control
List of AIPSTZ peripheral access control configuration.

Values:

enumerator kAIPSTZ_PeripheralAllowUntrustedMaster

enumerator kAIPSTZ_PeripheralWriteProtected

enumerator kAIPSTZ_PeripheralRequireSupervisor

enumerator kAIPSTZ_PeripheralAllowBufferedWrite

enum _aipstz_peripheral
List of AIPSTZ peripherals. Organized by register offset for higher 32 bits, width for the 8-15 bits and shift for lower 8 bits.

Values:

enumerator kAIPSTZ_Peripheral0

enumerator kAIPSTZ_Peripheral1

enumerator kAIPSTZ_Peripheral2

enumerator kAIPSTZ_Peripheral3

enumerator kAIPSTZ_Peripheral4

enumerator kAIPSTZ_Peripheral5

enumerator kAIPSTZ_Peripheral6

enumerator kAIPSTZ_Peripheral7

enumerator kAIPSTZ_Peripheral8

enumerator kAIPSTZ_Peripheral9

enumerator kAIPSTZ_Peripheral10

enumerator kAIPSTZ_Peripheral11

enumerator kAIPSTZ_Peripheral12

enumerator kAIPSTZ_Peripheral13

enumerator kAIPSTZ_Peripheral14

enumerator kAIPSTZ_Peripheral15

enumerator kAIPSTZ_Peripheral16

enumerator kAIPSTZ_Peripheral17

enumerator kAIPSTZ_Peripheral18

enumerator kAIPSTZ_Peripheral19

enumerator kAIPSTZ_Peripheral20

enumerator kAIPSTZ_Peripheral21

enumerator kAIPSTZ_Peripheral22

enumerator kAIPSTZ_Peripheral23

enumerator kAIPSTZ_Peripheral24

enumerator kAIPSTZ_Peripheral25

enumerator kAIPSTZ_Peripheral26

enumerator kAIPSTZ_Peripheral27

enumerator kAIPSTZ_Peripheral28

enumerator kAIPSTZ_Peripheral29

enumerator kAIPSTZ_Peripheral30

enumerator kAIPSTZ_Peripheral31

enumerator kAIPSTZ_Peripheral32

enumerator kAIPSTZ_Peripheral33

typedef enum *_aipstz_master_privilege_level* aipstz_master_privilege_level_t

List of AIPSTZ privilege configuration.

typedef enum *_aipstz_master* aipstz_master_t

List of AIPSTZ masters. Organized by width for the 8-15 bits and shift for lower 8 bits.

typedef enum *_aipstz_peripheral_access_control* aipstz_peripheral_access_control_t

List of AIPSTZ peripheral access control configuration.

typedef enum *_aipstz_peripheral* aipstz_peripheral_t

List of AIPSTZ peripherals. Organized by register offset for higher 32 bits, width for the 8-15 bits and shift for lower 8 bits.

2.2 ASRC: Asynchronous sample rate converter

2.3 ASRC Driver

uint32_t ASRC_GetInstance(ASRC_Type *base)

Get instance number of the ASRC peripheral.

Parameters

- base – ASRC base pointer.

void ASRC_Init(ASRC_Type *base)

brief Initializes the asrc peripheral.

This API gates the asrc clock. The asrc module can't operate unless ASRC_Init is called to enable the clock.

param base asrc base pointer.

void ASRC_Deinit(ASRC_Type *base)

De-initializes the ASRC peripheral.

This API gates the ASRC clock and disable ASRC module. The ASRC module can't operate unless ASRC_Init

Parameters

- base – ASRC base pointer.

void ASRC_GetContextDefaultConfig(*asrc_context_config_t* *config, uint32_t channels, uint32_t inSampleRate, uint32_t outSampleRate)

ASRC get context default configuration.

Parameters

- config – ASRC context configuration pointer.
- channels – input audio data channel numbers.
- inSampleRate – input sample rate.
- outSampleRate – output sample rate.

status_t ASRC_SetContextConfig(ASRC_Type *base, *asrc_context_t* context, *asrc_context_config_t* *config)

ASRC configure context.

Parameters

- base – ASRC base pointer.
- context – index of asrc context, reference asrc_context_t.
- config – ASRC context configuration pointer.

Return values

kStatus_InvalidArgument – invalid parameters. kStatus_ASRCConfigureFailed context configure failed. kStatus_Success context configure success.

status_t ASRC_SetContextOutputConfig(ASRC_Type *base, *asrc_context_t* context, *asrc_context_output_config_t* *config)

ASRC configure context output.

Parameters

- base – ASRC base pointer.
- context – index of asrc context, reference asrc_context_t.
- config – ASRC context output configuration pointer.

status_t ASRC_SetContextInputConfig(ASRC_Type *base, *asrc_context_t* context, *asrc_context_input_config_t* *config)

ASRC configure context input.

Parameters

- base – ASRC base pointer.
- context – index of asrc context, reference asrc_context_t.

- config – ASRC context input configuration pointer.

static inline void ASRC_EnableContextRun(ASRC_Type *base, *asrc_context_t* context, bool enable)

ASRC context enable run. All control fileds associated with a context must be stable prior to setting context run enable.

Parameters

- base – ASRC base pointer.
- context – ASRC context index.
- enable – true is enable, inform the datapath begin processing sample data for the context. false is disable, data processing will halt immediately.

static inline void ASRC_EnableContextRunStop(ASRC_Type *base, *asrc_context_t* context, bool enable)

ASRC context enable run stop. This function used to flush the ASRC pipeline and completely end processing for a context.

Parameters

- base – ASRC base pointer.
- context – ASRC context index.
- enable – true is enable, false is disable.

static inline void ASRC_EnableContextInDMA(ASRC_Type *base, *asrc_context_t* context, bool enable)

ASRC context input DMA request enable.

Parameters

- base – ASRC base pointer.
- context – ASRC context index.
- enable – true is enable, false is disable.

static inline void ASRC_EnableContextOutDMA(ASRC_Type *base, *asrc_context_t* context, bool enable)

ASRC context output DMA request enable.

Parameters

- base – ASRC base pointer.
- context – ASRC context index.
- enable – true is enable, false is disable.

static inline void ASRC_EnablePreFilterBypass(ASRC_Type *base, *asrc_context_t* context, bool bypass)

ASRC prefilter bypass mode This function enable or disable the ASRC prefilter bypass mode.

Parameters

- base – ASRC peripheral base address.
- context – context processor number.
- bypass – true is bypass, false is normal mode.

static inline void ASRC_EnableResamplerBypass(ASRC_Type *base, *asrc_context_t* context, bool bypass)

ASRC resampler bypass mode This function enable or disable the ASRC resampler bypass mode.

Parameters

- base – ASRC peripheral base address.
- context – context processor number.
- bypass – true is bypass, false is normal mode.

static inline void ASRC_SetContextChannelNumber(ASRC_Type *base, *asrc_context_t* context, uint32_t channels)

ASRC set context channel number. Note: The maximum channel number in one context can not exceed 32.

Parameters

- base – ASRC peripheral base address.
- context – context number.
- channels – channel number, should <= 32.

uint32_t ASRC_GetContextOutSampleSize(uint32_t inSampleRate, uint32_t inSamplesSize, uint32_t inWidth, uint32_t outSampleRate, uint32_t outWidth)

ASRC get output sample count.

Parameters

- inSampleRate – output sample rate.
- inSamplesSize – input sample rate.
- inWidth – input samples buffer size, the size of buffer should be converted to align with 4 byte .
- outSampleRate – input sample width.
- outWidth – Output width.

Return values

output – samples size.

static inline void ASRC_EnableInterrupt(ASRC_Type *base, uint32_t mask)

ASRC interrupt enable This function enable the ASRC interrupt with the provided mask.

Parameters

- base – ASRC peripheral base address.
- mask – The interrupts to enable. Logical OR of _asrc_interrupt_mask.

static inline void ASRC_DisableInterrupt(ASRC_Type *base, uint32_t mask)

ASRC interrupt disable This function disable the ASRC interrupt with the provided mask.

Parameters

- base – ASRC peripheral base address.
- mask – The interrupts to disable. Logical OR of _asrc_interrupt_mask.

static inline uint32_t ASRC_GetInterruptStatus(ASRC_Type *base)

Gets the ASRC interrupt status flag state.

Parameters

- base – ASRC base pointer

Returns

ASRC Tx status flag value. Use the Status Mask to get the status value needed.

static inline void ASRC_ClearInterruptStatus(ASRC_Type *base, uint32_t status)

clear the ASRC interrupt status flag state.

Parameters

- base – ASRC base pointer
- status – status flag to be cleared.

static inline uint32_t ASRC_GetFifoStatus(ASRC_Type *base, *asrc_context_t* context)

Gets the ASRC fifo status flag.

Parameters

- base – ASRC base pointer
- context – context id

static inline void ASRC_WriteContextFifo(ASRC_Type *base, *asrc_context_t* context, uint32_t data)

write the ASRC context fifo.

Parameters

- base – ASRC base pointer.
- context – context id.
- data – data to write.

static inline uint32_t ASRC_ReadContextFifo(ASRC_Type *base, *asrc_context_t* context)

read the ASRC context fifo.

Parameters

- base – ASRC base pointer.
- context – context id.

Return values

read – data.

static inline uint32_t ASRC_GetWriteContextFifoAddr(ASRC_Type *base, *asrc_context_t* context)

Get ASRC write fifo address.

Parameters

- base – ASRC base pointer.
- context – context id.

Return values

write – fifo address.

static inline uint32_t ASRC_GetReadContextFifoAddr(ASRC_Type *base, *asrc_context_t* context)

Get the ASRC read context fifo address.

Parameters

- base – ASRC base pointer.
- context – context id.

Return values

read – fifo address.

uint32_t ASRC_ReadFIFORemainedSample(ASRC_Type *base, *asrc_context_t* context, uint32_t *outAddr, uint32_t outWidth, uint32_t sampleCount)

Get the ASRC read fifo remained samples. Since the DMA request will be triggered only when the sample group in read fifo is bigger then the watermark, so when the data size cannot be divisible by the (watermark + 1), then part of sample will left in read fifo, application should call this api to get the left samples.

Parameters

- base – ASRC base pointer.
- context – context id.
- outAddr – address to receive remained sample in read fifo.
- outWidth – output data width.
- sampleCount – specify the read sample count.

Return values

sample – counts actual read from output fifo.

status_t ASRC_TransferBlocking(ASRC_Type *base, *asrc_context_t* context, *asrc_transfer_t* *xfer)

ASRC blocking convert audio sample rate. This function depends on the configuration of input and output, so it should be called after the ASRC_SetContextConfig. The data format it supports: 1.16bit 16bit per sample in input buffer, input buffer size should be calculate as: samples 2U output buffer size can be calculated by call function ASRC_GetContextOutSampleSize, the parameter outWidth should be 2 2.20bit 24bit per sample in input buffer, input buffer size should be calculate as: samples 3U output buffer size can be calculated by call function ASRC_GetContextOutSampleSize, the outWidth should be 3. 3.24bit 24bit per sample in input buffer, input buffer size should be calculate as: samples * 3U output buffer size can be calculated by call function ASRC_GetContextOutSampleSize, the outWidth should be 3. 4.32bit 32bit per sample in input buffer, input buffer size should be calculate as: samples * 4U output buffer size can be calculated by call function ASRC_GetContextOutSampleSize, the outWidth should be 4.

Parameters

- base – ASRC base pointer.
- context – context id.
- xfer – .xfer configuration.

Return values

kStatus_Success. –

FSL_ASRC_DRIVER_VERSION

Version 2.0.7

ASRC return status, _asrc_status.

Values:

enumerator kStatus_ASRCIdle

ASRC is idle.

enumerator kStatus_ASRCBusy

ASRC is busy.

enumerator kStatus_ASRCInvalidArgument

ASRC invalid argument.

enumerator kStatus_ASRCConfigureFailed

ASRC configure failed

enumerator kStatus_ASRCConvertError
ASRC convert error failed

enumerator kStatus_ASRCNotSupport
ASRC not support

enumerator kStatus_ASRCQueueFull
ASRC queue full

enumerator kStatus_ASRCQueueIdle
ASRC quue idle

enumerator kStatus_ASRCLoadFirmwareFailed
ASRC load firmware failed

enumerator kStatus_ASRCResamplerConfigureFailed
ASRC resampler configured failed

enumerator kStatus_ASRCPrefilterConfigureFailed
ASRC prefilter configured failed

enum _asrc_context
asrc context id

Values:

enumerator kASRC_Context0
Context 0 value

enumerator kASRC_Context1
Context 1 value

enumerator kASRC_Context2
Context 2 value

enumerator kASRC_Context3
Context 3 value

The ASRC interrupt enable flag, _asrc_interrupt_mask.

Values:

enumerator kASRC_Context0InputFifoOverflow
context 0 input fifo overflow

enumerator kASRC_Context1InputFifoOverflow
context 1 input fifo overflow

enumerator kASRC_Context2InputFifoOverflow
context 2 input fifo overflow

enumerator kASRC_Context3InputFifoOverflow
context 3 input fifo overflow

enumerator kASRC_Context0OutFifoReadEmpty
context 0 out fifo read empty

enumerator kASRC_Context1OutFifoReadEmpty
context 1 out fifo read empty

enumerator kASRC_Context2OutFifoReadEmpty
context 2 out fifo read empty

enumerator kASRC_Context3OutFifoReadEmpty
context 3 out fifo read empty

enumerator kASRC_Context0RunStopDone
context 0 run stop done interrupt

enumerator kASRC_Context1RunStopDone
context 1 run stop done interrupt

enumerator kASRC_Context2RunStopDone
context 2 run stop done interrupt

enumerator kASRC_Context3RunStopDone
context 3 run stop done interrupt

enumerator kASRC_ContextAllInterruptStatus
all the context interrupt status

ASRC fifo status, _asrc_fifo_status.

Values:

enumerator kASRC_FifoStatusInputFifoWatermarkFlag
input water mark flag raised

enumerator kASRC_FifoStatusOutputFifoWatermarkFlag
output water mark flag raised

enum _asrc_data_endianness
arsc data endianness

Values:

enumerator kASRC_DataEndianLittle
context data little endian

enumerator kASRC_DataEndianBig
context data big endian

enum _asrc_data_width
data width

Values:

enumerator kASRC_DataWidth32Bit
data width 32bit

enumerator kASRC_DataWidth24Bit
data width 24bit

enumerator kASRC_DataWidth20Bit
data width 20bit

enumerator kASRC_DataWidth16Bit
data width 16bit

enum _asrc_data_type
data type

Values:

enumerator kASRC_DataTypeInteger
data type int

enumerator kASRC_DataTypeFloat
data type float, single precision floating point format

enum _asrc_data_sign
sign extension

Values:

enumerator kASRC_DataSigned
input data is signed

enumerator kASRC_DataUnsigned
input data is unsinged

enum _asrc_sampleBuffer_init_mode
asrc prefilter and resampler sample buffer init mode

Values:

enumerator kASRC_SampleBufferNoPreFillOnInit
do not pre-fill

enumerator kASRC_SampleBufferFillFirstSampleOnInit
replicate the first sample to fill the right half of the sample buffer

enumerator kASRC_SampleBufferFillZeroOnInit
zero fill the right half og the sample buffer

enum _asrc_sampleBuffer_stop_mode
asrc prefilter and resampler sample buffer stop mode

Values:

enumerator kASRC_SampleBufferFillLastSampleOnStop
replicate the last sample to fill the left half of the sample buffer

enumerator kASRC_SampleBufferFillZeroOnStop
zero fill the left half of the sample buffer

enum _asrc_prefilter_stage1_result
ASRC prefilter stage1 result format.

Values:

enumerator kASRC_PrefilterStage1ResultInt
prefilter stage1 results are stored in 32 bit int format

enumerator kASRC_PrefilterStage1ResultFloat
prefilter stage1 results are stored in 32 bit float format

enum _asrc_resampler_taps
ASRC resampler taps.

Values:

enumerator kASRC_ResamplerTaps_32
resampler taps 32

enumerator kASRC_ResamplerTaps_64
resampler taps 64

enumerator kASRC_ResamplerTaps_128
resampler taps 128

ASRC support sample rate, _asrc_sample_rate.

Values:

enumerator kASRC_SampleRate_8000
8K sample rate

enumerator kASRC_SampleRate_11025
11025 sample rate

enumerator kASRC_SampleRate_12000
12K sample rate

enumerator kASRC_SampleRate_16000
16K sample rate

enumerator kASRC_SampleRate_22050
22.05K sample rate

enumerator kASRC_SampleRate_24000
24K sample rate

enumerator kASRC_SampleRate_32000
32K sample rate

enumerator kASRC_SampleRate_44100
44.1K sample rate

enumerator kASRC_SampleRate_48000
48K sample rate

enumerator kASRC_SampleRate_64000
64K sample rate

enumerator kASRC_SampleRate_88200
88.2K sample rate

enumerator kASRC_SampleRate_96000
96K sample rate

enumerator kASRC_SampleRate_128000
128K sample rate

enumerator kASRC_SampleRate_176400
176K sample rate

enumerator kASRC_SampleRate_192000
256K sample rate

enumerator kASRC_SampleRate_256000
256K sample rate

enumerator kASRC_SampleRate_352800
352.8K sample rate

enumerator kASRC_SampleRate_384000
384K sample rate

enumerator kASRC_SampleRate_768000
768K sample rate

typedef enum *_asrc_context* asrc_context_t
: asrc context id

typedef enum *_asrc_data_endianness* asrc_data_endianness_t
: arsc data endianness

typedef enum *_asrc_data_width* asrc_data_width_t
: data width

typedef enum *_asrc_data_type* asrc_data_type_t
: data type

typedef enum *_asrc_data_sign* asrc_data_sign_t
: sign extension

typedef enum *_asrc_sampleBuffer_init_mode* asrc_sampleBuffer_init_mode_t
: asrc prefilter and resampler sample buffer init mode

typedef enum *_asrc_sampleBuffer_stop_mode* asrc_sampleBuffer_stop_mode_t
: asrc prefilter and resampler sample buffer stop mode

typedef enum *_asrc_prefilter_stage1_result* asrc_prefilter_stage1_result_t
: ASRC prefilter stage1 result format.

typedef enum *_asrc_resampler_taps* asrc_resampler_taps_t
: ASRC resampler taps.

typedef struct *_asrc_data_format* asrc_data_format_t
: asrc context data format

typedef struct *_asrc_access_ctrl* asrc_access_ctrl_t
: asrc context access control The ASRC provides interleaving support in hardware to ensure that a variety of sample source can be internally combined tp confir with this format. The interleave patter is controlled using 3 register fields: GROUP_LENGTH, ACCESS_LENGTH, ITERATIONIS. This is intended to support hardware configurations which distribute a single context across samples from multiple audio sources. Take a example as below: accessGroupLen = 6, the sample group length is 6 samples accessIterations = 2, the 2 sequential ACCESS_LENGTH read from single source accessLen = 2, the 2 samples fetch from one source.

typedef struct *_asrc_context_input_config* asrc_context_input_config_t
: asrc context input configuration

typedef struct *_asrc_context_output_config* asrc_context_output_config_t
: asrc context output configuration

typedef struct *_asrc_context_prefilter_config* asrc_context_prefilter_config_t
: asrc context prefilter configuration

typedef struct *_asrc_context_resampler_config* asrc_context_resampler_config_t
: asrc context resampler configuration

typedef struct *_asrc_context_config* asrc_context_config_t
: asrc context configuration

typedef struct *_asrc_transfer* asrc_transfer_t
: ASRC transfer.

FSL_ASRC_INPUT_FIFO_DEPTH
: ASRC fifo depth.

FSL_ASRC_OUTPUT_FIFO_DEPTH

ASRC_SUPPORT_MAXIMUM_CONTEXT_PROCESSOR_NUMBER
ASRC support maximum channel number of context.

struct _asrc_data_format
#include <fsl_asrc.h> asrc context data format

Public Members

uint8_t dataPosition
context input data sample position

asrc_data_endianness_t dataEndianness
context input data endianness

asrc_data_width_t dataWidth
context input data width

asrc_data_type_t dataType
context input data type

asrc_data_sign_t dataSign
context input data signed or unsigned

struct _asrc_access_ctrl
#include <fsl_asrc.h> asrc context access control The ASRC provides interleaving support in hardware to ensure that a variety of sample source can be internally combined tp confir with this format. The interleave patter is controlled using 3 register fields: GROUP_LENGTH, ACCESS_LENGTH, ITERATIONIS. This is intended to support hardware configurations which distribute a single context across samples from multiple audio sources. Take a example as below: accessGroupLen = 6, the sample group length is 6 samples accessIterations = 2, the 2 sequential ACCESS_LENGTH read from single source accessLen = 2, the 2 samples fetch from one source.

Public Members

uint8_t accessIterations
number of sequential fetches per source

uint8_t accessGroupLen
number of channels in a context

uint8_t accessLen
number of channels per source1

struct _asrc_context_input_config
#include <fsl_asrc.h> asrc context input configuration

Public Members

uint32_t sampleRate
input audio data sample rate

uint8_t watermark
input water mark per samples

asrc_access_ctrl_t accessCtrl
input access control

asrc_data_format_t dataFormat
input data format

struct __asrc__context__output__config
#include <fsl_asrc.h> asrc context output configuration

Public Members

uint32_t sampleRate
output audio data sample rate

uint8_t watermark
output water mark per samples

asrc_access_ctrl_t accessCtrl
output access control

asrc_data_format_t dataFormat
output data format

bool enableDither
output path contains a TPDF dither function. The dither function support all fixed output modes(16, 20, 24, 32bits) dither is not supported in 32bit floating point output mode

bool enableIEC60958
output IEC60958 bit field insertion enable

struct __asrc__context__prefilter__config
#include <fsl_asrc.h> asrc context prefilter configuration

Public Members

asrc_sampleBuffer_init_mode_t initMode
prefilter initial mode

asrc_sampleBuffer_stop_mode_t stopMode
prefilter stop mode

asrc_prefilter_stage1_result_t stage1Result
stage1 data store format

uint32_t filterSt1Taps
prefilter stage1 taps

uint32_t filterSt2Taps
prefilter stage2 taps

uint32_t filterSt1Exp
prefilter stage1 expansion factor

const uint32_t *filterCoeffAddress
prefilter coeff address

struct __asrc__context__resampler__config
#include <fsl_asrc.h> asrc context resampler configuration

Public Members

asrc_sampleBuffer_init_mode_t initMode
: initial mode

asrc_sampleBuffer_stop_mode_t stopMode
: resampler stop mode

asrc_resampler_taps_t tap
: resampleer taps

uint32_t filterPhases
: interpolation phases

uint64_t filterCenterTap
: interpolation center tap

const uint32_t *filterCoeffAddress
: interpolation coeff address

struct __asrc__context__config
: *#include <fsl_asrc.h>* asrc context configuration

Public Members

uint8_t contextChannelNums
: context channel numbers

asrc_context_input_config_t contextInput
: context input configuration

asrc_context_output_config_t contextOutput
: context output configuration

asrc_context_prefilter_config_t contextPrefilter
: context pre filter configuration

asrc_context_resampler_config_t contextResampler
: context resampler configuration

struct __asrc__transfer
: *#include <fsl_asrc.h>* ASRC transfer.

Public Members

uint32_t *inDataAddr
: address of audio data to be converted

uint32_t inDataSize
: size of the audio data

uint32_t *outDataAddr
: address of audio data that is been converted

uint32_t outDataSize
: size of the audio data

2.4 ASRC SDMA Driver

void ASRC_TransferInCreateHandleSDMA(ASRC_Type *base, *asrc_sdma_handle_t* *handle, *asrc_sdma_callback_t* callback, *sdma_handle_t* *dmaHandle, uint32_t eventSource, *asrc_context_t* context, const *asrc_p2p_sdma_config_t* *periphConfig, void *userData)

Initializes the ASRC input SDMA handle.

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

Parameters

- base – ASRC base pointer.
- handle – ASRC SDMA handle pointer.
- base – ASRC peripheral base address.
- callback – Pointer to user callback function.
- dmaHandle – SDMA handle pointer, this handle shall be static allocated by users.
- eventSource – ASRC input sdma event source.
- context – ASRC context number.
- periphConfig – peripheral configurations, used for case.
- userData – User parameter passed to the callback function.

void ASRC_TransferOutCreateHandleSDMA(ASRC_Type *base, *asrc_sdma_handle_t* *handle, *asrc_sdma_callback_t* callback, *sdma_handle_t* *dmaHandle, uint32_t eventSource, *asrc_context_t* context, const *asrc_p2p_sdma_config_t* *periphConfig, void *userData)

Initializes the ASRC output SDMA handle.

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

Parameters

- base – ASRC base pointer.
- handle – ASRC SDMA handle pointer.
- callback – ASRC outcallback.
- dmaHandle – SDMA handle pointer, this handle shall be static allocated by users.
- eventSource – ASRC output event source.
- context – ASRC context number.
- periphConfig – peripheral configurations, used for case.
- userData – User parameter passed to the callback function.

status_t ASRC_TransferSetContextConfigSDMA(ASRC_Type *base, *asrc_sdma_handle_t* *handle, *asrc_context_config_t* *asrcConfig)

Configures the ASRC context.

Parameters

- base – ASRC base pointer.
- handle – ASRC SDMA handle pointer.
- asrcConfig – asrc context configurations.

status_t ASRC_TransferSDMA(ASRC_Type *base, *asrc_sdma_handle_t* *handle, *asrc_transfer_t* *xfer)

Performs a non-blocking ASRC transfer using DMA.

Parameters

- base – ASRC base pointer.
- handle – ASRC SDMA handle pointer.
- xfer – ASRC xfer configurations pointer.

Return values

- kStatus_Success – Start a ASRC SDMA send successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_TxBusy – ASRC is busy sending data.

void ASRC_TransferAbortInSDMA(ASRC_Type *base, *asrc_sdma_handle_t* *handle)

Aborts a ASRC in transfer using SDMA.

Parameters

- base – ASRC base pointer.
- handle – ASRC SDMA handle pointer.

void ASRC_TransferAbortOutSDMA(ASRC_Type *base, *asrc_sdma_handle_t* *handle)

brief Aborts a ASRC out transfer using SDMA.

param base ASRC base pointer. param handle ASRC SDMA handle pointer.

FSL_ASRC_SDMA_DRIVER_VERSION

Version 2.0.4

typedef struct *_asrc_sdma_handle* asrc_sdma_handle_t

ASRC sdma handle prototype.

typedef void (*asrc_sdma_callback_t)(ASRC_Type *base, *asrc_sdma_handle_t* *handle, *status_t* status, void *userData)

ASRC SDMA transfer callback function for finish and error.

typedef void (*asrc_start_peripheral_t)(bool start)

ASRC trigger peripheral function pointer.

typedef struct *_asrc_p2p_sdma_config* asrc_p2p_sdma_config_t

destination peripheral configuration

typedef struct *_asrc_sdma_in_handle* asrc_sdma_in_handle_t

ASRC sdma in handle.

typedef struct *_asrc_sdma_out_handle* asrc_sdma_out_handle_t

ASRC sdma out handle.

ASRC_XFER_IN_QUEUE_SIZE

ASRC xfer queue size.

ASRC_XFER_OUT_QUEUE_SIZE

struct asrc_p2p_sdma_config

#include <fsl_asrc_sdma.h> destination peripheral configuration

Public Members

uint32_t eventSource
: peripheral event source

uint8_t watermark
: peripheral watermark

uint8_t channel
: peripheral channel number

uint8_t fifoWidth
: peripheral fifo width

bool enableContinuous
: true is the amount of samples to be transferred is unknown and script will keep on transferring as long as both events are detected and script must be stopped by application, false is The amount of samples to be transferred is equal to the count field of mode word

asrc_start_peripheral_t startPeripheral
: trigger peripheral start

struct __asrc__sdma__in__handle
: *#include <fsl_asrc_sdma.h>* ASRC sdma in handle.

Public Members

sdma_handle_t *sdmaHandle
: DMA handler for ASRC

uint32_t eventSource
: ASRC event source number

asrc_sdma_callback_t callback
: Callback for users while transfer finish or error occurs

void *userData
: User callback parameter

sdma_buffer_descriptor_t bdPool[4U]
: BD pool for SDMA transfer.

uint8_t asrcInWatermark
: The transfer data count in a DMA request

uint8_t bytesPerSample
: Bytes in a sample

uint32_t *asrcQueue[4U]
: Transfer queue storing queued transfer.

size_t sdmaTransferSize[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

const *asrc_p2p_sdma_config_t* *peripheralConfig
peripheral configuration

uint32_t state
Internal state for ASRC SDMA transfer

struct __asrc__sdma__out__handle
#include <fsl_asrc_sdma.h> ASRC sdma out handle.

Public Members

sdma_handle_t *sdmaHandle
DMA handler for ASRC

void *userData
User callback parameter

uint32_t state
Internal state for ASRC SDMA transfer

uint8_t bytesPerSample
Bytes in a sample

uint32_t eventSource
ASRC event source number

asrc_sdma_callback_t callback
Callback for users while transfer finish or error occurs

uint8_t asrcOutWatermark
The transfer data count in a DMA request

sdma_buffer_descriptor_t bdPool[(4U * 2U)]
BD pool for SDMA transfer.

uint32_t *asrcQueue[(4U * 2U)]
Transfer queue storing queued transfer.

size_t sdmaTransferSize[(4U * 2U)]
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

const *asrc_p2p_sdma_config_t* *peripheralConfig
peripheral configuration

uint32_t nonAlignSize
non align size

void *nonAlignAddr
non align address

struct __asrc__sdma__handle
#include <fsl_asrc_sdma.h> ASRC DMA transfer handle, users should not touch the content of the handle.

Public Members

asrc_sdma_in_handle_t inDMAHandle
input dma handle

asrc_sdma_out_handle_t outDMAHandle
output dma handle

asrc_context_t context
ASRC context number

uint8_t dataChannels
ASRC process data channel number

2.5 Audiomix

enum _audiomix_attch_clk
audio mix attch clk id

Values:

enumerator kAUDIOMIX_Attach_SAI1_MCLK2_To_SAI1_ROOT
attach SAI1 MCLK2 to SAI1 root

enumerator kAUDIOMIX_Attach_SAI1_MCLK2_To_SAI2_ROOT
attach SAI1 MCLK2 to SAI2 root

enumerator kAUDIOMIX_Attach_SAI1_MCLK2_To_SAI3_ROOT
attach SAI1 MCLK2 to SAI3 root

enumerator kAUDIOMIX_Attach_SAI1_MCLK2_To_SAI5_ROOT
attach SAI1 MCLK2 to SAI5 root

enumerator kAUDIOMIX_Attach_SAI1_MCLK2_To_SAI6_ROOT
attach SAI1 MCLK2 to SAI6 root

enumerator kAUDIOMIX_Attach_SAI1_MCLK2_To_SAI7_ROOT
attach SAI1 MCLK2 to SAI7 root

enumerator kAUDIOMIX_Attach_SAI1_MCLK2_To_SAI1_MCLK
attach SAI1 MCLK2 to SAI1 MCLK

enumerator kAUDIOMIX_Attach_SAI1_MCLK2_To_SAI2_MCLK
attach SAI1 MCLK2 to SAI2 MCLK

enumerator kAUDIOMIX_Attach_SAI1_MCLK2_To_SAI3_MCLK
attach SAI1 MCLK2 to SAI3 MCLK

enumerator kAUDIOMIX_Attach_SAI1_MCLK2_To_SAI5_MCLK
attach SAI1 MCLK2 to SAI5 MCLK

enumerator kAUDIOMIX_Attach_SAI1_MCLK2_To_SAI6_MCLK
attach SAI1 MCLK2 to SAI6 MCLK

enumerator kAUDIOMIX_Attach_SAI1_MCLK2_To_SAI7_MCLK
attach SAI1 MCLK2 to SAI7 MCLK

enumerator kAUDIOMIX_Attach_SAI1_MCLK1_To_SAI1_ROOT
attach SAI1 MCLK1 to SAI1 root

enumerator kAUDIOMIX_Attach_SAI1_MCLK1_To_SAI1_MCLK
attach SAI1 MCLK1 to SAI1 MCLK

enumerator kAUDIOMIX_Attach_SAI2_MCLK2_To_SAI1_ROOT
attach SAI2 MCLK2 to SAI1 root

enumerator kAUDIOMIX_Attach_SAI2_MCLK2_To_SAI2_ROOT
attach SAI2 MCLK2 to SAI2 root

enumerator kAUDIOMIX_Attach_SAI2_MCLK2_To_SAI3_ROOT
attach SAI2 MCLK2 to SAI3 root

enumerator kAUDIOMIX_Attach_SAI2_MCLK2_To_SAI5_ROOT
attach SAI2 MCLK2 to SAI5 root

enumerator kAUDIOMIX_Attach_SAI2_MCLK2_To_SAI6_ROOT
attach SAI2 MCLK2 to SAI6 root

enumerator kAUDIOMIX_Attach_SAI2_MCLK2_To_SAI7_ROOT
attach SAI2 MCLK2 to SAI7 root

enumerator kAUDIOMIX_Attach_SAI2_MCLK2_To_SAI1_MCLK
attach SAI2 MCLK2 to SAI1 MCLK

enumerator kAUDIOMIX_Attach_SAI2_MCLK2_To_SAI2_MCLK
attach SAI2 MCLK2 to SAI2 MCLK

enumerator kAUDIOMIX_Attach_SAI2_MCLK2_To_SAI3_MCLK
attach SAI2 MCLK2 to SAI3 MCLK

enumerator kAUDIOMIX_Attach_SAI2_MCLK2_To_SAI5_MCLK
attach SAI2 MCLK2 to SAI5 MCLK

enumerator kAUDIOMIX_Attach_SAI2_MCLK2_To_SAI6_MCLK
attach SAI2 MCLK2 to SAI6 MCLK

enumerator kAUDIOMIX_Attach_SAI2_MCLK2_To_SAI7_MCLK
attach SAI2 MCLK2 to SAI7 MCLK

enumerator kAUDIOMIX_Attach_SAI2_MCLK2_To_SPDIF_ExtCLK
attach SAI2 MCLK2 to SPDIF EXTCLK

enumerator kAUDIOMIX_Attach_SAI2_MCLK1_To_SAI2_ROOT
attach SAI2 MCLK1 to SAI2 root

enumerator kAUDIOMIX_Attach_SAI2_MCLK1_To_SAI2_MCLK
attach SAI2 MCLK1 to SAI2 MCLK

enumerator kAUDIOMIX_Attach_SAI3_MCLK2_To_SAI1_ROOT
attach SAI3 MCLK2 to SAI1 root

enumerator kAUDIOMIX_Attach_SAI3_MCLK2_To_SAI2_ROOT
attach SAI3 MCLK2 to SAI2 root

enumerator kAUDIOMIX_Attach_SAI3_MCLK2_To_SAI3_ROOT
attach SAI3 MCLK2 to SAI3 root

enumerator kAUDIOMIX_Attach_SAI3_MCLK2_To_SAI5_ROOT
attach SAI3 MCLK2 to SAI5 root

enumerator kAUDIOMIX_Attach_SAI3_MCLK2_To_SAI6_ROOT
attach SAI3 MCLK2 to SAI6 root

enumerator kAUDIOMIX_Attach_SAI3_MCLK2_To_SAI7_ROOT
attach SAI3 MCLK2 to SAI7 root

enumerator kAUDIOMIX_Attach_SAI3_MCLK2_To_SAI1_MCLK
attach SAI3 MCLK2 to SAI1 MCLK

enumerator kAUDIOMIX_Attach_SAI3_MCLK2_To_SAI2_MCLK
attach SAI3 MCLK2 to SAI2 MCLK

enumerator kAUDIOMIX_Attach_SAI3_MCLK2_To_SAI3_MCLK
attach SAI3 MCLK2 to SAI3 MCLK

enumerator kAUDIOMIX_Attach_SAI3_MCLK2_To_SAI5_MCLK
attach SAI3 MCLK2 to SAI5 MCLK

enumerator kAUDIOMIX_Attach_SAI3_MCLK2_To_SAI6_MCLK
attach SAI3 MCLK2 to SAI6 MCLK

enumerator kAUDIOMIX_Attach_SAI3_MCLK2_To_SAI7_MCLK
attach SAI3 MCLK2 to SAI7 MCLK

enumerator kAUDIOMIX_Attach_SAI3_MCLK2_To_SPDIF_ExtCLK
attach SAI3 MCLK2 to SPDIF EXTCLK

enumerator kAUDIOMIX_Attach_SAI3_MCLK1_To_SAI3_ROOT
attach SAI3 MCLK1 to SAI3 root

enumerator kAUDIOMIX_Attach_SAI3_MCLK1_To_SAI3_MCLK
attach SAI3 MCLK1 to SAI3 MCLK

enumerator kAUDIOMIX_Attach_SAI5_MCLK2_To_SAI1_ROOT
attach SAI5 MCLK2 to SAI1 root

enumerator kAUDIOMIX_Attach_SAI5_MCLK2_To_SAI2_ROOT
attach SAI5 MCLK2 to SAI2 root

enumerator kAUDIOMIX_Attach_SAI5_MCLK2_To_SAI3_ROOT
attach SAI5 MCLK2 to SAI3 root

enumerator kAUDIOMIX_Attach_SAI5_MCLK2_To_SAI5_ROOT
attach SAI5 MCLK2 to SAI5 root

enumerator kAUDIOMIX_Attach_SAI5_MCLK2_To_SAI6_ROOT
attach SAI5 MCLK2 to SAI6 root

enumerator kAUDIOMIX_Attach_SAI5_MCLK2_To_SAI7_ROOT
attach SAI5 MCLK2 to SAI7 root

enumerator kAUDIOMIX_Attach_SAI5_MCLK2_To_SAI1_MCLK
attach SAI5 MCLK2 to SAI1 MCLK

enumerator kAUDIOMIX_Attach_SAI5_MCLK2_To_SAI2_MCLK
attach SAI5 MCLK2 to SAI2 MCLK

enumerator kAUDIOMIX_Attach_SAI5_MCLK2_To_SAI3_MCLK
attach SAI5 MCLK2 to SAI3 MCLK

enumerator kAUDIOMIX_Attach_SAI5_MCLK2_To_SAI5_MCLK
attach SAI5 MCLK2 to SAI5 MCLK

enumerator kAUDIOMIX_Attach_SAI5_MCLK2_To_SAI6_MCLK
attach SAI5 MCLK2 to SAI6 MCLK

enumerator kAUDIOMIX_Attach_SAI5_MCLK2_To_SAI7_MCLK
attach SAI5 MCLK2 to SAI7 MCLK

enumerator kAUDIOMIX_Attach_SAI5_MCLK2_To_SPDIF_ExtCLK
attach SAI5 MCLK2 to SPDIF EXTCLK

enumerator kAUDIOMIX_Attach_SAI5_MCLK1_To_SAI5_ROOT
attach SAI5 MCLK1 to SAI5 root

enumerator kAUDIOMIX_Attach_SAI5_MCLK1_To_SAI5_MCLK
attach SAI5 MCLK1 to SAI5 MCLK

enumerator kAUDIOMIX_Attach_SAI6_MCLK2_To_SAI1_ROOT
attach SAI6 MCLK2 to SAI1 root

enumerator kAUDIOMIX_Attach_SAI6_MCLK2_To_SAI2_ROOT
attach SAI6 MCLK2 to SAI2 root

enumerator kAUDIOMIX_Attach_SAI6_MCLK2_To_SAI3_ROOT
attach SAI6 MCLK2 to SAI3 root

enumerator kAUDIOMIX_Attach_SAI6_MCLK2_To_SAI5_ROOT
attach SAI6 MCLK2 to SAI5 root

enumerator kAUDIOMIX_Attach_SAI6_MCLK2_To_SAI6_ROOT
attach SAI6 MCLK2 to SAI6 root

enumerator kAUDIOMIX_Attach_SAI6_MCLK2_To_SAI7_ROOT
attach SAI6 MCLK2 to SAI7 root

enumerator kAUDIOMIX_Attach_SAI6_MCLK2_To_SAI1_MCLK
attach SAI6 MCLK2 to SAI1 MCLK

enumerator kAUDIOMIX_Attach_SAI6_MCLK2_To_SAI2_MCLK
attach SAI6 MCLK2 to SAI2 MCLK

enumerator kAUDIOMIX_Attach_SAI6_MCLK2_To_SAI3_MCLK
attach SAI6 MCLK2 to SAI3 MCLK

enumerator kAUDIOMIX_Attach_SAI6_MCLK2_To_SAI5_MCLK
attach SAI6 MCLK2 to SAI5 MCLK

enumerator kAUDIOMIX_Attach_SAI6_MCLK2_To_SAI6_MCLK
attach SAI6 MCLK2 to SAI6 MCLK

enumerator kAUDIOMIX_Attach_SAI6_MCLK2_To_SAI7_MCLK
attach SAI6 MCLK2 to SAI7 MCLK

enumerator kAUDIOMIX_Attach_SAI6_MCLK2_To_SPDIF_ExtCLK
attach SAI6 MCLK2 to SPDIF EXTCLK

enumerator kAUDIOMIX_Attach_SAI6_MCLK1_To_SAI6_ROOT
attach SAI6 MCLK2 to SAI6 root

enumerator kAUDIOMIX_Attach_SAI6_MCLK1_To_SAI6_MCLK
attach SAI6 MCLK2 to SAI6 MCLK

enumerator kAUDIOMIX_Attach_SAI7_MCLK2_To_SAI1_ROOT
attach SAI7 MCLK2 to SAI1 root

enumerator kAUDIOMIX_Attach_SAI7_MCLK2_To_SAI2_ROOT
attach SAI7 MCLK2 to SAI2 root

enumerator kAUDIOMIX_Attach_SAI7_MCLK2_To_SAI3_ROOT
attach SAI7 MCLK2 to SAI3 root

enumerator kAUDIOMIX_Attach_SAI7_MCLK2_To_SAI5_ROOT
attach SAI7 MCLK2 to SAI5 root

enumerator kAUDIOMIX_Attach_SAI7_MCLK2_To_SAI6_ROOT
attach SAI7 MCLK2 to SAI6 root

enumerator kAUDIOMIX_Attach_SAI7_MCLK2_To_SAI7_ROOT
attach SAI7 MCLK2 to SAI7 root

enumerator kAUDIOMIX_Attach_SAI7_MCLK2_To_SAI1_MCLK
attach SAI7 MCLK2 to SAI1 MCLK

enumerator kAUDIOMIX_Attach_SAI7_MCLK2_To_SAI2_MCLK
attach SAI7 MCLK2 to SAI2 MCLK

enumerator kAUDIOMIX_Attach_SAI7_MCLK2_To_SAI3_MCLK
attach SAI7 MCLK2 to SAI3 MCLK

enumerator kAUDIOMIX_Attach_SAI7_MCLK2_To_SAI5_MCLK
attach SAI7 MCLK2 to SAI5 MCLK

enumerator kAUDIOMIX_Attach_SAI7_MCLK2_To_SAI6_MCLK
attach SAI7 MCLK2 to SAI6 MCLK

enumerator kAUDIOMIX_Attach_SAI7_MCLK2_To_SAI7_MCLK
attach SAI7 MCLK2 to SAI7 MCLK

enumerator kAUDIOMIX_Attach_SAI7_MCLK2_To_SPDIF_ExtCLK
attach SAI7 MCLK2 to SPDIF_ExtCLK

enumerator kAUDIOMIX_Attach_SAI7_MCLK1_To_SAI7_ROOT
attach SAI7 MCLK1 to SAI7 root

enumerator kAUDIOMIX_Attach_SAI7_MCLK1_To_SAI7_MCLK
attach SAI7 MCLK1 to SAI7 MCLK

enumerator kAUDIOMIX_Attach_PDM_Root_to_CCM_PDM
attach PDM ROOT to CCM PDM clock

enumerator kAUDIOMIX_Attach_PDM_Root_to_SAI_PLL_DVI2
attach PDM ROOT to sai pll div2

enumerator kAUDIOMIX_Attach_PDM_Root_to_SAI1_MCLK
attach PDM ROOT to SAI1 mclk

enum _audiomix_power_ctrl
audio mix power control

Values:

enumerator kAUDIOMIX_PowerCtrlEDMA
edma power control

enumerator kAUDIOMIX_PowerCtrlSDMA2
SDMA2 power control

enumerator kAUDIOMIX_PowerCtrlSDMA3
SDMA3 power control

enumerator kAUDIOMIX_PowerCtrlPDM
PDM power control

enumerator kAUDIOMIX_PowerCtrlSAI1
SAI1 power control

enumerator kAUDIOMIX_PowerCtrlSAI2
SAI2 power control

enumerator kAUDIOMIX_PowerCtrlSAI3
SAI3 power control

enumerator kAUDIOMIX_PowerCtrlSAI5
SAI5 power control

enumerator kAUDIOMIX_PowerCtrlSAI6
SAI6 power control

enumerator kAUDIOMIX_PowerCtrlSAI7
SAI7 power control

typedef enum *_audiomix_attch_clk* audiomix_attch_clk_t
audio mix attch clk id

FSL_AUDIOMIX_DRIVER_VERSION
AUDIOMIX driver version 2.0.1.

AUDIOMIX_ATTACH_ID(offset, mask, value)
AUDIOMIX attach id combination.

GET_AUDIOMIX_ATTACH_ID_OFFSET(id)

GET_AUDIOMIX_ATTACH_ID_MASK(id)

GET_AUDIOMIX_ATTACH_ID_value(id)

void AUDIOMIX_AttachClk(AUDIOMIX_Type *base, *audiomix_attch_clk_t* id)
audiomix attach clock.

Parameters

- base – audiomix base address.
- id – attach clock id.

static inline uint32_t AUDIOMIX_GetIPStopAck(AUDIOMIX_Type *base, uint32_t ip)
audiomix low power ack bit status.

Parameters

- base – audiomix base address.
- ip – reference _audiomix_power_ctrl, can be a value or combine value in _audiomix_power_ctrl

static inline void AUDIOMIX_SetIPStop(AUDIOMIX_Type *base, uint32_t ip)
audiomix low power stop mode

Parameters

- base – audiomix base address.
- ip – reference _audiomix_power_ctrl, can be a value or combine value in _audiomix_power_ctrl

static inline void AUDIOMIX_SetEARCReset(AUDIOMIX_Type *base, bool enable)

audiomix earc reset

Parameters

- base – audiomix base address.
- enable – true is reset, flase is release.

static inline void AUDIOMIX_SetEARCPhyReset(AUDIOMIX_Type *base, bool enable)

audiomix earc PHY reset

Parameters

- base – audiomix base address.
- enable – true is reset, flase is release.

void AUDIOMIX_InitAudioPll(AUDIOMIX_Type *base, const *ccm_analog_frac_pll_config_t* *config)

Initializes the SAI PLL. note This function can't detect whether the AUDIO PLL has been enabled and used by some IPs.

Parameters

- base – audiomix base address.
- config – Pointer to the configuration structure(see ref ccm_analog_frac_pll_config_t enumeration).

void AUDIOMIX_DeinitAudioPll1(AUDIOMIX_Type *base)

brief De-initialize the SAI PLL.

uint32_t AUDIOMIX_GetAudioPllFreq(AUDIOMIX_Type *base)

Get the sai PLL output freq.

Return values

sai – pll output freq.

2.6 CACHE: ARMV7-M7 CACHE Memory Controller

static inline void L1CACHE_EnableICache(void)

Enables cortex-m7 L1 instruction cache.

static inline void L1CACHE_DisableICache(void)

Disables cortex-m7 L1 instruction cache.

static inline void L1CACHE_InvalidateICache(void)

Invalidate cortex-m7 L1 instruction cache.

void L1CACHE_InvalidateICacheByRange(uint32_t address, uint32_t size_byte)

Invalidate cortex-m7 L1 instruction cache by range.

Note: The start address and size_byte should be 32-byte(FSL_FEATURE_L1ICACHE_LINESIZE_BYTE) aligned. The startAddr here will be forced to align to L1 I-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 start address of the memory to be invalidated.
- size_byte – The memory size.

static inline void L1CACHE_EnableDCache(void)

Enables cortex-m7 L1 data cache.

static inline void L1CACHE_DisableDCache(void)

Disables cortex-m7 L1 data cache.

static inline void L1CACHE_InvalidateDCache(void)

Invalidates cortex-m7 L1 data cache.

static inline void L1CACHE_CleanDCache(void)

Cleans cortex-m7 L1 data cache.

static inline void L1CACHE_CleanInvalidateDCache(void)

Cleans and Invalidates cortex-m7 L1 data cache.

static inline void L1CACHE_InvalidateDCacheByRange(uint32_t address, uint32_t size_byte)

Invalidates cortex-m7 L1 data cache by range.

Note: The start address and size_byte should be 32-byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned. The startAddr here will be forced to align to L1 D-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 start address of the memory to be invalidated.
- size_byte – The memory size.

static inline void L1CACHE_CleanDCacheByRange(uint32_t address, uint32_t size_byte)

Cleans cortex-m7 L1 data cache by range.

Note: The start address and size_byte should be 32-byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned. The startAddr here will be forced to align to L1 D-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 start address of the memory to be cleaned.
- size_byte – The memory size.

static inline void L1CACHE_CleanInvalidateDCacheByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates cortex-m7 L1 data cache by range.

Note: The start address and size_byte should be 32-byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned. The startAddr here will be forced to align to L1 D-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 start address of the memory to be clean and invalidated.
- size_byte – The memory size.

void ICACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates all instruction caches by range.

Both cortex-m7 L1 cache line and L2 PL310 cache line length is 32-byte.

Note: address and size should be aligned to cache line size 32-Byte due to the cache operation unit is one cache line. 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.

void DCACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates all data caches by range.

Both cortex-m7 L1 cache line and L2 PL310 cache line length is 32-byte.

Note: address and size should be aligned to cache line size 32-Byte due to the cache operation unit is one cache line. 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.

void DCACHE_CleanByRange(uint32_t address, uint32_t size_byte)

Cleans all data caches by range.

Both cortex-m7 L1 cache line and L2 PL310 cache line length is 32-byte.

Note: address and size should be aligned to cache line size 32-Byte due to the cache operation unit is one cache line. 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.

void DCACHE_CleanInvalidateByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates all data caches by range.

Both cortex-m7 L1 cache line and L2 PL310 cache line length is 32-byte.

Note: address and size should be aligned to cache line size 32-Byte due to the cache operation unit is one cache line. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be cleaned and invalidated.

FSL_CACHE_DRIVER_VERSION

cache driver version 2.0.4.

2.7 Clock

enum _clock_name

Clock name used to get clock frequency.

Values:

enumerator kCLOCK_CoreM7Clk

ARM M7 Core clock

enumerator kCLOCK_AxiClk

Main AXI bus clock.

enumerator kCLOCK_AhbClk

AHB bus clock.

enumerator kCLOCK_IpgClk

IPG bus clock.

enumerator kCLOCK_PerClk

Peripheral Clock.

enumerator kCLOCK_EnetIpgClk

ENET IPG Clock.

enumerator kCLOCK_Osc24MClk

OSC 24M clock.

enumerator kCLOCK_ArmPllClk

Arm PLL clock.

enumerator kCLOCK_DramPllClk

Dram PLL clock.

enumerator kCLOCK_VpuPllClk

Vpu PLL clock.

enumerator kCLOCK_SysPll1Clk

Sys PLL1 clock.

enumerator kCLOCK_SysPll1Div2Clk

Sys PLL1 clock divided by 2.

enumerator kCLOCK_SysPll1Div3Clk

Sys PLL1 clock divided by 3.

enumerator kCLOCK_SysPll1Div4Clk
Sys PLL1 clock divided by 4.

enumerator kCLOCK_SysPll1Div5Clk
Sys PLL1 clock divided by 5.

enumerator kCLOCK_SysPll1Div6Clk
Sys PLL1 clock divided by 6.

enumerator kCLOCK_SysPll1Div8Clk
Sys PLL1 clock divided by 8.

enumerator kCLOCK_SysPll1Div10Clk
Sys PLL1 clock divided by 10.

enumerator kCLOCK_SysPll1Div20Clk
Sys PLL1 clock divided by 20.

enumerator kCLOCK_SysPll2Clk
Sys PLL2 clock.

enumerator kCLOCK_SysPll2Div2Clk
Sys PLL2 clock divided by 2.

enumerator kCLOCK_SysPll2Div3Clk
Sys PLL2 clock divided by 3.

enumerator kCLOCK_SysPll2Div4Clk
Sys PLL2 clock divided by 4.

enumerator kCLOCK_SysPll2Div5Clk
Sys PLL2 clock divided by 5.

enumerator kCLOCK_SysPll2Div6Clk
Sys PLL2 clock divided by 6.

enumerator kCLOCK_SysPll2Div8Clk
Sys PLL2 clock divided by 8.

enumerator kCLOCK_SysPll2Div10Clk
Sys PLL2 clock divided by 10.

enumerator kCLOCK_SysPll2Div20Clk
Sys PLL2 clock divided by 20.

enumerator kCLOCK_SysPll3Clk
Sys PLL3 clock.

enumerator kCLOCK_AudioPll1Clk
Audio PLL1 clock.

enumerator kCLOCK_AudioPll2Clk
Audio PLL2 clock.

enumerator kCLOCK_VideoPll1Clk
Video PLL1 clock.

enumerator kCLOCK_ExtClk1
External clock1.

enumerator kCLOCK_ExtClk2
External clock2.

enumerator kCLOCK_ExtClk3
External clock3.

enumerator kCLOCK_ExtClk4
External clock4.

enumerator kCLOCK_NoneName
None Clock Name.

enum _clock_ip_name
CCM CCGR gate control.

Values:

enumerator kCLOCK_IpInvalid

enumerator kCLOCK_Debug
DEBUG Clock Gate.

enumerator kCLOCK_Dram
DRAM Clock Gate.

enumerator kCLOCK_Ecspi1
ECSPI1 Clock Gate.

enumerator kCLOCK_Ecspi2
ECSPI2 Clock Gate.

enumerator kCLOCK_Ecspi3
ECSPI3 Clock Gate.

enumerator kCLOCK_Enet1
ENET1 Clock Gate.

enumerator kCLOCK_Gpio1
GPIO1 Clock Gate.

enumerator kCLOCK_Gpio2
GPIO2 Clock Gate.

enumerator kCLOCK_Gpio3
GPIO3 Clock Gate.

enumerator kCLOCK_Gpio4
GPIO4 Clock Gate.

enumerator kCLOCK_Gpio5
GPIO5 Clock Gate.

enumerator kCLOCK_Gpt1
GPT1 Clock Gate.

enumerator kCLOCK_Gpt2
GPT2 Clock Gate.

enumerator kCLOCK_Gpt3
GPT3 Clock Gate.

enumerator kCLOCK_Gpt4
GPT4 Clock Gate.

enumerator kCLOCK_Gpt5
GPT5 Clock Gate.

enumerator kCLOCK_Gpt6
GPT6 Clock Gate.

enumerator kCLOCK_I2c1
I2C1 Clock Gate.

enumerator kCLOCK_I2c2
I2C2 Clock Gate.

enumerator kCLOCK_I2c3
I2C3 Clock Gate.

enumerator kCLOCK_I2c4
I2C4 Clock Gate.

enumerator kCLOCK_I2c5
I2C5 Clock Gate.

enumerator kCLOCK_I2c6
I2C6 Clock Gate.

enumerator kCLOCK_Can1
FlexCAN1 Clock Gate.

enumerator kCLOCK_Can2
FlexCAN2 Clock Gate.

enumerator kCLOCK_Enet_Qos
ENET QOS Clock Gate.

enumerator kCLOCK_Iomux
IOMUX Clock Gate.

enumerator kCLOCK_Ipmux1
IPMUX1 Clock Gate.

enumerator kCLOCK_Ipmux2
IPMUX2 Clock Gate.

enumerator kCLOCK_Ipmux3
IPMUX3 Clock Gate.

enumerator kCLOCK_Mu
MU Clock Gate.

enumerator kCLOCK_Ocram
OCRAM Clock Gate.

enumerator kCLOCK_OcramS
OCRAM S Clock Gate.

enumerator kCLOCK_Pwm1
PWM1 Clock Gate.

enumerator kCLOCK_Pwm2
PWM2 Clock Gate.

enumerator kCLOCK_Pwm3
PWM3 Clock Gate.

enumerator kCLOCK_Pwm4
PWM4 Clock Gate.

enumerator kCLOCK_Qspi
QSPI Clock Gate.

enumerator kCLOCK_Nand
NAND Clock Gate.

enumerator kCLOCK_Rdc
RDC Clock Gate.

enumerator kCLOCK_Sdma1
SDMA1 Clock Gate.

enumerator kCLOCK_Sec_Debug
SEC_DEBUG Clock Gate.

enumerator kCLOCK_Sema42_1
RDC SEMA42 Clock Gate.

enumerator kCLOCK_Sema42_2
RDC SEMA42 Clock Gate.

enumerator kCLOCK_Sim_enet
SIM_ENET Clock Gate.

enumerator kCLOCK_Sim_m
SIM_M Clock Gate.

enumerator kCLOCK_Sim_main
SIM_MAIN Clock Gate.

enumerator kCLOCK_Sim_s
SIM_S Clock Gate.

enumerator kCLOCK_Sim_wakeup
SIM_WAKEUP Clock Gate.

enumerator kCLOCK_Gpu2D
GPU2D Clock Gate.

enumerator kCLOCK_Gpu3D
GPU3D Clock Gate.

enumerator kCLOCK_Uart1
UART1 Clock Gate.

enumerator kCLOCK_Uart2
UART2 Clock Gate.

enumerator kCLOCK_Uart3
UART3 Clock Gate.

enumerator kCLOCK_Uart4
UART4 Clock Gate.

enumerator kCLOCK_Usdhc1
USDHC1 Clock Gate.

enumerator kCLOCK_Usdhc2
USDHC2 Clock Gate.

enumerator kCLOCK_Wdog1
WDOG1 Clock Gate.

enumerator kCLOCK_Wdog2
WDOG2 Clock Gate.

enumerator kCLOCK_Wdog3
WDOG3 Clock Gate.

enumerator kCLOCK_Vpu_G1
VPU_G1 Clock Gate.

enumerator kCLOCK_Gpu
GPU Clock Gate.

enumerator kCLOCK_Vpu_Vc8ke
VPU_VC8KE Clock Gate.

enumerator kCLOCK_Vpu_G2
VPU_G2 Clock Gate.

enumerator kCLOCK_Npu
NPU Clock Gate.

enumerator kCLOCK_Hsio
HSIO Clock Gate.

enumerator kCLOCK_Media
MEDIA Clock Gate.

enumerator kCLOCK_Usdhc3
USDHC3 Clock Gate.

enumerator kCLOCK_Hdmi
HDMI Clock Gate.

enumerator kCLOCK_TempSensor
TempSensor Clock Gate.

enumerator kCLOCK_Audio
AUDIO Clock Gate.

enumerator kCLOCK_Earc
EARC clock gate

enumerator kCLOCK_AudioDspDebug
AUDIO DSP DEBUG clock gate

enumerator kCLOCK_AudioDsp
audio dsp clock gate

enumerator kCLOCK_Spba2
SPBA2 clock gate

enumerator kCLOCK_Sdma3
SDMA3 clock gate

enumerator kCLOCK_Sdma2
SDMA2 clock gate

enumerator kCLOCK_Pdm
PDM clock gate

enumerator kCLOCK_Asrc
ASRC clock gate

enumerator kCLOCK_Sai7_Mclk3
SAI7 MCLK3 clock gate

enumerator kCLOCK_Sai7_Mclk2
SAI7 MCLK2 clock gate

enumerator kCLOCK_Sai7_Mclk1
SAI7 MCLK1 clock gate

enumerator kCLOCK_Sai7
SAI7 clock gate

enumerator kCLOCK_Sai6_Mclk3
SAI6 MCLK3 clock gate

enumerator kCLOCK_Sai6_Mclk2
SAI6 MCLK2 clock gate

enumerator kCLOCK_Sai6_Mclk1
SAI6 MCLK1 clock gate

enumerator kCLOCK_Sai6
SAI6 clock gate

enumerator kCLOCK_Sai5_Mclk3
SAI5 MCLK3 clock gate

enumerator kCLOCK_Sai5_Mclk2
sai5 MCLK2 clock gate

enumerator kCLOCK_Sai5_Mclk1
SAI5 MCLK1 clock gate

enumerator kCLOCK_Sai5
SAI5 clock gate

enumerator kCLOCK_Sai3_Mclk3
SAI3 MCLK3 clock gate

enumerator kCLOCK_Sai3_Mclk2
SAI3 MCLK2 clock gate

enumerator kCLOCK_Sai3_Mclk1
SAI3 MCLK1 clock gate

enumerator kCLOCK_Sai3
SAI3 clock gate

enumerator kCLOCK_Sai2_Mclk3
SAI2 MCLK3 clock gate

enumerator kCLOCK_Sai2_Mclk2
SAI2 MCLK2 clock gate

enumerator kCLOCK_Sai2_Mclk1
SAI2 MCLK1 clock gate

enumerator kCLOCK_Sai2
SAI2 clock gate

enumerator kCLOCK_Sai1_Mclk3
SAI1 MCLK3 clock gate

enumerator kCLOCK_Sai1_Mclk2
SAI1 MCLK2 clock gate

enumerator kCLOCK_Sai1_Mclk1
SAI1 MCLK1 clock gate

enumerator kCLOCK_Sai1
SAI1 clock gate

enumerator kCLOCK_EarcPhy
EARC PHY clock gate

enumerator kCLOCK_Mu3
MU3 clock gate

enumerator kCLOCK_Mu2
MU2 clock gate

enumerator kCLOCK_Pll
PLL clock gate

enumerator kCLOCK_Edma
EDMA clock gate

enumerator kCLOCK_Aud2htx
AUD2HTX clock gate

enumerator kCLOCK_Ocram_A
OCRAM A clock gate

enum _clock_root_control
ccm root name used to get clock frequency.

Values:

enumerator kCLOCK_RootM7
ARM Cortex-M7 Clock control name.

enumerator kCLOCK_RootHsioAxi
HSIO AXI Clock control name.

enumerator kCLOCK_RootMainAxi
MAIN AXI Clock control name.

enumerator kCLOCK_RootEnetAxi
ENET AXI Clock control name.

enumerator kCLOCK_RootNandUsdhcBus
NAND USDHC BUS Clock control name.

enumerator kCLOCK_RootVpuBus
VPU BUS Clock control name.

enumerator kCLOCK_RootMediaAxi
MEDIA AXI Clock control name.

enumerator kCLOCK_RootMediaApb
MEDIA APB Clock control name.

enumerator kCLOCK_RootHdmiApb
HDMI APB Clock control name.

enumerator kCLOCK_RootNoc
NOC Clock control name.

enumerator kCLOCK_RootAhb
AHB Clock control name.

enumerator kCLOCK_RootIpg
IPG Clock control name.

enumerator kCLOCK_RootAudioAhb
Audio AHB Clock control name.

enumerator kCLOCK_RootAudioIpg
Audio IPG Clock control name.

enumerator kCLOCK_RootDramAlt
DRAM ALT Clock control name.

enumerator kCLOCK_RootFlexCan1
FLEXCAN1 Clock control name.

enumerator kCLOCK_RootFlexCan2
FLEXCAN2 Clock control name.

enumerator kCLOCK_RootSai1
SAI1 Clock control name.

enumerator kCLOCK_RootSai2
SAI2 Clock control name.

enumerator kCLOCK_RootSai3
SAI3 Clock control name.

enumerator kCLOCK_RootSai5
SAI5 Clock control name.

enumerator kCLOCK_RootSai6
SAI6 Clock control name.

enumerator kCLOCK_RootSai7
SAI7 Clock control name.

enumerator kCLOCK_RootEnetQos
ENET QOS Clock control name.

enumerator kCLOCK_RootEnetQosTimer
ENET QOS TIMER Clock control name.

enumerator kCLOCK_RootEnetRef
ENET Clock control name.

enumerator kCLOCK_RootEnetTimer
ENET TIMER Clock control name.

enumerator kCLOCK_RootEnetPhy
ENET PHY Clock control name.

enumerator kCLOCK_RootNand
NAND Clock control name.

enumerator kCLOCK_RootQspi
QSPI Clock control name.

enumerator kCLOCK_RootUsdhc1
USDHC1 Clock control name.

enumerator kCLOCK_RootUsdhc2
USDHC2 Clock control name.

enumerator kCLOCK_RootUsdhc3
USDHC3 Clock control name.

enumerator kCLOCK_RootI2c1
I2C1 Clock control name.

enumerator kCLOCK_RootI2c2
I2C2 Clock control name.

enumerator kCLOCK_RootI2c3
I2C3 Clock control name.

enumerator kCLOCK_RootI2c4
I2C4 Clock control name.

enumerator kCLOCK_RootI2c5
I2C5 Clock control name.

enumerator kCLOCK_RootI2c6
I2C6 Clock control name.

enumerator kCLOCK_RootUart1
UART1 Clock control name.

enumerator kCLOCK_RootUart2
UART2 Clock control name.

enumerator kCLOCK_RootUart3
UART3 Clock control name.

enumerator kCLOCK_RootUart4
UART4 Clock control name.

enumerator kCLOCK_RootGic
GIC Clock control name.

enumerator kCLOCK_RootEcspi1
ECSPI1 Clock control name.

enumerator kCLOCK_RootEcspi2
ECSPI2 Clock control name.

enumerator kCLOCK_RootEcspi3
ECSPI3 Clock control name.

enumerator kCLOCK_RootPwm1
PWM1 Clock control name.

enumerator kCLOCK_RootPwm2
PWM2 Clock control name.

enumerator kCLOCK_RootPwm3
PWM3 Clock control name.

enumerator kCLOCK_RootPwm4
PWM4 Clock control name.

enumerator kCLOCK_RootGpt1
GPT1 Clock control name.

enumerator kCLOCK_RootGpt2
GPT2 Clock control name.

enumerator kCLOCK_RootGpt3
GPT3 Clock control name.

enumerator kCLOCK_RootGpt4
GPT4 Clock control name.

enumerator kCLOCK_RootGpt5
GPT5 Clock control name.

enumerator kCLOCK_RootGpt6
GPT6 Clock control name.

enumerator kCLOCK_RootWdog
WDOG Clock control name.

enumerator kCLOCK_RootPdm
PDM Clock control name.

enum _clock_root
ccm clock root used to get clock frequency.

Values:

enumerator kCLOCK_M7ClkRoot
ARM Cortex-M7 Clock control name.

enumerator kCLOCK_HsioAxiClkRoot
HSIO AXI Clock control name.

enumerator kCLOCK_MainAxiClkRoot
MAIN AXI Clock control name.

enumerator kCLOCK_EnetAxiClkRoot
ENET AXI Clock control name.

enumerator kCLOCK_NandUsdhcBusClkRoot
NAND USDHC BUS Clock control name.

enumerator kCLOCK_VpuBusClkRoot
VPU BUS Clock control name.

enumerator kCLOCK_MediaAxiClkRoot
MEDIA AXI Clock control name.

enumerator kCLOCK_MediaApbClkRoot
MEDIA APB Clock control name.

enumerator kCLOCK_HdmiApbClkRoot
HDMI APB Clock control name.

enumerator kCLOCK_NocClkRoot
NOC Clock control name.

enumerator kCLOCK_AhbClkRoot
AHB Clock control name.

enumerator kCLOCK_IpgClkRoot
IPG Clock control name.

enumerator kCLOCK_AudioAhbClkRoot
Audio AHB Clock control name.

enumerator kCLOCK_AudioIpgClkRoot
Audio IPG Clock control name.

enumerator kCLOCK_DramAltClkRoot
DRAM ALT Clock control name.

enumerator kCLOCK_FlexCan1ClkRoot
FLEXCAN1 Clock control name.

enumerator kCLOCK_FlexCan2ClkRoot
FLEXCAN2 Clock control name.

enumerator kCLOCK_Sai1ClkRoot
SAI1 Clock control name.

enumerator kCLOCK_Sai2ClkRoot
SAI2 Clock control name.

enumerator kCLOCK_Sai3ClkRoot
SAI3 Clock control name.

enumerator kCLOCK_Sai5ClkRoot
SAI5 Clock control name.

enumerator kCLOCK_Sai6ClkRoot
SAI6 Clock control name.

enumerator kCLOCK_Sai7ClkRoot
SAI7 Clock control name.

enumerator kCLOCK_EnetQosClkRoot
ENET QOS Clock control name.

enumerator kCLOCK_EnetQosTimerClkRoot
ENET QOS TIMER Clock control name.

enumerator kCLOCK_EnetRefClkRoot
ENET Clock control name.

enumerator kCLOCK_EnetTimerClkRoot
ENET TIMER Clock control name.

enumerator kCLOCK_EnetPhyClkRoot
ENET PHY Clock control name.

enumerator kCLOCK_NandClkRoot
NAND Clock control name.

enumerator kCLOCK_QspiClkRoot
QSPI Clock control name.

enumerator kCLOCK_Usdhc1ClkRoot
USDHC1 Clock control name.

enumerator kCLOCK_Usdhc2ClkRoot
USDHC2 Clock control name.

enumerator kCLOCK_Usdhc3ClkRoot
USDHC3 Clock control name.

enumerator kCLOCK_I2c1ClkRoot
I2C1 Clock control name.

enumerator kCLOCK_I2c2ClkRoot
I2C2 Clock control name.

enumerator kCLOCK_I2c3ClkRoot
I2C3 Clock control name.

enumerator kCLOCK_I2c4ClkRoot
I2C4 Clock control name.

enumerator kCLOCK_I2c5ClkRoot
I2C5 Clock control name.

enumerator kCLOCK_I2c6ClkRoot
I2C6 Clock control name.

enumerator kCLOCK_Uart1ClkRoot
UART1 Clock control name.

enumerator kCLOCK_Uart2ClkRoot
UART2 Clock control name.

enumerator kCLOCK_Uart3ClkRoot
UART3 Clock control name.

enumerator kCLOCK_Uart4ClkRoot
UART4 Clock control name.

enumerator kCLOCK_GicClkRoot
GIC Clock control name.

enumerator kCLOCK_Ecspi1ClkRoot
ECSPI1 Clock control name.

enumerator kCLOCK_Ecspi2ClkRoot
ECSPI2 Clock control name.

enumerator kCLOCK_Ecspi3ClkRoot
ECSPI3 Clock control name.

enumerator kCLOCK_Pwm1ClkRoot
PWM1 Clock control name.

enumerator kCLOCK_Pwm2ClkRoot
PWM2 Clock control name.

enumerator kCLOCK_Pwm3ClkRoot
PWM3 Clock control name.

enumerator kCLOCK_Pwm4ClkRoot
PWM4 Clock control name.

enumerator kCLOCK_Gpt1ClkRoot
GPT1 Clock control name.

enumerator kCLOCK_Gpt2ClkRoot
GPT2 Clock control name.

enumerator kCLOCK_Gpt3ClkRoot
GPT3 Clock control name.

enumerator kCLOCK_Gpt4ClkRoot
GPT4 Clock control name.

enumerator kCLOCK_Gpt5ClkRoot
GPT5 Clock control name.

enumerator kCLOCK_Gpt6ClkRoot
GPT6 Clock control name.

enumerator kCLOCK_WdogClkRoot
WDOG Clock control name.

enumerator kCLOCK_PdmClkRoot
PDM Clock control name.

enum __clock_rootmux_m7_clk_sel
Root clock select enumeration for ARM Cortex-M7 core.

Values:

enumerator kCLOCK_M7RootmuxOsc24M
ARM Cortex-M7 Clock from OSC 24M.

enumerator kCLOCK_M7RootmuxSysPll2Div5
ARM Cortex-M7 Clock from SYSTEM PLL2 divided by 5.

enumerator kCLOCK_M7RootmuxSysPll2Div4
ARM Cortex-M7 Clock from SYSTEM PLL2 divided by 4.

enumerator kCLOCK_M7RootmuxSysVpuPll
ARM Cortex-M7 Clock from VPU PLL.

enumerator kCLOCK_M7RootmuxSysPll1
ARM Cortex-M7 Clock from SYSTEM PLL1.

enumerator kCLOCK_M7RootmuxAudioPll1
ARM Cortex-M7 Clock from AUDIO PLL1.

enumerator kCLOCK_M7RootmuxVideoPll1
ARM Cortex-M7 Clock from VIDEO PLL1.

enumerator kCLOCK_M7RootmuxSysPll3
ARM Cortex-M7 Clock from SYSTEM PLL3.

enum __clock_rootmux_axi_clk_sel
Root clock select enumeration for MAIN AXI bus.

Values:

enumerator kCLOCK_AxiRootmuxOsc24M
ARM MAIN AXI Clock from OSC 24M.

enumerator kCLOCK_AxiRootmuxSysPll2Div3
ARM MAIN AXI Clock from SYSTEM PLL2 divided by 3.

enumerator kCLOCK_AxiRootmuxSysPll1
ARM MAIN AXI Clock from SYSTEM PLL1.

enumerator kCLOCK_AxiRootmuxSysPll2Div4
ARM MAIN AXI Clock from SYSTEM PLL2 divided by 4.

enumerator kCLOCK_AxiRootmuxSysPll2
ARM MAIN AXI Clock from SYSTEM PLL2.

enumerator kCLOCK_AxiRootmuxAudioPll1
ARM MAIN AXI Clock from AUDIO PLL1.

enumerator kCLOCK_AxiRootmuxVideoPll1
ARM MAIN AXI Clock from VIDEO PLL1.

enumerator kCLOCK_AxiRootmuxSysPll1Div8
ARM MAIN AXI Clock from SYSTEM PLL1 divided by 8.

enum _clock_rootmux_ahb_clk_sel
Root clock select enumeration for AHB bus.

Values:

enumerator kCLOCK_AhbRootmuxOsc24M
ARM AHB Clock from OSC 24M.

enumerator kCLOCK_AhbRootmuxSysPll1Div6
ARM AHB Clock from SYSTEM PLL1 divided by 6.

enumerator kCLOCK_AhbRootmuxSysPll1
ARM AHB Clock from SYSTEM PLL1.

enumerator kCLOCK_AhbRootmuxSysPll1Div2
ARM AHB Clock from SYSTEM PLL1 divided by 2.

enumerator kCLOCK_AhbRootmuxSysPll2Div8
ARM AHB Clock from SYSTEM PLL2 divided by 8.

enumerator kCLOCK_AhbRootmuxSysPll3
ARM AHB Clock from SYSTEM PLL3.

enumerator kCLOCK_AhbRootmuxAudioPll1
ARM AHB Clock from AUDIO PLL1.

enumerator kCLOCK_AhbRootmuxVideoPll1
ARM AHB Clock from VIDEO PLL1.

enum _clock_rootmux_audio_ahb_clk_sel
Root clock select enumeration for Audio AHB bus.

Values:

enumerator kCLOCK_AudioAhbRootmuxOsc24M
ARM Audio AHB Clock from OSC 24M.

enumerator kCLOCK_AudioAhbRootmuxSysPll2Div2
ARM Audio AHB Clock from SYSTEM PLL2 divided by 2.

enumerator kCLOCK_AudioAhbRootmuxSysPll1
ARM Audio AHB Clock from SYSTEM PLL1.

enumerator kCLOCK_AudioAhbRootmuxSysPll2
ARM Audio AHB Clock from SYSTEM PLL2.

enumerator kCLOCK_AudioAhbRootmuxSysPll2Div6
ARM Audio AHB Clock from SYSTEM PLL2 divided by 6.

enumerator kCLOCK_AudioAhbRootmuxSysPll3
ARM Audio AHB Clock from SYSTEM PLL3.

enumerator kCLOCK_AudioAhbRootmuxAudioPll1
ARM Audio AHB Clock from AUDIO PLL1.

enumerator kCLOCK_AudioAhbRootmuxVideoPll1
ARM Audio AHB Clock from VIDEO PLL1.

enum _clock_rootmux_qspi_clk_sel
Root clock select enumeration for QSPI peripheral.

Values:

enumerator kCLOCK_QspiRootmuxOsc24M
ARM QSPI Clock from OSC 24M.

enumerator kCLOCK_QspiRootmuxSysPll1Div2
ARM QSPI Clock from SYSTEM PLL1 divided by 2.

enumerator kCLOCK_QspiRootmuxSysPll2Div3
ARM QSPI Clock from SYSTEM PLL2 divided by 3.

enumerator kCLOCK_QspiRootmuxSysPll2Div2
ARM QSPI Clock from SYSTEM PLL2 divided by 2.

enumerator kCLOCK_QspiRootmuxAudioPll2
ARM QSPI Clock from AUDIO PLL2.

enumerator kCLOCK_QspiRootmuxSysPll1Div3
ARM QSPI Clock from SYSTEM PLL1 divided by 3

enumerator kCLOCK_QspiRootmuxSysPll3
ARM QSPI Clock from SYSTEM PLL3.

enumerator kCLOCK_QspiRootmuxSysPll1Div8
ARM QSPI Clock from SYSTEM PLL1 divided by 8.

enum _clock_rootmux_ecspi_clk_sel
Root clock select enumeration for ECSPI peripheral.

Values:

enumerator kCLOCK_EcspiRootmuxOsc24M
ECSPI Clock from OSC 24M.

enumerator kCLOCK_EcspiRootmuxSysPll2Div5
ECSPI Clock from SYSTEM PLL2 divided by 5.

enumerator kCLOCK_EcspiRootmuxSysPll1Div20
ECSPI Clock from SYSTEM PLL1 divided by 20.

enumerator kCLOCK_EcspiRootmuxSysPll1Div5
ECSPI Clock from SYSTEM PLL1 divided by 5.

enumerator kCLOCK_EcspiRootmuxSysPll1
ECSPI Clock from SYSTEM PLL1.

enumerator kCLOCK_EcspiRootmuxSysPll3
ECSPI Clock from SYSTEM PLL3.

enumerator kCLOCK_EcspiRootmuxSysPll2Div4
ECSPI Clock from SYSTEM PLL2 divided by 4.

enumerator kCLOCK_EcspiRootmuxAudioPll2
ECSPI Clock from AUDIO PLL2.

enum __clock__rootmux__enet__axi__clk__sel

Root clock select enumeration for ENET AXI bus.

Values:

enumerator kCLOCK__EnetAxiRootmuxOsc24M
ENET AXI Clock from OSC 24M.

enumerator kCLOCK__EnetAxiRootmuxSysPll1Div3
ENET AXI Clock from SYSTEM PLL1 divided by 3.

enumerator kCLOCK__EnetAxiRootmuxSysPll1
ENET AXI Clock from SYSTEM PLL1.

enumerator kCLOCK__EnetAxiRootmuxSysPll2Div4
ENET AXI Clock from SYSTEM PLL2 divided by 4.

enumerator kCLOCK__EnetAxiRootmuxSysPll2Div5
ENET AXI Clock from SYSTEM PLL2 divided by 5.

enumerator kCLOCK__EnetAxiRootmuxAudioPll1
ENET AXI Clock from AUDIO PLL1.

enumerator kCLOCK__EnetAxiRootmuxVideoPll1
ENET AXI Clock from VIDEO PLL1.

enumerator kCLOCK__EnetAxiRootmuxSysPll3
ENET AXI Clock from SYSTEM PLL3.

enum __clock__rootmux__enet__qos__clk__sel

Root clock select enumeration for ENET QOS Clcok.

Values:

enumerator kCLOCK__EnetQosRootmuxOsc24M
ENET QOS Clock from OSC 24M.

enumerator kCLOCK__EnetQosRootmuxSysPll2Div8
ENET QOS Clock from SYSTEM PLL2 divided by 8.

enumerator kCLOCK__EnetQosRootmuxSysPll2Div20
ENET QOS Clock from SYSTEM PLL2 divided by 20.

enumerator kCLOCK__EnetQosRootmuxSysPll2Div10
ENET QOS Clock from SYSTEM PLL2 divided by 10.

enumerator kCLOCK__EnetQosRootmuxSysPll1Div5
ENET QOS Clock from SYSTEM PLL1 divided by 5.

enumerator kCLOCK__EnetQosRootmuxAudioPll1
ENET QOS Clock from AUDIO PLL1.

enumerator kCLOCK__EnetQosRootmuxVideoPll1
ENET QOS Clock from VIDEO PLL1.

enumerator kCLOCK__EnetQosRootmuxExtClk4
ENET QOS Clock from External Clock 4.

enum __clock__rootmux__enet__ref__clk__sel

Root clock select enumeration for ENET REF Clcok.

Values:

enumerator kCLOCK_EnetRefRootmuxOsc24M
ENET REF Clock from OSC 24M.

enumerator kCLOCK_EnetRefRootmuxSysPll2Div8
ENET REF Clock from SYSTEM PLL2 divided by 8.

enumerator kCLOCK_EnetRefRootmuxSysPll2Div20
ENET REF Clock from SYSTEM PLL2 divided by 20.

enumerator kCLOCK_EnetRefRootmuxSysPll2Div10
ENET REF Clock from SYSTEM PLL2 divided by 10.

enumerator kCLOCK_EnetRefRootmuxSysPll1Div5
ENET REF Clock from SYSTEM PLL1 divided by 5.

enumerator kCLOCK_EnetRefRootmuxAudioPll1
ENET REF Clock from AUDIO PLL1.

enumerator kCLOCK_EnetRefRootmuxVideoPll1
ENET REF Clock from VIDEO PLL1.

enumerator kCLOCK_EnetRefRootmuxExtClk4
ENET REF Clock from External Clock 4.

enum __clock_rootmux_enet_qos_timer_clk_sel
Root clock select enumeration for ENET QOS TIMER Clcok.

Values:

enumerator kCLOCK_EnetQosTimerRootmuxOsc24M
ENET QOS TIMER Clock from OSC 24M.

enumerator kCLOCK_EnetQosTimerRootmuxSysPll2Div10
ENET QOS TIMER Clock from SYSTEM PLL2 divided by 10.

enumerator kCLOCK_EnetQosTimerRootmuxAudioPll1
ENET QOS TIMER Clock from AUDIO PLL1.

enumerator kCLOCK_EnetQosTimerRootmuxExtClk1
ENET QOS TIMER Clock from External Clock 1.

enumerator kCLOCK_EnetQosTimerRootmuxExtClk2
ENET QOS TIMER Clock External Clock 2.

enumerator kCLOCK_EnetQosTimerRootmuxExtClk3
ENET QOS TIMER Clock from External Clock 3.

enumerator kCLOCK_EnetQosTimerRootmuxExtClk4
ENET QOS TIMER Clock from External Clock 4.

enumerator kCLOCK_EnetQosTimerRootmuxVideoPll1
ENET QOS TIMER Clock from VIDEO PLL1.

enum __clock_rootmux_enet_timer_clk_sel
Root clock select enumeration for ENET TIMER Clcok.

Values:

enumerator kCLOCK_EnetTimerRootmuxOsc24M
ENET TIMER Clock from OSC 24M.

enumerator kCLOCK_EnetTimerRootmuxSysPll2Div10
ENET TIMER Clock from SYSTEM PLL2 divided by 10.

enumerator kCLOCK_EnetTimerRootmuxAudioPll1
ENET TIMER Clock from AUDIO PLL1.

enumerator kCLOCK_EnetTimerRootmuxExtClk1
ENET TIMER Clock from External Clock 1.

enumerator kCLOCK_EnetTimerRootmuxExtClk2
ENET TIMER Clock External Clock 2.

enumerator kCLOCK_EnetTimerRootmuxExtClk3
ENET TIMER Clock from External Clock 3.

enumerator kCLOCK_EnetTimerRootmuxExtClk4
ENET TIMER Clock from External Clock 4.

enumerator kCLOCK_EnetTimerRootmuxVideoPll1
ENET TIMER Clock from VIDEO PLL1.

enum __clock_rootmux_enet_phy_clk_sel
Root clock select enumeration for ENET PHY Clcok.

Values:

enumerator kCLOCK_EnetPhyRootmuxOsc24M
ENET PHY Clock from OSC 24M.

enumerator kCLOCK_EnetPhyRootmuxSysPll2Div20
ENET PHY Clock from SYSTEM PLL2 divided by 20.

enumerator kCLOCK_EnetPhyRootmuxSysPll2Div8
ENET PHY Clock from SYSTEM PLL2 divided by 8.

enumerator kCLOCK_EnetPhyRootmuxSysPll2Div5
ENET PHY Clock from SYSTEM PLL2 divided by 5.

enumerator kCLOCK_EnetPhyRootmuxSysPll2Div2
ENET PHY Clock from SYSTEM PLL2 divided by 2.

enumerator kCLOCK_EnetPhyRootmuxAudioPll1
ENET PHY Clock from AUDIO PLL1.

enumerator kCLOCK_EnetPhyRootmuxVideoPll1
ENET PHY Clock from VIDEO PLL1.

enumerator kCLOCK_EnetPhyRootmuxAudioPll2
ENET PHY Clock from AUDIO PLL2.

enum __clock_rootmux_flexcan_clk_sel
Root clock select enumeration for FLEXCAN peripheral.

Values:

enumerator kCLOCK_FlexCanRootmuxOsc24M
FLEXCAN Clock from OSC 24M.

enumerator kCLOCK_FlexCanRootmuxSysPll2Div5
FLEXCAN Clock from SYSTEM PLL2 divided by 5.

enumerator kCLOCK_FlexCanRootmuxSysPll1Div20
FLEXCAN Clock from SYSTEM PLL1 divided by 20.

enumerator kCLOCK_FlexCanRootmuxSysPll1Div5
FLEXCAN Clock from SYSTEM PLL1 divided by 5.

enumerator kCLOCK_FlexCanRootmuxSysPll1
FLEXCAN Clock from SYSTEM PLL1.

enumerator kCLOCK_FlexCanRootmuxSysPll3
FLEXCAN Clock from SYSTEM PLL3.

enumerator kCLOCK_FlexCanRootmuxSysPll2Div4
FLEXCAN Clock from SYSTEM PLL2 divided by 4.

enumerator kCLOCK_FlexCanRootmuxAudioPll2
FLEXCAN Clock from AUDIO PLL2.

enum _clock_rootmux_i2c_clk_sel
Root clock select enumeration for I2C peripheral.

Values:

enumerator kCLOCK_I2cRootmuxOsc24M
I2C Clock from OSC 24M.

enumerator kCLOCK_I2cRootmuxSysPll1Div5
I2C Clock from SYSTEM PLL1 divided by 5.

enumerator kCLOCK_I2cRootmuxSysPll2Div20
I2C Clock from SYSTEM PLL2 divided by 20.

enumerator kCLOCK_I2cRootmuxSysPll3
I2C Clock from SYSTEM PLL3 .

enumerator kCLOCK_I2cRootmuxAudioPll1
I2C Clock from AUDIO PLL1.

enumerator kCLOCK_I2cRootmuxVideoPll1
I2C Clock from VIDEO PLL1.

enumerator kCLOCK_I2cRootmuxAudioPll2
I2C Clock from AUDIO PLL2.

enumerator kCLOCK_I2cRootmuxSysPll1Div6
I2C Clock from SYSTEM PLL1 divided by 6.

enum _clock_rootmux_uart_clk_sel
Root clock select enumeration for UART peripheral.

Values:

enumerator kCLOCK_UartRootmuxOsc24M
UART Clock from OSC 24M.

enumerator kCLOCK_UartRootmuxSysPll1Div10
UART Clock from SYSTEM PLL1 divided by 10.

enumerator kCLOCK_UartRootmuxSysPll2Div5
UART Clock from SYSTEM PLL2 divided by 5.

enumerator kCLOCK_UartRootmuxSysPll2Div10
UART Clock from SYSTEM PLL2 divided by 10.

enumerator kCLOCK_UartRootmuxSysPll3
UART Clock from SYSTEM PLL3.

enumerator kCLOCK_UartRootmuxExtClk2
UART Clock from External Clock 2.

enumerator kCLOCK_UartRootmuxExtClk34
UART Clock from External Clock 3, External Clock 4.

enumerator kCLOCK_UartRootmuxAudioPll2
UART Clock from Audio PLL2.

enum _clock_rootmux_gpt
Root clock select enumeration for GPT peripheral.

Values:

enumerator kCLOCK_GptRootmuxOsc24M
GPT Clock from OSC 24M.

enumerator kCLOCK_GptRootmuxSystemPll2Div10
GPT Clock from SYSTEM PLL2 divided by 10.

enumerator kCLOCK_GptRootmuxSysPll1Div2
GPT Clock from SYSTEM PLL1 divided by 2.

enumerator kCLOCK_GptRootmuxSysPll1Div20
GPT Clock from SYSTEM PLL1 divided by 20.

enumerator kCLOCK_GptRootmuxVideoPll1
GPT Clock from VIDEO PLL1.

enumerator kCLOCK_GptRootmuxSystemPll1Div10
GPT Clock from SYSTEM PLL1 divided by 10.

enumerator kCLOCK_GptRootmuxAudioPll1
GPT Clock from AUDIO PLL1.

enumerator kCLOCK_GptRootmuxExtClk123
GPT Clock from External Clock1, External Clock2, External Clock3.

enum _clock_rootmux_wdog_clk_sel
Root clock select enumeration for WDOG peripheral.

Values:

enumerator kCLOCK_WdogRootmuxOsc24M
WDOG Clock from OSC 24M.

enumerator kCLOCK_WdogRootmuxSysPll1Div6
WDOG Clock from SYSTEM PLL1 divided by 6.

enumerator kCLOCK_WdogRootmuxSysPll1Div5
WDOG Clock from SYSTEM PLL1 divided by 5.

enumerator kCLOCK_WdogRootmuxVpuPll
WDOG Clock from VPU DLL.

enumerator kCLOCK_WdogRootmuxSystemPll2Div8
WDOG Clock from SYSTEM PLL2 divided by 8.

enumerator kCLOCK_WdogRootmuxSystemPll3
WDOG Clock from SYSTEM PLL3.

enumerator kCLOCK_WdogRootmuxSystemPll1Div10
WDOG Clock from SYSTEM PLL1 divided by 10.

enumerator kCLOCK_WdogRootmuxSystemPll2Div6
WDOG Clock from SYSTEM PLL2 divided by 6.

enum _clock_rootmux_pwm_clk_sel

Root clock select enumeration for PWM peripheral.

Values:

enumerator kCLOCK_PwmRootmuxOsc24M

PWM Clock from OSC 24M.

enumerator kCLOCK_PwmRootmuxSysPll2Div10

PWM Clock from SYSTEM PLL2 divided by 10.

enumerator kCLOCK_PwmRootmuxSysPll1Div5

PWM Clock from SYSTEM PLL1 divided by 5.

enumerator kCLOCK_PwmRootmuxSysPll1Div20

PWM Clock from SYSTEM PLL1 divided by 20.

enumerator kCLOCK_PwmRootmuxSystemPll3

PWM Clock from SYSTEM PLL3.

enumerator kCLOCK_PwmRootmuxExtClk12

PWM Clock from External Clock1, External Clock2.

enumerator kCLOCK_PwmRootmuxSystemPll1Div10

PWM Clock from SYSTEM PLL1 divided by 10.

enumerator kCLOCK_PwmRootmuxVideoPll1

PWM Clock from VIDEO PLL1.

enum _clock_rootmux_sai_clk_sel

Root clock select enumeration for SAI peripheral.

Values:

enumerator kCLOCK_SaiRootmuxOsc24M

SAI Clock from OSC 24M.

enumerator kCLOCK_SaiRootmuxAudioPll1

SAI Clock from AUDIO PLL1.

enumerator kCLOCK_SaiRootmuxAudioPll2

SAI Clock from AUDIO PLL2.

enumerator kCLOCK_SaiRootmuxVideoPll1

SAI Clock from VIDEO PLL1.

enumerator kCLOCK_SaiRootmuxSysPll1Div6

SAI Clock from SYSTEM PLL1 divided by 6.

enumerator kCLOCK_SaiRootmuxOsc26m

SAI Clock from OSC HDMI 26M.

enumerator kCLOCK_SaiRootmuxExtClk1

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

enumerator kCLOCK_SaiRootmuxExtClk2

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

enum _clock_rootmux_pdm_clk_sel

Root clock select enumeration for PDM peripheral.

Values:

enumerator kCLOCK_PdmRootmuxOsc24M
PDM Clock from OSC 24M.

enumerator kCLOCK_PdmRootmuxSysPll2Div10
PDM Clock from SYSTEM PLL2 divided by 10.

enumerator kCLOCK_PdmRootmuxAudioPll1
PDM Clock from AUDIO PLL1.

enumerator kCLOCK_PdmRootmuxSysPll1
PDM Clock from SYSTEM PLL1.

enumerator kCLOCK_PdmRootmuxSysPll2
PDM Clock from SYSTEM PLL2.

enumerator kCLOCK_PdmRootmuxSysPll3
PDM Clock from SYSTEM PLL3.

enumerator kCLOCK_PdmRootmuxExtClk3
PDM Clock from External Clock3.

enumerator kCLOCK_PdmRootmuxAudioPll2
PDM Clock from AUDIO PLL2.

enum _clock_rootmux_noc_clk_sel
Root clock select enumeration for NOC CLK.

Values:

enumerator kCLOCK_NocRootmuxOsc24M
NOC Clock from OSC 24M.

enumerator kCLOCK_NocRootmuxSysPll1
NOC Clock from SYSTEM PLL1.

enumerator kCLOCK_NocRootmuxSysPll3
NOC Clock from SYSTEM PLL3.

enumerator kCLOCK_NocRootmuxSysPll2
NOC Clock from SYSTEM PLL2.

enumerator kCLOCK_NocRootmuxSysPll2Div2
NOC Clock from SYSTEM PLL2 divided by 2.

enumerator kCLOCK_NocRootmuxAudioPll1
NOC Clock from AUDIO PLL1.

enumerator kCLOCK_NocRootmuxVideoPll1
NOC Clock from VIDEO PLL1.

enumerator kCLOCK_NocRootmuxAudioPll2
NOC Clock from AUDIO PLL2.

enum _clock_pll_gate
CCM PLL gate control.

Values:

enumerator kCLOCK_ArmPllGate
ARM PLL Gate.

enumerator kCLOCK_GpuPllGate
GPU PLL Gate.

enumerator kCLOCK_VpuPllGate
VPU PLL Gate.

enumerator kCLOCK_DramPllGate
DRAM PLL1 Gate.

enumerator kCLOCK_SysPll1Gate
SYSTEM PLL1 Gate.

enumerator kCLOCK_SysPll1Div2Gate
SYSTEM PLL1 Div2 Gate.

enumerator kCLOCK_SysPll1Div3Gate
SYSTEM PLL1 Div3 Gate.

enumerator kCLOCK_SysPll1Div4Gate
SYSTEM PLL1 Div4 Gate.

enumerator kCLOCK_SysPll1Div5Gate
SYSTEM PLL1 Div5 Gate.

enumerator kCLOCK_SysPll1Div6Gate
SYSTEM PLL1 Div6 Gate.

enumerator kCLOCK_SysPll1Div8Gate
SYSTEM PLL1 Div8 Gate.

enumerator kCLOCK_SysPll1Div10Gate
SYSTEM PLL1 Div10 Gate.

enumerator kCLOCK_SysPll1Div20Gate
SYSTEM PLL1 Div20 Gate.

enumerator kCLOCK_SysPll2Gate
SYSTEM PLL2 Gate.

enumerator kCLOCK_SysPll2Div2Gate
SYSTEM PLL2 Div2 Gate.

enumerator kCLOCK_SysPll2Div3Gate
SYSTEM PLL2 Div3 Gate.

enumerator kCLOCK_SysPll2Div4Gate
SYSTEM PLL2 Div4 Gate.

enumerator kCLOCK_SysPll2Div5Gate
SYSTEM PLL2 Div5 Gate.

enumerator kCLOCK_SysPll2Div6Gate
SYSTEM PLL2 Div6 Gate.

enumerator kCLOCK_SysPll2Div8Gate
SYSTEM PLL2 Div8 Gate.

enumerator kCLOCK_SysPll2Div10Gate
SYSTEM PLL2 Div10 Gate.

enumerator kCLOCK_SysPll2Div20Gate
SYSTEM PLL2 Div20 Gate.

enumerator kCLOCK_SysPll3Gate
SYSTEM PLL3 Gate.

enumerator kCLOCK_AudioPll1Gate
AUDIO PLL1 Gate.

enumerator kCLOCK_AudioPll2Gate
AUDIO PLL2 Gate.

enumerator kCLOCK_VideoPll1Gate
VIDEO PLL1 Gate.

enumerator kCLOCK_VideoPll2Gate
VIDEO PLL2 Gate.

enum _clock_gate_value
CCM gate control value.

Values:

enumerator kCLOCK_ClockNotNeeded
Clock always disabled.

enumerator kCLOCK_ClockNeededRun
Clock enabled when CPU is running.

enumerator kCLOCK_ClockNeededRunWait
Clock enabled when CPU is running or in WAIT mode.

enumerator kCLOCK_ClockNeededAll
Clock always enabled.

enum _clock_pll_bypass_ctrl
PLL control names for PLL bypass.

These constants define the PLL control names for PLL bypass.

- 0:15: REG offset to CCM_ANALOG_BASE in bytes.
- 16:20: bypass bit shift.

Values:

enumerator kCLOCK_AudioPll1BypassCtrl
CCM Audio PLL1 bypass Control.

enumerator kCLOCK_AudioPll2BypassCtrl
CCM Audio PLL2 bypass Control.

enumerator kCLOCK_VideoPll1BypassCtrl
CCM Video Pll1 bypass Control.

enumerator kCLOCK_DramPllInternalPll1BypassCtrl
CCM DRAM PLL bypass Control.

enumerator kCLOCK_ArmPllPwrBypassCtrl
CCM Arm PLL bypass Control.

enumerator kCLOCK_SysPll1InternalPll1BypassCtrl
CCM System PLL1 bypass Control.

enumerator kCLOCK_SysPll2InternalPll1BypassCtrl
CCM System PLL2 bypass Control.

enumerator kCLOCK_SysPll3InternalPll1BypassCtrl
CCM System PLL3 bypass Control.

enum _ccm_analog_pll_clke

PLL clock names for clock enable/disable settings.

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

- 0:15: REG offset to CCM_ANALOG_BASE in bytes.
- 16:20: Clock enable bit shift.

Values:

enumerator kCLOCK_AudioPll1Clke
Audio pll1 clke

enumerator kCLOCK_AudioPll2Clke
Audio pll2 clke

enumerator kCLOCK_VideoPll1Clke
Video pll1 clke

enumerator kCLOCK_DramPllClke
Dram pll clke

enumerator kCLOCK_ArmPllClke
Arm pll clke

enumerator kCLOCK_SystemPll1Clke
System pll1 clke

enumerator kCLOCK_SystemPll1Div2Clke
System pll1 Div2 clke

enumerator kCLOCK_SystemPll1Div3Clke
System pll1 Div3 clke

enumerator kCLOCK_SystemPll1Div4Clke
System pll1 Div4 clke

enumerator kCLOCK_SystemPll1Div5Clke
System pll1 Div5 clke

enumerator kCLOCK_SystemPll1Div6Clke
System pll1 Div6 clke

enumerator kCLOCK_SystemPll1Div8Clke
System pll1 Div8 clke

enumerator kCLOCK_SystemPll1Div10Clke
System pll1 Div10 clke

enumerator kCLOCK_SystemPll1Div20Clke
System pll1 Div20 clke

enumerator kCLOCK_SystemPll2Clke
System pll2 clke

enumerator kCLOCK_SystemPll2Div2Clke
System pll2 Div2 clke

enumerator kCLOCK_SystemPll2Div3Clke
System pll2 Div3 clke

enumerator kCLOCK_SystemPll2Div4Clke
System pll2 Div4 clke

enumerator kCLOCK_SystemPll2Div5Clke
System pll2 Div5 clke

enumerator kCLOCK_SystemPll2Div6Clke
System pll2 Div6 clke

enumerator kCLOCK_SystemPll2Div8Clke
System pll2 Div8 clke

enumerator kCLOCK_SystemPll2Div10Clke
System pll2 Div10 clke

enumerator kCLOCK_SystemPll2Div20Clke
System pll2 Div20 clke

enumerator kCLOCK_SystemPll3Clke
System pll3 clke

enum _clock_pll_ctrl
ANALOG Power down override control.

Values:

enumerator kCLOCK_AudioPll1Ctrl

enumerator kCLOCK_AudioPll2Ctrl

enumerator kCLOCK_VideoPll1Ctrl

enumerator kCLOCK_DramPllCtrl

enumerator kCLOCK_VpuPllCtrl

enumerator kCLOCK_ArmPllCtrl

enumerator kCLOCK_SystemPll1Ctrl

enumerator kCLOCK_SystemPll2Ctrl

enumerator kCLOCK_SystemPll3Ctrl

PLL reference clock select.

Values:

enumerator kANALOG_PllRefOsc24M
reference OSC 24M

enumerator kANALOG_PllPadClk
reference PAD CLK

typedef enum *_clock_name* clock_name_t
Clock name used to get clock frequency.

typedef enum *_clock_ip_name* clock_ip_name_t
CCM CCGR gate control.

typedef enum *_clock_root_control* clock_root_control_t
ccm root name used to get clock frequency.

typedef enum *_clock_root* clock_root_t
ccm clock root used to get clock frequency.

typedef enum *_clock_rootmux_m7_clk_sel* clock_rootmux_m7_clk_sel_t
Root clock select enumeration for ARM Cortex-M7 core.

typedef enum *_clock_rootmux_axi_clk_sel* clock_rootmux_axi_clk_sel_t
Root clock select enumeration for MAIN AXI bus.

typedef enum *_clock_rootmux_ahb_clk_sel* clock_rootmux_ahb_clk_sel_t
Root clock select enumeration for AHB bus.

typedef enum *_clock_rootmux_audio_ahb_clk_sel* clock_rootmux_audio_ahb_clk_sel_t
Root clock select enumeration for Audio AHB bus.

typedef enum *_clock_rootmux_qspi_clk_sel* clock_rootmux_qspi_clk_sel_t
Root clock select enumeration for QSPI peripheral.

typedef enum *_clock_rootmux_ecspi_clk_sel* clock_rootmux_ecspi_clk_sel_t
Root clock select enumeration for ECSPI peripheral.

typedef enum *_clock_rootmux_enet_axi_clk_sel* clock_rootmux_enet_axi_clk_sel_t
Root clock select enumeration for ENET AXI bus.

typedef enum *_clock_rootmux_enet_qos_clk_sel* clock_rootmux_enet_qos_clk_sel_t
Root clock select enumeration for ENET QOS Clcok.

typedef enum *_clock_rootmux_enet_ref_clk_sel* clock_rootmux_enet_ref_clk_sel_t
Root clock select enumeration for ENET REF Clcok.

typedef enum *_clock_rootmux_enet_qos_timer_clk_sel* clock_rootmux_enet_qos_timer_clk_sel_t
Root clock select enumeration for ENET QOS TIMER Clcok.

typedef enum *_clock_rootmux_enet_timer_clk_sel* clock_rootmux_enet_timer_clk_sel_t
Root clock select enumeration for ENET TIMER Clcok.

typedef enum *_clock_rootmux_enet_phy_clk_sel* clock_rootmux_enet_phy_clk_sel_t
Root clock select enumeration for ENET PHY Clcok.

typedef enum *_clock_rootmux_flexcan_clk_sel* clock_rootmux_flexcan_clk_sel_t
Root clock select enumeration for FLEXCAN peripheral.

typedef enum *_clock_rootmux_i2c_clk_sel* clock_rootmux_i2c_clk_sel_t
Root clock select enumeration for I2C peripheral.

typedef enum *_clock_rootmux_uart_clk_sel* clock_rootmux_uart_clk_sel_t
Root clock select enumeration for UART peripheral.

typedef enum *_clock_rootmux_gpt* clock_rootmux_gpt_t
Root clock select enumeration for GPT peripheral.

typedef enum *_clock_rootmux_wdog_clk_sel* clock_rootmux_wdog_clk_sel_t
Root clock select enumeration for WDOG peripheral.

typedef enum *_clock_rootmux_pwm_clk_sel* clock_rootmux_Pwm_clk_sel_t
Root clock select enumeration for PWM peripheral.

typedef enum *_clock_rootmux_sai_clk_sel* clock_rootmux_sai_clk_sel_t
Root clock select enumeration for SAI peripheral.

typedef enum *_clock_rootmux_pdm_clk_sel* clock_rootmux_pdm_clk_sel_t
Root clock select enumeration for PDM peripheral.

typedef enum *_clock_rootmux_noc_clk_sel* clock_rootmux_noc_clk_sel_t
Root clock select enumeration for NOC CLK.

typedef enum *_clock_pll_gate* clock_pll_gate_t

CCM PLL gate control.

typedef enum *_clock_gate_value* clock_gate_value_t

CCM gate control value.

typedef enum *_clock_pll_bypass_ctrl* clock_pll_bypass_ctrl_t

PLL control names for PLL bypass.

These constants define the PLL control names for PLL bypass.

- 0:15: REG offset to CCM_ANALOG_BASE in bytes.
- 16:20: bypass bit shift.

typedef enum *_ccm_analog_pll_clke* clock_pll_clke_t

PLL clock names for clock enable/disable settings.

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

- 0:15: REG offset to CCM_ANALOG_BASE in bytes.
- 16:20: Clock enable bit shift.

typedef enum *_clock_pll_ctrl* clock_pll_ctrl_t

ANALOG Power down override control.

typedef struct *_ccm_analog_frac_pll_config* ccm_analog_frac_pll_config_t

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

typedef struct *_ccm_analog_integer_pll_config* ccm_analog_integer_pll_config_t

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

FSL_CLOCK_DRIVER_VERSION

CLOCK driver version 2.2.0.

SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY

OSC24M_CLK_FREQ

XTAL 24M clock frequency.

CLKPAD_FREQ

pad clock frequency.

ECSPI_CLOCKS

Clock ip name array for ECSPI.

EDMA_CLOCKS

Clock ip name array for EDMA.

ENET_CLOCKS

Clock ip name array for ENET.

ENETQOS_CLOCKS

Clock ip name array for ENET_QOS.

FLEXCAN_CLOCKS

Clock ip name array for FLEXCAN.

GPIO_CLOCKS

Clock ip name array for GPIO.

GPT_CLOCKS
: Clock ip name array for GPT.

I2C_CLOCKS
: Clock ip name array for I2C.

IOMUX_CLOCKS
: Clock ip name array for IOMUX.

IPMUX_CLOCKS
: Clock ip name array for IPMUX.

PWM_CLOCKS
: Clock ip name array for PWM.

RDC_CLOCKS
: Clock ip name array for RDC.

SAI_CLOCKS
: Clock ip name array for SAI.

RDC_SEMA42_CLOCKS
: Clock ip name array for RDC SEMA42.

UART_CLOCKS
: Clock ip name array for UART.

USDHC_CLOCKS
: Clock ip name array for USDHC.

WDOG_CLOCKS
: Clock ip name array for WDOG.

TMU_CLOCKS
: Clock ip name array for TEMPSENSOR.

SDMA_CLOCKS
: Clock ip name array for SDMA.

MU_CLOCKS
: Clock ip name array for MU.

QSPI_CLOCKS
: Clock ip name array for QSPI.

PDM_CLOCKS
: Clock ip name array for PDM.

ASRC_CLOCKS
: Clock ip name array for ASRC.

CCM_BIT_FIELD_EXTRACTION(val, mask, shift)
: CCM reg macros to extract corresponding registers bit field.

CCM_REG_OFF(root, off)
: CCM reg macros to map corresponding registers.

CCM_REG(root)

CCM_REG_SET(root)

CCM_REG_CLR(root)

AUDIO_PLL1_GEN_CTRL_OFFSET
: CCM Analog registers offset.

AUDIO_PLL2_GEN_CTRL_OFFSET

VIDEO_PLL1_GEN_CTRL_OFFSET

GPU_PLL_GEN_CTRL_OFFSET

VPU_PLL_GEN_CTRL_OFFSET

ARM_PLL_GEN_CTRL_OFFSET

SYS_PLL1_GEN_CTRL_OFFSET

SYS_PLL2_GEN_CTRL_OFFSET

SYS_PLL3_GEN_CTRL_OFFSET

DRAM_PLL_GEN_CTRL_OFFSET

CCM_ANALOG_TUPLE(reg, shift)
: CCM ANALOG tuple macros to map corresponding registers and bit fields.

CCM_ANALOG_TUPLE_SHIFT(tuple)

CCM_ANALOG_TUPLE_REG_OFF(base, tuple, off)

CCM_ANALOG_TUPLE_REG(base, tuple)

CLOCK_GATE_IN_AUDIOMIX
: CCM CCGR and root tuple.

CLOCK_GATE_IN_CCM

CLOCK_GATE_TYPE(tuple)

CCM_TUPLE(ccgr, root)

CCM_TUPLE_CCGR(tuple)

CCM_TUPLE_ROOT(tuple)

AUDIOMIX_TUPLE(offset, gate, root)
: audio mix CCGR

AUDIOMIX_TUPLE_OFFSET(tuple)

AUDIOMIX_TUPLE_GATE(tuple)

AUDIOMIX_TUPLE_ROOT(tuple)

CLOCK_ROOT_SOURCE
: clock root source

CLOCK_ROOT_CONTROL_TUPLE

kCLOCK_CoreSysClk
: For compatible with other platforms without CCM.

CLOCK_GetCoreSysClkFreq
: For compatible with other platforms without CCM.

static inline void CLOCK_SetRootMux(*clock_root_control_t* rootClk, uint32_t mux)

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

Parameters

- rootClk – Root clock control (see clock_root_control_t enumeration).
- mux – Root mux value (see _ccm_rootmux_xxx enumeration).

static inline uint32_t CLOCK_GetRootMux(*clock_root_control_t* rootClk)

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

Parameters

- rootClk – Root clock control (see clock_root_control_t enumeration).

Returns

Root mux value (see _ccm_rootmux_xxx enumeration).

static inline void CLOCK_EnableRoot(*clock_root_control_t* rootClk)

Enable clock root.

Parameters

- rootClk – Root clock control (see clock_root_control_t enumeration)

static inline void CLOCK_DisableRoot(*clock_root_control_t* rootClk)

Disable clock root.

Parameters

- rootClk – Root control (see clock_root_control_t enumeration)

static inline bool CLOCK_IsRootEnabled(*clock_root_control_t* rootClk)

Check whether clock root is enabled.

Parameters

- rootClk – Root control (see clock_root_control_t enumeration)

Returns

CCM root enabled or not.

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

void CLOCK_UpdateRoot(*clock_root_control_t* ccmRootClk, uint32_t mux, uint32_t pre, uint32_t post)

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

Parameters

- ccmRootClk – Root control (see clock_root_control_t enumeration)
- mux – mux value (see _ccm_rootmux_xxx enumeration)
- pre – Pre divider value (0-7, divider=n+1)
- post – Post divider value (0-63, divider=n+1)

void CLOCK_SetRootDivider(*clock_root_control_t* ccmRootClk, uint32_t pre, uint32_t post)

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

Parameters

- ccmRootClk – Root control (see clock_root_control_t enumeration)
- pre – Pre divider value (1-8)
- post – Post divider value (1-64)

static inline uint32_t CLOCK_GetRootPreDivider(*clock_root_control_t* rootClk)

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

Parameters

- rootClk – Root clock name (see clock_root_control_t enumeration).

Returns

Root Pre divider value.

static inline uint32_t CLOCK_GetRootPostDivider(*clock_root_control_t* rootClk)

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

Parameters

- rootClk – Root clock name (see clock_root_control_t enumeration).

Returns

Root Post divider value.

static inline void CLOCK_ControlGate(uintptr_t ccmGate, *clock_gate_value_t* control)

Set PLL or CCGR gate control.

Parameters

- ccmGate – Gate control (see clock_pll_gate_t and clock_ip_name_t enumeration)
- control – Gate control value (see clock_gate_value_t)

void CLOCK_EnableClock(*clock_ip_name_t* ccmGate)

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

Parameters

- ccmGate – Gate control for each module (see clock_ip_name_t enumeration).

void CLOCK_DisableClock(*clock_ip_name_t* ccmGate)

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

Parameters

- ccmGate – Gate control for each module (see clock_ip_name_t enumeration).

static inline void CLOCK_PowerUpPll(CCM_ANALOG_Type *base, *clock_pll_ctrl_t* pllControl)

Power up PLL.

Parameters

- base – CCM_ANALOG base pointer.
- pllControl – PLL control name (see clock_pll_ctrl_t enumeration)

static inline void CLOCK_PowerDownPll(CCM_ANALOG_Type *base, *clock_pll_ctrl_t* pllControl)

Power down PLL.

Parameters

- base – CCM_ANALOG base pointer.
- pllControl – PLL control name (see clock_pll_ctrl_t enumeration)

static inline void CLOCK_SetPllBypass(CCM_ANALOG_Type *base, *clock_pll_bypass_ctrl_t* pllControl, bool bypass)

PLL bypass setting.

Parameters

- base – CCM_ANALOG base pointer.
- pllControl – PLL control name (see clock_pll_bypass_ctrl_t enumeration)
- bypass – Bypass the PLL.
  - true: Bypass the PLL.
  - false: Do not bypass the PLL.

static inline bool CLOCK_IsPllBypassed(CCM_ANALOG_Type *base, *clock_pll_bypass_ctrl_t* pllControl)

Check if PLL is bypassed.

Parameters

- base – CCM_ANALOG base pointer.
- pllControl – PLL control name (see clock_pll_bypass_ctrl_t enumeration)

Returns

PLL bypass status.

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

static inline bool CLOCK_IsPllLocked(CCM_ANALOG_Type *base, *clock_pll_ctrl_t* pllControl)

Check if PLL clock is locked.

Parameters

- base – CCM_ANALOG base pointer.
- pllControl – PLL control name (see clock_pll_ctrl_t enumeration)

Returns

PLL lock status.

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

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

Enable PLL clock.

Parameters

- base – CCM_ANALOG base pointer.
- pllClock – PLL clock name (see clock_pll_clke_t enumeration)

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

Disable PLL clock.

Parameters

- base – CCM_ANALOG base pointer.
- pllClock – PLL clock name (see clock_pll_clke_t enumeration)

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

Override PLL clock output enable.

Parameters

- base – CCM_ANALOG base pointer.
- ovClock – PLL clock name (see clock_pll_clke_t enumeration)
- override – Override the PLL.
  - true: Override the PLL clke, CCM will handle it.
  - false: Do not override the PLL clke.

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

Override PLL power down.

Parameters

- base – CCM_ANALOG base pointer.
- pdClock – PLL clock name (see clock_pll_ctrl_t enumeration)
- override – Override the PLL.
  - true: Override the PLL clke, CCM will handle it.
  - false: Do not override the PLL clke.

void CLOCK_InitArmPll(const *ccm_analog_integer_pll_config_t* *config)

Initializes the ANALOG ARM PLL.

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

Parameters

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

void CLOCK_DeinitArmPll(void)

De-initialize the ARM PLL.

void CLOCK_InitSysPll1(const *ccm_analog_integer_pll_config_t* *config)

Initializes the ANALOG SYS PLL1.

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

Parameters

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

void CLOCK_DeinitSysPll1(void)

De-initialize the System PLL1.

void CLOCK_InitSysPll2(const *ccm_analog_integer_pll_config_t* *config)

Initializes the ANALOG SYS PLL2.

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

Parameters

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

void CLOCK_DeinitSysPll2(void)

De-initialize the System PLL2.

void CLOCK_InitSysPll3(const *ccm_analog_integer_pll_config_t* *config)

Initializes the ANALOG SYS PLL3.

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

Parameters

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

void CLOCK_DeinitSysPll3(void)

De-initialize the System PLL3.

void CLOCK_InitAudioPll1(const *ccm_analog_frac_pll_config_t* *config)

Initializes the ANALOG AUDIO PLL1.

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

Parameters

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

void CLOCK_DeinitAudioPll1(void)

De-initialize the Audio PLL1.

void CLOCK_InitAudioPll2(const *ccm_analog_frac_pll_config_t* *config)

Initializes the ANALOG AUDIO PLL2.

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

Parameters

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

void CLOCK_DeinitAudioPll2(void)

De-initialize the Audio PLL2.

void CLOCK_InitVideoPll1(const *ccm_analog_frac_pll_config_t* *config)

Initializes the ANALOG VIDEO PLL1.

Parameters

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

void CLOCK_DeinitVideoPll1(void)

De-initialize the Video PLL1.

void CLOCK_InitIntegerPll(CCM_ANALOG_Type *base, const *ccm_analog_integer_pll_config_t* *config, *clock_pll_ctrl_t* type)

Initializes the ANALOG Integer PLL.

Parameters

- base – CCM ANALOG base address
- config – Pointer to the configuration structure(see ccm_analog_integer_pll_config_t enumeration).
- type – integer pll type

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

Get the ANALOG Integer PLL clock frequency.

Parameters

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

Returns

Clock frequency

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

Initializes the ANALOG Fractional PLL.

Parameters

- base – CCM ANALOG base address.
- config – Pointer to the configuration structure(see ccm_analog_frac_pll_config_t enumeration).
- type – fractional pll type.

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

Gets the ANALOG Fractional PLL clock frequency.

Parameters

- base – CCM_ANALOG base pointer.
- type – fractional pll type.

- refClkFreq – Reference clock frequency.

Returns
Clock frequency

uint32_t CLOCK_GetPllFreq(*clock_pll_ctrl_t* pll)

Gets PLL clock frequency.

Parameters

- pll – fractional pll type.

Returns
Clock frequency

uint32_t CLOCK_GetPllRefClkFreq(*clock_pll_ctrl_t* ctrl)

Gets PLL reference clock frequency.

Parameters

- ctrl – fractional pll type.

Returns
Clock frequency

uint32_t CLOCK_GetFreq(*clock_name_t* clockName)

Gets the clock frequency for a specific clock name.

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

Parameters

- clockName – Clock names defined in clock_name_t

Returns
Clock frequency value in hertz

uint32_t CLOCK_GetClockRootFreq(*clock_root_t* clockRoot)

Gets the frequency of selected clock root.

Parameters

- clockRoot – The clock root used to get the frequency, please refer to clock_root_t.

Returns
The frequency of selected clock root.

uint32_t CLOCK_GetCoreM7Freq(void)

Get the CCM Cortex M7 core frequency.

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

uint32_t CLOCK_GetAxiFreq(void)

Get the CCM Axi bus frequency.

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

uint32_t CLOCK_GetAhbFreq(void)

Get the CCM Ahb bus frequency.

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

uint32_t CLOCK_GetEnetAxiFreq(void)

brief Get the CCM Enet AXI bus frequency.

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

uint8_t refSel

pll reference clock sel

uint32_t mainDiv

Value of the 10-bit programmable main-divider, range must be 64~1023

uint32_t dsm

Value of 16-bit DSM

uint8_t preDiv

Value of the 6-bit programmable pre-divider, range must be 1~63

uint8_t postDiv

Value of the 3-bit programmable Scaler, range must be 0~6

uint8_t refSel

pll reference clock sel

uint32_t mainDiv

Value of the 10-bit programmable main-divider, range must be 64~1023

uint8_t preDiv

Value of the 6-bit programmable pre-divider, range must be 1~63

uint8_t postDiv

Value of the 3-bit programmable Scaler, range must be 0~6

struct __ccm_analog_frac_pll_config

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

struct __ccm_analog_integer_pll_config

#include <fsl_clock.h> Integer PLL configuration. Note: all the dividers in this configuration structure are the actually divider, software will map it to register value.

2.8 EDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

void EDMA_Init(DMA_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.

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(DMA_Type *base)

Deinitializes the eDMA peripheral.

This function gates the eDMA clock.

Parameters

- base – eDMA peripheral base address.

void EDMA_InstallTCD(DMA_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.enableMasterIdReplication = true;
config.enableHaltOnError = true;
config.enableRoundRobinArbitration = false;
config.enableDebugMode = false;
config.enableBufferedWrites = false;
```

Parameters

- config – A pointer to the eDMA configuration structure.

static inline void EDMA_EnableAllChannelLink(DMA_Type *base, bool enable)

Enables/disables all channel linking.

This function enables/disables all channel linking in the management page. For specific channel linking enablement & configuration, please refer to EDMA_SetChannelLink and EDMA_TcdSetChannelLink APIs.

For example, to disable all channel linking in the DMA0 management page:

```
EDMA_EnableAllChannelLink(DMA0, false);
```

Parameters

- base – eDMA peripheral base address.
- enable – Switcher of the channel linking feature for all channels. "true" means to enable. "false" means not.

void EDMA_ResetChannel(DMA_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(DMA_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_config_t config;
edma_tcd_t tcd;
config.srcAddr = ..;
config.destAddr = ..;
...
EDMA_SetTransferConfig(DMA0, channel, &config, &stcd);
```

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(DMA_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.

static inline void EDMA_SetChannelArbitrationGroup(DMA_Type *base, uint32_t channel, uint32_t group)

Configures the eDMA channel arbitration group.

This function configures the channel arbitration group. The arbitration group priorities are evaluated by numeric value from highest group number to lowest.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number
- group – Fixed-priority arbitration group number for the channel.

static inline void EDMA_SetChannelPreemptionConfig(DMA_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.

static inline uint32_t EDMA_GetChannelSystemBusInformation(DMA_Type *base, uint32_t channel)

Gets the eDMA 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.

void EDMA_SetChannelLink(DMA_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(DMA_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(DMA_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_EnableAsyncRequest(DMA_Type *base, uint32_t channel, bool enable)

Enables an async request for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- enable – The command to enable (true) or disable (false).

static inline void EDMA_EnableAutoStopRequest(DMA_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(DMA_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(DMA_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_TcdReset(*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

- 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.

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The STCD 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_config_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.

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.

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.

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.

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.

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.

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.

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.

static inline void EDMA_EnableChannelRequest(DMA_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(DMA_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(DMA_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(DMA_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(DMA_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(DMA_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(DMA_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.

void EDMA_CreateHandle(*edma_handle_t* *handle, DMA_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.

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.

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.

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).

Parameters

- config – The user configuration structure of type edma_transfer_config_t.
- srcAddr – eDMA transfer source address.
- srcWidth – eDMA transfer source address width(bytes).
- srcOffset – eDMA transfer source address offset
- destAddr – eDMA transfer destination address.
- destWidth – eDMA transfer destination address width(bytes).
- destOffset – eDMA transfer 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* transferType)

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_config_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.
- transferType – eDMA transfer type.

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. If submitting the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time.

Parameters

- handle – 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.

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.

static inline *edma_transfer_size_t* EDMA_GetTransferSize(uint32_t width)

Get the transfer size.

This function gets the transfer size.

Parameters

- width – transfer width(bytes).

Returns

The transfer size.

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.

Parameters

- handle – eDMA handle pointer.

FSL_EDMA_DRIVER_VERSION

eDMA driver version

Version 2.5.0.

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

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 channel error status flags, _edma_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_TransferCanceledFlag
Transfer cancelled

enumerator kEDMA_ErrorChannelFlag
Error channel number of the cancelled channel number

enumerator kEDMA_ValidFlag
No error occurred, this bit is 0. Otherwise, it is 1.

eDMA channel system bus information, _edma_channel_sys_bus_info

Values:

enumerator kEDMA_AttributeOutput
DMA's AHB system bus attribute output value.

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_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

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.

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_interrupt_enable* edma_interrupt_enable_t
eDMA interrupt source

typedef enum *_edma_transfer_type* edma_transfer_type_t
eDMA transfer type

typedef struct *_edma_config* edma_config_t
eDMA global configuration structure.

typedef struct *_edma_transfer_config* edma_transfer_config_t
eDMA transfer configuration

This structure configures the source/destination transfer attribute.

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 struct *_edma_tcd* 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 void (*edma_callback)(struct *_edma_handle* *handle, void *userData, bool transferDone, uint32_t tcds)
Define callback function for eDMA.

typedef uint32_t (*edma_memorymap_callback)(uint32_t addr)
Memroy map function callback for DMA.

typedef struct *_edma_handle* edma_handle_t
eDMA transfer handle structure

struct _edma_config
#include <fsl_edma.h> eDMA global configuration structure.

Public Members

bool enableMasterIdReplication
Enable (true) master ID replication. If Master ID replication is disabled, the privileged protection level (supervisor mode) for DMA transfers is used.

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 enableRoundRobinArbitration

Enable (true) round robin channel arbitration method or fixed priority arbitration is used for channel selection

bool enableDebugMode

Enable(true) eDMA debug mode. When in debug mode, the eDMA stalls the start of a new channel. Executing channels are allowed to complete.

bool enableBufferedWrites

Enable(true) buffered writes. When buffered writes are enabled, all writes except for the last write sequence of the minor loop are signaled by the eDMA as bufferable.

struct __edma__transfer__config

#include <fsl_edma.h> eDMA transfer configuration

This structure configures the source/destination transfer attribute.

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 applied to the current source address to form the next-state value as each source read is completed.

int16_t destOffset

Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.

uint32_t minorLoopBytes

Bytes to transfer in a minor loop

uint32_t majorLoopCounts

Major loop iteration count.

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_tcd
#include <fsl_edma.h> 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.

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 stcd 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_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.

DMA_Type *base
eDMA peripheral base address.

edma_tcd_t *tcdPool
Pointer to memory stored TCDs.

uint8_t channel
eDMA channel number.

volatile int8_t header
The first TCD index.

volatile int8_t tail
The last TCD index.

volatile int8_t tcdUsed
The number of used TCD slots.

volatile int8_t tcdSize
The total number of TCD slots in the queue.

uint8_t flags
The status of the current channel.

2.9 ECSPI: Enhanced Configurable Serial Peripheral Interface Driver

2.10 ECSPI Driver

void ECSPI_MasterGetDefaultConfig(*ecspi_master_config_t* *config)
Sets the ECSPI configuration structure to default values.

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

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

Parameters

- config – pointer to config structure

void ECSPI_MasterInit(ECSPI_Type *base, const *ecspi_master_config_t* *config, uint32_t srcClock_Hz)
Initializes the ECSPI with configuration.

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

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

Parameters

- base – ECSPI base pointer
- config – pointer to master configuration structure
- srcClock_Hz – Source clock frequency.

void ECSPI_SlaveGetDefaultConfig(*ecspi_slave_config_t* *config)

Sets the ECSPI configuration structure to default values.

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

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

Parameters

- config – pointer to config structure

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

Initializes the ECSPI with configuration.

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

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

Parameters

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

void ECSPI_Deinit(ECSPI_Type *base)

De-initializes the ECSPI.

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

Parameters

- base – ECSPI base pointer

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

Enables or disables the ECSPI.

Parameters

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

static inline uint32_t ECSPI_GetStatusFlags(ECSPI_Type *base)

Gets the status flag.

Parameters

- base – ECSPI base pointer

Returns

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

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

Clear the status flag.

Parameters

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

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

Enables the interrupt for the ECSPI.

Parameters

- base – ECSPI base pointer
- mask – ECSPI interrupt source. The parameter can be any combination of the following values:
  - kECSPI_TxfifoEmptyInterruptEnable
  - kECSPI_TxFifoDataRequstInterruptEnable
  - kECSPI_TxFifoFullInterruptEnable
  - kECSPI_RxFifoReadyInterruptEnable
  - kECSPI_RxFifoDataRequstInterruptEnable
  - kECSPI_RxFifoFullInterruptEnable
  - kECSPI_RxFifoOverFlowInterruptEnable
  - kECSPI_TransferCompleteInterruptEnable
  - kECSPI_AllInterruptEnable

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

Disables the interrupt for the ECSPI.

Parameters

- base – ECSPI base pointer
- mask – ECSPI interrupt source. The parameter can be any combination of the following values:
  - kECSPI_TxfifoEmptyInterruptEnable
  - kECSPI_TxFifoDataRequstInterruptEnable
  - kECSPI_TxFifoFullInterruptEnable
  - kECSPI_RxFifoReadyInterruptEnable
  - kECSPI_RxFifoDataRequstInterruptEnable
  - kECSPI_RxFifoFullInterruptEnable
  - kECSPI_RxFifoOverFlowInterruptEnable
  - kECSPI_TransferCompleteInterruptEnable

– kECSPI_AllInterruptEnable

static inline void ECSPI_SoftwareReset(ECSPI_Type *base)

Software reset.

Parameters

- base – ECSPI base pointer

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

Mode check.

Parameters

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

Returns

mode of channel

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

Enables the DMA source for ECSPI.

Parameters

- base – ECSPI base pointer
- mask – ECSPI DMA source. The parameter can be any of the following values:
  - kECSPI_TxDmaEnable
  - kECSPI_RxDmaEnable
  - kECSPI_DmaAllEnable
- enable – True means enable DMA, false means disable DMA

static inline uint8_t ECSPI_GetTxFifoCount(ECSPI_Type *base)

Get the Tx FIFO data count.

Parameters

- base – ECSPI base pointer.

Returns

the number of words in Tx FIFO buffer.

static inline uint8_t ECSPI_GetRxFifoCount(ECSPI_Type *base)

Get the Rx FIFO data count.

Parameters

- base – ECSPI base pointer.

Returns

the number of words in Rx FIFO buffer.

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

Set channel select for transfer.

Parameters

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

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

Set channel select configuration for transfer.

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

Parameters

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

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

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

Parameters

- base – ECSPI base pointer
- baudRate_Bps – baud rate needed in Hz.
- srcClock_Hz – ECSPI source clock frequency in Hz.

status_t ECSPI_WriteBlocking(ECSPI_Type *base, const uint32_t *buffer, size_t size)

Sends a buffer of data bytes using a blocking method.

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

Parameters

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

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_ECSPI_Timeout – The transfer timed out and was aborted.

static inline void ECSPI_WriteData(ECSPI_Type *base, uint32_t data)

Writes a data into the ECSPI data register.

Parameters

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

static inline uint32_t ECSPI_ReadData(ECSPI_Type *base)

Gets a data from the ECSPI data register.

Parameters

- base – ECSPI base pointer

Returns

Data in the register.

void ECSPI_MasterTransferCreateHandle(ECSPI_Type *base, *ecspi_master_handle_t* *handle, *ecspi_master_callback_t* callback, void *userData)

Initializes the ECSPI master handle.

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

Parameters

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

status_t ECSPI_MasterTransferBlocking(ECSPI_Type *base, *ecspi_transfer_t* *xfer)

Transfers a block of data using a polling method.

Parameters

- base – SPI base pointer
- xfer – pointer to spi_xfer_config_t structure

Return values

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

status_t ECSPI_MasterTransferNonBlocking(ECSPI_Type *base, *ecspi_master_handle_t* *handle, *ecspi_transfer_t* *xfer)

Performs a non-blocking ECSPI interrupt transfer.

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

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

Parameters

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

Return values

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

status_t ECSPI_MasterTransferGetCount(ECSPI_Type *base, *ecspi_master_handle_t* *handle, size_t *count)

Gets the bytes of the ECSPI interrupt transferred.

Parameters

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

Return values

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

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

Aborts an ECSPI transfer using interrupt.

Parameters

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

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

Interrupts the handler for the ECSPI.

Parameters

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

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

Initializes the ECSPI slave handle.

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

Parameters

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

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

Performs a non-blocking ECSPI slave interrupt transfer.

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

Parameters

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

Return values

- kStatus_Success – Successfully start a transfer.

- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_ECSPI_Busy – ECSPI is not idle, is running another transfer.

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

Gets the bytes of the ECSPI interrupt transferred.

Parameters

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

Return values

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

static inline void ECSPI_SlaveTransferAbort(ECSPI_Type *base, *ecspi_slave_handle_t* *handle)

Aborts an ECSPI slave transfer using interrupt.

Parameters

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

void ECSPI_SlaveTransferHandleIRQ(ECSPI_Type *base, *ecspi_slave_handle_t* *handle)

Interrupts a handler for the ECSPI slave.

Parameters

- base – ECSPI peripheral base address.
- handle – pointer to ecspi_slave_handle_t structure which stores the transfer state

FSL_ECSPI_DRIVER_VERSION

ECSPI driver version.

Return status for the ECSPI driver.

Values:

enumerator kStatus_ECSPI_Busy

ECSPI bus is busy

enumerator kStatus_ECSPI_Idle

ECSPI is idle

enumerator kStatus_ECSPI_Error

ECSPI error

enumerator kStatus_ECSPI_HardwareOverFlow

ECSPI hardware overflow

enumerator kStatus_ECSPI_Timeout

ECSPI timeout polling status flags.

enum _ecspi_clock_polarity

ECSPI clock polarity configuration.

Values:

enumerator kECSPI_PolarityActiveHigh
Active-high ECSPI polarity high (idles low).

enumerator kECSPI_PolarityActiveLow
Active-low ECSPI polarity low (idles high).

enum _ecspi_clock_phase
ECSPI clock phase configuration.

Values:

enumerator kECSPI_ClockPhaseFirstEdge
First edge on SPSCK occurs at the middle of the first cycle of a data transfer.

enumerator kECSPI_ClockPhaseSecondEdge
First edge on SPSCK occurs at the start of the first cycle of a data transfer.

ECSPI interrupt sources.

Values:

enumerator kECSPI_TxfifoEmptyInterruptEnable
Transmit FIFO buffer empty interrupt

enumerator kECSPI_TxFifoDataRequstInterruptEnable
Transmit FIFO data requst interrupt

enumerator kECSPI_TxFifoFullInterruptEnable
Transmit FIFO full interrupt

enumerator kECSPI_RxFifoReadyInterruptEnable
Receiver FIFO ready interrupt

enumerator kECSPI_RxFifoDataRequstInterruptEnable
Receiver FIFO data requst interrupt

enumerator kECSPI_RxFifoFullInterruptEnable
Receiver FIFO full interrupt

enumerator kECSPI_RxFifoOverFlowInterruptEnable
Receiver FIFO buffer overflow interrupt

enumerator kECSPI_TransferCompleteInterruptEnable
Transfer complete interrupt

enumerator kECSPI_AllInterruptEnable
All interrupt

ECSPI status flags.

Values:

enumerator kECSPI_TxfifoEmptyFlag
Transmit FIFO buffer empty flag

enumerator kECSPI_TxFifoDataRequstFlag
Transmit FIFO data requst flag

enumerator kECSPI_TxFifoFullFlag
Transmit FIFO full flag

enumerator kECSPI_RxFifoReadyFlag
Receiver FIFO ready flag

enumerator kECSPI_RxFifoDataRequstFlag
Receiver FIFO data requst flag

enumerator kECSPI_RxFifoFullFlag
Receiver FIFO full flag

enumerator kECSPI_RxFifoOverFlowFlag
Receiver FIFO buffer overflow flag

enumerator kECSPI_TransferCompleteFlag
Transfer complete flag

ECSPI DMA enable.

Values:

enumerator kECSPI_TxDmaEnable
Tx DMA request source

enumerator kECSPI_RxDmaEnable
Rx DMA request source

enumerator kECSPI_DmaAllEnable
All DMA request source

enum _ecspi_data_ready
ECSPI SPI_RDY signal configuration.

Values:

enumerator kECSPI_DataReadyIgnore
SPI_RDY signal is ignored

enumerator kECSPI_DataReadyFallingEdge
SPI_RDY signal will be triggerd by the falling edge

enumerator kECSPI_DataReadyLowLevel
SPI_RDY signal will be triggerd by a low level

enum _ecspi_channel_source
ECSPI channel select source.

Values:

enumerator kECSPI_Channel0
Channel 0 is selectd

enumerator kECSPI_Channel1
Channel 1 is selectd

enumerator kECSPI_Channel2
Channel 2 is selectd

enumerator kECSPI_Channel3
Channel 3 is selectd

enum _ecspi_master_slave_mode
ECSPI master or slave mode configuration.

Values:

enumerator kECSPI_Slave
ECSPI peripheral operates in slave mode.

enumerator kECSPI_Master
ECSPI peripheral operates in master mode.

enum _ecspi_data_line_inactive_state_t
ECSPI data line inactive state configuration.

Values:

enumerator kECSPI_DataLineInactiveStateHigh
The data line inactive state stays high.

enumerator kECSPI_DataLineInactiveStateLow
The data line inactive state stays low.

enum _ecspi_clock_inactive_state_t
ECSPI clock inactive state configuration.

Values:

enumerator kECSPI_ClockInactiveStateLow
The SCLK inactive state stays low.

enumerator kECSPI_ClockInactiveStateHigh
The SCLK inactive state stays high.

enum _ecspi_chip_select_active_state_t
ECSPI active state configuration.

Values:

enumerator kECSPI_ChipSelectActiveStateLow
The SS signal line active stays low.

enumerator kECSPI_ChipSelectActiveStateHigh
The SS signal line active stays high.

enum _ecspi_sample_period_clock_source
ECSPI sample period clock configuration.

Values:

enumerator kECSPI_spiClock
The sample period clock source is SCLK.

enumerator kECSPI_lowFreqClock
The sample seriod clock source is low_frequency reference clock(32.768 kHz).

typedef enum *_ecspi_clock_polarity* ecspi_clock_polarity_t
ECSPI clock polarity configuration.

typedef enum *_ecspi_clock_phase* ecspi_clock_phase_t
ECSPI clock phase configuration.

typedef enum *_ecspi_data_ready* ecspi_Data_ready_t
ECSPI SPI_RDY signal configuration.

typedef enum *_ecspi_channel_source* ecspi_channel_source_t
ECSPI channel select source.

typedef enum *_ecspi_master_slave_mode* ecspi_master_slave_mode_t
ECSPI master or slave mode configuration.

typedef enum *_ecspi_data_line_inactive_state_t* ecspi_data_line_inactive_state_t
ECSPI data line inactive state configuration.

typedef enum *_ecspi_clock_inactive_state_t* ecspi_clock_inactive_state_t
ECSPI clock inactive state configuration.

typedef enum *_ecspi_chip_select_active_state_t* ecspi_chip_select_active_state_t
ECSPI active state configuration.

typedef enum *_ecspi_sample_period_clock_source* ecspi_sample_period_clock_source_t
ECSPI sample period clock configuration.

typedef struct *_ecspi_channel_config* ecspi_channel_config_t
ECSPI user channel configure structure.

typedef struct *_ecspi_master_config* ecspi_master_config_t
ECSPI master configure structure.

typedef struct *_ecspi_slave_config* ecspi_slave_config_t
ECSPI slave configure structure.

typedef struct *_ecspi_transfer* ecspi_transfer_t
ECSPI transfer structure.

typedef struct *_ecspi_master_handle* ecspi_master_handle_t

typedef *ecspi_master_handle_t* ecspi_slave_handle_t
Slave handle is the same with master handle

typedef void (*ecspi_master_callback_t)(ECSPI_Type *base, *ecspi_master_handle_t* *handle, *status_t* status, void *userData)
ECSPI master callback for finished transmit.

typedef void (*ecspi_slave_callback_t)(ECSPI_Type *base, *ecspi_slave_handle_t* *handle, *status_t* status, void *userData)
ECSPI slave callback for finished transmit.

uint32_t ECSPI_GetInstance(ECSPI_Type *base)
Get the instance for ECSPI module.

Parameters

- base – ECSPI base address

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

SPI_RETRY_TIMES
Retry times for waiting flag.

struct _ecspi_channel_config
#include <fsl_ecspi.h> ECSPI user channel configure structure.

Public Members

ecspi_master_slave_mode_t channelMode
Channel mode

ecspi_clock_inactive_state_t clockInactiveState
Clock line (SCLK) inactive state

ecspi_data_line_inactive_state_t dataLineInactiveState
Data line (MOSI&MISO) inactive state

ecspi_chip_select_active_state_t chipSlectActiveState
Chip select(SS) line active state

ecspi_clock_polarity_t polarity
Clock polarity

ecspi_clock_phase_t phase
Clock phase

struct _ecspi_master_config
#include <fsl_ecspi.h> ECSPI master configure structure.

Public Members

ecspi_channel_source_t channel
Channel number

ecspi_channel_config_t channelConfig
Channel configuration

ecspi_sample_period_clock_source_t samplePeriodClock
Sample period clock source

uint16_t burstLength
Burst length. The length shall be less than 4096 bits

uint8_t chipSelectDelay
SS delay time

uint16_t samplePeriod
Sample period

uint8_t txFifoThreshold
TX Threshold

uint8_t rxFifoThreshold
RX Threshold

uint32_t baudRate_Bps
ECSPI baud rate for master mode

bool enableLoopback
Enable the ECSPI loopback test.

struct _ecspi_slave_config
#include <fsl_ecspi.h> ECSPI slave configure structure.

Public Members

uint16_t burstLength
Burst length. The length shall be less than 4096 bits

uint8_t txFifoThreshold
TX Threshold

uint8_t rxFifoThreshold
RX Threshold

ecspi_channel_config_t channelConfig
Channel configuration

struct __ecspi__transfer
#include <fsl_ecspi.h> ECSPI transfer structure.

Public Members

const uint32_t *txData
Send buffer

uint32_t *rxData
Receive buffer

size_t dataSize
Transfer bytes

ecspi_channel_source_t channel
ECSPI channel select

struct __ecspi__master__handle
#include <fsl_ecspi.h> ECSPI master handle structure.

Public Members

ecspi_channel_source_t channel
Channel number

const uint32_t *volatile txData
Transfer buffer

uint32_t *volatile rxData
Receive buffer

volatile size_t txRemainingBytes
Send data remaining in bytes

volatile size_t rxRemainingBytes
Receive data remaining in bytes

volatile uint32_t state
ECSPI internal state

size_t transferSize
Bytes to be transferred

ecspi_master_callback_t callback
ECSPI callback

void *userData
Callback parameter

2.11 ECSPI SDMA Driver

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

Initialize the ECSPI master SDMA handle.

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

Parameters

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

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

Initialize the ECSPI Slave SDMA handle.

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

Parameters

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

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

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

Perform a non-blocking ECSPI master transfer using SDMA.

Note: This interface returned immediately after transfer initiates.

Parameters

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

Return values

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

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

Perform a non-blocking ECSPI slave transfer using SDMA.

Note: This interface returned immediately after transfer initiates.

Parameters

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

Return values

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

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

Abort a ECSPI master transfer using SDMA.

Parameters

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

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

Abort a ECSPI slave transfer using SDMA.

Parameters

- base – ECSPI peripheral base address.

- handle – ECSPI SDMA handle pointer.

FSL_ECSPI_FREERTOS_DRIVER_VERSION
ECSPI FreeRTOS driver version.

typedef struct *_ecspi_sdma_handle* ecspi_sdma_handle_t

typedef void (*ecspi_sdma_callback_t)(ECSPI_Type *base, *ecspi_sdma_handle_t* *handle, *status_t* status, void *userData)
ECSPI SDMA callback called at the end of transfer.

struct _ecspi_sdma_handle
#include <fsl_ecspi_sdma.h> ECSPI SDMA transfer handle, users should not touch the content of the handle.

Public Members

bool txInProgress
Send transfer finished

bool rxInProgress
Receive transfer finished

sdma_handle_t *txSdmaHandle
DMA handler for ECSPI send

sdma_handle_t *rxSdmaHandle
DMA handler for ECSPI receive

ecspi_sdma_callback_t callback
Callback for ECSPI SDMA transfer

void *userData
User Data for ECSPI SDMA callback

uint32_t state
Internal state of ECSPI SDMA transfer

uint32_t ChannelTx
Channel for send handle

uint32_t ChannelRx
Channel for receive handler

2.12 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 dulpex for MAC.

Parameters

- base – ENET peripheral base address.
- speed – The speed of the RMII mode.
- duplex – The duplex of the RMII mode.

void ENET_SetSMI(ENET_Type *base, 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.

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_GetRxFrameSize(*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_GetRxFrameSize, 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_GetRxFrameSize 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_GetRxFrameSize(&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);
```

(continues on next page)

(continued from previous page)

```
        Comments: Call stack input API to deliver the data to stack
    }
}
else if (status == kStatus_ENET_RxFrameError)
{
    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 isEnable, 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.
- isEnable – 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 interupt 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 finishs.
- 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/mutli-ring (frame 0).

Parameters

- base – ENET peripheral base address.

FSL_ENET_DRIVER_VERSION
Defines the driver version.

FSL_FEATURE_ENET_QUEUE
Defines the queue number.

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_LENVLIOLATE_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_SOFTOWENER1_MASK
Software owner one mask.

ENET_BUFFDESCRIPTOR_TX_WRAP_MASK
Wrap buffer descriptor mask.

ENET_BUFFDESCRIPTOR_TX_SOFTOWENER2_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.

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.

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_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 macSpecialConfig 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 rxFifoFullThreshold 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).

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_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_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 macSpecialConfig 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 rxFifoFullThreshold 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_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_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 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 decrese 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.

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 txConsumIdx
: 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.

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.

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__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_FULL 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 decrese 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_rx_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 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.

uint8_t ringNum
: Number of used rings. default with 1 — 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 txConsumIdx
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

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.

2.13 EQOS-TSN: Ethernet QoS with TSN Driver

2.14 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_StartSMIRead(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_StartExtC45SMIRead(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_MDIORead(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_MDIOC45Write(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_MDIOC45Read(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 fist 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 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 the rx descriptor tail. This is called after you setup a new rx descriptor to update the tail pointer to make the new descriptor accessible by DMA and to anouse the rx poll command for DMA.

Parameters

- base – ENET peripheral base address.
- channel – The rx DMA channel.
- rxDescTailAddrAlign – The new rx tail pointer address.

static inline uint32_t ENET_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 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_GetRxFrameSize(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_GetRxFrameSize, ENET_QOS_ReadFrame() should be called to update the receive buffers If the result is not "kStatus_ENET_QOS_RxFrameEmpty".

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_RxFrameEmpty – 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);
```

(continues on next page)

(continued from previous page)

```
    }
    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 OQS 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 OQS 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 OQS 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 offest 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
Maxmimum 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, these is no need to set this 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_MacPmt

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_PacketDCBCPQ

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_PtpPpsIstance0
PPS instance 0.

enumerator kENET_QOS_PtpPpsIstance1
PPS instance 1.

enumerator kENET_QOS_PtpPpsIstance2
PPS instance 2.

enumerator kENET_QOS_PtpPpsIstance3
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 enmueration 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 enmueration 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 enmueration 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, these is no need to set this 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-bot 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-bot 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_rxqueuemap* enet_qos_mtl_rxqueuemap_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 enmueration of ETS list length.

typedef enum *_enet_qos_ets_gccr_addr* enet_qos_ets_gccr_addr_t

Defines the enmueration of ETS gate control address.

typedef enum *_enet_qos_rxp_dma_chn* enet_qos_rxp_dma_chn_t

Defines the enmueration 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

TDES 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 / (refClk_Hz / ENET_QOS_MICRSECS_ONESECOND / 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:

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.

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[(5U)]
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[(5U)]
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[(5U)]
: Rx buffer cache maintain enabled.

enet_qos_rx_bd_ring_t rxBdRing[(5U)]
: Receive buffer descriptor.

enet_qos_tx_bd_ring_t txBdRing[(5U)]
: Transmit buffer descriptor.

enet_qos_tx_dirty_ring_t txDirtyRing[(5U)]
: Transmit dirty buffers addresses.

uint32_t *rxBufferStartAddr[(5U)]
: 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.

2.15 FlexCAN: Flex Controller Area Network Driver

2.16 FlexCAN Driver

void FLEXCAN_EnterFreezeMode(CAN_Type *base)
Enter FlexCAN Freeze Mode.

This function makes the FlexCAN work under Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

void FLEXCAN_ExitFreezeMode(CAN_Type *base)
Exit FlexCAN Freeze Mode.

This function makes the FlexCAN leave Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

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, 40000000UL);
```

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the user-defined configuration structure.
- sourceClock_Hz – FlexCAN Protocol Engine clock source frequency in Hz.

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_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.

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.

static inline uint32_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, uint32_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 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_EnableInterrupts(CAN_Type *base, uint32_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, uint32_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_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_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.

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_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_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.

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.

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.

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.

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.

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_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_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.

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
UnHadled Interrupt asserted.

enumerator kStatus_FLEXCAN_RxRemote
Rx Remote Message Received in Mail box.

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_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_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_WakeUpInterruptEnable
Self Wake Up interrupt, use bit 26.

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_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_WakeUpIntFlag
Self Wake-Up Interrupt Flag.

enumerator kFLEXCAN_ErrorFlag

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 ditermine which error is generated.

Values:

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.

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_rx_fifo_filter_type* flexcan_rx_fifo_filter_type_t

FlexCAN Rx Fifo Filter type.

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 struct *_flexcan_frame* flexcan_frame_t

FlexCAN message frame structure.

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 struct *_flexcan_rx_fifo_config* flexcan_rx_fifo_config_t

FlexCAN Legacy 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, uint32_t result, void *userData)

FLEXCAN_WAIT_TIMEOUT

FLEXCAN_ID_STD(id)

FlexCAN frame length helper macro.

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.

FLEXCAN_ERROR_AND_STATUS_INT_FLAG
FlexCAN interrupt/status flag helper macro.

FLEXCAN_PNWAKE_UP_FLAG

FLEXCAN_WAKE_UP_FLAG

FLEXCAN_MEMORY_ERROR_INT_FLAG

FLEXCAN_MEMORY_ENHANCED_RX_FIFO_INT_FLAG
FlexCAN Enhanced Rx FIFO base address helper macro.

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_frame

#include <fsl_flexcan.h> FlexCAN message frame structure.

struct _flexcan_timing_config

#include <fsl_flexcan.h> FlexCAN protocol timing characteristic configuration structure.

Public Members

uint16_t preDivider

Classic CAN or CAN FD nominal phase bit rate prescaler.

uint8_t rJumpwidth

Classic CAN or CAN FD nominal phase Re-sync Jump Width.

uint8_t phaseSeg1

Classic CAN or CAN FD nominal phase Segment 1.

uint8_t phaseSeg2

Classic CAN or CAN FD nominal phase Segment 2.

uint8_t propSeg

Classic CAN or CAN FD nominal 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.

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 enableSelfWakeup

Enable or Disable Self Wakeup Mode.

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 enableSupervisorMode

Enable or Disable Supervisor Mode, enable this mode will make registers allow only Supervisor access.

bool enableRemoteRequestFrameStored

true: Store Remote Request Frame in the same fashion of data frame. false: Generate an automatic Remote Response 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).

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_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_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 Ehanced 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_frame_t *volatile rxFifoFrameBuf

The buffer for received CAN data from Legacy Rx FIFO.

size_t rxFifoFrameNum

The number of CAN messages remaining to be received from Legacy or Ehanced Rx FIFO.

size_t rxFifoTransferTotalNum

Total CAN Message number need to be received from Legacy or Ehanced 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 ___unnamed19___

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 ___unnamed21___

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 ___unnamed23___

Public Members

struct __flexcan__frame

struct __flexcan__frame

struct ___unnamed25___

Public Members

uint32_t dataWord0

CAN Frame payload word0.

uint32_t dataWord1

CAN Frame payload word1.

struct ___unnamed27___

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 ___unnamed29___

Public Members

struct _flexcan_config

struct _flexcan_config

struct ___unnamed31___

Public Members

uint32_t baudRate
FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.

struct ___unnamed33___

Public Members

uint32_t bitRate
FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.

2.17 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 confiugration 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.

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.

2.18 FLEXSPI: Flexible Serial Peripheral Interface Driver

uint32_t FLEXSPI_GetInstance(FLEXSPI_Type *base)

Get the instance number for FLEXSPI.

Parameters

- base – FLEXSPI base pointer.

status_t FLEXSPI_CheckAndClearError(FLEXSPI_Type *base, uint32_t status)

Check and clear IP command execution errors.

Parameters

- base – FLEXSPI base pointer.
- status – interrupt status.

void FLEXSPI_Init(FLEXSPI_Type *base, const *flexspi_config_t* *config)

Initializes the FLEXSPI module and internal state.

This function enables the clock for FLEXSPI and also configures the FLEXSPI with the input configure parameters. Users should call this function before any FLEXSPI operations.

Parameters

- base – FLEXSPI peripheral base address.
- config – FLEXSPI configure structure.

void FLEXSPI_GetDefaultConfig(*flexspi_config_t* *config)

Gets default settings for FLEXSPI.

Parameters

- config – FLEXSPI configuration structure.

void FLEXSPI_Deinit(FLEXSPI_Type *base)

Deinitializes the FLEXSPI module.

Clears the FLEXSPI state and FLEXSPI module registers.

Parameters

- base – FLEXSPI peripheral base address.

void FLEXSPI_UpdateDllValue(FLEXSPI_Type *base, *flexspi_device_config_t* *config, *flexspi_port_t* port)

Update FLEXSPI DLL value depending on currently flexspi root clock.

Parameters

- base – FLEXSPI peripheral base address.
- config – Flash configuration parameters.
- port – FLEXSPI Operation port.

void FLEXSPI_SetFlashConfig(FLEXSPI_Type *base, *flexspi_device_config_t* *config, *flexspi_port_t* port)

Configures the connected device parameter.

This function configures the connected device relevant parameters, such as the size, command, and so on. The flash configuration value cannot have a default value. The user needs to configure it according to the connected device.

Parameters

- base – FLEXSPI peripheral base address.
- config – Flash configuration parameters.
- port – FLEXSPI Operation port.

void FLEXSPI_SoftwareReset(FLEXSPI_Type *base)

Software reset for the FLEXSPI logic.

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

Parameters

- base – FLEXSPI peripheral base address.

static inline void FLEXSPI_Enable(FLEXSPI_Type *base, bool enable)

Enables or disables the FLEXSPI module.

Parameters

- base – FLEXSPI peripheral base address.
- enable – True means enable FLEXSPI, false means disable.

static inline void FLEXSPI_EnableInterrupts(FLEXSPI_Type *base, uint32_t mask)

Enables the FLEXSPI interrupts.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

static inline void FLEXSPI_DisableInterrupts(FLEXSPI_Type *base, uint32_t mask)

Disable the FLEXSPI interrupts.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

static inline void FLEXSPI_EnableTxDMA(FLEXSPI_Type *base, bool enable)

Enables or disables FLEXSPI IP Tx FIFO DMA requests.

Parameters

- base – FLEXSPI peripheral base address.
- enable – Enable flag for transmit DMA request. Pass true for enable, false for disable.

static inline void FLEXSPI_EnableRxDMA(FLEXSPI_Type *base, bool enable)

Enables or disables FLEXSPI IP Rx FIFO DMA requests.

Parameters

- base – FLEXSPI peripheral base address.
- enable – Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t FLEXSPI_GetTxFifoAddress(FLEXSPI_Type *base)

Gets FLEXSPI IP tx fifo address for DMA transfer.

Parameters

- base – FLEXSPI peripheral base address.

Return values

The – tx fifo address.

static inline uint32_t FLEXSPI_GetRxFifoAddress(FLEXSPI_Type *base)

Gets FLEXSPI IP rx fifo address for DMA transfer.

Parameters

- base – FLEXSPI peripheral base address.

Return values

The – rx fifo address.

static inline void FLEXSPI_ResetFifos(FLEXSPI_Type *base, bool txFifo, bool rxFifo)

Clears the FLEXSPI IP FIFO logic.

Parameters

- base – FLEXSPI peripheral base address.
- txFifo – Pass true to reset TX FIFO.
- rxFifo – Pass true to reset RX FIFO.

static inline void FLEXSPI_GetFifoCounts(FLEXSPI_Type *base, size_t *txCount, size_t *rxCount)

Gets the valid data entries in the FLEXSPI FIFOs.

Parameters

- base – FLEXSPI peripheral base address.
- txCount – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.

- rxCount – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

static inline uint32_t FLEXSPI_GetInterruptStatusFlags(FLEXSPI_Type *base)

Get the FLEXSPI interrupt status flags.

Parameters

- base – FLEXSPI peripheral base address.

Return values

interrupt – status flag, use status flag to AND flexspi_flags_t could get the related status.

static inline void FLEXSPI_ClearInterruptStatusFlags(FLEXSPI_Type *base, uint32_t mask)

Get the FLEXSPI interrupt status flags.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

static inline void FLEXSPI_GetDataLearningPhase(FLEXSPI_Type *base, uint8_t *portAPhase, uint8_t *portBPhase)

Gets the sampling clock phase selection after Data Learning.

Parameters

- base – FLEXSPI peripheral base address.
- portAPhase – Pointer to a uint8_t type variable to receive the selected clock phase on PORTA.
- portBPhase – Pointer to a uint8_t type variable to receive the selected clock phase on PORTB.

static inline *flexspi_arb_command_source_t* FLEXSPI_GetArbitratorCommandSource(FLEXSPI_Type *base)

Gets the trigger source of current command sequence granted by arbitrator.

Parameters

- base – FLEXSPI peripheral base address.

Return values

trigger – source of current command sequence.

static inline *flexspi_ip_error_code_t* FLEXSPI_GetIPCommandErrorCode(FLEXSPI_Type *base, uint8_t *index)

Gets the error code when IP command error detected.

Parameters

- base – FLEXSPI peripheral base address.
- index – Pointer to a uint8_t type variable to receive the sequence index when error detected.

Return values

error – code when IP command error detected.

static inline *flexspi_ahb_error_code_t* FLEXSPI_GetAHBCommandErrorCode(FLEXSPI_Type *base, uint8_t *index)

Gets the error code when AHB command error detected.

Parameters

- base – FLEXSPI peripheral base address.
- index – Pointer to a uint8_t type variable to receive the sequence index when error detected.

Return values

error – code when AHB command error detected.

static inline bool FLEXSPI_GetBusIdleStatus(FLEXSPI_Type *base)

Returns whether the bus is idle.

Parameters

- base – FLEXSPI peripheral base address.

Return values

- true – Bus is idle.
- false – Bus is busy.

void FLEXSPI_UpdateRxSampleClock(FLEXSPI_Type *base, *flexspi_read_sample_clock_t* clockSource)

Update read sample clock source.

Parameters

- base – FLEXSPI peripheral base address.
- clockSource – clockSource of type flexspi_read_sample_clock_t

static inline void FLEXSPI_EnableIPParallelMode(FLEXSPI_Type *base, bool enable)

Enables/disables the FLEXSPI IP command parallel mode.

Parameters

- base – FLEXSPI peripheral base address.
- enable – True means enable parallel mode, false means disable parallel mode.

static inline void FLEXSPI_EnableAHBParallelMode(FLEXSPI_Type *base, bool enable)

Enables/disables the FLEXSPI AHB command parallel mode.

Parameters

- base – FLEXSPI peripheral base address.
- enable – True means enable parallel mode, false means disable parallel mode.

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 sequencen error detected
- kStatus_FLEXSPI_IpCommandGrantTimeout – IP command grant timeout detected

status_t FLEXSPI_TransferBlocking(FLEXSPI_Type *base, *flexspi_transfer_t* *xfer)

Execute command to transfer a buffer data bytes using a blocking method.

Parameters

- base – FLEXSPI peripheral base address
- xfer – pointer to the transfer structure.

Return values

- kStatus_Success – command transfer success without error
- kStatus_FLEXSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_FLEXSPI_IpCommandSequenceError – IP command sequence error detected
- kStatus_FLEXSPI_IpCommandGrantTimeout – IP command grant timeout detected

void FLEXSPI_TransferCreateHandle(FLEXSPI_Type *base, *flexspi_handle_t* *handle, *flexspi_transfer_callback_t* callback, void *userData)

Initializes the FLEXSPI handle which is used in transactional functions.

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to flexspi_handle_t structure to store the transfer state.
- callback – pointer to user callback function.
- userData – user parameter passed to the callback function.

status_t FLEXSPI_TransferNonBlocking(FLEXSPI_Type *base, *flexspi_handle_t* *handle, *flexspi_transfer_t* *xfer)

Performs a interrupt non-blocking transfer on the FLEXSPI bus.

Note: Calling the API returns immediately after transfer initiates. The user needs to call FLEXSPI_GetTransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_FLEXSPI_Busy, the transfer is finished. For FLEXSPI_Read, the dataSize should be multiple of rx watermark level, or FLEXSPI could not read data properly.

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to flexspi_handle_t structure which stores the transfer state.
- xfer – pointer to flexspi_transfer_t structure.

Return values

- kStatus_Success – Successfully start the data transmission.

- kStatus_FLEXSPI_Busy – Previous transmission still not finished.

status_t FLEXSPI_TransferGetCount(FLEXSPI_Type *base, *flexspi_handle_t* *handle, size_t *count)

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

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to flexspi_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

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

void FLEXSPI_TransferAbort(FLEXSPI_Type *base, *flexspi_handle_t* *handle)

Aborts an interrupt non-blocking transfer early.

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

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to flexspi_handle_t structure which stores the transfer state

void FLEXSPI_TransferHandleIRQ(FLEXSPI_Type *base, *flexspi_handle_t* *handle)

Master interrupt handler.

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to flexspi_handle_t structure.

FSL_FLEXSPI_DRIVER_VERSION

FLEXSPI driver version.

Status structure of FLEXSPI.

Values:

enumerator kStatus_FLEXSPI_Busy

FLEXSPI is busy

enumerator kStatus_FLEXSPI_SequenceExecutionTimeout

Sequence execution timeout error occurred during FLEXSPI transfer.

enumerator kStatus_FLEXSPI_IpCommandSequenceError

IP command Sequence execution timeout error occurred during FLEXSPI transfer.

enumerator kStatus_FLEXSPI_IpCommandGrantTimeout

IP command grant timeout error occurred during FLEXSPI transfer.

CMD definition of FLEXSPI, use to form LUT instruction, _flexspi_command.

Values:

enumerator kFLEXSPI_Command_STOP
Stop execution, deassert CS.

enumerator kFLEXSPI_Command_SDR
Transmit Command code to Flash, using SDR mode.

enumerator kFLEXSPI_Command_RADDR_SDR
Transmit Row Address to Flash, using SDR mode.

enumerator kFLEXSPI_Command_CADDR_SDR
Transmit Column Address to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE1_SDR
Transmit 1-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE2_SDR
Transmit 2-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE4_SDR
Transmit 4-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE8_SDR
Transmit 8-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_WRITE_SDR
Transmit Programming Data to Flash, using SDR mode.

enumerator kFLEXSPI_Command_READ_SDR
Receive Read Data from Flash, using SDR mode.

enumerator kFLEXSPI_Command_LEARN_SDR
Receive Read Data or Preamble bit from Flash, SDR mode.

enumerator kFLEXSPI_Command_DATSZ_SDR
Transmit Read/Program Data size (byte) to Flash, SDR mode.

enumerator kFLEXSPI_Command_DUMMY_SDR
Leave data lines undriven by FlexSPI controller.

enumerator kFLEXSPI_Command_DUMMY_RWDS_SDR
Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

enumerator kFLEXSPI_Command_DDR
Transmit Command code to Flash, using DDR mode.

enumerator kFLEXSPI_Command_RADDR_DDR
Transmit Row Address to Flash, using DDR mode.

enumerator kFLEXSPI_Command_CADDR_DDR
Transmit Column Address to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE1_DDR
Transmit 1-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE2_DDR
Transmit 2-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE4_DDR
Transmit 4-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE8_DDR
Transmit 8-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_WRITE_DDR
Transmit Programming Data to Flash, using DDR mode.

enumerator kFLEXSPI_Command_READ_DDR
Receive Read Data from Flash, using DDR mode.

enumerator kFLEXSPI_Command_LEARN_DDR
Receive Read Data or Preamble bit from Flash, DDR mode.

enumerator kFLEXSPI_Command_DATSZ_DDR
Transmit Read/Program Data size (byte) to Flash, DDR mode.

enumerator kFLEXSPI_Command_DUMMY_DDR
Leave data lines undriven by FlexSPI controller.

enumerator kFLEXSPI_Command_DUMMY_RWDS_DDR
Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

enumerator kFLEXSPI_Command_JUMP_ON_CS
Stop execution, deassert CS and save operand[7:0] as the instruction start pointer for next sequence

enum _flexspi_pad
pad definition of FLEXSPI, use to form LUT instruction.

Values:

enumerator kFLEXSPI_1PAD
Transmit command/address and transmit/receive data only through DATA0/DATA1.

enumerator kFLEXSPI_2PAD
Transmit command/address and transmit/receive data only through DATA[1:0].

enumerator kFLEXSPI_4PAD
Transmit command/address and transmit/receive data only through DATA[3:0].

enumerator kFLEXSPI_8PAD
Transmit command/address and transmit/receive data only through DATA[7:0].

enum _flexspi_flags
FLEXSPI interrupt status flags.

Values:

enumerator kFLEXSPI_SequenceExecutionTimeoutFlag
Sequence execution timeout.

enumerator kFLEXSPI_AhbBusTimeoutFlag
AHB Bus timeout.

enumerator kFLEXSPI_SckStoppedBecauseTxEmptyFlag
SCK is stopped during command sequence because Async TX FIFO empty.

enumerator kFLEXSPI_SckStoppedBecauseRxFullFlag
SCK is stopped during command sequence because Async RX FIFO full.

enumerator kFLEXSPI_DataLearningFailedFlag
Data learning failed.

enumerator kFLEXSPI_IpTxFifoWatermarkEmptyFlag
IP TX FIFO WaterMark empty.

enumerator kFLEXSPI_IpRxFifoWatermarkAvailableFlag
IP RX FIFO WaterMark available.

enumerator kFLEXSPI_AhbCommandSequenceErrorFlag
AHB triggered Command Sequences Error.

enumerator kFLEXSPI_IpCommandSequenceErrorFlag
IP triggered Command Sequences Error.

enumerator kFLEXSPI_AhbCommandGrantTimeoutFlag
AHB triggered Command Sequences Grant Timeout.

enumerator kFLEXSPI_IpCommandGrantTimeoutFlag
IP triggered Command Sequences Grant Timeout.

enumerator kFLEXSPI_IpCommandExecutionDoneFlag
IP triggered Command Sequences Execution finished.

enumerator kFLEXSPI_AllInterruptFlags
All flags.

enum _flexspi_read_sample_clock
FLEXSPI sample clock source selection for Flash Reading.

Values:

enumerator kFLEXSPI_ReadSampleClkLoopbackInternally
Dummy Read strobe generated by FlexSPI Controller and loopback internally.

enumerator kFLEXSPI_ReadSampleClkLoopbackFromDqsPad
Dummy Read strobe generated by FlexSPI Controller and loopback from DQS pad.

enumerator kFLEXSPI_ReadSampleClkLoopbackFromSckPad
SCK output clock and loopback from SCK pad.

enumerator kFLEXSPI_ReadSampleClkExternalInputFromDqsPad
Flash provided Read strobe and input from DQS pad.

enum _flexspi_cs_interval_cycle_unit
FLEXSPI interval unit for flash device select.

Values:

enumerator kFLEXSPI_CsIntervalUnit1SckCycle
Chip selection interval: CSINTERVAL * 1 serial clock cycle.

enumerator kFLEXSPI_CsIntervalUnit256SckCycle
Chip selection interval: CSINTERVAL * 256 serial clock cycle.

enum _flexspi_ahb_write_wait_unit
FLEXSPI AHB wait interval unit for writing.

Values:

enumerator kFLEXSPI_AhbWriteWaitUnit2AhbCycle
AWRWAIT unit is 2 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit8AhbCycle
AWRWAIT unit is 8 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit32AhbCycle
AWRWAIT unit is 32 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit128AhbCycle
AWRWAIT unit is 128 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit512AhbCycle
AWRWAIT unit is 512 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit2048AhbCycle
AWRWAIT unit is 2048 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit8192AhbCycle
AWRWAIT unit is 8192 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit32768AhbCycle
AWRWAIT unit is 32768 ahb clock cycle.

enum _flexspi_ip_error_code
Error Code when IP command Error detected.

Values:

enumerator kFLEXSPI_IpCmdErrorNoError
No error.

enumerator kFLEXSPI_IpCmdErrorJumpOnCsInIpCmd
IP command with JMP_ON_CS instruction used.

enumerator kFLEXSPI_IpCmdErrorUnknownOpCode
Unknown instruction opcode in the sequence.

enumerator kFLEXSPI_IpCmdErrorSdrDummyInDdrSequence
Instruction DUMMY_SDR/DUMMY_RWDS_SDR used in DDR sequence.

enumerator kFLEXSPI_IpCmdErrorDdrDummyInSdrSequence
Instruction DUMMY_DDR/DUMMY_RWDS_DDR used in SDR sequence.

enumerator kFLEXSPI_IpCmdErrorInvalidAddress
Flash access start address exceed the whole flash address range (A1/A2/B1/B2).

enumerator kFLEXSPI_IpCmdErrorSequenceExecutionTimeout
Sequence execution timeout.

enumerator kFLEXSPI_IpCmdErrorFlashBoundaryAcrosss
Flash boundary crossed.

enum _flexspi_ahb_error_code
Error Code when AHB command Error detected.

Values:

enumerator kFLEXSPI_AhbCmdErrorNoError
No error.

enumerator kFLEXSPI_AhbCmdErrorJumpOnCsInWriteCmd
AHB Write command with JMP_ON_CS instruction used in the sequence.

enumerator kFLEXSPI_AhbCmdErrorUnknownOpCode
Unknown instruction opcode in the sequence.

enumerator kFLEXSPI_AhbCmdErrorSdrDummyInDdrSequence
Instruction DUMMY_SDR/DUMMY_RWDS_SDR used in DDR sequence.

enumerator kFLEXSPI_AhbCmdErrorDdrDummyInSdrSequence
Instruction DUMMY_DDR/DUMMY_RWDS_DDR used in SDR sequence.

enumerator kFLEXSPI_AhbCmdSequenceExecutionTimeout
Sequence execution timeout.

enum _flexspi_port
FLEXSPI operation port select.

Values:

enumerator kFLEXSPI_PortA1
Access flash on A1 port.

enumerator kFLEXSPI_PortA2
Access flash on A2 port.

enumerator kFLEXSPI_PortB1
Access flash on B1 port.

enumerator kFLEXSPI_PortB2
Access flash on B2 port.

enumerator kFLEXSPI_PortCount

enum _flexspi_arb_command_source
Trigger source of current command sequence granted by arbitrator.

Values:

enumerator kFLEXSPI_AhbReadCommand

enumerator kFLEXSPI_AhbWriteCommand

enumerator kFLEXSPI_IpCommand

enumerator kFLEXSPI_SuspendedCommand

enum _flexspi_command_type
Command type.

Values:

enumerator kFLEXSPI_Command
FlexSPI operation: Only command, both TX and Rx buffer are ignored.

enumerator kFLEXSPI_Config
FlexSPI operation: Configure device mode, the TX fifo size is fixed in LUT.

enumerator kFLEXSPI_Read

enumerator kFLEXSPI_Write

typedef enum *_flexspi_pad* flexspi_pad_t
pad definition of FLEXSPI, use to form LUT instruction.

typedef enum *_flexspi_flags* flexspi_flags_t
FLEXSPI interrupt status flags.

typedef enum *_flexspi_read_sample_clock* flexspi_read_sample_clock_t
FLEXSPI sample clock source selection for Flash Reading.

typedef enum *_flexspi_cs_interval_cycle_unit* flexspi_cs_interval_cycle_unit_t
FLEXSPI interval unit for flash device select.

typedef enum *_flexspi_ahb_write_wait_unit* flexspi_ahb_write_wait_unit_t
FLEXSPI AHB wait interval unit for writing.

typedef enum *_flexspi_ip_error_code* flexspi_ip_error_code_t
Error Code when IP command Error detected.

typedef enum *_flexspi_ahb_error_code* flexspi_ahb_error_code_t
Error Code when AHB command Error detected.

typedef enum *_flexspi_port* flexspi_port_t
FLEXSPI operation port select.

typedef enum *_flexspi_arb_command_source* flexspi_arb_command_source_t
Trigger source of current command sequence granted by arbitrator.

typedef enum *_flexspi_command_type* flexspi_command_type_t
Command type.

typedef struct *_flexspi_ahbBuffer_config* flexspi_ahbBuffer_config_t

typedef struct *_flexspi_config* flexspi_config_t
FLEXSPI configuration structure.

typedef struct *_flexspi_device_config* flexspi_device_config_t
External device configuration items.

typedef struct *_flexspi_transfer* flexspi_transfer_t
Transfer structure for FLEXSPI.

typedef struct *_flexspi_handle* flexspi_handle_t

typedef void (*flexspi_transfer_callback_t)(FLEXSPI_Type *base, *flexspi_handle_t* *handle, *status_t* status, void *userData)
FLEXSPI transfer callback function.

typedef struct *_flexspi_addr_map_config* flexspi_addr_map_config_t
Address mapping configuration structure.

FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNT

FLEXSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)
Formula to form FLEXSPI instructions in LUT table.

struct _flexspi_ahbBuffer_config
#include <fsl_flexspi.h>

Public Members

uint8_t priority
This priority for AHB Master Read which this AHB RX Buffer is assigned.

uint8_t masterIndex
AHB Master ID the AHB RX Buffer is assigned.

uint16_t bufferSize
AHB buffer size in byte.

bool enablePrefetch
AHB Read Prefetch Enable for current AHB RX Buffer corresponding Master, allows prefetch disable/enable separately for each master.

struct _flexspi_config
#include <fsl_flexspi.h> FLEXSPI configuration structure.

Public Members

flexspi_read_sample_clock_t rxSampleClock
: Sample Clock source selection for Flash Reading.

bool enableSckFreeRunning
: Enable/disable SCK output free-running.

bool enableCombination
: Enable/disable combining PORT A and B Data Pins (SIOA[3:0] and SIOB[3:0]) to support Flash Octal mode.

bool enableDoze
: Enable/disable doze mode support.

bool enableHalfSpeedAccess
: Enable/disable divide by 2 of the clock for half speed commands.

bool enableSckBDiffOpt
: Enable/disable SCKB pad use as SCKA differential clock output, when enable, Port B flash access is not available.

bool enableSameConfigForAll
: Enable/disable same configuration for all connected devices when enabled, same configuration in FLASHA1CRx is applied to all.

uint16_t seqTimeoutCycle
: Timeout wait cycle for command sequence execution, timeout after ahbGrantTimeoutCyle*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.

flexspi_cs_interval_cycle_unit_t CSIntervalUnit
: CS interval unit, 1 or 256 cycle.

uint16_t CSInterval
: CS line assert interval, multiply CS interval unit to get the CS line assert interval cycles.

uint8_t CSHoldTime
: CS line hold time.

uint8_t CSSetupTime
: CS line setup time.

uint8_t dataValidTime
: Data valid time for external device.

uint8_t columnspace
: Column space size.

bool enableWordAddress
: If enable word address.

uint8_t AWRSeqIndex
: Sequence ID for AHB write command.

uint8_t AWRSeqNumber
: Sequence number for AHB write command.

uint8_t ARDSeqIndex
: Sequence ID for AHB read command.

uint8_t ARDSeqNumber
: Sequence number for AHB read command.

flexspi_ahb_write_wait_unit_t AHBWriteWaitUnit
: AHB write wait unit.

uint16_t AHBWriteWaitInterval
: AHB write wait interval, multiply AHB write interval unit to get the AHB write wait cycles.

bool enableWriteMask
: Enable/Disable FLEXSPI drive DQS pin as write mask when writing to external device.

struct __flexspi__transfer
: *#include <fsl_flexspi.h>* Transfer structure for FLEXSPI.

Public Members

uint32_t deviceAddress
: Operation device address.

flexspi_port_t port
: Operation port.

flexspi_command_type_t cmdType
: Execution command type.

uint8_t seqIndex
: Sequence ID for command.

uint8_t SeqNumber
: Sequence number for command.

uint32_t *data
: Data buffer.

size_t dataSize
Data size in bytes.

struct __flexspi__handle
#include <fsl_flexspi.h> Transfer handle structure for FLEXSPI.

Public Members

uint32_t state
Internal state for FLEXSPI transfer

uint8_t *data
Data buffer.

size_t dataSize
Remaining Data size in bytes.

size_t transferTotalSize
Total Data size in bytes.

flexspi_transfer_callback_t completionCallback
Callback for users while transfer finish or error occurred

void *userData
FLEXSPI callback function parameter.

struct __flexspi__addr__map__config
#include <fsl_flexspi.h> Address mapping configuration structure.

Public Members

uint32_t addrStart
Remapping start address.

uint32_t addrEnd
Remapping end address.

uint32_t addrOffset
Address offset.

bool remapEnable
Enable address remapping.

struct ahbConfig

Public Members

bool enableAHBWriteIpTxFifo
Enable AHB bus write access to IP TX FIFO.

bool enableAHBWriteIpRxFifo
Enable AHB bus write access to IP RX FIFO.

uint8_t ahbGrantTimeoutCycle
Timeout wait cycle for AHB command grant, timeout after ahbGrantTimeoutCyle*1024 AHB clock cycles.

uint16_t ahbBusTimeoutCycle

Timeout wait cycle for AHB read/write access, timeout after ahbBusTimeoutCycle*1024 AHB clock cycles.

uint8_t resumeWaitCycle

Wait cycle for idle state before suspended command sequence resume, timeout after ahbBusTimeoutCycle AHB clock cycles.

flexspi_ahbBuffer_config_t buffer[FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNTn(0)]

AHB buffer size.

bool enableClearAHBBufferOpt

Enable/disable automatically clean AHB RX Buffer and TX Buffer when FLEXSPI returns STOP mode ACK.

bool enableReadAddressOpt

Enable/disable remove AHB read burst start address alignment limitation. when enable, there is no AHB read burst start address alignment limitation.

bool enableAHBPrefetch

Enable/disable AHB read prefetch feature, when enabled, FLEXSPI will fetch more data than current AHB burst.

bool enableAHBBufferable

Enable/disable AHB bufferable write access support, when enabled, FLEXSPI return before waiting for command execution finished.

bool enableAHBCachable

Enable AHB bus cachable read access support.

2.19 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 kFLEXPSI_EDMAnSize1Bytes(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.

enum _flexspi_edma_ntransfer_size

eDMA transfer configuration

Values:

enumerator kFLEXPSI_EDMAnSize1Bytes
Source/Destination data transfer size is 1 byte every time

enumerator kFLEXPSI_EDMAnSize2Bytes
Source/Destination data transfer size is 2 bytes every time

enumerator kFLEXPSI_EDMAnSize4Bytes
Source/Destination data transfer size is 4 bytes every time

enumerator kFLEXPSI_EDMAnSize8Bytes
Source/Destination data transfer size is 8 bytes every time

enumerator kFLEXPSI_EDMAnSize32Bytes
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

2.20 GPC: General Power Controller Driver

FSL_GPC_DRIVER_VERSION
GPC driver version 2.2.0.

enum _gpc_lpm_mode
GPC LPM mode definition.

Values:

enumerator kGPC_RunMode
run mode

enumerator kGPC_WaitMode
wait mode

enumerator kGPC_StopMode
stop mode

enum _gpc_pgc_ack_sel
PGC ack signal selection

Values:

enumerator kGPC_DummyPGCPowerUpAck
dummy power up ack signal

enumerator kGPC_VirtualPGCPowerUpAck
virtual pgc power up ack signal

enumerator kGPC_DummyPGCPowerDownAck
dummy power down ack signal

enumerator kGPC_VirtualPGCPowerDownAck
virtual pgc power down ack signal

enumerator kGPC_NocPGCPowerUpAck
NOC power up ack signal

enumerator kGPC_NocPGCPowerDownAck
NOC power

enum _gpc_standby_count
Standby counter which GPC will wait between PMIC_STBY_REQ negation and assertion of PMIC_READY

Values:

enumerator kGPC_StandbyCounter4CkilClk
4 ckil clocks

enumerator kGPC_StandbyCounter8CkilClk
8 ckil clocks

enumerator kGPC_StandbyCounter16CkilClk
16 ckil clocks

enumerator kGPC_StandbyCounter32CkilClk
32 ckil clocks

enumerator kGPC_StandbyCounter64CkilClk
64 ckil clocks

enumerator kGPC_StandbyCounter128CkilClk
128 ckil clocks

enumerator kGPC_StandbyCounter256CkilClk
256 ckil clocks

enumerator kGPC_StandbyCounter512CkilClk
512 ckil clocks

typedef struct *_gpc_lpm_config* gpc_lpm_config_t
configuration for enter DSM mode

typedef struct *_gpc_dsm_config* gpc_dsm_config_t

static inline void GPC_AllowIRQs(GPC_Type *base)
Allow all the IRQ/Events within the charge of GPC.

Parameters

- base – GPC peripheral base address.

static inline void GPC_DisallowIRQs(GPC_Type *base)
Disallow all the IRQ/Events within the charge of GPC.

Parameters

- base – GPC peripheral base address.

static inline uint32_t GPC_GetLpmMode(GPC_Type *base)
Get current LPM mode.

Parameters

- base – GPC peripheral base address.

Return values
lpm – mode, reference _gpc_lpm_mode

void GPC_EnableIRQ(GPC_Type *base, uint32_t irqId)
Enable the IRQ.

Parameters

- base – GPC peripheral base address.
- irqId – ID number of IRQ to be enabled, available range is 0-127,reference SOC headerfile IRQn_Type.

void GPC_DisableIRQ(GPC_Type *base, uint32_t irqId)
Disable the IRQ.

Parameters

- base – GPC peripheral base address.
- irqId – ID number of IRQ to be disabled, available range is 0-127,reference SOC headerfile IRQn_Type.

bool GPC_GetIRQStatusFlag(GPC_Type *base, uint32_t irqId)
Get the IRQ/Event flag.

Parameters

- base – GPC peripheral base address.
- irqId – ID number of IRQ to be enabled, available range is 0-127,reference SOC headerfile IRQn_Type.

Returns
Indicated IRQ/Event is asserted or not.

static inline void GPC_DsmTriggerMask(GPC_Type *base, bool enable)
Mask the DSM trigger.

Parameters

- base – GPC peripheral base address.
- enable – true to enable mask, false to disable mask.

static inline void GPC_WFIMask(GPC_Type *base, bool enable)
Mask the WFI.

Parameters

- base – GPC peripheral base address.
- enable – true to enable mask, false to disable mask.

static inline void GPC_SelectPGCAckSignal(GPC_Type *base, uint32_t mask)
Select the PGC ACK signal.

Parameters

- base – GPC peripheral base address.
- mask – reference _gpc_pgc_ack_sel.

static inline void GPC_PowerDownRequestMask(GPC_Type *base, bool enable)
Power down request to virtual PGC mask or not.

Parameters

- base – GPC peripheral base address.
- enable – true to mask, false to not mask.

static inline void GPC_PGCMapping(GPC_Type *base, uint32_t mask)
PGC CPU Mapping.

Parameters

- base – GPC peripheral base address.
- mask – mask value reference PGC CPU mapping definition.

static inline void GPC_TimeSlotConfigureForPUS(GPC_Type *base, uint8_t slotIndex, uint32_t value)
Time slot configure.

Parameters

- base – GPC peripheral base address.
- slotIndex – time slot index.
- value – value to be configured

void GPC_EnterWaitMode(GPC_Type *base, *gpc_lpm_config_t* *config)
Enter WAIT mode.

Parameters

- base – GPC peripheral base address.
- config – lpm mode configurations.

void GPC_EnterStopMode(GPC_Type *base, *gpc_lpm_config_t* *config)

Enter STOP mode.

Parameters

- base – GPC peripheral base address.
- config – lpm mode configurations.

void GPC_Init(GPC_Type *base, uint32_t powerUpSlot, uint32_t powerDownSlot)

GPC init function.

Parameters

- base – GPC peripheral base address.
- powerUpSlot – power up slot number.
- powerDownSlot – power down slot number.

GPC_PCG_TIME_SLOT_TOTAL_NUMBER

Total number of the timeslot.

struct _gpc_lpm_config

#include <fsl_gpc.h>

Public Members

bool enFastWakeUp

enable fast wake up from lpm mode

bool enCpuClk

enable CPU clock when LPM enter

bool enVirtualPGCPowerup

enable virtual PGC power up with LPM enter

bool enVirtualPGCPowerdown

enable virtual PGC power down with LPM enter

bool enWfiMask

enable WFI Mask

bool enDsmMask

enable DSM Mask

struct _gpc_dsm_config

#include <fsl_gpc.h>

Public Members

bool disableRamLpctl

Memory can be defined to go to retention mode or not

bool enPMICStandBy

PMIC can be defined to be stand-by mode or not

uint8_t pmicStandByCounter

PMIC standby counter, reference _gpc_standby_count

uint8_t regBypassCounter

if PMIC standby is request, regulator bypass should be enable, and the counter can be defined

2.21 GPIO: General-Purpose Input/Output Driver

void GPIO_PinInit(GPIO_Type *base, uint32_t pin, const *gpio_pin_config_t* *Config)

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

Parameters

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

void GPIO_PinWrite(GPIO_Type *base, uint32_t pin, uint8_t output)

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

Parameters

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

static inline void GPIO_WritePinOutput(GPIO_Type *base, uint32_t pin, uint8_t output)

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

Deprecated:

Do not use this function. It has been superceded by GPIO_PinWrite.

static inline void GPIO_PortSet(GPIO_Type *base, uint32_t mask)

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

Parameters

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

static inline void GPIO_SetPinsOutput(GPIO_Type *base, uint32_t mask)

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

Deprecated:

Do not use this function. It has been superceded by GPIO_PortSet.

static inline void GPIO_PortClear(GPIO_Type *base, uint32_t mask)

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

Parameters

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

static inline void GPIO_ClearPinsOutput(GPIO_Type *base, uint32_t mask)

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

Deprecated:

Do not use this function. It has been superceded by GPIO_PortClear.

static inline void GPIO_PortToggle(GPIO_Type *base, uint32_t mask)

Reverses the current output logic of the multiple GPIO pins.

Parameters

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

static inline uint32_t GPIO_PinRead(GPIO_Type *base, uint32_t pin)

Reads the current input value of the GPIO port.

Parameters

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

Return values

GPIO – port input value.

static inline uint32_t GPIO_ReadPinInput(GPIO_Type *base, uint32_t pin)

Reads the current input value of the GPIO port.

Deprecated:

Do not use this function. It has been superceded by GPIO_PinRead.

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

Reads the current GPIO pin pad status.

Parameters

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

Return values

GPIO – pin pad status value.

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

Reads the current GPIO pin pad status.

Deprecated:

Do not use this function. It has been superceded by GPIO_PinReadPadStatus.

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

Sets the current pin interrupt mode.

Parameters

- base – GPIO base pointer.
- pin – GPIO port pin number.
- pinInterruptMode – pointer to a gpio_interrupt_mode_t structure that contains the interrupt mode information.

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

Sets the current pin interrupt mode.

Deprecated:

Do not use this function. It has been superceded by GPIO_PinSetInterruptConfig.

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

Enables the specific pin interrupt.

Parameters

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

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

Enables the specific pin interrupt.

Parameters

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

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

Disables the specific pin interrupt.

Parameters

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

static inline void GPIO_DisableInterrupts(GPIO_Type *base, uint32_t mask)

Disables the specific pin interrupt.

Deprecated:

Do not use this function. It has been superceded by GPIO_PortDisableInterrupts.

static inline uint32_t GPIO_PortGetInterruptFlags(GPIO_Type *base)

Reads individual pin interrupt status.

Parameters

- base – GPIO base pointer.

Return values

current – pin interrupt status flag.

static inline uint32_t GPIO_GetPinsInterruptFlags(GPIO_Type *base)

Reads individual pin interrupt status.

Parameters

- base – GPIO base pointer.

Return values

current – pin interrupt status flag.

static inline void GPIO_PortClearInterruptFlags(GPIO_Type *base, uint32_t mask)

Clears pin interrupt flag. Status flags are cleared by writing a 1 to the corresponding bit position.

Parameters

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

static inline void GPIO_ClearPinsInterruptFlags(GPIO_Type *base, uint32_t mask)

Clears pin interrupt flag. Status flags are cleared by writing a 1 to the corresponding bit position.

Parameters

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

FSL_GPIO_DRIVER_VERSION

GPIO driver version.

enum _gpio_pin_direction

GPIO direction definition.

Values:

enumerator kGPIO_DigitalInput

Set current pin as digital input.

enumerator kGPIO_DigitalOutput

Set current pin as digital output.

enum _gpio_interrupt_mode

GPIO interrupt mode definition.

Values:

enumerator kGPIO_NoIntmode

Set current pin general IO functionality.

enumerator kGPIO_IntLowLevel

Set current pin interrupt is low-level sensitive.

enumerator kGPIO_IntHighLevel

Set current pin interrupt is high-level sensitive.

enumerator kGPIO_IntRisingEdge

Set current pin interrupt is rising-edge sensitive.

enumerator kGPIO_IntFallingEdge

Set current pin interrupt is falling-edge sensitive.

enumerator kGPIO_IntRisingOrFallingEdge

Enable the edge select bit to override the ICR register's configuration.

typedef enum *_gpio_pin_direction* gpio_pin_direction_t

GPIO direction definition.

typedef enum *_gpio_interrupt_mode* gpio_interrupt_mode_t

GPIO interrupt mode definition.

typedef struct *_gpio_pin_config* gpio_pin_config_t

GPIO Init structure definition.

struct _gpio_pin_config

#include <fsl_gpio.h> GPIO Init structure definition.

Public Members

gpio_pin_direction_t direction
Specifies the pin direction.

uint8_t outputLogic
Set a default output logic, which has no use in input

gpio_interrupt_mode_t interruptMode
Specifies the pin interrupt mode, a value of gpio_interrupt_mode_t.

2.22 GPT: General Purpose Timer

void GPT_Init(GPT_Type *base, const *gpt_config_t* *initConfig)
Initialize GPT to reset state and initialize running mode.

Parameters

- base – GPT peripheral base address.
- initConfig – GPT mode setting configuration.

void GPT_Deinit(GPT_Type *base)
Disables the module and gates the GPT clock.

Parameters

- base – GPT peripheral base address.

void GPT_GetDefaultConfig(*gpt_config_t* *config)
Fills in the GPT configuration structure with default settings.

The default values are:

```
config->clockSource = kGPT_ClockSource_Periph;
config->divider = 1U;
config->enableRunInStop = true;
config->enableRunInWait = true;
config->enableRunInDoze = false;
config->enableRunInDbg = false;
config->enableFreeRun = false;
config->enableMode  = true;
```

Parameters

- config – Pointer to the user configuration structure.

static inline void GPT_SoftwareReset(GPT_Type *base)
Software reset of GPT module.

Parameters

- base – GPT peripheral base address.

static inline void GPT_SetClockSource(GPT_Type *base, *gpt_clock_source_t* gptClkSource)
Set clock source of GPT.

Parameters

- base – GPT peripheral base address.
- gptClkSource – Clock source (see gpt_clock_source_t typedef enumeration).

static inline *gpt_clock_source_t* GPT_GetClockSource(GPT_Type *base)
Get clock source of GPT.

Parameters

- base – GPT peripheral base address.

Returns
clock source (see gpt_clock_source_t typedef enumeration).

static inline void GPT_SetClockDivider(GPT_Type *base, uint32_t divider)
Set pre scaler of GPT.

Parameters

- base – GPT peripheral base address.
- divider – Divider of GPT (1-4096).

static inline uint32_t GPT_GetClockDivider(GPT_Type *base)
Get clock divider in GPT module.

Parameters

- base – GPT peripheral base address.

Returns
clock divider in GPT module (1-4096).

static inline void GPT_SetOscClockDivider(GPT_Type *base, uint32_t divider)
OSC 24M pre-scaler before selected by clock source.

Parameters

- base – GPT peripheral base address.
- divider – OSC Divider(1-16).

static inline uint32_t GPT_GetOscClockDivider(GPT_Type *base)
Get OSC 24M clock divider in GPT module.

Parameters

- base – GPT peripheral base address.

Returns
OSC clock divider in GPT module (1-16).

static inline void GPT_StartTimer(GPT_Type *base)
Start GPT timer.

Parameters

- base – GPT peripheral base address.

static inline void GPT_StopTimer(GPT_Type *base)
Stop GPT timer.

Parameters

- base – GPT peripheral base address.

static inline uint32_t GPT_GetCurrentTimerCount(GPT_Type *base)
Reads the current GPT counting value.

Parameters

- base – GPT peripheral base address.

Returns
Current GPT counter value.

static inline void GPT_SetInputOperationMode(GPT_Type *base, *gpt_input_capture_channel_t* channel, *gpt_input_operation_mode_t* mode)

Set GPT operation mode of input capture channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT capture channel (see gpt_input_capture_channel_t typedef enumeration).
- mode – GPT input capture operation mode (see gpt_input_operation_mode_t typedef enumeration).

static inline *gpt_input_operation_mode_t* GPT_GetInputOperationMode(GPT_Type *base, *gpt_input_capture_channel_t* channel)

Get GPT operation mode of input capture channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT capture channel (see gpt_input_capture_channel_t typedef enumeration).

Returns

GPT input capture operation mode (see gpt_input_operation_mode_t typedef enumeration).

static inline uint32_t GPT_GetInputCaptureValue(GPT_Type *base, *gpt_input_capture_channel_t* channel)

Get GPT input capture value of certain channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT capture channel (see gpt_input_capture_channel_t typedef enumeration).

Returns

GPT input capture value.

static inline void GPT_SetOutputOperationMode(GPT_Type *base, *gpt_output_compare_channel_t* channel, *gpt_output_operation_mode_t* mode)

Set GPT operation mode of output compare channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).
- mode – GPT output operation mode (see gpt_output_operation_mode_t typedef enumeration).

static inline *gpt_output_operation_mode_t* GPT_GetOutputOperationMode(GPT_Type *base, *gpt_output_compare_channel_t* channel)

Get GPT operation mode of output compare channel.

Parameters

- base – GPT peripheral base address.

- channel – GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).

Returns
GPT output operation mode (see gpt_output_operation_mode_t typedef enumeration).

static inline void GPT_SetOutputCompareValue(GPT_Type *base, *gpt_output_compare_channel_t* channel, uint32_t value)

Set GPT output compare value of output compare channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).
- value – GPT output compare value.

static inline uint32_t GPT_GetOutputCompareValue(GPT_Type *base, *gpt_output_compare_channel_t* channel)

Get GPT output compare value of output compare channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).

Returns
GPT output compare value.

static inline void GPT_ForceOutput(GPT_Type *base, *gpt_output_compare_channel_t* channel)

Force GPT output action on output compare channel, ignoring comparator.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).

static inline void GPT_EnableInterrupts(GPT_Type *base, uint32_t mask)

Enables the selected GPT interrupts.

Parameters

- base – GPT peripheral base address.
- mask – The interrupts to enable. This is a logical OR of members of the enumeration gpt_interrupt_enable_t

static inline void GPT_DisableInterrupts(GPT_Type *base, uint32_t mask)

Disables the selected GPT interrupts.

Parameters

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

static inline uint32_t GPT_GetEnabledInterrupts(GPT_Type *base)

Gets the enabled GPT interrupts.

Parameters

- base – GPT peripheral base address

Returns
The enabled interrupts. This is the logical OR of members of the enumeration gpt_interrupt_enable_t

static inline uint32_t GPT_GetStatusFlags(GPT_Type *base, *gpt_status_flag_t* flags)

Get GPT status flags.

Parameters

- base – GPT peripheral base address.
- flags – GPT status flag mask (see gpt_status_flag_t for bit definition).

Returns
GPT status, each bit represents one status flag.

static inline void GPT_ClearStatusFlags(GPT_Type *base, *gpt_status_flag_t* flags)

Clears the GPT status flags.

Parameters

- base – GPT peripheral base address.
- flags – GPT status flag mask (see gpt_status_flag_t for bit definition).

FSL_GPT_DRIVER_VERSION

enum _gpt_clock_source

List of clock sources.

Note: Actual number of clock sources is SoC dependent

Values:

enumerator kGPT_ClockSource_Off
GPT Clock Source Off.

enumerator kGPT_ClockSource_Periph
GPT Clock Source from Peripheral Clock.

enumerator kGPT_ClockSource_HighFreq
GPT Clock Source from High Frequency Reference Clock.

enumerator kGPT_ClockSource_Ext
GPT Clock Source from external pin.

enumerator kGPT_ClockSource_LowFreq
GPT Clock Source from Low Frequency Reference Clock.

enumerator kGPT_ClockSource_Osc
GPT Clock Source from Crystal oscillator.

enum _gpt_input_capture_channel

List of input capture channel number.

Values:

enumerator kGPT_InputCapture_Channel1
GPT Input Capture Channel1.

enumerator kGPT_InputCapture_Channel2
GPT Input Capture Channel2.

enum _gpt_input_operation_mode

List of input capture operation mode.

Values:

enumerator kGPT_InputOperation_Disabled

Don't capture.

enumerator kGPT_InputOperation_RiseEdge

Capture on rising edge of input pin.

enumerator kGPT_InputOperation_FallEdge

Capture on falling edge of input pin.

enumerator kGPT_InputOperation_BothEdge

Capture on both edges of input pin.

enum _gpt_output_compare_channel

List of output compare channel number.

Values:

enumerator kGPT_OutputCompare_Channel1

Output Compare Channel1.

enumerator kGPT_OutputCompare_Channel2

Output Compare Channel2.

enumerator kGPT_OutputCompare_Channel3

Output Compare Channel3.

enum _gpt_output_operation_mode

List of output compare operation mode.

Values:

enumerator kGPT_OutputOperation_Disconnected

Don't change output pin.

enumerator kGPT_OutputOperation_Toggle

Toggle output pin.

enumerator kGPT_OutputOperation_Clear

Set output pin low.

enumerator kGPT_OutputOperation_Set

Set output pin high.

enumerator kGPT_OutputOperation_Activelow

Generate a active low pulse on output pin.

enum _gpt_interrupt_enable

List of GPT interrupts.

Values:

enumerator kGPT_OutputCompare1InterruptEnable

Output Compare Channel1 interrupt enable

enumerator kGPT_OutputCompare2InterruptEnable

Output Compare Channel2 interrupt enable

enumerator kGPT_OutputCompare3InterruptEnable

Output Compare Channel3 interrupt enable

enumerator kGPT_InputCapture1InterruptEnable
Input Capture Channel1 interrupt enable

enumerator kGPT_InputCapture2InterruptEnable
Input Capture Channel1 interrupt enable

enumerator kGPT_RollOverFlagInterruptEnable
Counter rolled over interrupt enable

enum _gpt_status_flag
Status flag.

Values:

enumerator kGPT_OutputCompare1Flag
Output compare channel 1 event.

enumerator kGPT_OutputCompare2Flag
Output compare channel 2 event.

enumerator kGPT_OutputCompare3Flag
Output compare channel 3 event.

enumerator kGPT_InputCapture1Flag
Input Capture channel 1 event.

enumerator kGPT_InputCapture2Flag
Input Capture channel 2 event.

enumerator kGPT_RollOverFlag
Counter reaches maximum value and rolled over to 0 event.

typedef enum *_gpt_clock_source* gpt_clock_source_t
List of clock sources.

Note: Actual number of clock sources is SoC dependent

typedef enum *_gpt_input_capture_channel* gpt_input_capture_channel_t
List of input capture channel number.

typedef enum *_gpt_input_operation_mode* gpt_input_operation_mode_t
List of input capture operation mode.

typedef enum *_gpt_output_compare_channel* gpt_output_compare_channel_t
List of output compare channel number.

typedef enum *_gpt_output_operation_mode* gpt_output_operation_mode_t
List of output compare operation mode.

typedef enum *_gpt_interrupt_enable* gpt_interrupt_enable_t
List of GPT interrupts.

typedef enum *_gpt_status_flag* gpt_status_flag_t
Status flag.

typedef struct *_gpt_init_config* gpt_config_t
Structure to configure the running mode.

struct _gpt_init_config
#include <fsl_gpt.h> Structure to configure the running mode.

Public Members

gpt_clock_source_t clockSource
clock source for GPT module.

uint32_t divider
clock divider (prescaler+1) from clock source to counter.

bool enableFreeRun
true: FreeRun mode, false: Restart mode.

bool enableRunInWait
GPT enabled in wait mode.

bool enableRunInStop
GPT enabled in stop mode.

bool enableRunInDoze
GPT enabled in doze mode.

bool enableRunInDbg
GPT enabled in debug mode.

bool enableMode
true: counter reset to 0 when enabled; false: counter retain its value when enabled.

2.23 I2C: Inter-Integrated Circuit Driver

2.24 I2C Driver

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

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

Note: This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can be custom filled or it can be set with default values by using the I2C_MasterGetDefaultConfig(). After calling this API, the master is ready to transfer. This is an example.

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

Parameters

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

void I2C_MasterDeinit(I2C_Type *base)

De-initializes the I2C master peripheral. Call this API to gate the I2C clock. The I2C master module can't work unless the I2C_MasterInit is called.

Parameters

- base – I2C base pointer

void I2C_MasterGetDefaultConfig(*i2c_master_config_t* *masterConfig)

Sets the I2C master configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in the I2C_MasterInit(). Use the initialized structure unchanged in the I2C_MasterInit() or modify the structure before calling the I2C_MasterInit(). This is an example.

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

Parameters

- masterConfig – A pointer to the master configuration structure.

void I2C_SlaveInit(I2C_Type *base, const *i2c_slave_config_t* *slaveConfig)

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

Note: This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can partly be set with default values by I2C_SlaveGetDefaultConfig() or it can be custom filled by the user. This is an example.

```
i2c_slave_config_t config = {
.enableSlave = true,
.slaveAddress = 0x1DU,
};
I2C_SlaveInit(I2C0, &config);
```

Parameters

- base – I2C base pointer
- slaveConfig – A pointer to the slave configuration structure

void I2C_SlaveDeinit(I2C_Type *base)

De-initializes the I2C slave peripheral. Calling this API gates the I2C clock. The I2C slave module can't work unless the I2C_SlaveInit is called to enable the clock.

Parameters

- base – I2C base pointer

void I2C_SlaveGetDefaultConfig(*i2c_slave_config_t* *slaveConfig)

Sets the I2C slave configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in the I2C_SlaveInit(). Modify fields of the structure before calling the I2C_SlaveInit(). This is an example.

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

Parameters

- slaveConfig – A pointer to the slave configuration structure.

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

Enables or disables the I2C peripheral operation.

Parameters

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

static inline uint32_t I2C_MasterGetStatusFlags(I2C_Type *base)

Gets the I2C status flags.

Parameters

- base – I2C base pointer

Returns

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

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

Clears the I2C status flag state.

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

Parameters

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

static inline uint32_t I2C_SlaveGetStatusFlags(I2C_Type *base)

Gets the I2C status flags.

Parameters

- base – I2C base pointer

Returns

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

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

Clears the I2C status flag state.

The following status register flags can be cleared kI2C_ArbitrationLostFlag and kI2C_IntPendingFlag

Parameters

- base – I2C base pointer
- statusMask – The status flag mask, defined in type i2c_status_flag_t. The parameter can be any combination of the following values:
  - kI2C_IntPendingFlagFlag

void I2C_EnableInterrupts(I2C_Type *base, uint32_t mask)

Enables I2C interrupt requests.

Parameters

- base – I2C base pointer

- mask – interrupt source The parameter can be combination of the following source if defined:
  - kI2C_GlobalInterruptEnable
  - kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable
  - kI2C_SdaTimeoutInterruptEnable

void I2C_DisableInterrupts(I2C_Type *base, uint32_t mask)

Disables I2C interrupt requests.

Parameters

- base – I2C base pointer
- mask – interrupt source The parameter can be combination of the following source if defined:
  - kI2C_GlobalInterruptEnable
  - kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable
  - kI2C_SdaTimeoutInterruptEnable

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

Sets the I2C master transfer baud rate.

Parameters

- base – I2C base pointer
- baudRate_Bps – the baud rate value in bps
- srcClock_Hz – Source clock

status_t I2C_MasterStart(I2C_Type *base, uint8_t address, *i2c_direction_t* direction)

Sends a START on the I2C bus.

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

Parameters

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

Return values

- kStatus_Success – Successfully send the start signal.
- kStatus_I2C_Busy – Current bus is busy.

status_t I2C_MasterStop(I2C_Type *base)

Sends a STOP signal on the I2C bus.

Return values

- kStatus_Success – Successfully send the stop signal.
- kStatus_I2C_Timeout – Send stop signal failed, timeout.

status_t I2C_MasterRepeatedStart(I2C_Type *base, uint8_t address, *i2c_direction_t* direction)

Sends a REPEATED START on the I2C bus.

Parameters

- base – I2C peripheral base pointer
- address – 7-bit slave device address.

- direction – Master transfer directions(transmit/receive).

Return values

- kStatus_Success – Successfully send the start signal.
- kStatus_I2C_Busy – Current bus is busy but not occupied by current I2C master.

status_t I2C_MasterWriteBlocking(I2C_Type *base, const uint8_t *txBuff, size_t txSize, uint32_t flags)

Performs a polling send transaction on the I2C bus.

Parameters

- base – The I2C peripheral base pointer.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.
- flags – Transfer control flag to decide whether need to send a stop, use kI2C_TransferDefaultFlag to issue a stop and kI2C_TransferNoStop to not send a stop.

Return values

- kStatus_Success – Successfully complete the data transmission.
- kStatus_I2C_ArbitrationLost – Transfer error, arbitration lost.
- kStataus_I2C_Nak – Transfer error, receive NAK during transfer.

status_t I2C_MasterReadBlocking(I2C_Type *base, uint8_t *rxBuff, size_t rxSize, uint32_t flags)

Performs a polling receive transaction on the I2C bus.

Note: The I2C_MasterReadBlocking function stops the bus before reading the final byte. Without stopping the bus prior for the final read, the bus issues another read, resulting in garbage data being read into the data register.

Parameters

- base – I2C peripheral base pointer.
- rxBuff – The pointer to the data to store the received data.
- rxSize – The length in bytes of the data to be received.
- flags – Transfer control flag to decide whether need to send a stop, use kI2C_TransferDefaultFlag to issue a stop and kI2C_TransferNoStop to not send a stop.

Return values

- kStatus_Success – Successfully complete the data transmission.
- kStatus_I2C_Timeout – Send stop signal failed, timeout.

status_t I2C_SlaveWriteBlocking(I2C_Type *base, const uint8_t *txBuff, size_t txSize)

Performs a polling send transaction on the I2C bus.

Parameters

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

Return values

- kStatus_Success – Successfully complete the data transmission.
- kStatus_I2C_ArbitrationLost – Transfer error, arbitration lost.
- kStataus_I2C_Nak – Transfer error, receive NAK during transfer.

status_t I2C_SlaveReadBlocking(I2C_Type *base, uint8_t *rxBuff, size_t rxSize)

Performs a polling receive transaction on the I2C bus.

Parameters

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

status_t I2C_MasterTransferBlocking(I2C_Type *base, *i2c_master_transfer_t* *xfer)

Performs a master polling transfer on the I2C bus.

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

Parameters

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

Return values

- kStatus_Success – Successfully complete the data transmission.
- kStatus_I2C_Busy – Previous transmission still not finished.
- kStatus_I2C_Timeout – Transfer error, wait signal timeout.
- kStatus_I2C_ArbitrationLost – Transfer error, arbitration lost.
- kStataus_I2C_Nak – Transfer error, receive NAK during transfer.

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

Initializes the I2C handle which is used in transactional functions.

Parameters

- base – I2C base pointer.
- handle – pointer to i2c_master_handle_t structure to store the transfer state.
- callback – pointer to user callback function.
- userData – user parameter passed to the callback function.

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

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

Note: Calling the API returns immediately after transfer initiates. The user needs to call I2C_MasterGetTransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_I2C_Busy, the transfer is finished.

Parameters

- base – I2C base pointer.
- handle – pointer to i2c_master_handle_t structure which stores the transfer state.
- xfer – pointer to i2c_master_transfer_t structure.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_I2C_Busy – Previous transmission still not finished.
- kStatus_I2C_Timeout – Transfer error, wait signal timeout.

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

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

Parameters

- base – I2C base pointer.
- handle – pointer to i2c_master_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

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

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

Aborts an interrupt non-blocking transfer early.

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

Parameters

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

Return values

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

void I2C_MasterTransferHandleIRQ(I2C_Type *base, void *i2cHandle)

Master interrupt handler.

Parameters

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

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

Initializes the I2C handle which is used in transactional functions.

Parameters

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

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

Starts accepting slave transfers.

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

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

Parameters

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

Return values

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

void I2C_SlaveTransferAbort(I2C_Type *base, *i2c_slave_handle_t* *handle)

Aborts the slave transfer.

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

Parameters

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

status_t I2C_SlaveTransferGetCount(I2C_Type *base, *i2c_slave_handle_t* *handle, size_t *count)

Gets the slave transfer remaining bytes during a interrupt non-blocking transfer.

Parameters

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

Return values

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

void I2C_SlaveTransferHandleIRQ(I2C_Type *base, void *i2cHandle)

Slave interrupt handler.

Parameters

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

FSL_I2C_DRIVER_VERSION

I2C driver version.

I2C status return codes.

Values:

enumerator kStatus_I2C_Busy

I2C is busy with current transfer.

enumerator kStatus_I2C_Idle

Bus is Idle.

enumerator kStatus_I2C_Nak

NAK received during transfer.

enumerator kStatus_I2C_ArbitrationLost

Arbitration lost during transfer.

enumerator kStatus_I2C_Timeout

Timeout polling status flags.

enumerator kStatus_I2C_Addr_Nak

NAK received during the address probe.

enum _i2c_flags

I2C peripheral flags.

The following status register flags can be cleared:

- kI2C_ArbitrationLostFlag
- kI2C_IntPendingFlag

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

Values:

enumerator kI2C_ReceiveNakFlag

I2C receive NAK flag.

enumerator kI2C_IntPendingFlag

I2C interrupt pending flag.

enumerator kI2C_TransferDirectionFlag

I2C transfer direction flag.

enumerator kI2C_ArbitrationLostFlag

I2C arbitration lost flag.

enumerator kI2C_BusBusyFlag
I2C bus busy flag.

enumerator kI2C_AddressMatchFlag
I2C address match flag.

enumerator kI2C_TransferCompleteFlag
I2C transfer complete flag.

enum _i2c_interrupt_enable
I2C feature interrupt source.

Values:

enumerator kI2C_GlobalInterruptEnable
I2C global interrupt.

enum _i2c_direction
The direction of master and slave transfers.

Values:

enumerator kI2C_Write
Master transmits to the slave.

enumerator kI2C_Read
Master receives from the slave.

enum _i2c_master_transfer_flags
I2C transfer control flag.

Values:

enumerator kI2C_TransferDefaultFlag
A transfer starts with a start signal, stops with a stop signal.

enumerator kI2C_TransferNoStartFlag
A transfer starts without a start signal, only support write only or write+read with no start flag, do not support read only with no start flag.

enumerator kI2C_TransferRepeatedStartFlag
A transfer starts with a repeated start signal.

enumerator kI2C_TransferNoStopFlag
A transfer ends without a stop signal.

enum _i2c_slave_transfer_event
Set of events sent to the callback for nonblocking slave transfers.

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

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

Values:

enumerator kI2C_SlaveAddressMatchEvent
Received the slave address after a start or repeated start.

enumerator kI2C_SlaveTransmitEvent

A callback is requested to provide data to transmit (slave-transmitter role).

enumerator kI2C_SlaveReceiveEvent

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

enumerator kI2C_SlaveTransmitAckEvent

A callback needs to either transmit an ACK or NACK.

enumerator kI2C_SlaveCompletionEvent

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

enumerator kI2C_SlaveAllEvents

A bit mask of all available events.

typedef enum *_i2c_direction* i2c_direction_t

The direction of master and slave transfers.

typedef struct *_i2c_master_config* i2c_master_config_t

I2C master user configuration.

typedef struct *_i2c_master_handle* i2c_master_handle_t

I2C master handle typedef.

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

I2C master transfer callback typedef.

typedef struct *_i2c_master_transfer* i2c_master_transfer_t

I2C master transfer structure.

typedef enum *_i2c_slave_transfer_event* i2c_slave_transfer_event_t

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

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

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

typedef struct *_i2c_slave_handle* i2c_slave_handle_t

I2C slave handle typedef.

typedef struct *_i2c_slave_config* i2c_slave_config_t

I2C slave user configuration.

typedef struct *_i2c_slave_transfer* i2c_slave_transfer_t

I2C slave transfer structure.

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

I2C slave transfer callback typedef.

I2C_RETRY_TIMES

Retry times for waiting flag.

struct _i2c_master_config

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

Public Members

bool enableMaster
: Enables the I2C peripheral at initialization time.

uint32_t baudRate_Bps
: Baud rate configuration of I2C peripheral.

struct __i2c_master_transfer
: *#include <fsl_i2c.h>* I2C master transfer structure.

Public Members

uint32_t flags
: A transfer flag which controls the transfer.

uint8_t slaveAddress
: 7-bit slave address.

i2c_direction_t direction
: A transfer direction, read or write.

uint32_t subaddress
: A sub address. Transferred MSB first.

uint8_t subaddressSize
: A size of the command buffer.

uint8_t *volatile data
: A transfer buffer.

volatile size_t dataSize
: A transfer size.

struct __i2c_master_handle
: *#include <fsl_i2c.h>* I2C master handle structure.

Public Members

i2c_master_transfer_t transfer
: I2C master transfer copy.

size_t transferSize
: Total bytes to be transferred.

uint8_t state
: A transfer state maintained during transfer.

i2c_master_transfer_callback_t completionCallback
: A callback function called when the transfer is finished.

void *userData
: A callback parameter passed to the callback function.

struct __i2c_slave_config
: *#include <fsl_i2c.h>* I2C slave user configuration.

Public Members

bool enableSlave
: Enables the I2C peripheral at initialization time.

uint16_t slaveAddress
: A slave address configuration.

struct __i2c_slave_transfer
: *#include <fsl_i2c.h>* I2C slave transfer structure.

Public Members

i2c_slave_transfer_event_t event
: A reason that the callback is invoked.

uint8_t *volatile data
: A transfer buffer.

volatile size_t dataSize
: A transfer size.

status_t completionStatus
: Success or error code describing how the transfer completed. Only applies for kI2C_SlaveCompletionEvent.

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

struct __i2c_slave_handle
: *#include <fsl_i2c.h>* I2C slave handle structure.

Public Members

volatile uint8_t state
: A transfer state maintained during transfer.

i2c_slave_transfer_t transfer
: I2C slave transfer copy.

uint32_t eventMask
: A mask of enabled events.

i2c_slave_transfer_callback_t callback
: A callback function called at the transfer event.

void *userData
: A callback parameter passed to the callback.

2.25 Iomuxc_driver

static inline void IOMUXC_SetPinMux(uintptr_t muxRegister, uint32_t muxMode, uintptr_t inputRegister, uint32_t inputDaisy, uintptr_t configRegister, uint32_t inputOnfield)
: Sets the IOMUXC pin mux mode.

This is an example to set the I2C4_SDA as the pwm1_OUT:

```
IOMUXC_SetPinMux(IOMUXC_I2C4_SDA_PWM1_OUT, 0);
```

Note: The first five parameters can be filled with the pin function ID macros.

Parameters

- muxRegister – The pin mux register_
- muxMode – The pin mux mode_
- inputRegister – The select input register_
- inputDaisy – The input daisy_
- configRegister – The config register_
- inputOnfield – The pad->module input inversion_

static inline void IOMUXC_SetPinConfig(uintptr_t muxRegister, uint32_t muxMode, uintptr_t inputRegister, uint32_t inputDaisy, uintptr_t configRegister, uint32_t configValue)

Sets the IOMUXC pin configuration.

This is an example to set pin configuration for IOMUXC_I2C4_SDA_PWM1_OUT:

```
IOMUXC_SetPinConfig(IOMUXC_I2C4_SDA_PWM1_OUT, IOMUXC_SW_PAD_CTL_PAD_
↪ODE_MASK | IOMUXC0_SW_PAD_CTL_PAD_DSE(2U))
```

Note: The previous five parameters can be filled with the pin function ID macros.

Parameters

- muxRegister – The pin mux register_
- muxMode – The pin mux mode_
- inputRegister – The select input register_
- inputDaisy – The input daisy_
- configRegister – The config register_
- configValue – The pin config value_

FSL_IOMUXC_DRIVER_VERSION

IOMUXC driver version 2.0.4.

IOMUXC_BOOT_MODE0_SRC_BOOT_MODE0

IOMUXC_BOOT_MODE1_SRC_BOOT_MODE1

IOMUXC_BOOT_MODE2_SRC_BOOT_MODE2

IOMUXC_BOOT_MODE3_SRC_BOOT_MODE3

IOMUXC_JTAG_MOD_JTAG_MODE

IOMUXC_JTAG_TDI_JTAG_TDI

IOMUXC_JTAG_TMS_JTAG_TMS

IOMUXC_JTAG_TCK_JTAG_TCK

IOMUXC_JTAG_TDO_JTAG_TDO

IOMUXC_RTC_XTALI_SNVS_RTC

IOMUXC_PMIC_STBY_REQ_CCM_PMIC_STBY_REQ

IOMUXC_PMIC_ON_REQ_SNVS_PMIC_ON_REQ

IOMUXC_ONOFF_SNVS_ONOFF

IOMUXC_POR_B_SNVS_POR_B

IOMUXC_GPIO1_IO00_GPIO1_IO00

IOMUXC_GPIO1_IO00_CCM_ENET_PHY_REF_CLK_ROOT

IOMUXC_GPIO1_IO00_ISP_FL_TRIG_0

IOMUXC_GPIO1_IO00_ANAMIX_REF_CLK_32K

IOMUXC_GPIO1_IO00_CCM_EXT_CLK1

IOMUXC_GPIO1_IO01_GPIO1_IO01

IOMUXC_GPIO1_IO01_PWM1_OUT

IOMUXC_GPIO1_IO01_ISP_SHUTTER_TRIG_0

IOMUXC_GPIO1_IO01_ANAMIX_REF_CLK_24M

IOMUXC_GPIO1_IO01_CCM_EXT_CLK2

IOMUXC_GPIO1_IO02_GPIO1_IO02

IOMUXC_GPIO1_IO02_WDOG1_WDOG_B

IOMUXC_GPIO1_IO02_ISP_FLASH_TRIG_0

IOMUXC_GPIO1_IO02_WDOG1_WDOG_ANY

IOMUXC_GPIO1_IO02_SJC_DE_B

IOMUXC_GPIO1_IO03_GPIO1_IO03

IOMUXC_GPIO1_IO03_USDHC1_VSELECT

IOMUXC_GPIO1_IO03_ISP_PRELIGHT_TRIG_0

IOMUXC_GPIO1_IO03_SDMA1_EXT_EVENT0

IOMUXC_GPIO1_IO04_GPIO1_IO04

IOMUXC_GPIO1_IO04_USDHC2_VSELECT

IOMUXC_GPIO1_IO04_ISP_SHUTTER_OPEN_0

IOMUXC_GPIO1_IO04_SDMA1_EXT_EVENT1

IOMUXC_GPIO1_IO05_GPIO1_IO05

IOMUXC_GPIO1_IO05_M7_NMI

IOMUXC_GPIO1_IO05_ISP_FL_TRIG_1

IOMUXC_GPIO1_IO05_CCM_PMIC_READY

IOMUXC_GPIO1_IO06_GPIO1_IO06

IOMUXC_GPIO1_IO06_ENET_QOS_MDC

IOMUXC_GPIO1_IO06_ISP_SHUTTER_TRIG_1

IOMUXC_GPIO1_IO06_USDHC1_CD_B

IOMUXC_GPIO1_IO06_CCM_EXT_CLK3

IOMUXC_GPIO1_IO07_GPIO1_IO07

IOMUXC_GPIO1_IO07_ENET_QOS_MDIO

IOMUXC_GPIO1_IO07_ISP_FLASH_TRIG_1

IOMUXC_GPIO1_IO07_USDHC1_WP

IOMUXC_GPIO1_IO07_CCM_EXT_CLK4

IOMUXC_GPIO1_IO08_GPIO1_IO08

IOMUXC_GPIO1_IO08_ENET_QOS_1588_EVENT0_IN

IOMUXC_GPIO1_IO08_PWM1_OUT

IOMUXC_GPIO1_IO08_ISP_PRELIGHT_TRIG_1

IOMUXC_GPIO1_IO08_ENET_QOS_1588_EVENT0_AUX_IN

IOMUXC_GPIO1_IO08_USDHC2_RESET_B

IOMUXC_GPIO1_IO09_GPIO1_IO09

IOMUXC_GPIO1_IO09_ENET_QOS_1588_EVENT0_OUT

IOMUXC_GPIO1_IO09_PWM2_OUT

IOMUXC_GPIO1_IO09_ISP_SHUTTER_OPEN_1

IOMUXC_GPIO1_IO09_USDHC3_RESET_B

IOMUXC_GPIO1_IO09_SDMA2_EXT_EVENT0

IOMUXC_GPIO1_IO10_GPIO1_IO10

IOMUXC_GPIO1_IO10_USB1_ID

IOMUXC_GPIO1_IO10_PWM3_OUT

IOMUXC_GPIO1_IO11_GPIO1_IO11

IOMUXC_GPIO1_IO11_USB2_ID

IOMUXC_GPIO1_IO11_PWM2_OUT

IOMUXC_GPIO1_IO11_USDHC3_VSELECT

IOMUXC_GPIO1_IO11_CCM_PMIC_READY

IOMUXC_GPIO1_IO12_GPIO1_IO12

IOMUXC_GPIO1_IO12_USB1_PWR

IOMUXC_GPIO1_IO12_SDMA2_EXT_EVENT1

IOMUXC_GPIO1_IO13_GPIO1_IO13

IOMUXC_GPIO1_IO13_USB1_OC

IOMUXC_GPIO1_IO13_PWM2_OUT

IOMUXC_GPIO1_IO14_GPIO1_IO14

IOMUXC_GPIO1_IO14_USB2_PWR

IOMUXC_GPIO1_IO14_USDHC3_CD_B

IOMUXC_GPIO1_IO14_PWM3_OUT

IOMUXC_GPIO1_IO14_CCM_CLKO1

IOMUXC_GPIO1_IO15_GPIO1_IO15

IOMUXC_GPIO1_IO15_USB2_OC

IOMUXC_GPIO1_IO15_USDHC3_WP

IOMUXC_GPIO1_IO15_PWM4_OUT

IOMUXC_GPIO1_IO15_CCM_CLKO2

IOMUXC_ENET_MDC_ENET_QOS_MDC

IOMUXC_ENET_MDC_AUDIOMIX_SAI6_TX_DATA0

IOMUXC_ENET_MDC_GPIO1_IO16

IOMUXC_ENET_MDC_USDHC3_STROBE

IOMUXC_ENET_MDIO_ENET_QOS_MDIO

IOMUXC_ENET_MDIO_AUDIOMIX_SAI6_TX_SYNC

IOMUXC_ENET_MDIO_AUDIOMIX_PDM_BIT_STREAM3

IOMUXC_ENET_MDIO_GPIO1_IO17

IOMUXC_ENET_MDIO_USDHC3_DATA5

IOMUXC_ENET_TD3_ENET_QOS_RGMII_TD3

IOMUXC_ENET_TD3_AUDIOMIX_SAI6_TX_BCLK

IOMUXC_ENET_TD3_AUDIOMIX_PDM_BIT_STREAM2

IOMUXC_ENET_TD3_GPIO1_IO18

IOMUXC_ENET_TD3_USDHC3_DATA6

IOMUXC_ENET_TD2_ENET_QOS_RGMII_TD2

IOMUXC_ENET_TD2_CCM_ENET_QOS_CLOCK_GENERATE_REF_CLK

IOMUXC_ENET_TD2_AUDIOMIX_SAI6_RX_DATA0

IOMUXC_ENET_TD2_AUDIOMIX_PDM_BIT_STREAM1

IOMUXC_ENET_TD2_GPIO1_IO19

IOMUXC_ENET_TD2_USDHC3_DATA7

IOMUXC_ENET_TD1_ENET_QOS_RGMII_TD1

IOMUXC_ENET_TD1_AUDIOMIX_SAI6_RX_SYNC

IOMUXC_ENET_TD1_AUDIOMIX_PDM_BIT_STREAM0

IOMUXC_ENET_TD1_GPIO1_IO20

IOMUXC_ENET_TD1_USDHC3_CD_B

IOMUXC_ENET_TD0_ENET_QOS_RGMII_TD0

IOMUXC_ENET_TD0_AUDIOMIX_SAI6_RX_BCLK

IOMUXC_ENET_TD0_AUDIOMIX_PDM_CLK

IOMUXC_ENET_TD0_GPIO1_IO21

IOMUXC_ENET_TD0_USDHC3_WP

IOMUXC_ENET_TX_CTL_ENET_QOS_RGMII_TX_CTL

IOMUXC_ENET_TX_CTL_AUDIOMIX_SAI6_MCLK

IOMUXC_ENET_TX_CTL_AUDIOMIX_SPDIF1_OUT

IOMUXC_ENET_TX_CTL_GPIO1_IO22

IOMUXC_ENET_TX_CTL_USDHC3_DATA0

IOMUXC_ENET_TXC_CCM_ENET_QOS_CLOCK_GENERATE_TX_CLK

IOMUXC_ENET_TXC_ENET_QOS_TX_ER

IOMUXC_ENET_TXC_AUDIOMIX_SAI7_TX_DATA0

IOMUXC_ENET_TXC_GPIO1_IO23

IOMUXC_ENET_TXC_USDHC3_DATA1

IOMUXC_ENET_RX_CTL_ENET_QOS_RGMII_RX_CTL

IOMUXC_ENET_RX_CTL_AUDIOMIX_SAI7_TX_SYNC

IOMUXC_ENET_RX_CTL_AUDIOMIX_PDM_BIT_STREAM3

IOMUXC_ENET_RX_CTL_GPIO1_IO24

IOMUXC_ENET_RX_CTL_USDHC3_DATA2

IOMUXC_ENET_RXC_CCM_ENET_QOS_CLOCK_GENERATE_RX_CLK

IOMUXC_ENET_RXC_ENET_QOS_RX_ER

IOMUXC_ENET_RXC_AUDIOMIX_SAI7_TX_BCLK

IOMUXC_ENET_RXC_AUDIOMIX_PDM_BIT_STREAM2

IOMUXC_ENET_RXC_GPIO1_IO25

IOMUXC_ENET_RXC_USDHC3_DATA3

IOMUXC_ENET_RD0_ENET_QOS_RGMII_RD0

IOMUXC_ENET_RD0_AUDIOMIX_SAI7_RX_DATA0

IOMUXC_ENET_RD0_AUDIOMIX_PDM_BIT_STREAM1

IOMUXC_ENET_RD0_GPIO1_IO26

IOMUXC_ENET_RD0_USDHC3_DATA4

IOMUXC_ENET_RD1_ENET_QOS_RGMII_RD1

IOMUXC_ENET_RD1_AUDIOMIX_SAI7_RX_SYNC

IOMUXC_ENET_RD1_AUDIOMIX_PDM_BIT_STREAM0

IOMUXC_ENET_RD1_GPIO1_IO27

IOMUXC_ENET_RD1_USDHC3_RESET_B

IOMUXC_ENET_RD2_ENET_QOS_RGMII_RD2

IOMUXC_ENET_RD2_AUDIOMIX_SAI7_RX_BCLK

IOMUXC_ENET_RD2_AUDIOMIX_PDM_CLK

IOMUXC_ENET_RD2_GPIO1_IO28

IOMUXC_ENET_RD2_USDHC3_CLK

IOMUXC_ENET_RD3_ENET_QOS_RGMII_RD3

IOMUXC_ENET_RD3_AUDIOMIX_SAI7_MCLK

IOMUXC_ENET_RD3_AUDIOMIX_SPDIF1_IN

IOMUXC_ENET_RD3_GPIO1_IO29

IOMUXC_ENET_RD3_USDHC3_CMD

IOMUXC_SD1_CLK_USDHC1_CLK

IOMUXC_SD1_CLK_ENET1_MDC

IOMUXC_SD1_CLK_I2C5_SCL

IOMUXC_SD1_CLK_UART1_TX

IOMUXC_SD1_CLK_UART1_RX

IOMUXC_SD1_CLK_GPIO2_IO00

IOMUXC_SD1_CMD_USDHC1_CMD

IOMUXC_SD1_CMD_ENET1_MDIO

IOMUXC_SD1_CMD_I2C5_SDA

IOMUXC_SD1_CMD_UART1_RX

IOMUXC_SD1_CMD_UART1_TX

IOMUXC_SD1_CMD_GPIO2_IO01

IOMUXC_SD1_DATA0_USDHC1_DATA0

IOMUXC_SD1_DATA0_ENET1_RGMII_TD1

IOMUXC_SD1_DATA0_I2C6_SCL

IOMUXC_SD1_DATA0_UART1_RTS_B

IOMUXC_SD1_DATA0_UART1_CTS_B

IOMUXC_SD1_DATA0_GPIO2_IO02

IOMUXC_SD1_DATA1_USDHC1_DATA1

IOMUXC_SD1_DATA1_ENET1_RGMII_TD0

IOMUXC_SD1_DATA1_I2C6_SDA

IOMUXC_SD1_DATA1_UART1_CTS_B

IOMUXC_SD1_DATA1_UART1_RTS_B

IOMUXC_SD1_DATA1_GPIO2_IO03

IOMUXC_SD1_DATA2_USDHC1_DATA2

IOMUXC_SD1_DATA2_ENET1_RGMII_RD0

IOMUXC_SD1_DATA2_I2C4_SCL

IOMUXC_SD1_DATA2_UART2_TX

IOMUXC_SD1_DATA2_UART2_RX

IOMUXC_SD1_DATA2_GPIO2_IO04

IOMUXC_SD1_DATA3_USDHC1_DATA3

IOMUXC_SD1_DATA3_ENET1_RGMII_RD1

IOMUXC_SD1_DATA3_I2C4_SDA

IOMUXC_SD1_DATA3_UART2_RX

IOMUXC_SD1_DATA3_UART2_TX

IOMUXC_SD1_DATA3_GPIO2_IO05

IOMUXC_SD1_DATA4_USDHC1_DATA4

IOMUXC_SD1_DATA4_ENET1_RGMII_TX_CTL

IOMUXC_SD1_DATA4_I2C1_SCL

IOMUXC_SD1_DATA4_UART2_RTS_B

IOMUXC_SD1_DATA4_UART2_CTS_B

IOMUXC_SD1_DATA4_GPIO2_IO06

IOMUXC_SD1_DATA5_USDHC1_DATA5

IOMUXC_SD1_DATA5_ENET1_TX_ER

IOMUXC_SD1_DATA5_I2C1_SDA

IOMUXC_SD1_DATA5_UART2_CTS_B

IOMUXC_SD1_DATA5_UART2_RTS_B

IOMUXC_SD1_DATA5_GPIO2_IO07

IOMUXC_SD1_DATA6_USDHC1_DATA6

IOMUXC_SD1_DATA6_ENET1_RGMII_RX_CTL

IOMUXC_SD1_DATA6_I2C2_SCL

IOMUXC_SD1_DATA6_UART3_TX

IOMUXC_SD1_DATA6_UART3_RX

IOMUXC_SD1_DATA6_GPIO2_IO08

IOMUXC_SD1_DATA7_USDHC1_DATA7

IOMUXC_SD1_DATA7_ENET1_RX_ER

IOMUXC_SD1_DATA7_I2C2_SDA

IOMUXC_SD1_DATA7_UART3_RX

IOMUXC_SD1_DATA7_UART3_TX

IOMUXC_SD1_DATA7_GPIO2_IO09

IOMUXC_SD1_RESET_B_USDHC1_RESET_B

IOMUXC_SD1_RESET_B_ENET1_TX_CLK

IOMUXC_SD1_RESET_B_I2C3_SCL

IOMUXC_SD1_RESET_B_UART3_RTS_B

IOMUXC_SD1_RESET_B_UART3_CTS_B

IOMUXC_SD1_RESET_B_GPIO2_IO10

IOMUXC_SD1_STROBE_USDHC1_STROBE

IOMUXC_SD1_STROBE_I2C3_SDA

IOMUXC_SD1_STROBE_UART3_CTS_B

IOMUXC_SD1_STROBE_UART3_RTS_B

IOMUXC_SD1_STROBE_GPIO2_IO11

IOMUXC_SD2_CD_B_USDHC2_CD_B

IOMUXC_SD2_CD_B_GPIO2_IO12

IOMUXC_SD2_CLK_USDHC2_CLK

IOMUXC_SD2_CLK_ECSPI2_SCLK

IOMUXC_SD2_CLK_UART4_RX

IOMUXC_SD2_CLK_UART4_TX

IOMUXC_SD2_CLK_GPIO2_IO13

IOMUXC_SD2_CMD_USDHC2_CMD

IOMUXC_SD2_CMD_ECSPI2_MOSI

IOMUXC_SD2_CMD_UART4_TX

IOMUXC_SD2_CMD_UART4_RX

IOMUXC_SD2_CMD_AUDIOMIX_PDM_CLK

IOMUXC_SD2_CMD_GPIO2_IO14

IOMUXC_SD2_DATA0_USDHC2_DATA0

IOMUXC_SD2_DATA0_I2C4_SDA

IOMUXC_SD2_DATA0_UART2_RX

IOMUXC_SD2_DATA0_UART2_TX

IOMUXC_SD2_DATA0_AUDIOMIX_PDM_BIT_STREAM0

IOMUXC_SD2_DATA0_GPIO2_IO15

IOMUXC_SD2_DATA1_USDHC2_DATA1

IOMUXC_SD2_DATA1_I2C4_SCL

IOMUXC_SD2_DATA1_UART2_TX

IOMUXC_SD2_DATA1_UART2_RX

IOMUXC_SD2_DATA1_AUDIOMIX_PDM_BIT_STREAM1

IOMUXC_SD2_DATA1_GPIO2_IO16

IOMUXC_SD2_DATA2_USDHC2_DATA2

IOMUXC_SD2_DATA2_ECSPI2_SS0

IOMUXC_SD2_DATA2_AUDIOMIX_SPDIF1_OUT

IOMUXC_SD2_DATA2_AUDIOMIX_PDM_BIT_STREAM2

IOMUXC_SD2_DATA2_GPIO2_IO17

IOMUXC_SD2_DATA3_USDHC2_DATA3

IOMUXC_SD2_DATA3_ECSPI2_MISO

IOMUXC_SD2_DATA3_AUDIOMIX_SPDIF1_IN

IOMUXC_SD2_DATA3_AUDIOMIX_PDM_BIT_STREAM3

IOMUXC_SD2_DATA3_GPIO2_IO18

IOMUXC_SD2_RESET_B_USDHC2_RESET_B

IOMUXC_SD2_RESET_B_GPIO2_IO19

IOMUXC_SD2_WP_USDHC2_WP

IOMUXC_SD2_WP_GPIO2_IO20

IOMUXC_SD2_WP_CORESIGHT_EVENTI

IOMUXC_NAND_ALE_NAND_ALE

IOMUXC_NAND_ALE_FLEXSPI_A_SCLK

IOMUXC_NAND_ALE_AUDIOMIX_SAI3_TX_BCLK

IOMUXC_NAND_ALE_ISP_FL_TRIG_0

IOMUXC_NAND_ALE_UART3_RX

IOMUXC_NAND_ALE_UART3_TX

IOMUXC_NAND_ALE_GPIO3_IO00

IOMUXC_NAND_ALE_CORESIGHT_TRACE_CLK

IOMUXC_NAND_CE0_B_NAND_CE0_B

IOMUXC_NAND_CE0_B_FLEXSPI_A_SS0_B

IOMUXC_NAND_CE0_B_AUDIOMIX_SAI3_TX_DATA0

IOMUXC_NAND_CE0_B_ISP_SHUTTER_TRIG_0

IOMUXC_NAND_CE0_B_UART3_TX

IOMUXC_NAND_CE0_B_UART3_RX

IOMUXC_NAND_CE0_B_GPIO3_IO01

IOMUXC_NAND_CE0_B_CORESIGHT_TRACE_CTL

IOMUXC_NAND_CE1_B_NAND_CE1_B

IOMUXC_NAND_CE1_B_FLEXSPI_A_SS1_B

IOMUXC_NAND_CE1_B_USDHC3_STROBE

IOMUXC_NAND_CE1_B_I2C4_SCL

IOMUXC_NAND_CE1_B_GPIO3_IO02

IOMUXC_NAND_CE1_B_CORESIGHT_TRACE00

IOMUXC_NAND_CE2_B_NAND_CE2_B

IOMUXC_NAND_CE2_B_FLEXSPI_B_SS0_B

IOMUXC_NAND_CE2_B_USDHC3_DATA5

IOMUXC_NAND_CE2_B_I2C4_SDA

IOMUXC_NAND_CE2_B_GPIO3_IO03

IOMUXC_NAND_CE2_B_CORESIGHT_TRACE01

IOMUXC_NAND_CE3_B_NAND_CE3_B

IOMUXC_NAND_CE3_B_FLEXSPI_B_SS1_B

IOMUXC_NAND_CE3_B_USDHC3_DATA6

IOMUXC_NAND_CE3_B_I2C3_SDA

IOMUXC_NAND_CE3_B_GPIO3_IO04

IOMUXC_NAND_CE3_B_CORESIGHT_TRACE02

IOMUXC_NAND_CLE_NAND_CLE

IOMUXC_NAND_CLE_FLEXSPI_B_SCLK

IOMUXC_NAND_CLE_USDHC3_DATA7

IOMUXC_NAND_CLE_UART4_RX

IOMUXC_NAND_CLE_UART4_TX

IOMUXC_NAND_CLE_GPIO3_IO05

IOMUXC_NAND_CLE_CORESIGHT_TRACE03

IOMUXC_NAND_DATA00_NAND_DATA00

IOMUXC_NAND_DATA00_FLEXSPI_A_DATA0

IOMUXC_NAND_DATA00_AUDIOMIX_SAI3_RX_DATA0

IOMUXC_NAND_DATA00_ISP_FLASH_TRIG_0

IOMUXC_NAND_DATA00_UART4_RX

IOMUXC_NAND_DATA00_UART4_TX

IOMUXC_NAND_DATA00_GPIO3_IO06

IOMUXC_NAND_DATA00_CORESIGHT_TRACE04

IOMUXC_NAND_DATA01_NAND_DATA01

IOMUXC_NAND_DATA01_FLEXSPI_A_DATA1

IOMUXC_NAND_DATA01_AUDIOMIX_SAI3_TX_SYNC

IOMUXC_NAND_DATA01_ISP_PRELIGHT_TRIG_0

IOMUXC_NAND_DATA01_UART4_TX

IOMUXC_NAND_DATA01_UART4_RX

IOMUXC_NAND_DATA01_GPIO3_IO07

IOMUXC_NAND_DATA01_CORESIGHT_TRACE05

IOMUXC_NAND_DATA02_NAND_DATA02

IOMUXC_NAND_DATA02_FLEXSPI_A_DATA2

IOMUXC_NAND_DATA02_USDHC3_CD_B

IOMUXC_NAND_DATA02_UART4_CTS_B

IOMUXC_NAND_DATA02_UART4_RTS_B

IOMUXC_NAND_DATA02_I2C4_SDA

IOMUXC_NAND_DATA02_GPIO3_IO08

IOMUXC_NAND_DATA02_CORESIGHT_TRACE06

IOMUXC_NAND_DATA03_NAND_DATA03

IOMUXC_NAND_DATA03_FLEXSPI_A_DATA3

IOMUXC_NAND_DATA03_USDHC3_WP

IOMUXC_NAND_DATA03_UART4_RTS_B

IOMUXC_NAND_DATA03_UART4_CTS_B

IOMUXC_NAND_DATA03_ISP_FL_TRIG_1

IOMUXC_NAND_DATA03_GPIO3_IO09

IOMUXC_NAND_DATA03_CORESIGHT_TRACE07

IOMUXC_NAND_DATA04_NAND_DATA04

IOMUXC_NAND_DATA04_FLEXSPI_B_DATA0

IOMUXC_NAND_DATA04_USDHC3_DATA0

IOMUXC_NAND_DATA04_FLEXSPI_A_DATA4

IOMUXC_NAND_DATA04_ISP_SHUTTER_TRIG_1

IOMUXC_NAND_DATA04_GPIO3_IO10

IOMUXC_NAND_DATA04_CORESIGHT_TRACE08

IOMUXC_NAND_DATA05_NAND_DATA05

IOMUXC_NAND_DATA05_FLEXSPI_B_DATA1

IOMUXC_NAND_DATA05_USDHC3_DATA1

IOMUXC_NAND_DATA05_FLEXSPI_A_DATA5

IOMUXC_NAND_DATA05_ISP_FLASH_TRIG_1

IOMUXC_NAND_DATA05_GPIO3_IO11

IOMUXC_NAND_DATA05_CORESIGHT_TRACE09

IOMUXC_NAND_DATA06_NAND_DATA06

IOMUXC_NAND_DATA06_FLEXSPI_B_DATA2

IOMUXC_NAND_DATA06_USDHC3_DATA2

IOMUXC_NAND_DATA06_FLEXSPI_A_DATA6

IOMUXC_NAND_DATA06_ISP_PRELIGHT_TRIG_1

IOMUXC_NAND_DATA06_GPIO3_IO12

IOMUXC_NAND_DATA06_CORESIGHT_TRACE10

IOMUXC_NAND_DATA07_NAND_DATA07

IOMUXC_NAND_DATA07_FLEXSPI_B_DATA3

IOMUXC_NAND_DATA07_USDHC3_DATA3

IOMUXC_NAND_DATA07_FLEXSPI_A_DATA7

IOMUXC_NAND_DATA07_ISP_SHUTTER_OPEN_1

IOMUXC_NAND_DATA07_GPIO3_IO13

IOMUXC_NAND_DATA07_CORESIGHT_TRACE11

IOMUXC_NAND_DQS_NAND_DQS

IOMUXC_NAND_DQS_FLEXSPI_A_DQS

IOMUXC_NAND_DQS_AUDIOMIX_SAI3_MCLK

IOMUXC_NAND_DQS_ISP_SHUTTER_OPEN_0

IOMUXC_NAND_DQS_I2C3_SCL

IOMUXC_NAND_DQS_GPIO3_IO14

IOMUXC_NAND_DQS_CORESIGHT_TRACE12

IOMUXC_NAND_RE_B_NAND_RE_B

IOMUXC_NAND_RE_B_FLEXSPI_B_DQS

IOMUXC_NAND_RE_B_USDHC3_DATA4

IOMUXC_NAND_RE_B_UART4_TX

IOMUXC_NAND_RE_B_UART4_RX

IOMUXC_NAND_RE_B_GPIO3_IO15

IOMUXC_NAND_RE_B_CORESIGHT_TRACE13

IOMUXC_NAND_READY_B_NAND_READY_B

IOMUXC_NAND_READY_B_USDHC3_RESET_B

IOMUXC_NAND_READY_B_I2C3_SCL

IOMUXC_NAND_READY_B_GPIO3_IO16

IOMUXC_NAND_READY_B_CORESIGHT_TRACE14

IOMUXC_NAND_WE_B_NAND_WE_B

IOMUXC_NAND_WE_B_USDHC3_CLK

IOMUXC_NAND_WE_B_I2C3_SDA

IOMUXC_NAND_WE_B_GPIO3_IO17

IOMUXC_NAND_WE_B_CORESIGHT_TRACE15

IOMUXC_NAND_WP_B_NAND_WP_B

IOMUXC_NAND_WP_B_USDHC3_CMD

IOMUXC_NAND_WP_B_I2C4_SCL

IOMUXC_NAND_WP_B_GPIO3_IO18

IOMUXC_NAND_WP_B_CORESIGHT_EVENTO

IOMUXC_SAI5_RXFS_AUDIOMIX_SAI5_RX_SYNC

IOMUXC_SAI5_RXFS_AUDIOMIX_SAI1_TX_DATA0

IOMUXC_SAI5_RXFS_PWM4_OUT

IOMUXC_SAI5_RXFS_I2C6_SCL

IOMUXC_SAI5_RXFS_GPIO3_IO19

IOMUXC_SAI5_RXC_AUDIOMIX_SAI5_RX_BCLK

IOMUXC_SAI5_RXC_AUDIOMIX_SAI1_TX_DATA1

IOMUXC_SAI5_RXC_PWM3_OUT

IOMUXC_SAI5_RXC_I2C6_SDA

IOMUXC_SAI5_RXC_AUDIOMIX_PDM_CLK

IOMUXC_SAI5_RXC_GPIO3_IO20

IOMUXC_SAI5_RXD0_AUDIOMIX_SAI5_RX_DATA0

IOMUXC_SAI5_RXD0_AUDIOMIX_SAI1_TX_DATA2

IOMUXC_SAI5_RXD0_PWM2_OUT

IOMUXC_SAI5_RXD0_I2C5_SCL

IOMUXC_SAI5_RXD0_AUDIOMIX_PDM_BIT_STREAM0

IOMUXC_SAI5_RXD0_GPIO3_IO21

IOMUXC_SAI5_RXD1_AUDIOMIX_SAI5_RX_DATA1

IOMUXC_SAI5_RXD1_AUDIOMIX_SAI1_TX_DATA3

IOMUXC_SAI5_RXD1_AUDIOMIX_SAI1_TX_SYNC

IOMUXC_SAI5_RXD1_AUDIOMIX_SAI5_TX_SYNC

IOMUXC_SAI5_RXD1_AUDIOMIX_PDM_BIT_STREAM1

IOMUXC_SAI5_RXD1_GPIO3_IO22

IOMUXC_SAI5_RXD1_CAN1_TX

IOMUXC_SAI5_RXD2_AUDIOMIX_SAI5_RX_DATA2

IOMUXC_SAI5_RXD2_AUDIOMIX_SAI1_TX_DATA4

IOMUXC_SAI5_RXD2_AUDIOMIX_SAI1_TX_SYNC

IOMUXC_SAI5_RXD2_AUDIOMIX_SAI5_TX_BCLK

IOMUXC_SAI5_RXD2_AUDIOMIX_PDM_BIT_STREAM2

IOMUXC_SAI5_RXD2_GPIO3_IO23

IOMUXC_SAI5_RXD2_CAN1_RX

IOMUXC_SAI5_RXD3_AUDIOMIX_SAI5_RX_DATA3

IOMUXC_SAI5_RXD3_AUDIOMIX_SAI1_TX_DATA5

IOMUXC_SAI5_RXD3_AUDIOMIX_SAI1_TX_SYNC

IOMUXC_SAI5_RXD3_AUDIOMIX_SAI5_TX_DATA0

IOMUXC_SAI5_RXD3_AUDIOMIX_PDM_BIT_STREAM3

IOMUXC_SAI5_RXD3_GPIO3_IO24

IOMUXC_SAI5_RXD3_CAN2_TX

IOMUXC_SAI5_MCLK_AUDIOMIX_SAI5_MCLK

IOMUXC_SAI5_MCLK_AUDIOMIX_SAI1_TX_BCLK

IOMUXC_SAI5_MCLK_PWM1_OUT

IOMUXC_SAI5_MCLK_I2C5_SDA

IOMUXC_SAI5_MCLK_GPIO3_IO25

IOMUXC_SAI5_MCLK_CAN2_RX

IOMUXC_SAI1_RXFS_AUDIOMIX_SAI1_RX_SYNC

IOMUXC_SAI1_RXFS_ENET1_1588_EVENT0_IN

IOMUXC_SAI1_RXFS_GPIO4_IO00

IOMUXC_SAI1_RXC_AUDIOMIX_SAI1_RX_BCLK

IOMUXC_SAI1_RXC_AUDIOMIX_PDM_CLK

IOMUXC_SAI1_RXC_ENET1_1588_EVENT0_OUT

IOMUXC_SAI1_RXC_GPIO4_IO01

IOMUXC_SAI1_RXD0_AUDIOMIX_SAI1_RX_DATA0

IOMUXC_SAI1_RXD0_AUDIOMIX_SAI1_TX_DATA1

IOMUXC_SAI1_RXD0_AUDIOMIX_PDM_BIT_STREAM0

IOMUXC_SAI1_RXD0_ENET1_1588_EVENT1_IN

IOMUXC_SAI1_RXD0_GPIO4_IO02

IOMUXC_SAI1_RXD1_AUDIOMIX_SAI1_RX_DATA1

IOMUXC_SAI1_RXD1_AUDIOMIX_PDM_BIT_STREAM1

IOMUXC_SAI1_RXD1_ENET1_1588_EVENT1_OUT

IOMUXC_SAI1_RXD1_GPIO4_IO03

IOMUXC_SAI1_RXD2_AUDIOMIX_SAI1_RX_DATA2

IOMUXC_SAI1_RXD2_AUDIOMIX_PDM_BIT_STREAM2

IOMUXC_SAI1_RXD2_ENET1_MDC

IOMUXC_SAI1_RXD2_GPIO4_IO04

IOMUXC_SAI1_RXD3_AUDIOMIX_SAI1_RX_DATA3

IOMUXC_SAI1_RXD3_AUDIOMIX_PDM_BIT_STREAM3

IOMUXC_SAI1_RXD3_ENET1_MDIO

IOMUXC_SAI1_RXD3_GPIO4_IO05

IOMUXC_SAI1_RXD4_AUDIOMIX_SAI1_RX_DATA4

IOMUXC_SAI1_RXD4_AUDIOMIX_SAI6_TX_BCLK

IOMUXC_SAI1_RXD4_AUDIOMIX_SAI6_RX_BCLK

IOMUXC_SAI1_RXD4_ENET1_RGMII_RD0

IOMUXC_SAI1_RXD4_GPIO4_IO06

IOMUXC_SAI1_RXD5_AUDIOMIX_SAI1_RX_DATA5

IOMUXC_SAI1_RXD5_AUDIOMIX_SAI6_TX_DATA0

IOMUXC_SAI1_RXD5_AUDIOMIX_SAI6_RX_DATA0

IOMUXC_SAI1_RXD5_AUDIOMIX_SAI1_RX_SYNC

IOMUXC_SAI1_RXD5_ENET1_RGMII_RD1

IOMUXC_SAI1_RXD5_GPIO4_IO07

IOMUXC_SAI1_RXD6_AUDIOMIX_SAI1_RX_DATA6

IOMUXC_SAI1_RXD6_AUDIOMIX_SAI6_TX_SYNC

IOMUXC_SAI1_RXD6_AUDIOMIX_SAI6_RX_SYNC

IOMUXC_SAI1_RXD6_ENET1_RGMII_RD2

IOMUXC_SAI1_RXD6_GPIO4_IO08

IOMUXC_SAI1_RXD7_AUDIOMIX_SAI1_RX_DATA7

IOMUXC_SAI1_RXD7_AUDIOMIX_SAI6_MCLK

IOMUXC_SAI1_RXD7_AUDIOMIX_SAI1_TX_SYNC

IOMUXC_SAI1_RXD7_AUDIOMIX_SAI1_TX_DATA4

IOMUXC_SAI1_RXD7_ENET1_RGMII_RD3

IOMUXC_SAI1_RXD7_GPIO4_IO09

IOMUXC_SAI1_TXFS_AUDIOMIX_SAI1_TX_SYNC

IOMUXC_SAI1_TXFS_ENET1_RGMII_RX_CTL

IOMUXC_SAI1_TXFS_GPIO4_IO10

IOMUXC_SAI1_TXC_AUDIOMIX_SAI1_TX_BCLK

IOMUXC_SAI1_TXC_ENET1_RGMII_RXC

IOMUXC_SAI1_TXC_GPIO4_IO11

IOMUXC_SAI1_TXD0_AUDIOMIX_SAI1_TX_DATA0

IOMUXC_SAI1_TXD0_ENET1_RGMII_TD0

IOMUXC_SAI1_TXD0_GPIO4_IO12

IOMUXC_SAI1_TXD1_AUDIOMIX_SAI1_TX_DATA1

IOMUXC_SAI1_TXD1_ENET1_RGMII_TD1

IOMUXC_SAI1_TXD1_GPIO4_IO13

IOMUXC_SAI1_TXD2_AUDIOMIX_SAI1_TX_DATA2

IOMUXC_SAI1_TXD2_ENET1_RGMII_TD2

IOMUXC_SAI1_TXD2_GPIO4_IO14

IOMUXC_SAI1_TXD3_AUDIOMIX_SAI1_TX_DATA3

IOMUXC_SAI1_TXD3_ENET1_RGMII_TD3

IOMUXC_SAI1_TXD3_GPIO4_IO15

IOMUXC_SAI1_TXD4_AUDIOMIX_SAI1_TX_DATA4

IOMUXC_SAI1_TXD4_AUDIOMIX_SAI6_RX_BCLK

IOMUXC_SAI1_TXD4_AUDIOMIX_SAI6_TX_BCLK

IOMUXC_SAI1_TXD4_ENET1_RGMII_TX_CTL

IOMUXC_SAI1_TXD4_GPIO4_IO16

IOMUXC_SAI1_TXD5_AUDIOMIX_SAI1_TX_DATA5

IOMUXC_SAI1_TXD5_AUDIOMIX_SAI6_RX_DATA0

IOMUXC_SAI1_TXD5_AUDIOMIX_SAI6_TX_DATA0

IOMUXC_SAI1_TXD5_ENET1_RGMII_TXC

IOMUXC_SAI1_TXD5_GPIO4_IO17

IOMUXC_SAI1_TXD6_AUDIOMIX_SAI1_TX_DATA6

IOMUXC_SAI1_TXD6_AUDIOMIX_SAI6_RX_SYNC

IOMUXC_SAI1_TXD6_AUDIOMIX_SAI6_TX_SYNC

IOMUXC_SAI1_TXD6_ENET1_RX_ER

IOMUXC_SAI1_TXD6_GPIO4_IO18

IOMUXC_SAI1_TXD7_AUDIOMIX_SAI1_TX_DATA7

IOMUXC_SAI1_TXD7_AUDIOMIX_SAI6_MCLK

IOMUXC_SAI1_TXD7_AUDIOMIX_PDM_CLK

IOMUXC_SAI1_TXD7_ENET1_TX_ER

IOMUXC_SAI1_TXD7_GPIO4_IO19

IOMUXC_SAI1_MCLK_AUDIOMIX_SAI1_MCLK

IOMUXC_SAI1_MCLK_AUDIOMIX_SAI1_TX_BCLK

IOMUXC_SAI1_MCLK_ENET1_TX_CLK

IOMUXC_SAI1_MCLK_GPIO4_IO20

IOMUXC_SAI2_RXFS_AUDIOMIX_SAI2_RX_SYNC

IOMUXC_SAI2_RXFS_AUDIOMIX_SAI5_TX_SYNC

IOMUXC_SAI2_RXFS_AUDIOMIX_SAI5_TX_DATA1

IOMUXC_SAI2_RXFS_AUDIOMIX_SAI2_RX_DATA1

IOMUXC_SAI2_RXFS_UART1_TX

IOMUXC_SAI2_RXFS_UART1_RX

IOMUXC_SAI2_RXFS_GPIO4_IO21

IOMUXC_SAI2_RXFS_AUDIOMIX_PDM_BIT_STREAM2

IOMUXC_SAI2_RXC_AUDIOMIX_SAI2_RX_BCLK

IOMUXC_SAI2_RXC_AUDIOMIX_SAI5_TX_BCLK

IOMUXC_SAI2_RXC_CAN1_TX

IOMUXC_SAI2_RXC_UART1_RX

IOMUXC_SAI2_RXC_UART1_TX

IOMUXC_SAI2_RXC_GPIO4_IO22

IOMUXC_SAI2_RXC_AUDIOMIX_PDM_BIT_STREAM1

IOMUXC_SAI2_RXD0_AUDIOMIX_SAI2_RX_DATA0

IOMUXC_SAI2_RXD0_AUDIOMIX_SAI5_TX_DATA0

IOMUXC_SAI2_RXD0_ENET_QOS_1588_EVENT2_OUT

IOMUXC_SAI2_RXD0_AUDIOMIX_SAI2_TX_DATA1

IOMUXC_SAI2_RXD0_UART1_RTS_B

IOMUXC_SAI2_RXD0_UART1_CTS_B

IOMUXC_SAI2_RXD0_GPIO4_IO23

IOMUXC_SAI2_RXD0_AUDIOMIX_PDM_BIT_STREAM3

IOMUXC_SAI2_TXFS_AUDIOMIX_SAI2_TX_SYNC

IOMUXC_SAI2_TXFS_AUDIOMIX_SAI5_TX_DATA1

IOMUXC_SAI2_TXFS_ENET_QOS_1588_EVENT3_OUT

IOMUXC_SAI2_TXFS_AUDIOMIX_SAI2_TX_DATA1

IOMUXC_SAI2_TXFS_UART1_CTS_B

IOMUXC_SAI2_TXFS_UART1_RTS_B

IOMUXC_SAI2_TXFS_GPIO4_IO24

IOMUXC_SAI2_TXFS_AUDIOMIX_PDM_BIT_STREAM2

IOMUXC_SAI2_TXC_AUDIOMIX_SAI2_TX_BCLK

IOMUXC_SAI2_TXC_AUDIOMIX_SAI5_TX_DATA2

IOMUXC_SAI2_TXC_CAN1_RX

IOMUXC_SAI2_TXC_GPIO4_IO25

IOMUXC_SAI2_TXC_AUDIOMIX_PDM_BIT_STREAM1

IOMUXC_SAI2_TXD0_AUDIOMIX_SAI2_TX_DATA0

IOMUXC_SAI2_TXD0_AUDIOMIX_SAI5_TX_DATA3

IOMUXC_SAI2_TXD0_ENET_QOS_1588_EVENT2_IN

IOMUXC_SAI2_TXD0_CAN2_TX

IOMUXC_SAI2_TXD0_ENET_QOS_1588_EVENT2_AUX_IN

IOMUXC_SAI2_TXD0_GPIO4_IO26

IOMUXC_SAI2_MCLK_AUDIOMIX_SAI2_MCLK

IOMUXC_SAI2_MCLK_AUDIOMIX_SAI5_MCLK

IOMUXC_SAI2_MCLK_ENET_QOS_1588_EVENT3_IN

IOMUXC_SAI2_MCLK_CAN2_RX

IOMUXC_SAI2_MCLK_ENET_QOS_1588_EVENT3_AUX_IN

IOMUXC_SAI2_MCLK_GPIO4_IO27

IOMUXC_SAI2_MCLK_AUDIOMIX_SAI3_MCLK

IOMUXC_SAI3_RXFS_AUDIOMIX_SAI3_RX_SYNC

IOMUXC_SAI3_RXFS_AUDIOMIX_SAI2_RX_DATA1

IOMUXC_SAI3_RXFS_AUDIOMIX_SAI5_RX_SYNC

IOMUXC_SAI3_RXFS_AUDIOMIX_SAI3_RX_DATA1

IOMUXC_SAI3_RXFS_AUDIOMIX_SPDIF1_IN

IOMUXC_SAI3_RXFS_GPIO4_IO28

IOMUXC_SAI3_RXFS_AUDIOMIX_PDM_BIT_STREAM0

IOMUXC_SAI3_RXC_AUDIOMIX_SAI3_RX_BCLK

IOMUXC_SAI3_RXC_AUDIOMIX_SAI2_RX_DATA2

IOMUXC_SAI3_RXC_AUDIOMIX_SAI5_RX_BCLK

IOMUXC_SAI3_RXC_GPT1_CLK

IOMUXC_SAI3_RXC_UART2_CTS_B

IOMUXC_SAI3_RXC_UART2_RTS_B

IOMUXC_SAI3_RXC_GPIO4_IO29

IOMUXC_SAI3_RXC_AUDIOMIX_PDM_CLK

IOMUXC_SAI3_RXD_AUDIOMIX_SAI3_RX_DATA0

IOMUXC_SAI3_RXD_AUDIOMIX_SAI2_RX_DATA3

IOMUXC_SAI3_RXD_AUDIOMIX_SAI5_RX_DATA0

IOMUXC_SAI3_RXD_UART2_RTS_B

IOMUXC_SAI3_RXD_UART2_CTS_B

IOMUXC_SAI3_RXD_GPIO4_IO30

IOMUXC_SAI3_RXD_AUDIOMIX_PDM_BIT_STREAM1

IOMUXC_SAI3_TXFS_AUDIOMIX_SAI3_TX_SYNC

IOMUXC_SAI3_TXFS_AUDIOMIX_SAI2_TX_DATA1

IOMUXC_SAI3_TXFS_AUDIOMIX_SAI5_RX_DATA1

IOMUXC_SAI3_TXFS_AUDIOMIX_SAI3_TX_DATA1

IOMUXC_SAI3_TXFS_UART2_RX

IOMUXC_SAI3_TXFS_UART2_TX

IOMUXC_SAI3_TXFS_GPIO4_IO31

IOMUXC_SAI3_TXFS_AUDIOMIX_PDM_BIT_STREAM3

IOMUXC_SAI3_TXC_AUDIOMIX_SAI3_TX_BCLK

IOMUXC_SAI3_TXC_AUDIOMIX_SAI2_TX_DATA2

IOMUXC_SAI3_TXC_AUDIOMIX_SAI5_RX_DATA2

IOMUXC_SAI3_TXC_GPT1_CAPTURE1

IOMUXC_SAI3_TXC_UART2_TX

IOMUXC_SAI3_TXC_UART2_RX

IOMUXC_SAI3_TXC_GPIO5_IO00

IOMUXC_SAI3_TXC_AUDIOMIX_PDM_BIT_STREAM2

IOMUXC_SAI3_TXD_AUDIOMIX_SAI3_TX_DATA0

IOMUXC_SAI3_TXD_AUDIOMIX_SAI2_TX_DATA3

IOMUXC_SAI3_TXD_AUDIOMIX_SAI5_RX_DATA3

IOMUXC_SAI3_TXD_GPT1_CAPTURE2

IOMUXC_SAI3_TXD_AUDIOMIX_SPDIF1_EXT_CLK

IOMUXC_SAI3_TXD_GPIO5_IO01

IOMUXC_SAI3_MCLK_AUDIOMIX_SAI3_MCLK

IOMUXC_SAI3_MCLK_PWM4_OUT

IOMUXC_SAI3_MCLK_AUDIOMIX_SAI5_MCLK

IOMUXC_SAI3_MCLK_AUDIOMIX_SPDIF1_OUT

IOMUXC_SAI3_MCLK_GPIO5_IO02

IOMUXC_SAI3_MCLK_AUDIOMIX_SPDIF1_IN

IOMUXC_SPDIF_TX_AUDIOMIX_SPDIF1_OUT

IOMUXC_SPDIF_TX_PWM3_OUT

IOMUXC_SPDIF_TX_I2C5_SCL

IOMUXC_SPDIF_TX_GPT1_COMPARE1

IOMUXC_SPDIF_TX_CAN1_TX

IOMUXC_SPDIF_TX_GPIO5_IO03

IOMUXC_SPDIF_RX_AUDIOMIX_SPDIF1_IN

IOMUXC_SPDIF_RX_PWM2_OUT

IOMUXC_SPDIF_RX_I2C5_SDA

IOMUXC_SPDIF_RX_GPT1_COMPARE2

IOMUXC_SPDIF_RX_CAN1_RX

IOMUXC_SPDIF_RX_GPIO5_IO04

IOMUXC_SPDIF_EXT_CLK_AUDIOMIX_SPDIF1_EXT_CLK

IOMUXC_SPDIF_EXT_CLK_PWM1_OUT

IOMUXC_SPDIF_EXT_CLK_GPT1_COMPARE3

IOMUXC_SPDIF_EXT_CLK_GPIO5_IO05

IOMUXC_ECSPI1_SCLK_ECSPI1_SCLK

IOMUXC_ECSPI1_SCLK_UART3_RX

IOMUXC_ECSPI1_SCLK_UART3_TX

IOMUXC_ECSPI1_SCLK_I2C1_SCL

IOMUXC_ECSPI1_SCLK_AUDIOMIX_SAI7_RX_SYNC

IOMUXC_ECSPI1_SCLK_GPIO5_IO06

IOMUXC_ECSPI1_MOSI_ECSPI1_MOSI

IOMUXC_ECSPI1_MOSI_UART3_TX

IOMUXC_ECSPI1_MOSI_UART3_RX

IOMUXC_ECSPI1_MOSI_I2C1_SDA

IOMUXC_ECSPI1_MOSI_AUDIOMIX_SAI7_RX_BCLK

IOMUXC_ECSPI1_MOSI_GPIO5_IO07

IOMUXC_ECSPI1_MISO_ECSPI1_MISO

IOMUXC_ECSPI1_MISO_UART3_CTS_B

IOMUXC_ECSPI1_MISO_UART3_RTS_B

IOMUXC_ECSPI1_MISO_I2C2_SCL

IOMUXC_ECSPI1_MISO_AUDIOMIX_SAI7_RX_DATA0

IOMUXC_ECSPI1_MISO_GPIO5_IO08

IOMUXC_ECSPI1_SS0_ECSPI1_SS0

IOMUXC_ECSPI1_SS0_UART3_RTS_B

IOMUXC_ECSPI1_SS0_UART3_CTS_B

IOMUXC_ECSPI1_SS0_I2C2_SDA

IOMUXC_ECSPI1_SS0_AUDIOMIX_SAI7_TX_SYNC

IOMUXC_ECSPI1_SS0_GPIO5_IO09

IOMUXC_ECSPI2_SCLK_ECSPI2_SCLK

IOMUXC_ECSPI2_SCLK_UART4_RX

IOMUXC_ECSPI2_SCLK_UART4_TX

IOMUXC_ECSPI2_SCLK_I2C3_SCL

IOMUXC_ECSPI2_SCLK_AUDIOMIX_SAI7_TX_BCLK

IOMUXC_ECSPI2_SCLK_GPIO5_IO10

IOMUXC_ECSPI2_MOSI_ECSPI2_MOSI

IOMUXC_ECSPI2_MOSI_UART4_TX

IOMUXC_ECSPI2_MOSI_UART4_RX

IOMUXC_ECSPI2_MOSI_I2C3_SDA

IOMUXC_ECSPI2_MOSI_AUDIOMIX_SAI7_TX_DATA0

IOMUXC_ECSPI2_MOSI_GPIO5_IO11

IOMUXC_ECSPI2_MISO_ECSPI2_MISO

IOMUXC_ECSPI2_MISO_UART4_CTS_B

IOMUXC_ECSPI2_MISO_UART4_RTS_B

IOMUXC_ECSPI2_MISO_I2C4_SCL

IOMUXC_ECSPI2_MISO_AUDIOMIX_SAI7_MCLK

IOMUXC_ECSPI2_MISO_CCM_CLKO1

IOMUXC_ECSPI2_MISO_GPIO5_IO12

IOMUXC_ECSPI2_SS0_ECSPI2_SS0

IOMUXC_ECSPI2_SS0_UART4_RTS_B

IOMUXC_ECSPI2_SS0_UART4_CTS_B

IOMUXC_ECSPI2_SS0_I2C4_SDA

IOMUXC_ECSPI2_SS0_CCM_CLKO2

IOMUXC_ECSPI2_SS0_GPIO5_IO13

IOMUXC_I2C1_SCL_I2C1_SCL

IOMUXC_I2C1_SCL_ENET_QOS_MDC

IOMUXC_I2C1_SCL_ECSPI1_SCLK

IOMUXC_I2C1_SCL_GPIO5_IO14

IOMUXC_I2C1_SDA_I2C1_SDA

IOMUXC_I2C1_SDA_ENET_QOS_MDIO

IOMUXC_I2C1_SDA_ECSPI1_MOSI

IOMUXC_I2C1_SDA_GPIO5_IO15

IOMUXC_I2C2_SCL_I2C2_SCL

IOMUXC_I2C2_SCL_ENET_QOS_1588_EVENT1_IN

IOMUXC_I2C2_SCL_USDHC3_CD_B

IOMUXC_I2C2_SCL_ECSPI1_MISO

IOMUXC_I2C2_SCL_ENET_QOS_1588_EVENT1_AUX_IN

IOMUXC_I2C2_SCL_GPIO5_IO16

IOMUXC_I2C2_SDA_I2C2_SDA

IOMUXC_I2C2_SDA_ENET_QOS_1588_EVENT1_OUT

IOMUXC_I2C2_SDA_USDHC3_WP

IOMUXC_I2C2_SDA_ECSPI1_SS0

IOMUXC_I2C2_SDA_GPIO5_IO17

IOMUXC_I2C3_SCL_I2C3_SCL

IOMUXC_I2C3_SCL_PWM4_OUT

IOMUXC_I2C3_SCL_GPT2_CLK

IOMUXC_I2C3_SCL_ECSPI2_SCLK

IOMUXC_I2C3_SCL_GPIO5_IO18

IOMUXC_I2C3_SDA_I2C3_SDA

IOMUXC_I2C3_SDA_PWM3_OUT

IOMUXC_I2C3_SDA_GPT3_CLK

IOMUXC_I2C3_SDA_ECSPI2_MOSI

IOMUXC_I2C3_SDA_GPIO5_IO19

IOMUXC_I2C4_SCL_I2C4_SCL

IOMUXC_I2C4_SCL_PWM2_OUT

IOMUXC_I2C4_SCL_PCIE_CLKREQ_B

IOMUXC_I2C4_SCL_ECSPI2_MISO

IOMUXC_I2C4_SCL_GPIO5_IO20

IOMUXC_I2C4_SDA_I2C4_SDA

IOMUXC_I2C4_SDA_PWM1_OUT

IOMUXC_I2C4_SDA_ECSPI2_SS0

IOMUXC_I2C4_SDA_GPIO5_IO21

IOMUXC_UART1_RXD_UART1_RX

IOMUXC_UART1_RXD_UART1_TX

IOMUXC_UART1_RXD_ECSPI3_SCLK

IOMUXC_UART1_RXD_GPIO5_IO22

IOMUXC_UART1_TXD_UART1_TX

IOMUXC_UART1_TXD_UART1_RX

IOMUXC_UART1_TXD_ECSPI3_MOSI

IOMUXC_UART1_TXD_GPIO5_IO23

IOMUXC_UART2_RXD_UART2_RX

IOMUXC_UART2_RXD_UART2_TX

IOMUXC_UART2_RXD_ECSPI3_MISO

IOMUXC_UART2_RXD_GPT1_COMPARE3

IOMUXC_UART2_RXD_GPIO5_IO24

IOMUXC_UART2_TXD_UART2_TX

IOMUXC_UART2_TXD_UART2_RX

IOMUXC_UART2_TXD_ECSPI3_SS0

IOMUXC_UART2_TXD_GPT1_COMPARE2

IOMUXC_UART2_TXD_GPIO5_IO25

IOMUXC_UART3_RXD_UART3_RX

IOMUXC_UART3_RXD_UART3_TX

IOMUXC_UART3_RXD_UART1_CTS_B

IOMUXC_UART3_RXD_UART1_RTS_B

IOMUXC_UART3_RXD_USDHC3_RESET_B

IOMUXC_UART3_RXD_GPT1_CAPTURE2

IOMUXC_UART3_RXD_CAN2_TX

IOMUXC_UART3_RXD_GPIO5_IO26

IOMUXC_UART3_TXD_UART3_TX

IOMUXC_UART3_TXD_UART3_RX

IOMUXC_UART3_TXD_UART1_RTS_B

IOMUXC_UART3_TXD_UART1_CTS_B

IOMUXC_UART3_TXD_USDHC3_VSELECT

IOMUXC_UART3_TXD_GPT1_CLK

IOMUXC_UART3_TXD_CAN2_RX

IOMUXC_UART3_TXD_GPIO5_IO27

IOMUXC_UART4_RXD_UART4_RX

IOMUXC_UART4_RXD_UART4_TX

IOMUXC_UART4_RXD_UART2_CTS_B

IOMUXC_UART4_RXD_UART2_RTS_B

IOMUXC_UART4_RXD_PCIE_CLKREQ_B

IOMUXC_UART4_RXD_GPT1_COMPARE1

IOMUXC_UART4_RXD_I2C6_SCL

IOMUXC_UART4_RXD_GPIO5_IO28

IOMUXC_UART4_TXD_UART4_TX

IOMUXC_UART4_TXD_UART4_RX

IOMUXC_UART4_TXD_UART2_RTS_B

IOMUXC_UART4_TXD_UART2_CTS_B

IOMUXC_UART4_TXD_GPT1_CAPTURE1

IOMUXC_UART4_TXD_I2C6_SDA

IOMUXC_UART4_TXD_GPIO5_IO29

IOMUXC_HDMI_DDC_SCL_HDMIMIX_HDMI_SCL

IOMUXC_HDMI_DDC_SCL_I2C5_SCL

IOMUXC_HDMI_DDC_SCL_CAN1_TX

IOMUXC_HDMI_DDC_SCL_GPIO3_IO26

IOMUXC_HDMI_DDC_SCL_EARC_TEST_OUT0

IOMUXC_HDMI_DDC_SDA_HDMIMIX_HDMI_SDA

IOMUXC_HDMI_DDC_SDA_I2C5_SDA

IOMUXC_HDMI_DDC_SDA_CAN1_RX

IOMUXC_HDMI_DDC_SDA_GPIO3_IO27

IOMUXC_HDMI_DDC_SDA_EARC_TEST_OUT1

IOMUXC_HDMI_CEC_HDMIMIX_HDMI_CEC

IOMUXC_HDMI_CEC_I2C6_SCL

IOMUXC_HDMI_CEC_CAN2_TX

IOMUXC_HDMI_CEC_GPIO3_IO28

IOMUXC_HDMI_HPD_HDMIMIX_HDMI_HPD

IOMUXC_HDMI_HPD_AUDIOMIX_HDMI_HPD_O

IOMUXC_HDMI_HPD_I2C6_SDA

IOMUXC_HDMI_HPD_CAN2_RX

IOMUXC_HDMI_HPD_GPIO3_IO29

FSL_COMPONENT_ID

2.26 IRQSTEER: Interrupt Request Steering Driver

void IRQSTEER_Init(IRQSTEER_Type *base)

Initializes the IRQSTEER module.

This function enables the clock gate for the specified IRQSTEER.

Parameters

- base – IRQSTEER peripheral base address.

void IRQSTEER_Deinit(IRQSTEER_Type *base)

Deinitializes an IRQSTEER instance for operation.

The clock gate for the specified IRQSTEER is disabled.

Parameters

- base – IRQSTEER peripheral base address.

static inline void IRQSTEER_EnableInterrupt(IRQSTEER_Type *base, IRQn_Type irq)

Enables an interrupt source.

Parameters

- base – IRQSTEER peripheral base address.
- irq – Interrupt to be routed. The interrupt must be an IRQSTEER source.

static inline void IRQSTEER_DisableInterrupt(IRQSTEER_Type *base, IRQn_Type irq)

Disables an interrupt source.

Parameters

- base – IRQSTEER peripheral base address.
- irq – Interrupt source number. The interrupt must be an IRQSTEER source.

static inline bool IRQSTEER_InterruptIsEnabled(IRQSTEER_Type *base, IRQn_Type irq)

Check if an interrupt source is enabled.

Parameters

- base – IRQSTEER peripheral base address.
- irq – Interrupt to be queried. The interrupt must be an IRQSTEER source.

Returns

true if the interrupt is not masked, false otherwise.

static inline void IRQSTEER_SetInterrupt(IRQSTEER_Type *base, IRQn_Type irq, bool set)

Sets/Forces an interrupt.

Note: This function is not affected by the function IRQSTEER_DisableInterrupt and IRQSTEER_EnableInterrupt.

Parameters

- base – IRQSTEER peripheral base address.
- irq – Interrupt to be set/forced. The interrupt must be an IRQSTEER source.
- set – Switcher of the interrupt set/force function. "true" means to set. "false" means not (normal operation).

static inline void IRQSTEER_EnableMasterInterrupt(IRQSTEER_Type *base, *irqsteer_int_master_t* intMasterIndex)

Enables a master interrupt. By default, all the master interrupts are enabled.

For example, to enable the interrupt sources of master 1:

```
IRQSTEER_EnableMasterInterrupt(IRQSTEER_M4_0, kIRQSTEER_InterruptMaster1);
```

Parameters

- base – IRQSTEER peripheral base address.
- intMasterIndex – Master index of interrupt sources to be routed, options available in enumeration irqsteer_int_master_t.

static inline void IRQSTEER_DisableMasterInterrupt(IRQSTEER_Type *base, *irqsteer_int_master_t* intMasterIndex)

Disables a master interrupt.

For example, to disable the interrupt sources of master 1:

```
IRQSTEER_DisableMasterInterrupt(IRQSTEER_M4_0, kIRQSTEER_InterruptMaster1);
```

Parameters

- base – IRQSTEER peripheral base address.
- intMasterIndex – Master index of interrupt sources to be disabled, options available in enumeration irqsteer_int_master_t.

static inline bool IRQSTEER_IsInterruptSet(IRQSTEER_Type *base, IRQn_Type irq)

Checks the status of one specific IRQSTEER interrupt.

For example, to check whether interrupt from output 0 of Display 1 is set:

```
if (IRQSTEER_IsInterruptSet(IRQSTEER_DISPLAY1_INT_OUT0)
{
    ...
}
```

Parameters

- base – IRQSTEER peripheral base address.
- irq – Interrupt source status to be checked. The interrupt must be an IRQSTEER source.

Returns

The interrupt status. "true" means interrupt set. "false" means not.

static inline bool IRQSTEER_IsMasterInterruptSet(IRQSTEER_Type *base)

Checks the status of IRQSTEER master interrupt. The master interrupt status represents at least one interrupt is asserted or not among ALL interrupts.

Note: The master interrupt status is not affected by the function IRQSTEER_DisableMasterInterrupt.

Parameters

- base – IRQSTEER peripheral base address.

Returns

The master interrupt status. “true” means at least one interrupt set. “false” means not.

static inline uint32_t IRQSTEER_GetGroupInterruptStatus(IRQSTEER_Type *base, *irqsteer_int_group_t* intGroupIndex)

Gets the status of IRQSTEER group interrupt. The group interrupt status represents all the interrupt status within the group specified. This API aims for facilitating the status return of one set of interrupts.

Parameters

- base – IRQSTEER peripheral base address.
- intGroupIndex – Index of the interrupt group status to get.

Returns

The mask of the group interrupt status. Bit[n] set means the source with bit offset n in group intGroupIndex of IRQSTEER is asserted.

IRQn_Type IRQSTEER_GetMasterNextInterrupt(IRQSTEER_Type *base, *irqsteer_int_master_t* intMasterIndex)

Gets the next interrupt source (currently set) of one specific master.

Parameters

- base – IRQSTEER peripheral base address.
- intMasterIndex – Master index of interrupt sources, options available in enumeration irqsteer_int_master_t.

Returns

The current set next interrupt source number of one specific master. Return IRQSTEER_INT_Invalid if no interrupt set.

uint32_t IRQSTEER_GetMasterIrqCount(IRQSTEER_Type *base, *irqsteer_int_master_t* intMasterIndex)

Get the number of interrupt for a given master.

Parameters

- base – IRQSTEER peripheral base address.
- intMasterIndex – Master index of interrupt sources, options available in enumeration irqsteer_int_master_t.

Returns

Number of interrupts for a given master.

uint64_t IRQSTEER_GetMasterInterruptsStatus(IRQSTEER_Type *base, *irqsteer_int_master_t* intMasterIndex)

Get the status of the interrupts a master is in charge of.

What this function does is it takes the CHn_STATUS registers associated with the interrupts a master is in charge of and puts them in 64-bit variable. The order they are put in the 64-bit variable is the following: CHn_STATUS[i] : CHn_STATUS[i + 1], where CHn_STATUS[i + 1] is placed in the least significant half of the 64-bit variable. Assuming a master is in charge of 64 interrupts, the user may use the result of this function as such: BIT(i) & IRQSTEER_GetMasterInterrupts() to check if interrupt i is asserted.

Parameters

- base – IRQSTEER peripheral base address.
- intMasterIndex – Master index of interrupt sources, options available in enumeration irqsteer_int_master_t.

Returns

64-bit variable containing the status of the interrupts a master is in charge of.

FSL_IRQSTEER_DRIVER_VERSION

Driver version.

enum _irqsteer_int_group

IRQSTEER interrupt groups.

Values:

enumerator kIRQSTEER_InterruptGroup0

Interrupt Group 0: interrupt source 31 - 0

enumerator kIRQSTEER_InterruptGroup1

Interrupt Group 1: interrupt source 63 - 32

enumerator kIRQSTEER_InterruptGroup2

Interrupt Group 2: interrupt source 95 - 64

enumerator kIRQSTEER_InterruptGroup3

Interrupt Group 3: interrupt source 127 - 96

enumerator kIRQSTEER_InterruptGroup4

Interrupt Group 4: interrupt source 159 - 128

enumerator kIRQSTEER_InterruptGroup5

Interrupt Group 5: interrupt source 191 - 160

enumerator kIRQSTEER_InterruptGroup6

Interrupt Group 6: interrupt source 223 - 192

enumerator kIRQSTEER_InterruptGroup7

Interrupt Group 7: interrupt source 255 - 224

enumerator kIRQSTEER_InterruptGroup8

Interrupt Group 8: interrupt source 287 - 256

enumerator kIRQSTEER_InterruptGroup9

Interrupt Group 9: interrupt source 319 - 288

enumerator kIRQSTEER_InterruptGroup10

Interrupt Group 10: interrupt source 351 - 320

enumerator kIRQSTEER_InterruptGroup11

Interrupt Group 11: interrupt source 383 - 352

enumerator kIRQSTEER_InterruptGroup12

Interrupt Group 12: interrupt source 415 - 384

enumerator kIRQSTEER_InterruptGroup13
Interrupt Group 13: interrupt source 447 - 416

enumerator kIRQSTEER_InterruptGroup14
Interrupt Group 14: interrupt source 479 - 448

enumerator kIRQSTEER_InterruptGroup15
Interrupt Group 15: interrupt source 511 - 480

enum _irqsteer_int_master
IRQSTEER master interrupts mapping.

Values:

enumerator kIRQSTEER_InterruptMaster0
Interrupt Master 0: interrupt source 63 - 0

enumerator kIRQSTEER_InterruptMaster1
Interrupt Master 1: interrupt source 127 - 64

enumerator kIRQSTEER_InterruptMaster2
Interrupt Master 2: interrupt source 191 - 128

enumerator kIRQSTEER_InterruptMaster3
Interrupt Master 3: interrupt source 255 - 192

enumerator kIRQSTEER_InterruptMaster4
Interrupt Master 4: interrupt source 319 - 256

enumerator kIRQSTEER_InterruptMaster5
Interrupt Master 5: interrupt source 383 - 320

enumerator kIRQSTEER_InterruptMaster6
Interrupt Master 6: interrupt source 447 - 384

enumerator kIRQSTEER_InterruptMaster7
Interrupt Master 7: interrupt source 511 - 448

typedef enum *_irqsteer_int_group* irqsteer_int_group_t
IRQSTEER interrupt groups.

typedef enum *_irqsteer_int_master* irqsteer_int_master_t
IRQSTEER master interrupts mapping.

FSL_IRQSTEER_USE_DRIVER_IRQ_HANDLER
Use the IRQSTEER driver IRQ Handler or not.

When define as 1, IRQSTEER driver implements the IRQSTEER ISR, otherwise user shall implement it. Currently the IRQSTEER ISR is only available for Cortex-M platforms.

FSL_IRQSTEER_ENABLE_MASTER_INT
IRQSTEER_Init/IRQSTEER_Deinit enables/disables IRQSTEER master interrupt or not.

When define as 1, IRQSTEER_Init will enable the IRQSTEER master interrupt in system level interrupt controller (such as NVIC, GIC), IRQSTEER_Deinit will disable it. Otherwise IRQSTEER_Init/IRQSTEER_Deinit won't touch.

IRQSTEER_INT_SRC_REG_WIDTH
IRQSTEER interrupt source register width.

IRQSTEER_INT_MASTER_AGGREGATED_INT_NUM
IRQSTEER aggregated interrupt source number for each master.

IRQSTEER_INT_SRC_REG_INDEX(irq)

IRQSTEER interrupt source mapping register index.

IRQSTEER_INT_SRC_BIT_OFFSET(irq)

IRQSTEER interrupt source mapping bit offset.

IRQSTEER_INT_SRC_NUM(regIndex, bitOffset)

IRQSTEER interrupt source number.

2.27 ISI: Image Sensing Interface

void ISI_Init(ISI_Type *base)

Initializes the ISI peripheral.

This function ungates the ISI clock, it should be called before any other ISI functions.

Parameters

- base – ISI peripheral base address.

void ISI_Deinit(ISI_Type *base)

Deinitializes the ISI peripheral.

This function gates the ISI clock.

Parameters

- base – ISI peripheral base address.

void ISI_Reset(ISI_Type *base)

Reset the ISI peripheral.

This function resets the ISI channel processing pipeline similar to a hardware reset. The channel will need to be reconfigured after reset before it can be used.

Parameters

- base – ISI peripheral base address.

static inline uint32_t ISI_EnableInterrupts(ISI_Type *base, uint32_t mask)

Enables ISI interrupts.

Parameters

- base – ISI peripheral base address
- mask – Interrupt source, OR'ed value of _isi_interrupt.

Returns

OR'ed value of the enabled interrupts before calling this function.

static inline uint32_t ISI_DisableInterrupts(ISI_Type *base, uint32_t mask)

Disables ISI interrupts.

Parameters

- base – ISI peripheral base address
- mask – Interrupt source, OR'ed value of _isi_interrupt.

Returns

OR'ed value of the enabled interrupts before calling this function.

static inline uint32_t ISI_GetInterruptStatus(ISI_Type *base)

Get the ISI interrupt pending flags.

All interrupt pending flags are returned, upper layer could compare with the OR'ed value of _isi_interrupt. For example, to check whether memory read completed, use like this:

```
uint32_t mask = ISI_GetInterruptStatus(ISI);
if (mask & kISI_MemReadCompletedInterrupt)
{
    memory read completed
}
```

Parameters

- base – ISI peripheral base address

Returns

The OR'ed value of the pending interrupt flags. of _isi_interrupt.

static inline void ISI_ClearInterruptStatus(ISI_Type *base, uint32_t mask)

Clear ISI interrupt pending flags.

This function could clear one or more flags at one time, the flags to clear are passed in as an OR'ed value of _isi_interrupt. For example, to clear both line received interrupt flag and frame received flag, use like this:

```
ISI_ClearInterruptStatus(ISI, kISI_LineReceivedInterrupt | kISI_FrameReceivedInterrupt);
```

Parameters

- base – ISI peripheral base address
- mask – The flags to clear, it is OR'ed value of _isi_interrupt.

void ISI_SetScalerConfig(ISI_Type *base, uint16_t inputWidth, uint16_t inputHeight, uint16_t outputWidth, uint16_t outputHeight)

Set the ISI channel scaler configurations.

This function sets the scaling configurations. If the ISI channel is bypassed, then the scaling feature could not be used.

ISI only supports down scaling but not up scaling.

Note: Total bytes in one line after down scaling must be more than 256 bytes.

Parameters

- base – ISI peripheral base address
- inputWidth – Input image width.
- inputHeight – Input image height.
- outputWidth – Output image width.
- outputHeight – Output image height.

void ISI_SetColorSpaceConversionConfig(ISI_Type *base, const *isi_csc_config_t* *config)

Set the ISI color space conversion configurations.

This function sets the color space conversion configurations. After setting the configuration, use the function ISI_EnableColorSpaceConversion to enable this feature. If the ISI channel is bypassed, then the color space conversion feature could not be used.

Parameters

- base – ISI peripheral base address
- config – Pointer to the configuration structure.

void ISI_ColorSpaceConversionGetDefaultConfig(*isi_csc_config_t* *config)

Get the ISI color space conversion default configurations.

The default value is:

```
config->mode = kISI_CscYUV2RGB;
config->A1 = 0.0;
config->A2 = 0.0;
config->A3 = 0.0;
config->B1 = 0.0;
config->B2 = 0.0;
config->B3 = 0.0;
config->C1 = 0.0;
config->C2 = 0.0;
config->C3 = 0.0;
config->D1 = 0;
config->D2 = 0;
config->D3 = 0;
```

Parameters

- config – Pointer to the configuration structure.

static inline void ISI_EnableColorSpaceConversion(ISI_Type *base, bool enable)

Enable or disable the ISI color space conversion.

If the ISI channel is bypassed, then the color space conversion feature could not be used even enable using this function.

Note: The CSC is enabled by default. Disable it if it is not required.

Parameters

- base – ISI peripheral base address
- enable – True to enable, false to disable.

void ISI_SetCropConfig(ISI_Type *base, const *isi_crop_config_t* *config)

Set the ISI cropping configurations.

This function sets the cropping configurations. After setting the configuration, use the function ISI_EnableCrop to enable the feature. Cropping still works when the ISI channel is bypassed.

Note: The upper left corner and lower right corner should be configured base on the image resolution output from the scaler.

Parameters

- base – ISI peripheral base address
- config – Pointer to the configuration structure.

void ISI_CropGetDefaultConfig(*isi_crop_config_t* *config)

Get the ISI cropping default configurations.

The default value is:

```
config->upperLeftX = 0U;
config->upperLeftY = 0U;
config->lowerRightX = 0U;
config->lowerRightY = 0U;
```

Parameters

- config – Pointer to the configuration structure.

static inline void ISI_EnableCrop(ISI_Type *base, bool enable)

Enable or disable the ISI cropping.

If the ISI channel is bypassed, the cropping still works.

Parameters

- base – ISI peripheral base address
- enable – True to enable, false to disable.

static inline void ISI_SetGlobalAlpha(ISI_Type *base, uint8_t alpha)

Set the global alpha value.

Parameters

- base – ISI peripheral base address
- alpha – The global alpha value.

static inline void ISI_EnableGlobalAlpha(ISI_Type *base, bool enable)

Enable the global alpha insertion.

Alpha still works when channel bypassed.

Parameters

- base – ISI peripheral base address
- enable – True to enable, false to disable.

void ISI_SetRegionAlphaConfig(ISI_Type *base, uint8_t index, const *isi_region_alpha_config_t* *config)

Set the alpha value for region of interest.

Set the alpha insertion configuration for specific region of interest. The function ISI_EnableRegionAlpha could be used to enable the alpha insertion. Alpha insertion still works when channel bypassed.

Note: The upper left corner and lower right corner should be configured base on the image resolution output from the scaler.

Parameters

- base – ISI peripheral base address
- index – Index of the region of interest, Could be 0, 1, 2, and 3.
- config – Pointer to the configuration structure.

void ISI_RegionAlphaGetDefaultConfig(*isi_region_alpha_config_t* *config)

Get the regional alpha insertion default configurations.

The default configuration is:

```
config->upperLeftX = 0U;
config->upperLeftY = 0U;
config->lowerRightX = 0U;
config->lowerRightY = 0U;
config->alpha = 0U;
```

Parameters

- config – Pointer to the configuration structure.

void ISI_EnableRegionAlpha(ISI_Type *base, uint8_t index, bool enable)

Enable or disable the alpha value insertion for region of interest.

Alpha insertion still works when channel bypassed.

Parameters

- base – ISI peripheral base address
- index – Index of the region of interest, Could be 0, 1, 2, and 3.
- enable – True to enable, false to disable.

void ISI_SetInputMemConfig(ISI_Type *base, const *isi_input_mem_config_t* *config)

Set the input memory configuration.

Parameters

- base – ISI peripheral base address
- config – Pointer to the configuration structure.

void ISI_InputMemGetDefaultConfig(*isi_input_mem_config_t* *config)

Get the input memory default configurations.

The default configuration is:

```
config->adddr = 0U;
config->linePitchBytes = 0U;
config->framePitchBytes = 0U;
config->format = kISI_InputMemBGR8P;
```

Parameters

- config – Pointer to the configuration structure.

static inline void ISI_SetInputMemAddr(ISI_Type *base, uint32_t addr)

Set the input memory address.

This function only sets the input memory address, it is used for fast run-time setting.

Parameters

- base – ISI peripheral base address
- addr – Input memory address.

void ISI_TriggerInputMemRead(ISI_Type *base)

Trigger the ISI pipeline to read the input memory.

Parameters

- base – ISI peripheral base address

static inline void ISI_SetFlipMode(ISI_Type *base, *isi_flip_mode_t* mode)

Set the ISI channel flipping mode.

Parameters

- base – ISI peripheral base address
- mode – Flipping mode.

void ISI_SetOutputBufferAddr(ISI_Type *base, uint8_t index, uint32_t addrY, uint32_t addrU, uint32_t addrV)

Set the ISI output buffer address.

This function sets the output buffer address and trigger the ISI to shadow the address, it is used for fast run-time setting.

Parameters

- base – ISI peripheral base address
- index – Index of output buffer, could be 0 and 1.
- addrY – RGB or Luma (Y) output buffer address.
- addrU – Chroma (U/Cb/UV/CbCr) output buffer address.
- addrV – Chroma (V/Cr) output buffer address.

static inline void ISI_Start(ISI_Type *base)

Start the ISI channel.

Start the ISI channel to work, this function should be called after all channel configuration finished.

Parameters

- base – ISI peripheral base address

static inline void ISI_Stop(ISI_Type *base)

Stop the ISI channel.

Parameters

- base – ISI peripheral base address

FSL_ISI_DRIVER_VERSION

ISI driver version.

enum _isi_interrupt

ISI interrupts.

Values:

enumerator kISI_MemReadCompletedInterrupt

Input memory read completed.

enumerator kISI_LineReceivedInterrupt

Line received.

enumerator kISI_FrameReceivedInterrupt

Frame received.

enumerator kISI_AxiWriteErrorVInterrupt

AXI Bus write error when storing V data to memory.

enumerator kISI_AxiWriteErrorUInterrupt

AXI Bus write error when storing U data to memory.

enumerator kISI_AxiWriteErrorYInterrupt

AXI Bus write error when storing Y data to memory.

enumerator kISI_AxiReadErrorInterrupt

AXI Bus error when reading the input memory.

enumerator kISI_OverflowAlertVInterrupt
V output buffer overflow threshold accrossed.

enumerator kISI_ExcessOverflowVInterrupt
V output buffer excess overflow interrupt.

enumerator kISI_OverflowVInterrupt
V output buffer overflow interrupt.

enumerator kISI_OverflowAlertUInterrupt
U output buffer overflow threshold accrossed.

enumerator kISI_ExcessOverflowUInterrupt
U output buffer excess overflow interrupt.

enumerator kISI_OverflowUInterrupt
U output buffer overflow interrupt.

enumerator kISI_OverflowAlertYInterrupt
V output buffer overflow threshold accrossed.

enumerator kISI_ExcessOverflowYInterrupt
V output buffer excess overflow interrupt.

enumerator kISI_OverflowYInterrupt
V output buffer overflow interrupt.

enum __isi_output_format
ISI output image format.

Values:

enumerator kISI_OutputRGBA8888
RGBA8888.

enumerator kISI_OutputABGR8888
ABGR8888.

enumerator kISI_OutputARGB8888
ARGB8888.

enumerator kISI_OutputRGBX8888
RGBX8888 unpacked and MSB aligned in 32-bit.

enumerator kISI_OutputXBGR8888
XBGR8888 unpacked and LSB aligned in 32-bit.

enumerator kISI_OutputXRGB8888
XRGB8888 unpacked and LSB aligned in 32-bit.

enumerator kISI_OutputRGB888
RGB888 packed into 32-bit.

enumerator kISI_OutputBGR888
BGR888 packed into 32-bit.

enumerator kISI_OutputA2BGR10
BGR format with 2-bits alpha in MSB; 10-bits per color component.

enumerator kISI_OutputA2RGB10
RGB format with 2-bits alpha in MSB; 10-bits per color component.

enumerator kISI_OutputRGB565
RGB565 packed into 32-bit.

enumerator kISI_OutputRaw8
8-bit raw data packed into 32-bit.

enumerator kISI_OutputRaw10
10-bit raw data packed into 16-bit with 6 LSBs wasted.

enumerator kISI_OutputRaw10P
10-bit raw data packed into 32-bit.

enumerator kISI_OutputRaw12P
16-bit raw data packed into 16-bit with 4 LSBs wasted.

enumerator kISI_OutputRaw16P
16-bit raw data packed into 32-bit.

enumerator kISI_OutputYUV444_1P8P
8-bits per color component; 1-plane, YUV interleaved packed bytes.

enumerator kISI_OutputYUV444_2P8P
8-bits per color component; 2-plane, UV interleaved packed bytes.

enumerator kISI_OutputYUV444_3P8P
8-bits per color component; 3-plane, non-interleaved packed bytes.

enumerator kISI_OutputYUV444_1P8
8-bits per color component; 1-plane YUV interleaved unpacked bytes (8 MSBs waste bits in 32-bit DWORD).

enumerator kISI_OutputYUV444_1P10
10-bits per color component; 1-plane, YUV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV444_2P10
10-bits per color component; 2-plane, UV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV444_3P10
10-bits per color component; 3-plane, non-interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV444_1P10P
10-bits per color component; 1-plane, YUV interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

enumerator kISI_OutputYUV444_2P10P
10-bits per color component; 2-plane, UV interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

enumerator kISI_OutputYUV444_3P10P
10-bits per color component; 3-plane, non-interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

enumerator kISI_OutputYUV444_1P12
12-bits per color component; 1-plane, YUV interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV444_2P12
12-bits per color component; 2-plane, UV interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV444_3P12

12-bits per color component; 3-plane, non-interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV422_1P8P

8-bits per color component; 1-plane, YUV interleaved packed bytes.

enumerator kISI_OutputYUV422_2P8P

8-bits per color component; 2-plane, UV interleaved packed bytes.

enumerator kISI_OutputYUV422_3P8P

8-bits per color component; 3-plane, non-interleaved packed bytes.

enumerator kISI_OutputYUV422_1P10

10-bits per color component; 1-plane, YUV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV422_2P10

10-bits per color component; 2-plane, UV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV422_3P10

10-bits per color component; 3-plane, non-interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV422_1P10P

10-bits per color component; 1-plane, YUV interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

enumerator kISI_OutputYUV422_2P10P

10-bits per color component; 2-plane, UV interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

enumerator kISI_OutputYUV422_3P10P

10-bits per color component; 3-plane, non-interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

enumerator kISI_OutputYUV422_1P12

12-bits per color component; 1-plane, YUV interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV422_2P12

12-bits per color component; 2-plane, UV interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV422_3P12

12-bits per color component; 3-plane, non-interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV420_2P8P

8-bits per color component; 2-plane, UV interleaved packed bytes.

enumerator kISI_OutputYUV420_3P8P

8-bits per color component; 3-plane, non-interleaved packed bytes.

enumerator kISI_OutputYUV420_2P10

10-bits per color component; 2-plane, UV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV420_3P10

10-bits per color component; 3-plane, non-interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV420_2P10P

10-bits per color component; 2-plane, UV interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

enumerator kISI_OutputYUV420_3P10P

10-bits per color component; 3-plane, non-interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

enumerator kISI_OutputYUV420_2P12

12-bits per color component; 2-plane, UV interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

enumerator kISI_OutputYUV420_3P12

12-bits per color component; 3-plane, non-interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

enum _isi_chain_mode

ISI line buffer chain mode.

Values:

enumerator kISI_ChainDisable

No line buffers chained, for 2048 or less horizontal resolution.

enumerator kISI_ChainTwo

Line buffers of channel n and n+1 chained, for 4096 horizontal resolution.

enum _isi_deint_mode

ISI de-interlacing mode.

Values:

enumerator kISI_DeintDisable

No de-interlacing.

enumerator kISI_DeintWeaveOddOnTop

Weave de-interlacing (Odd, Even) method used.

enumerator kISI_DeintWeaveEvenOnTop

Weave de-interlacing (Even, Odd) method used.

enumerator kISI_DeintBlendingOddFirst

Blending or linear interpolation (Odd + Even).

enumerator kISI_DeintBlendingEvenFirst

Blending or linear interpolation (Even + Odd).

enumerator kISI_DeintDoublingOdd

Doubling odd frame and discard even frame.

enumerator kISI_DeintDoublingEven

Doubling even frame and discard odd frame.

enum _isi_threshold

ISI overflow panic alert threshold.

Values:

enumerator kISI_ThresholdDisable

No panic alert will be asserted.

enumerator kISI_Threshold25Percent

Panic will assert when the buffers are 25% full.

enumerator kISI_Threshold50Percent
Panic will assert when the buffers are 50% full.

enumerator kISI_Threshold75Percent
Panic will assert when the buffers are 75% full.

enum _isi_csc_mode
ISI color space conversion mode.

Values:

enumerator kISI_CscYUV2RGB
Convert YUV to RGB.

enumerator kISI_CscYCbCr2RGB
Convert YCbCr to RGB.

enumerator kISI_CscRGB2YUV
Convert RGB to YUV.

enumerator kISI_CscRGB2YCbCr
Convert RGB to YCbCr.

enum _isi_flip_mode
ISI flipping mode.

Values:

enumerator kISI_FlipDisable
Flip disabled.

enumerator kISI_FlipHorizontal
Horizontal flip.

enumerator kISI_FlipVertical
Vertical flip.

enumerator kISI_FlipBoth
Flip both direction.

enum _isi_input_mem_format
ISI image format of the input memory.

Values:

enumerator kISI_InputMemBGR888
BGR format with 8-bits per color component, packed into 32-bit, 24 bits per pixel.

enumerator kISI_InputMemRGB888
RGB format with 8-bits per color component, packed into 32-bit, 24 bits per pixel.

enumerator kISI_InputMemXRGB8888
RGB format with 8-bits per color component, unpacked and LSB aligned in 32-bit, 32 bits per pixel.

enumerator kISI_InputMemRGBX8888
RGB format with 8-bits per color component, unpacked and MSB alinged in 32-bit, 32 bits per pixel.

enumerator kISI_InputMemXBGR8888
BGR format with 8-bits per color component, unpacked and LSB aligned in 32-bit, 32 bits per pixel.

enumerator kISI_InputMemRGB565
RGB format with 5-bits of R, B; 6-bits of G (packed into 32-bit)

enumerator kISI_InputMemA2BGR10
BGR format with 2-bits alpha in MSB; 10-bits per color component.

enumerator kISI_InputMemA2RGB10
RGB format with 2-bits alpha in MSB; 10-bits per color component.

enumerator kISI_InputMemYUV444_1P8P
8-bits per color component; 1-plane, YUV interleaved packed bytes.

enumerator kISI_InputMemYUV444_1P10
10-bits per color component; 1-plane, YUV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_InputMemYUV444_1P10P
10-bits per color component; 1-plane, YUV interleaved packed bytes (2 MSBs waste bits in 32-bit WORD).

enumerator kISI_InputMemYUV444_1P12
12-bits per color component; 1-plane, YUV interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

enumerator kISI_InputMemYUV444_1P8
8-bits per color component; 1-plane YUV interleaved unpacked bytes (8 MSBs waste bits in 32-bit DWORD).

enumerator kISI_InputMemYUV422_1P8P
8-bits per color component; 1-plane YUV interleaved packed bytes.

enumerator kISI_InputMemYUV422_1P10
10-bits per color component; 1-plane, YUV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

enumerator kISI_InputMemYUV422_1P12
12-bits per color component; 1-plane, YUV interleaved packed bytes (4 MSBs waste bits in 16-bit WORD).

typedef enum *_isi_output_format* isi_output_format_t
ISI output image format.

typedef enum *_isi_chain_mode* isi_chain_mode_t
ISI line buffer chain mode.

typedef enum *_isi_deint_mode* isi_deint_mode_t
ISI de-interlacing mode.

typedef enum *_isi_threshold* isi_threshold_t
ISI overflow panic alert threshold.

typedef struct *_isi_config* isi_config_t
ISI basic configuration.

typedef enum *_isi_csc_mode* isi_csc_mode_t
ISI color space conversion mode.

typedef struct *_isi_csc_config* isi_csc_config_t
ISI color space conversion configurations.

(a) RGB to YUV (or YCbCr) conversion

- Y = (A1 x R) + (A2 x G) + (A3 x B) + D1

- U = (B1 x R) + (B2 x G) + (B3 x B) + D2
- V = (C1 x R) + (C2 x G) + (C3 x B) + D3

(b) YUV (or YCbCr) to RGB conversion

- R = (A1 x (Y + D1)) + (A2 x (U + D2)) + (A3 x (V + D3))
- G = (B1 x (Y + D1)) + (B2 x (U + D2)) + (B3 x (V + D3))
- B = (C1 x (Y + D1)) + (C2 x (U + D2)) + (C3 x (V + D3))

Overflow for the three channels are saturated at 0x255 and underflow is saturated at 0x00.

typedef enum *_isi_flip_mode* isi_flip_mode_t

ISI flipping mode.

typedef struct *_isi_crop_config* isi_crop_config_t

ISI cropping configurations.

typedef struct *_isi_regoin_alpha_config* isi_region_alpha_config_t

ISI regional region alpha configurations.

typedef enum *_isi_input_mem_format* isi_input_mem_format_t

ISI image format of the input memory.

typedef struct *_isi_input_mem_config* isi_input_mem_config_t

ISI input memory configurations.

void ISI_SetConfig(ISI_Type *base, const *isi_config_t* *config)

Set the ISI channel basic configurations.

This function sets the basic configurations, generally the channel could be started to work after this function. To enable other features such as croping, flipping, please call the functions accordingly.

Parameters

- base – ISI peripheral base address
- config – Pointer to the configuration structure.

void ISI_GetDefaultConfig(*isi_config_t* *config)

Get the ISI channel default basic configurations.

The default value is:

```
config->isChannelBypassed = false;
config->isSourceMemory = false;
config->isYCbCr = false;
config->chainMode = kISI_ChainDisable;
config->deintMode = kISI_DeintDisable;
config->blankPixel = 0xFFU;
config->sourcePort = 0U;
config->mipiChannel = 0U;
config->inputHeight = 1080U;
config->inputWidth = 1920U;
config->outputFormat = kISI_OutputRGBA8888;
config->outputLinePitchBytes = 0U;
config->thresholdY = kISI_ThresholdDisable;
config->thresholdU = kISI_ThresholdDisable;
config->thresholdV = kISI_ThresholdDisable;
```

Parameters

- config – Pointer to the configuration structure.

struct __isi__config

#include <fsl_isi.h> ISI basic configuration.

Public Members

bool isChannelBypassed

Bypass the channel, if bypassed, the scaling and color space conversion could not work.

bool isSourceMemory

Whether the input source is memory or not.

bool isYCbCr

Whether the input source is YCbCr mode or not.

isi_chain_mode_t chainMode

The line buffer chain mode.

isi_deint_mode_t deintMode

The de-interlacing mode.

uint8_t blankPixel

The pixel to insert into image when overflow occors.

uint8_t sourcePort

Input source port selection.

uint8_t mipiChannel

MIPI virtual channel, ignored if input source is not MIPI CSI.

uint16_t inputHeight

Input image height(lines).

uint16_t inputWidth

Input image width(pixels).

isi_output_format_t outputFormat

Output image format.

isi_threshold_t thresholdY

Panic alert threshold for RGB or Luma (Y) buffer.

isi_threshold_t thresholdU

Panic alert threshold for Chroma (U/Cb/UV/CbCr) buffer.

isi_threshold_t thresholdV

Panic alert threshold for Chroma (V/Cr) buffer.

struct __isi__csc__config

#include <fsl_isi.h> ISI color space conversion configurations.

(a) RGB to YUV (or YCbCr) conversion

- Y = (A1 x R) + (A2 x G) + (A3 x B) + D1
- U = (B1 x R) + (B2 x G) + (B3 x B) + D2
- V = (C1 x R) + (C2 x G) + (C3 x B) + D3

(b) YUV (or YCbCr) to RGB conversion

- R = (A1 x (Y + D1)) + (A2 x (U + D2)) + (A3 x (V + D3))
- G = (B1 x (Y + D1)) + (B2 x (U + D2)) + (B3 x (V + D3))
- B = (C1 x (Y + D1)) + (C2 x (U + D2)) + (C3 x (V + D3))

Overflow for the three channels are saturated at 0x255 and underflow is saturated at 0x00.

Public Members

isi_csc_mode_t mode
Convertion mode.

float A1
Must be in the range of [-3.99609375, 3.99609375].

float A2
Must be in the range of [-3.99609375, 3.99609375].

float A3
Must be in the range of [-3.99609375, 3.99609375].

float B1
Must be in the range of [-3.99609375, 3.99609375].

float B2
Must be in the range of [-3.99609375, 3.99609375].

float B3
Must be in the range of [-3.99609375, 3.99609375].

float C1
Must be in the range of [-3.99609375, 3.99609375].

float C2
Must be in the range of [-3.99609375, 3.99609375].

float C3
Must be in the range of [-3.99609375, 3.99609375].

int32_t D1
Must be in the range of [-256, 255].

int32_t D2
Must be in the range of [-256, 255].

int32_t D3
Must be in the range of [-256, 255].

struct __isi_crop_config
#include <fsl_isi.h> ISI cropping configurations.

Public Members

uint16_t upperLeftX
X of upper left corner.

uint16_t upperLeftY
Y of upper left corner.

uint16_t lowerRightX
X of lower right corner.

uint16_t lowerRightY
Y of lower right corner.

struct __isi_regoin_alpha_config
#include <fsl_isi.h> ISI regional region alpha configurations.

Public Members

uint16_t upperLeftX
: X of upper left corner.

uint16_t upperLeftY
: Y of upper left corner.

uint16_t lowerRightX
: X of lower right corner.

uint16_t lowerRightY
: Y of lower right corner.

uint8_t alpha
: Alpha value.

struct _isi_input_mem_config
: *#include <fsl_isi.h>* ISI input memory configurations.

Public Members

uint32_t adddr
: Address of the input memory.

uint16_t linePitchBytes
: Line phtch in bytes.

uint16_t framePitchBytes
: Frame phtch in bytes.

isi_input_mem_format_t format
: Image format of the input memory.

2.28 Common Driver

FSL_COMMON_DRIVER_VERSION
: common driver version.

DEBUG_CONSOLE_DEVICE_TYPE_NONE
: No debug console.

DEBUG_CONSOLE_DEVICE_TYPE_UART
: Debug console based on UART.

DEBUG_CONSOLE_DEVICE_TYPE_LPUART
: Debug console based on LPUART.

DEBUG_CONSOLE_DEVICE_TYPE_LPSCI
: Debug console based on LPSCI.

DEBUG_CONSOLE_DEVICE_TYPE_USBCDC
: Debug console based on USBCDC.

DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM
: Debug console based on FLEXCOMM.

DEBUG_CONSOLE_DEVICE_TYPE_IUART
: Debug console based on i.MX UART.

DEBUG_CONSOLE_DEVICE_TYPE_VUSART
: Debug console based on LPC_VUSART.

DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART
: Debug console based on LPC_USART.

DEBUG_CONSOLE_DEVICE_TYPE_SWO
: Debug console based on SWO.

DEBUG_CONSOLE_DEVICE_TYPE_QSCI
: Debug console based on QSCI.

MIN(a, b)
: Computes the minimum of *a* and *b*.

MAX(a, b)
: Computes the maximum of *a* and *b*.

UINT16_MAX
: Max value of uint16_t type.

UINT32_MAX
: Max value of uint32_t type.

SDK_ATOMIC_LOCAL_ADD(addr, val)
: Add value *val* from the variable at address *address*.

SDK_ATOMIC_LOCAL_SUB(addr, val)
: Subtract value *val* to the variable at address *address*.

SDK_ATOMIC_LOCAL_SET(addr, bits)
: Set the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR(addr, bits)
: Clear the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_TOGGLE(addr, bits)
: Toggle the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR_AND_SET(addr, clearBits, setBits)
: For the variable at address *address*, clear the bits specifiled by *clearBits* and set the bits specifiled by *setBits*.

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

SDK_ATOMIC_LOCAL_TEST_AND_SET(addr, newValue)
: For the variable at address *address*, set as *newValue* value and return old value.

USEC_TO_COUNT(us, clockFreqInHz)
: Macro to convert a microsecond period to raw count value

COUNT_TO_USEC(count, clockFreqInHz)
: Macro to convert a raw count value to microsecond

MSEC_TO_COUNT(ms, clockFreqInHz)
: Macro to convert a millisecond period to raw count value

COUNT_TO_MSEC(count, clockFreqInHz)
: Macro to convert a raw count value to millisecond

SDK_ISR_EXIT_BARRIER

SDK_SIZEALIGN(var, alignbytes)

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

Macro to define a variable with L2 cache line size alignment

Macro to change a value to a given size aligned value

AT_NONCACHEABLE_SECTION(var)

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

AT_NONCACHEABLE_SECTION_ALIGN(var, alignbytes)

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

AT_NONCACHEABLE_SECTION_INIT(var)

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

AT_NONCACHEABLE_SECTION_ALIGN_INIT(var, alignbytes)

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

enum _status_groups

Status group numbers.

Values:

enumerator kStatusGroup_Generic

Group number for generic status codes.

enumerator kStatusGroup_FLASH

Group number for FLASH status codes.

enumerator kStatusGroup_LPSPI

Group number for LPSPI status codes.

enumerator kStatusGroup_FLEXIO_SPI

Group number for FLEXIO SPI status codes.

enumerator kStatusGroup_DSPI

Group number for DSPI status codes.

enumerator kStatusGroup_FLEXIO_UART

Group number for FLEXIO UART status codes.

enumerator kStatusGroup_FLEXIO_I2C

Group number for FLEXIO I2C status codes.

enumerator kStatusGroup_LPI2C

Group number for LPI2C status codes.

enumerator kStatusGroup_UART

Group number for UART status codes.

enumerator kStatusGroup_I2C

Group number for UART status codes.

enumerator kStatusGroup_LPSCI

Group number for LPSCI status codes.

enumerator kStatusGroup_LPUART

Group number for LPUART status codes.

enumerator kStatusGroup_SPI

Group number for SPI status code.

enumerator kStatusGroup_XRDC
Group number for XRDC status code.

enumerator kStatusGroup_SEMA42
Group number for SEMA42 status code.

enumerator kStatusGroup_SDHC
Group number for SDHC status code

enumerator kStatusGroup_SDMMC
Group number for SDMMC status code

enumerator kStatusGroup_SAI
Group number for SAI status code

enumerator kStatusGroup_MCG
Group number for MCG status codes.

enumerator kStatusGroup_SCG
Group number for SCG status codes.

enumerator kStatusGroup_SDSPI
Group number for SDSPI status codes.

enumerator kStatusGroup_FLEXIO_I2S
Group number for FLEXIO I2S status codes

enumerator kStatusGroup_FLEXIO_MCULCD
Group number for FLEXIO LCD status codes

enumerator kStatusGroup_FLASHIAP
Group number for FLASHIAP status codes

enumerator kStatusGroup_FLEXCOMM_I2C
Group number for FLEXCOMM I2C status codes

enumerator kStatusGroup_I2S
Group number for I2S status codes

enumerator kStatusGroup_IUART
Group number for IUART status codes

enumerator kStatusGroup_CSI
Group number for CSI status codes

enumerator kStatusGroup_MIPI_DSI
Group number for MIPI DSI status codes

enumerator kStatusGroup_SDRAMC
Group number for SDRAMC status codes.

enumerator kStatusGroup_POWER
Group number for POWER status codes.

enumerator kStatusGroup_ENET
Group number for ENET status codes.

enumerator kStatusGroup_PHY
Group number for PHY status codes.

enumerator kStatusGroup_TRGMUX
Group number for TRGMUX status codes.

enumerator kStatusGroup_SMARTCARD
Group number for SMARTCARD status codes.

enumerator kStatusGroup_LMEM
Group number for LMEM status codes.

enumerator kStatusGroup_QSPI
Group number for QSPI status codes.

enumerator kStatusGroup_DMA
Group number for DMA status codes.

enumerator kStatusGroup_EDMA
Group number for EDMA status codes.

enumerator kStatusGroup_DMAMGR
Group number for DMAMGR status codes.

enumerator kStatusGroup_FLEXCAN
Group number for FlexCAN status codes.

enumerator kStatusGroup_LTC
Group number for LTC status codes.

enumerator kStatusGroup_FLEXIO_CAMERA
Group number for FLEXIO CAMERA status codes.

enumerator kStatusGroup_LPC_SPI
Group number for LPC_SPI status codes.

enumerator kStatusGroup_LPC_USART
Group number for LPC_USART status codes.

enumerator kStatusGroup_DMIC
Group number for DMIC status codes.

enumerator kStatusGroup_SDIF
Group number for SDIF status codes.

enumerator kStatusGroup_SPIFI
Group number for SPIFI status codes.

enumerator kStatusGroup_OTP
Group number for OTP status codes.

enumerator kStatusGroup_MCAN
Group number for MCAN status codes.

enumerator kStatusGroup_CAAM
Group number for CAAM status codes.

enumerator kStatusGroup_ECSPI
Group number for ECSPI status codes.

enumerator kStatusGroup_USDHC
Group number for USDHC status codes.

enumerator kStatusGroup_LPC_I2C
Group number for LPC_I2C status codes.

enumerator kStatusGroup_DCP
Group number for DCP status codes.

enumerator kStatusGroup_MSCAN
Group number for MSCAN status codes.

enumerator kStatusGroup_ESAI
Group number for ESAI status codes.

enumerator kStatusGroup_FLEXSPI
Group number for FLEXSPI status codes.

enumerator kStatusGroup_MMDC
Group number for MMDC status codes.

enumerator kStatusGroup_PDM
Group number for MIC status codes.

enumerator kStatusGroup_SDMA
Group number for SDMA status codes.

enumerator kStatusGroup_ICS
Group number for ICS status codes.

enumerator kStatusGroup_SPDIF
Group number for SPDIF status codes.

enumerator kStatusGroup_LPC_MINISPI
Group number for LPC_MINISPI status codes.

enumerator kStatusGroup_HASHCRYPT
Group number for Hashcrypt status codes

enumerator kStatusGroup_LPC_SPI_SSP
Group number for LPC_SPI_SSP status codes.

enumerator kStatusGroup_I3C
Group number for I3C status codes

enumerator kStatusGroup_LPC_I2C_1
Group number for LPC_I2C_1 status codes.

enumerator kStatusGroup_NOTIFIER
Group number for NOTIFIER status codes.

enumerator kStatusGroup_DebugConsole
Group number for debug console status codes.

enumerator kStatusGroup_SEMC
Group number for SEMC status codes.

enumerator kStatusGroup_ApplicationRangeStart
Starting number for application groups.

enumerator kStatusGroup_IAP
Group number for IAP status codes

enumerator kStatusGroup_SFA
Group number for SFA status codes

enumerator kStatusGroup_SPC
Group number for SPC status codes.

enumerator kStatusGroup_PUF
Group number for PUF status codes.

enumerator kStatusGroup_TOUCH_PANEL
Group number for touch panel status codes

enumerator kStatusGroup_VBAT
Group number for VBAT status codes

enumerator kStatusGroup_XSPI
Group number for XSPI status codes

enumerator kStatusGroup_PNGDEC
Group number for PNGDEC status codes

enumerator kStatusGroup_JPEGDEC
Group number for JPEGDEC status codes

enumerator kStatusGroup_HAL_GPIO
Group number for HAL GPIO status codes.

enumerator kStatusGroup_HAL_UART
Group number for HAL UART status codes.

enumerator kStatusGroup_HAL_TIMER
Group number for HAL TIMER status codes.

enumerator kStatusGroup_HAL_SPI
Group number for HAL SPI status codes.

enumerator kStatusGroup_HAL_I2C
Group number for HAL I2C status codes.

enumerator kStatusGroup_HAL_FLASH
Group number for HAL FLASH status codes.

enumerator kStatusGroup_HAL_PWM
Group number for HAL PWM status codes.

enumerator kStatusGroup_HAL_RNG
Group number for HAL RNG status codes.

enumerator kStatusGroup_HAL_I2S
Group number for HAL I2S status codes.

enumerator kStatusGroup_HAL_ADC_SENSOR
Group number for HAL ADC SENSOR status codes.

enumerator kStatusGroup_TIMERMANAGER
Group number for TiMER MANAGER status codes.

enumerator kStatusGroup_SERIALMANAGER
Group number for SERIAL MANAGER status codes.

enumerator kStatusGroup_LED
Group number for LED status codes.

enumerator kStatusGroup_BUTTON
Group number for BUTTON status codes.

enumerator kStatusGroup_EXTERN_EEPROM
Group number for EXTERN EEPROM status codes.

enumerator kStatusGroup_SHELL
Group number for SHELL status codes.

enumerator kStatusGroup_MEM_MANAGER
Group number for MEM MANAGER status codes.

enumerator kStatusGroup_LIST
Group number for List status codes.

enumerator kStatusGroup_OSA
Group number for OSA status codes.

enumerator kStatusGroup_COMMON_TASK
Group number for Common task status codes.

enumerator kStatusGroup_MSG
Group number for messaging status codes.

enumerator kStatusGroup_SDK_OCOTP
Group number for OCOTP status codes.

enumerator kStatusGroup_SDK_FLEXSPINOR
Group number for FLEXSPINOR status codes.

enumerator kStatusGroup_CODEC
Group number for codec status codes.

enumerator kStatusGroup_ASRC
Group number for codec status ASRC.

enumerator kStatusGroup_OTFAD
Group number for codec status codes.

enumerator kStatusGroup_SDIOSLV
Group number for SDIOSLV status codes.

enumerator kStatusGroup_MECC
Group number for MECC status codes.

enumerator kStatusGroup_ENET_QOS
Group number for ENET_QOS status codes.

enumerator kStatusGroup_LOG
Group number for LOG status codes.

enumerator kStatusGroup_I3CBUS
Group number for I3CBUS status codes.

enumerator kStatusGroup_QSCI
Group number for QSCI status codes.

enumerator kStatusGroup_ELEMU
Group number for ELEMU status codes.

enumerator kStatusGroup_QUEUEDSPI
Group number for QSPI status codes.

enumerator kStatusGroup_POWER_MANAGER
Group number for POWER_MANAGER status codes.

enumerator kStatusGroup_IPED
Group number for IPED status codes.

enumerator kStatusGroup_ELS_PKC
Group number for ELS PKC status codes.

enumerator kStatusGroup_CSS_PKC
Group number for CSS PKC status codes.

enumerator kStatusGroup_HOSTIF
Group number for HOSTIF status codes.

enumerator kStatusGroup_CLIF
Group number for CLIF status codes.

enumerator kStatusGroup_BMA
Group number for BMA status codes.

enumerator kStatusGroup_NETC
Group number for NETC status codes.

enumerator kStatusGroup_ELE
Group number for ELE status codes.

enumerator kStatusGroup_GLIKEY
Group number for GLIKEY status codes.

enumerator kStatusGroup_AON_POWER
Group number for AON_POWER status codes.

enumerator kStatusGroup_AON_COMMON
Group number for AON_COMMON status codes.

enumerator kStatusGroup_ENDAT3
Group number for ENDAT3 status codes.

enumerator kStatusGroup_HIPERFACE
Group number for HIPERFACE status codes.

enumerator kStatusGroup_NPX
Group number for NPX status codes.

enumerator kStatusGroup_ELA_CSEC
Group number for ELA_CSEC status codes.

enumerator kStatusGroup_FLEXIO_T_FORMAT
Group number for T-format status codes.

enumerator kStatusGroup_FLEXIO_A_FORMAT
Group number for A-format status codes.

Generic status return codes.

Values:

enumerator kStatus_Success
Generic status for Success.

enumerator kStatus_Fail
Generic status for Fail.

enumerator kStatus_ReadOnly
Generic status for read only failure.

enumerator kStatus_OutOfRange
Generic status for out of range access.

enumerator kStatus_InvalidArgument
Generic status for invalid argument check.

enumerator kStatus_Timeout
Generic status for timeout.

enumerator kStatus_NoTransferInProgress
Generic status for no transfer in progress.

enumerator kStatus_Busy
Generic status for module is busy.

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

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

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

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

Parameters

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

Return values
The – allocated memory.

void SDK_Free(void *ptr)
Free memory.

Parameters

- ptr – The memory to be release.

void SDK_DelayAtLeastUs(uint32_t delayTime_us, uint32_t coreClock_Hz)
Delay at least for some time. Please note that, this API uses while loop for delay, different run-time environments make the time not precise, if precise delay count was needed, please implement a new delay function with hardware timer.

Parameters

- delayTime_us – Delay time in unit of microsecond.
- coreClock_Hz – Core clock frequency with Hz.

static inline *status_t* EnableIRQ(IRQn_Type interrupt)
Enable specific interrupt.

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

This function only enables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ number.

Return values

- kStatus_Success – Interrupt enabled successfully

- kStatus_Fail – Failed to enable the interrupt

static inline *status_t* DisableIRQ(IRQn_Type interrupt)

Disable specific interrupt.

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

This function only disables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ number.

Return values

- kStatus_Success – Interrupt disabled successfully
- kStatus_Fail – Failed to disable the interrupt

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

Enable the IRQ, and also set the interrupt priority.

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

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

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

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

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

Set the IRQ priority.

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

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

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

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline *status_t* IRQ_ClearPendingIRQ(IRQn_Type interrupt)

Clear the pending IRQ flag.

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

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The flag which IRQ to clear.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline uint32_t DisableGlobalIRQ(void)

Disable the global IRQ.

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

Returns

Current primask value.

static inline void EnableGlobalIRQ(uint32_t primask)

Enable the global IRQ.

Set the primask register with the provided primask value but not just enable the primask. The idea is for the convenience of integration of RTOS. some RTOS get its own management mechanism of primask. User is required to use the EnableGlobalIRQ() and DisableGlobalIRQ() in pair.

Parameters

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

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

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

FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ

Macro to use the default weak IRQ handler in drivers.

MAKE_STATUS(group, code)

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

MAKE_VERSION(major, minor, bugfix)

Construct the version number for drivers.

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

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

ARRAY_SIZE(x)

Computes the number of elements in an array.

UINT64_H(X)

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

UINT64_L(X)

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

SUPPRESS_FALL_THROUGH_WARNING()

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

MSDK_REG_SECURE_ADDR(x)

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

MSDK_REG_NONSECURE_ADDR(x)

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

MSDK_INVALID_IRQ_HANDLER

Invalid IRQ handler address.

2.29 LCDIF: LCD interface

status_t LCDIF_Init(LCDIF_Type *base)

Initialize the LCDIF.

This function initializes the LCDIF to work.

Parameters

- base – LCDIF peripheral base address.

Return values

kStatus_Success – Initialize successfully.

void LCDIF_Deinit(LCDIF_Type *base)

De-initialize the LCDIF.

This function disables the LCDIF peripheral clock.

Parameters

- base – LCDIF peripheral base address.

void LCDIF_DpiModeGetDefaultConfig(*lcdif_dpi_config_t* *config)

Get the default configuration for to initialize the LCDIF.

The default configuration value is:

```
config->panelWidth = 0;
config->panelHeight = 0;
config->hsw = 0;
config->hfp = 0;
config->hbp = 0;
config->vsw = 0;
config->vfp = 0;
config->vbp = 0;
config->polarityFlags = kLCDIF_VsyncActiveLow | kLCDIF_HsyncActiveLow | kLCDIF_
↪DataEnableActiveHigh |
kLCDIF_DriveDataOnFallingClkEdge; config->format = kLCDIF_Output24Bit;
```

Parameters

- config – Pointer to the LCDIF configuration.

status_t LCDIF_DpiModeSetConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_dpi_config_t* *config)

Initialize the LCDIF to work in DPI mode.

This function configures the LCDIF DPI display.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- config – Pointer to the configuration structure.

Return values

- kStatus_Success – Initialize successfully.
- kStatus_InvalidArgument – Initialize failed because of invalid argument.

status_t LCDIF_DbiModeSetConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_dbi_config_t* *config)

Initialize the LCDIF to work in DBI mode.

This function configures the LCDIF DBI display.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- config – Pointer to the configuration structure.

Return values

- kStatus_Success – Initialize successfully.
- kStatus_InvalidArgument – Initialize failed because of invalid argument.

void LCDIF_DbiModeGetDefaultConfig(*lcdif_dbi_config_t* *config)

Get the default configuration to initialize the LCDIF DBI mode.

The default configuration value is:

```
config->swizzle         = kLCDIF_DbiOutSwizzleRGB;
config->format          = kLCDIF_DbiOutD8RGB332;
config->acTimeUnit      = 0;
config->type            = kLCDIF_DbiTypeA_ClockedE;
config->reversePolarity = false;
config->writeWRPeriod   = 3U;
config->writeWRAssert   = 0U;
config->writeCSAssert   = 0U;
config->writeWRDeassert = 0U;
config->writeCSDeassert = 0U;
config->typeCTas        = 1U;
config->typeCSCLTwrl    = 1U;
config->typeCSCLTwrh    = 1U;
```

Parameters

- config – Pointer to the LCDIF DBI configuration.

static inline void LCDIF_DbiReset(LCDIF_Type *base, uint8_t displayIndex)

Reset the DBI module.

Parameters

- displayIndex – Display index.
- base – LCDIF peripheral base address.

void LCDIF_DbiSelectArea(LCDIF_Type *base, uint8_t displayIndex, uint16_t startX, uint16_t startY, uint16_t endX, uint16_t endY, bool isTiled)

Select the update area in DBI mode.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- startX – X coordinate for start pixel.
- startY – Y coordinate for start pixel.
- endX – X coordinate for end pixel.
- endY – Y coordinate for end pixel.
- isTiled – true if the pixel data is tiled.

static inline void LCDIF_DbiSendCommand(LCDIF_Type *base, uint8_t displayIndex, uint8_t cmd)

Send command to DBI port.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- cmd – the DBI command to send.

void LCDIF_DbiSendData(LCDIF_Type *base, uint8_t displayIndex, const uint8_t *data, uint32_t dataLen_Byte)

brief Send data to DBI port.

Can be used to send light weight data to panel. To send pixel data in frame buffer, use LCDIF_DbiWriteMem.

param base LCDIF peripheral base address. param displayIndex Display index. param data pointer to data buffer. param dataLen_Byte data buffer length in byte.

void LCDIF_DbiSendCommandAndData(LCDIF_Type *base, uint8_t displayIndex, uint8_t cmd, const uint8_t *data, uint32_t dataLen_Byte)

Send command followed by data to DBI port.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- cmd – the DBI command to send.
- data – pointer to data buffer.
- dataLen_Byte – data buffer length in byte.

static inline void LCDIF_DbiWriteMem(LCDIF_Type *base, uint8_t displayIndex)

Send pixel data in frame buffer to panel controller memory.

This function starts sending the pixel data in frame buffer to panel controller, user can monitor interrupt kLCDIF_Display0FrameDoneInterrupt to know when then data sending finished.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.

void LCDIF_SetFrameBufferConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_fb_config_t* *config)

Configure the LCDIF frame buffer.

@Note: For LCDIF of version DC8000 there can be 3 layers in the pre-processing, compared with the older version. Apart from the video layer, there are also 2 overlay layers which shares the same configurations. Use this API to configure the legacy video layer, and use LCDIF_SetOverlayFrameBufferConfig to configure the overlay layers.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- config – Pointer to the configuration structure.

void LCDIF_FrameBufferGetDefaultConfig(*lcdif_fb_config_t* *config)

Get default frame buffer configuration.

@Note: For LCDIF of version DC8000 there can be 3 layers in the pre-processing, compared with the older version. Apart from the video layer, there are also 2 overlay layers which shares the same configurations. Use this API to get the default configuration for all the 3 layers.

The default configuration is

```
config->enable = true;
config->enableGamma = false;
config->format = kLCDIF_PixelFormatRGB565;
```

Parameters

- config – Pointer to the configuration structure.

static inline void LCDIF_SetFrameBufferAddr(LCDIF_Type *base, uint8_t displayIndex, uint32_t address)

Set the frame buffer to LCDIF.

Note: The address must be 128 bytes aligned.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- address – Frame buffer address.

void LCDIF_SetFrameBufferStride(LCDIF_Type *base, uint8_t displayIndex, uint32_t strideBytes)

Set the frame buffer stride.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- strideBytes – The stride in byte.

void LCDIF_SetDitherConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_dither_config_t* *config)

Set the dither configuration.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Index to configure.
- config – Pointer to the configuration structure.

void LCDIF_SetGammaData(LCDIF_Type *base, uint8_t displayIndex, uint16_t startIndex, const uint32_t *gamma, uint16_t gammaLen)

Set the gamma translation values to the LCDIF gamma table.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- startIndex – Start index in the gamma table that the value will be set to.
- gamma – The gamma values to set to the gamma table in LCDIF, could be defined using LCDIF_MAKE_GAMMA_VALUE.
- gammaLen – The length of the gamma.

static inline void LCDIF_EnableInterrupts(LCDIF_Type *base, uint32_t mask)

Enables LCDIF interrupt requests.

Parameters

- base – LCDIF peripheral base address.
- mask – The interrupts to enable, pass in as OR'ed value of _lcdif_interrupt.

static inline void LCDIF_DisableInterrupts(LCDIF_Type *base, uint32_t mask)

Disable LCDIF interrupt requests.

Parameters

- base – LCDIF peripheral base address.
- mask – The interrupts to disable, pass in as OR'ed value of _lcdif_interrupt.

static inline uint32_t LCDIF_GetAndClearInterruptPendingFlags(LCDIF_Type *base)

Get and clear LCDIF interrupt pending status.

Note: The interrupt must be enabled, otherwise the interrupt flags will not assert.

Parameters

- base – LCDIF peripheral base address.

Returns

The interrupt pending status.

void LCDIF_CursorGetDefaultConfig(*lcdif_cursor_config_t* *config)

Get the hardware cursor default configuration.

The default configuration values are:

```
config->enable = true;
config->format = kLCDIF_CursorMasked;
config->hotspotOffsetX = 0;
config->hotspotOffsetY = 0;
```

Parameters

- config – Pointer to the hardware cursor configuration structure.

void LCDIF_SetCursorConfig(LCDIF_Type *base, const *lcdif_cursor_config_t* *config)

Configure the cursor.

Parameters

- base – LCDIF peripheral base address.
- config – Cursor configuration.

static inline void LCDIF_SetCursorHotspotPosition(LCDIF_Type *base, uint16_t x, uint16_t y)

Set the cursor hotspot postion.

Parameters

- base – LCDIF peripheral base address.
- x – X coordinate of the hotspot, range 0 ~ 8191.
- y – Y coordinate of the hotspot, range 0 ~ 8191.

static inline void LCDIF_SetCursorBufferAddress(LCDIF_Type *base, uint32_t address)

Set the cursor memory address.

Parameters

- base – LCDIF peripheral base address.
- address – Memory address.

void LCDIF_SetCursorColor(LCDIF_Type *base, uint32_t background, uint32_t foreground)

Set the cursor color.

Parameters

- base – LCDIF peripheral base address.
- background – Background color, could be defined use LCDIF_MAKE_CURSOR_COLOR
- foreground – Foreground color, could be defined use LCDIF_MAKE_CURSOR_COLOR

FSL_LCDIF_DRIVER_VERSION

enum _lcdif_polarity_flags

LCDIF signal polarity flags.

Values:

enumerator kLCDIF_VsyncActiveLow

VSYNC active low.

enumerator kLCDIF_VsyncActiveHigh

VSYNC active high.

enumerator kLCDIF_HsyncActiveLow

HSYNC active low.

enumerator kLCDIF_HsyncActiveHigh
HSYNC active high.

enumerator kLCDIF_DataEnableActiveLow
Data enable line active low.

enumerator kLCDIF_DataEnableActiveHigh
Data enable line active high.

enumerator kLCDIF_DriveDataOnFallingClkEdge
Drive data on falling clock edge, capture data on rising clock edge.

enumerator kLCDIF_DriveDataOnRisingClkEdge
Drive data on falling clock edge, capture data on rising clock edge.

enum _lcdif_output_format
LCDIF DPI output format.

Values:

enumerator kLCDIF_Output16BitConfig1
16-bit configuration 1. RGB565: XXXXXXXX_RRRRRGGG_GGGBBBBB.

enumerator kLCDIF_Output16BitConfig2
16-bit configuration 2. RGB565: XXXRRRRR_XXGGGGGG_XXXBBBBB.

enumerator kLCDIF_Output16BitConfig3
16-bit configuration 3. RGB565: XXRRRRRX_XXGGGGGG_XXBBBBBX.

enumerator kLCDIF_Output18BitConfig1
18-bit configuration 1. RGB666: XXXXXXRR_RRRRGGGG_GGBBBBBB.

enumerator kLCDIF_Output18BitConfig2
18-bit configuration 2. RGB666: XXRRRRRR_XXGGGGGG_XXBBBBBB.

enumerator kLCDIF_Output24Bit
24-bit.

enum _lcdif_fb_format
LCDIF frame buffer pixel format.

Values:

enumerator kLCDIF_PixelFormatXRGB444
XRGB4444, deprecated, use kLCDIF_PixelFormatXRGB4444 instead.

enumerator kLCDIF_PixelFormatXRGB4444
XRGB4444, 16-bit each pixel, 4-bit each element. R4G4B4 in reference manual.

enumerator kLCDIF_PixelFormatXRGB1555
XRGB1555, 16-bit each pixel, 5-bit each element. R5G5B5 in reference manual.

enumerator kLCDIF_PixelFormatRGB565
RGB565, 16-bit each pixel. R5G6B5 in reference manual.

enumerator kLCDIF_PixelFormatXRGB8888
XRGB8888, 32-bit each pixel, 8-bit each element. R8G8B8 in reference manual.

enum _lcdif_interrupt
LCDIF interrupt and status.

Values:

enumerator kLCDIF_Display0FrameDoneInterrupt
The last pixel of visible area in frame is shown.

enum _lcdif_cursor_format
LCDIF cursor format.

Values:

enumerator kLCDIF_CursorMasked
Masked format.

enumerator kLCDIF_CursorARGB8888
ARGB8888.

enum _lcdif_dbi_cmd_flag
LCDIF DBI command flag.

Values:

enumerator kLCDIF_DbiCmdAddress
Send address (or command).

enumerator kLCDIF_DbiCmdWriteMem
Start write memory.

enumerator kLCDIF_DbiCmdData
Send data.

enumerator kLCDIF_DbiCmdReadMem
Start read memory.

enum _lcdif_dbi_out_format
LCDIF DBI output format.

Values:

enumerator kLCDIF_DbiOutD8RGB332
8-bit data bus width, pixel RGB332. For type A or B. 1 pixel sent in 1 cycle.

enumerator kLCDIF_DbiOutD8RGB444
8-bit data bus width, pixel RGB444. For type A or B. 2 pixels sent in 3 cycles.

enumerator kLCDIF_DbiOutD8RGB565
8-bit data bus width, pixel RGB565. For type A or B. 1 pixel sent in 2 cycles.

enumerator kLCDIF_DbiOutD8RGB666
8-bit data bus width, pixel RGB666. For type A or B. 1 pixel sent in 3 cycles, data bus 2 LSB not used.

enumerator kLCDIF_DbiOutD8RGB888
8-bit data bus width, pixel RGB888. For type A or B. 1 pixel sent in 3 cycles.

enumerator kLCDIF_DbiOutD9RGB666
9-bit data bus width, pixel RGB666. For type A or B. 1 pixel sent in 2 cycles.

enumerator kLCDIF_DbiOutD16RGB332
16-bit data bus width, pixel RGB332. For type A or B. 2 pixels sent in 1 cycle.

enumerator kLCDIF_DbiOutD16RGB444
16-bit data bus width, pixel RGB444. For type A or B. 1 pixel sent in 1 cycle, data bus 4 MSB not used.

enumerator kLCDIF_DbiOutD16RGB565
16-bit data bus width, pixel RGB565. For type A or B. 1 pixel sent in 1 cycle.

enumerator kLCDIF_DbiOutD16RGB666Option1
16-bit data bus width, pixel RGB666. For type A or B. 2 pixels sent in 3 cycles.

enumerator kLCDIF_DbiOutD16RGB666Option2
16-bit data bus width, pixel RGB666. For type A or B. 1 pixel sent in 2 cycles.

enumerator kLCDIF_DbiOutD16RGB888Option1
16-bit data bus width, pixel RGB888. For type A or B. 2 pixels sent in 3 cycles.

enumerator kLCDIF_DbiOutD16RGB888Option2
16-bit data bus width, pixel RGB888. For type A or B. 1 pixel sent in 2 cycles.

enum _lcdif_dbi_type
LCDIF DBI type.

Values:

enumerator kLCDIF_DbiTypeA_FixedE
Selects DBI type A fixed E mode, 68000, Motorola mode.

enumerator kLCDIF_DbiTypeA_ClockedE
Selects DBI Type A Clocked E mode, 68000, Motorola mode.

enumerator kLCDIF_DbiTypeB
Selects DBI type B, 8080, Intel mode.

enum _lcdif_dbi_out_swizzle
LCDIF DBI output swizzle.

Values:

enumerator kLCDIF_DbiOutSwizzleRGB
RGB

enumerator kLCDIF_DbiOutSwizzleBGR
BGR

typedef enum *_lcdif_output_format* lcdif_output_format_t
LCDIF DPI output format.

typedef struct *_lcdif_dpi_config* lcdif_dpi_config_t
Configuration for LCDIF module to work in DBI mode.

typedef enum *_lcdif_fb_format* lcdif_fb_format_t
LCDIF frame buffer pixel format.

typedef struct *_lcdif_fb_config* lcdif_fb_config_t
LCDIF frame buffer configuration.

typedef enum *_lcdif_cursor_format* lcdif_cursor_format_t
LCDIF cursor format.

typedef struct *_lcdif_cursor_config* lcdif_cursor_config_t
LCDIF cursor configuration.

typedef struct *_lcdif_dither_config* lcdif_dither_config_t
LCDIF dither configuration.

a. Decide which bit of pixel color to enhance. This is configured by the lcdif_dither_config_t::redSize, lcdif_dither_config_t::greenSize, and lcdif_dither_config_t::blueSize. For example, setting redSize=6 means it is the 6th bit starting from the MSB that we want to enhance, in other words, it is the RedColor[2]bit from RedColor[7:0]. greenSize and blueSize function in the same way.

b. Create the look-up table. a. The Look-Up Table includes 16 entries, 4 bits for each. b. The Look-Up Table provides a value U[3:0] through the index X[1:0] and Y[1:0]. c. The color value RedColor[3:0] is used to compare with this U[3:0]. d. If RedColor[3:0] > U[3:0], and RedColor[7:2] is not 6'b111111, then the final color value is: NewRedColor = RedColor[7:2] + 1'b1. e. If RedColor[3:0] <= U[3:0], then NewRedColor = RedColor[7:2].

typedef enum *_lcdif_dbi_out_format* lcdif_dbi_out_format_t

LCDIF DBI output format.

typedef enum *_lcdif_dbi_type* lcdif_dbi_type_t

LCDIF DBI type.

typedef enum *_lcdif_dbi_out_swizzle* lcdif_dbi_out_swizzle_t

LCDIF DBI output swizzle.

typedef struct *_lcdif_dbi_config* lcdif_dbi_config_t

LCDIF DBI configuration.

LCDIF_MAKE_CURSOR_COLOR(r, g, b)

Construct the cursor color, every element should be in the range of 0 ~ 255.

LCDIF_MAKE_GAMMA_VALUE(r, g, b)

Construct the gamma value set to LCDIF gamma table, every element should be in the range of 0~255.

LCDIF_ALIGN_ADDR(addr, align)

Calculate the aligned address for LCDIF buffer.

LCDIF_FB_ALIGN

The frame buffer should be 128 byte aligned.

LCDIF_GAMMA_INDEX_MAX

Gamma index max value.

LCDIF_CURSOR_SIZE

The cursor size is 32 x 32.

LCDIF_FRAMEBUFFERCONFIG0_OUTPUT_MASK

LCDIF_ADDR_CPU_2_IP(addr)

struct __lcdif_dpi_config

#include <fsl_lcdif.h> Configuration for LCDIF module to work in DBI mode.

Public Members

uint16_t panelWidth

Display panel width, pixels per line.

uint16_t panelHeight

Display panel height, how many lines per panel.

uint8_t hsw

HSYNC pulse width.

uint8_t hfp

Horizontal front porch.

uint8_t hbp

Horizontal back porch.

uint8_t vsw
VSYNC pulse width.

uint8_t vfp
Vrtical front porch.

uint8_t vbp
Vertical back porch.

uint32_t polarityFlags
OR'ed value of _lcdif_polarity_flags, used to contol the signal polarity.

lcdif_output_format_t format
DPI output format.

struct __lcdif_fb_config
#include <fsl_lcdif.h> LCDIF frame buffer configuration.

Public Members

bool enable
Enable the frame buffer output.

bool enableGamma
Enable the gamma correction.

lcdif_fb_format_t format
Frame buffer pixel format.

struct __lcdif_cursor_config
#include <fsl_lcdif.h> LCDIF cursor configuration.

Public Members

bool enable
Enable the cursor or not.

lcdif_cursor_format_t format
Cursor format.

uint8_t hotspotOffsetX
Offset of the hotspot to top left point, range 0 ~ 31

uint8_t hotspotOffsetY
Offset of the hotspot to top left point, range 0 ~ 31

struct __lcdif_dither_config
#include <fsl_lcdif.h> LCDIF dither configuration.

a. Decide which bit of pixel color to enhance. This is configured by the lcdif_dither_config_t::redSize, lcdif_dither_config_t::greenSize, and lcdif_dither_config_t::blueSize. For example, setting redSize=6 means it is the 6th bit starting from the MSB that we want to enhance, in other words, it is the RedColor[2]bit from RedColor[7:0]. greenSize and blueSize function in the same way.

b. Create the look-up table. a. The Look-Up Table includes 16 entries, 4 bits for each. b. The Look-Up Table provides a value U[3:0] through the index X[1:0] and Y[1:0]. c. The color value RedColor[3:0] is used to compare with this U[3:0]. d. If RedColor[3:0] > U[3:0], and RedColor[7:2] is not 6'b111111, then the final color value is: NewRedColor = RedColor[7:2] + 1'b1. e. If RedColor[3:0] <= U[3:0], then NewRedColor = RedColor[7:2].

Public Members

bool enable

Enable or not.

uint8_t redSize

Red color size, valid region 4-8.

uint8_t greenSize

Green color size, valid region 4-8.

uint8_t blueSize

Blue color size, valid region 4-8.

uint32_t low

Low part of the look up table.

uint32_t high

High part of the look up table.

struct __lcdif_dbi_config

#include <fsl_lcdif.h> LCDIF DBI configuration.

Public Members

lcdif_dbi_out_swizzle_t swizzle

Swizzle.

lcdif_dbi_out_format_t format

Output format.

uint8_t acTimeUnit

Time unit for AC characteristics.

lcdif_dbi_type_t type

DBI type.

uint16_t writeWRPeriod

WR signal period, Cycle number = writeWRPeriod * (acTimeUnit + 1), must be no less than 3. Only for type A and type b.

uint8_t writeWRAssert

Cycle number = writeWRAssert * (acTimeUnit + 1), only for type A and type B. With kLCDIF_DbiTypeA_FixedE: Not used. With kLCDIF_DbiTypeA_ClockedE: Time to assert E. With kLCDIF_DbiTypeB: Time to assert WRX.

uint8_t writeCSAssert

Cycle number = writeCSAssert * (acTimeUnit + 1), only for type A and type B. With kLCDIF_DbiTypeA_FixedE: Time to assert CSX. With kLCDIF_DbiTypeA_ClockedE: Not used. With kLCDIF_DbiTypeB: Time to assert CSX.

uint16_t writeWRDeassert

Cycle number = writeWRDeassert * (acTimeUnit + 1), only for type A and type B. With kLCDIF_DbiTypeA_FixedE: Not used. With kLCDIF_DbiTypeA_ClockedE: Time to de-assert E. With kLCDIF_DbiTypeB: Time to de-assert WRX.

uint16_t writeCSDeassert

Cycle number = writeCSDeassert * (acTimeUnit + 1), only for type A and type B. With kLCDIF_DbiTypeA_FixedE: Time to de-assert CSX. With kLCDIF_DbiTypeA_ClockedE: Not used. With kLCDIF_DbiTypeB: Time to de-assert CSX.

2.30 MU: Messaging Unit

void MU_Init(MU_Type *base)

Initializes the MU module.

This function enables the MU clock only.

Parameters

- base – MU peripheral base address.

void MU_Deinit(MU_Type *base)

De-initializes the MU module.

This function disables the MU clock only.

Parameters

- base – MU peripheral base address.

static inline void MU_SendMsgNonBlocking(MU_Type *base, uint32_t regIndex, uint32_t msg)

Writes a message to the TX register.

This function writes a message to the specific TX register. It does not check whether the TX register is empty or not. The upper layer should make sure the TX register is empty before calling this function. This function can be used in ISR for better performance.

```
while (!(kMU_Tx0EmptyFlag & MU_GetStatusFlags(base))) { }  Wait for TX0 register empty.
MU_SendMsgNonBlocking(base, kMU_MsgReg0, MSG_VAL);  Write message to the TX0 register.
```

Parameters

- base – MU peripheral base address.
- regIndex – TX register index, see mu_msg_reg_index_t.
- msg – Message to send.

void MU_SendMsg(MU_Type *base, uint32_t regIndex, uint32_t msg)

Blocks to send a message.

This function waits until the TX register is empty and sends the message.

Parameters

- base – MU peripheral base address.
- regIndex – MU message register, see mu_msg_reg_index_t.
- msg – Message to send.

static inline uint32_t MU_ReceiveMsgNonBlocking(MU_Type *base, uint32_t regIndex)

Reads a message from the RX register.

This function reads a message from the specific RX register. It does not check whether the RX register is full or not. The upper layer should make sure the RX register is full before calling this function. This function can be used in ISR for better performance.

```
uint32_t msg;
while (!(kMU_Rx0FullFlag & MU_GetStatusFlags(base)))
{
}  Wait for the RX0 register full.

msg = MU_ReceiveMsgNonBlocking(base, kMU_MsgReg0);  Read message from RX0 register.
```

Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see mu_msg_reg_index_t.

Returns
The received message.

uint32_t MU_ReceiveMsg(MU_Type *base, uint32_t regIndex)

Blocks to receive a message.

This function waits until the RX register is full and receives the message.

Parameters

- base – MU peripheral base address.
- regIndex – MU message register, see mu_msg_reg_index_t

Returns
The received message.

static inline void MU_SetFlagsNonBlocking(MU_Type *base, uint32_t flags)

Sets the 3-bit MU flags reflect on the other MU side.

This function sets the 3-bit MU flags directly. Every time the 3-bit MU flags are changed, the status flag kMU_FlagsUpdatingFlag asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag kMU_FlagsUpdatingFlag is cleared by hardware. During the flags updating period, the flags cannot be changed. The upper layer should make sure the status flag kMU_FlagsUpdatingFlag is cleared before calling this function.

```
while (kMU_FlagsUpdatingFlag & MU_GetStatusFlags(base))
{
}  Wait for previous MU flags updating.

MU_SetFlagsNonBlocking(base, 0U);  Set the mU flags.
```

Parameters

- base – MU peripheral base address.
- flags – The 3-bit MU flags to set.

void MU_SetFlags(MU_Type *base, uint32_t flags)

Blocks setting the 3-bit MU flags reflect on the other MU side.

This function blocks setting the 3-bit MU flags. Every time the 3-bit MU flags are changed, the status flag kMU_FlagsUpdatingFlag asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag kMU_FlagsUpdatingFlag is cleared by hardware. During the flags updating period, the flags cannot be changed. This function waits for the MU status flag kMU_FlagsUpdatingFlag cleared and sets the 3-bit MU flags.

Parameters

- base – MU peripheral base address.
- flags – The 3-bit MU flags to set.

static inline uint32_t MU_GetFlags(MU_Type *base)

Gets the current value of the 3-bit MU flags set by the other side.

This function gets the current 3-bit MU flags on the current side.

Parameters

- base – MU peripheral base address.

Returns
flags Current value of the 3-bit flags.

static inline uint32_t MU_GetStatusFlags(MU_Type *base)

Gets the MU status flags.

This function returns the bit mask of the MU status flags. See _mu_status_flags.

```
uint32_t flags;
flags = MU_GetStatusFlags(base);  Get all status flags.
if (kMU_Tx0EmptyFlag & flags)
{
    The TX0 register is empty. Message can be sent.
    MU_SendMsgNonBlocking(base, kMU_MsgReg0, MSG0_VAL);
}
if (kMU_Tx1EmptyFlag & flags)
{
    The TX1 register is empty. Message can be sent.
    MU_SendMsgNonBlocking(base, kMU_MsgReg1, MSG1_VAL);
}
```

Parameters

- base – MU peripheral base address.

Returns
Bit mask of the MU status flags, see _mu_status_flags.

static inline uint32_t MU_GetRxStatusFlags(MU_Type *base)

Return the RX status flags.

This function return the RX status flags. Note: RFn bits of SR[27-24](mu status register) are mapped in reverse numerical order: RF0 -> SR[27] RF1 -> SR[26] RF2 -> SR[25] RF3 -> SR[24]

```
status_reg = MU_GetRxStatusFlags(base);
```

Parameters

- base – MU peripheral base address.

Returns
MU RX status

static inline uint32_t MU_GetInterruptsPending(MU_Type *base)

Gets the MU IRQ pending status of enabled interrupts.

This function returns the bit mask of the pending MU IRQs of enabled interrupts. Only these flags are checked. kMU_Tx0EmptyFlag kMU_Tx1EmptyFlag kMU_Tx2EmptyFlag kMU_Tx3EmptyFlag kMU_Rx0FullFlag kMU_Rx1FullFlag kMU_Rx2FullFlag kMU_Rx3FullFlag kMU_GenInt0Flag kMU_GenInt1Flag kMU_GenInt2Flag kMU_GenInt3Flag

Parameters

- base – MU peripheral base address.

Returns
Bit mask of the MU IRQs pending.

static inline void MU_ClearStatusFlags(MU_Type *base, uint32_t mask)

Clears the specific MU status flags.

This function clears the specific MU status flags. The flags to clear should be passed in as bit mask. See _mu_status_flags.

```
Clear general interrupt 0 and general interrupt 1 pending flags.
MU_ClearStatusFlags(base, kMU_GenInt0Flag | kMU_GenInt1Flag);
```

Parameters

- base – MU peripheral base address.
- mask – Bit mask of the MU status flags. See _mu_status_flags. The following flags are cleared by hardware, this function could not clear them.
  - kMU_Tx0EmptyFlag
  - kMU_Tx1EmptyFlag
  - kMU_Tx2EmptyFlag
  - kMU_Tx3EmptyFlag
  - kMU_Rx0FullFlag
  - kMU_Rx1FullFlag
  - kMU_Rx2FullFlag
  - kMU_Rx3FullFlag
  - kMU_EventPendingFlag
  - kMU_FlagsUpdatingFlag
  - kMU_OtherSideInResetFlag

static inline void MU_EnableInterrupts(MU_Type *base, uint32_t mask)

Enables the specific MU interrupts.

This function enables the specific MU interrupts. The interrupts to enable should be passed in as bit mask. See _mu_interrupt_enable.

```
  Enable general interrupt 0 and TX0 empty interrupt.
MU_EnableInterrupts(base, kMU_GenInt0InterruptEnable | kMU_Tx0EmptyInterruptEnable);
```

Parameters

- base – MU peripheral base address.
- mask – Bit mask of the MU interrupts. See _mu_interrupt_enable.

static inline void MU_DisableInterrupts(MU_Type *base, uint32_t mask)

Disables the specific MU interrupts.

This function disables the specific MU interrupts. The interrupts to disable should be passed in as bit mask. See _mu_interrupt_enable.

```
  Disable general interrupt 0 and TX0 empty interrupt.
MU_DisableInterrupts(base, kMU_GenInt0InterruptEnable | kMU_Tx0EmptyInterruptEnable);
```

Parameters

- base – MU peripheral base address.
- mask – Bit mask of the MU interrupts. See _mu_interrupt_enable.

status_t MU_TriggerInterrupts(MU_Type *base, uint32_t mask)

Triggers interrupts to the other core.

This function triggers the specific interrupts to the other core. The interrupts to trigger are passed in as bit mask. See _mu_interrupt_trigger. The MU should not trigger an interrupt to the other core when the previous interrupt has not been processed by the other core. This

function checks whether the previous interrupts have been processed. If not, it returns an error.

```
if (kStatus_Success != MU_TriggerInterrupts(base, kMU_GenInt0InterruptTrigger | kMU_
↪GenInt2InterruptTrigger))
{
    Previous general purpose interrupt 0 or general purpose interrupt 2
    has not been processed by the other core.
}
```

Parameters

- base – MU peripheral base address.
- mask – Bit mask of the interrupts to trigger. See _mu_interrupt_trigger.

Return values

- kStatus_Success – Interrupts have been triggered successfully.
- kStatus_Fail – Previous interrupts have not been accepted.

static inline void MU_ClearNmi(MU_Type *base)

Clear non-maskable interrupt (NMI) sent by the other core.

This function clears non-maskable interrupt (NMI) sent by the other core.

Parameters

- base – MU peripheral base address.

void MU_BootCoreB(MU_Type *base, mu_core_boot_mode_t mode)

Boots the core at B side.

This function sets the B side core's boot configuration and releases the core from reset.

Note: Only MU side A can use this function.

Parameters

- base – MU peripheral base address.
- mode – Core B boot mode.

static inline void MU_HoldCoreBReset(MU_Type *base)

Holds the core reset of B side.

This function causes the core of B side to be held in reset following any reset event.

Note: Only A side could call this function.

Parameters

- base – MU peripheral base address.

void MU_BootOtherCore(MU_Type *base, mu_core_boot_mode_t mode)

Boots the other core.

This function boots the other core with a boot configuration.

Parameters

- base – MU peripheral base address.
- mode – The other core boot mode.

static inline void MU_HoldOtherCoreReset(MU_Type *base)

Holds the other core reset.

This function causes the other core to be held in reset following any reset event.

Parameters

- base – MU peripheral base address.

static inline void 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.

Note: For some platforms, only MU side A could use this function, check reference manual for details.

Parameters

- base – MU peripheral base address.

void 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.

Example 1: Reset the other core, and no hold reset

```
MU_HardwareResetOtherCore(MU_A, true, false, bootMode);
```

In this example, the core at MU side B will reset with the specified boot mode.

Example 2: Reset the other core and hold it, then boot the other core later.

```
 Here the other core enters reset, and the reset is hold
MU_HardwareResetOtherCore(MU_A, true, true, modeDontCare);
 Current core boot the other core when necessary.
MU_BootOtherCore(MU_A, bootMode);
```

Parameters

- base – MU peripheral base address.
- waitReset – Wait the other core enters reset.
  - true: Wait until the other core enters reset, if the other core has masked the hardware reset, then this function will be blocked.
  - false: Don't wait the reset.
- holdReset – Hold the other core reset or not.
  - true: Hold the other core in reset, this function returns directly when the other core enters reset.
  - false: Don't hold the other core in reset, this function waits until the other core out of reset.
- bootMode – Boot mode of the other core, if holdReset is true, this parameter is useless.

static inline void MU_SetClockOnOtherCoreEnable(MU_Type *base, bool enable)

Enables or disables the clock on the other core.

This function enables or disables the platform clock on the other core when that core enters a stop mode. If disabled, the platform clock for the other core is disabled when it enters stop mode. If enabled, the platform clock keeps running on the other core in stop mode, until this core also enters stop mode.

Parameters

- base – MU peripheral base address.
- enable – Enable or disable the clock on the other core.

static inline mu_power_mode_t MU_GetOtherCorePowerMode(MU_Type *base)

Gets the power mode of the other core.

This function gets the power mode of the other core.

Parameters

- base – MU peripheral base address.

Returns

Power mode of the other core.

FSL_MU_DRIVER_VERSION

MU driver version.

enum _mu_status_flags

MU status flags.

Values:

enumerator kMU_Tx0EmptyFlag

TX0 empty.

enumerator kMU_Tx1EmptyFlag

TX1 empty.

enumerator kMU_Tx2EmptyFlag

TX2 empty.

enumerator kMU_Tx3EmptyFlag

TX3 empty.

enumerator kMU_Rx0FullFlag

RX0 full.

enumerator kMU_Rx1FullFlag

RX1 full.

enumerator kMU_Rx2FullFlag

RX2 full.

enumerator kMU_Rx3FullFlag

RX3 full.

enumerator kMU_GenInt0Flag

General purpose interrupt 0 pending.

enumerator kMU_GenInt1Flag

General purpose interrupt 1 pending.

enumerator kMU_GenInt2Flag

General purpose interrupt 2 pending.

enumerator kMU_GenInt3Flag
General purpose interrupt 3 pending.

enumerator kMU_EventPendingFlag
MU event pending.

enumerator kMU_FlagsUpdatingFlag
MU flags update is on-going.

enum _mu_interrupt_enable
MU interrupt source to enable.

Values:

enumerator kMU_Tx0EmptyInterruptEnable
TX0 empty.

enumerator kMU_Tx1EmptyInterruptEnable
TX1 empty.

enumerator kMU_Tx2EmptyInterruptEnable
TX2 empty.

enumerator kMU_Tx3EmptyInterruptEnable
TX3 empty.

enumerator kMU_Rx0FullInterruptEnable
RX0 full.

enumerator kMU_Rx1FullInterruptEnable
RX1 full.

enumerator kMU_Rx2FullInterruptEnable
RX2 full.

enumerator kMU_Rx3FullInterruptEnable
RX3 full.

enumerator kMU_GenInt0InterruptEnable
General purpose interrupt 0.

enumerator kMU_GenInt1InterruptEnable
General purpose interrupt 1.

enumerator kMU_GenInt2InterruptEnable
General purpose interrupt 2.

enumerator kMU_GenInt3InterruptEnable
General purpose interrupt 3.

enum _mu_interrupt_trigger
MU interrupt that could be triggered to the other core.

Values:

enumerator kMU_NmiInterruptTrigger
NMI interrupt.

enumerator kMU_GenInt0InterruptTrigger
General purpose interrupt 0.

enumerator kMU_GenInt1InterruptTrigger
General purpose interrupt 1.

enumerator kMU_GenInt2InterruptTrigger
General purpose interrupt 2.

enumerator kMU_GenInt3InterruptTrigger
General purpose interrupt 3.

enum __mu_msg_reg_index
MU message register.

Values:

enumerator kMU_MsgReg0

enumerator kMU_MsgReg1

enumerator kMU_MsgReg2

enumerator kMU_MsgReg3

typedef enum *_mu_msg_reg_index* mu_msg_reg_index_t
MU message register.

MU_CR_NMI_MASK

MU_GET_CORE_FLAG(flags)

MU_GET_STAT_FLAG(flags)

MU_GET_TX_FLAG(flags)

MU_GET_RX_FLAG(flags)

MU_GET_GI_FLAG(flags)

2.31 OCOTP: On Chip One-Time Programmable controller.

FSL_OCOTP_DRIVER_VERSION
OCOTP driver version.

_ocotp_status Error codes for the OCOTP driver.

Values:

enumerator kStatus_OCOTP_AccessError
eFuse and shadow register access error.

enumerator kStatus_OCOTP_CrcFail
CRC check failed.

enumerator kStatus_OCOTP_ReloadError
Error happens during reload shadow register.

enumerator kStatus_OCOTP_ProgramFail
Fuse programming failed.

enumerator kStatus_OCOTP_Locked
Fuse is locked and cannot be programmed.

void OCOTP_Init(OCOTP_Type *base, uint32_t srcClock_Hz)

Initializes OCOTP controller.

Parameters

- base – OCOTP peripheral base address.
- srcClock_Hz – source clock frequency in unit of Hz. When the macro FSL_FEATURE_OCOTP_HAS_TIMING_CTRL is defined as 0, this parameter is not used, application could pass in 0 in this case.

void OCOTP_Deinit(OCOTP_Type *base)

De-initializes OCOTP controller.

Return values

kStatus_Success – upon successful execution, error status otherwise.

static inline bool OCOTP_CheckBusyStatus(OCOTP_Type *base)

Checking the BUSY bit in CTRL register. Checking this BUSY bit will help confirm if the OCOTP controller is ready for access.

Parameters

- base – OCOTP peripheral base address.

Return values

true – for bit set and false for cleared.

static inline bool OCOTP_CheckErrorStatus(OCOTP_Type *base)

Checking the ERROR bit in CTRL register.

Parameters

- base – OCOTP peripheral base address.

Return values

true – for bit set and false for cleared.

static inline void OCOTP_ClearErrorStatus(OCOTP_Type *base)

Clear the error bit if this bit is set.

Parameters

- base – OCOTP peripheral base address.

status_t OCOTP_ReloadShadowRegister(OCOTP_Type *base)

Reload the shadow register. This function will help reload the shadow register without reseting the OCOTP module. Please make sure the OCOTP has been initialized before calling this API.

Parameters

- base – OCOTP peripheral base addess.

Return values

- kStatus_Success – Reload success.
- kStatus_OCOTP_ReloadError – Reload failed.

uint32_t OCOTP_ReadFuseShadowRegister(OCOTP_Type *base, uint32_t address)

Read the fuse shadow register with the fuse addess.

Deprecated:

Use OCOTP_ReadFuseShadowRegisterExt instead of this function.

Parameters

- base – OCOTP peripheral base address.
- address – the fuse address to be read from.

Returns
The read out data.

status_t OCOTP_ReadFuseShadowRegisterExt(OCOTP_Type *base, uint32_t address, uint32_t *data, uint8_t fuseWords)

Read the fuse shadow register from the fuse addess.

This function reads fuse from address, how many words to read is specified by the parameter fuseWords. This function could read at most OCOTP_READ_FUSE_DATA_COUNT fuse word one time.

Parameters

- base – OCOTP peripheral base address.
- address – the fuse address to be read from.
- data – Data array to save the readout fuse value.
- fuseWords – How many words to read.

Return values

- kStatus_Success – Read success.
- kStatus_Fail – Error occurs during read.

status_t OCOTP_WriteFuseShadowRegister(OCOTP_Type *base, uint32_t address, uint32_t data)

Write the fuse shadow register with the fuse addess and data. Please make sure the wrtie address is not locked while calling this API.

Parameters

- base – OCOTP peripheral base address.
- address – the fuse address to be written.
- data – the value will be writen to fuse address.

Return values
write – status, kStatus_Success for success and kStatus_Fail for failed.

status_t OCOTP_WriteFuseShadowRegisterWithLock(OCOTP_Type *base, uint32_t address, uint32_t data, bool lock)

Write the fuse shadow register and lock it.

Please make sure the wrtie address is not locked while calling this API.

Some OCOTP controller supports ECC mode and redundancy mode (see reference mananual for more details). OCOTP controller will auto select ECC or redundancy mode to program the fuse word according to fuse map definition. In ECC mode, the 32 fuse bits in one word can only be written once. In redundancy mode, the word can be written more than once as long as they are different fuse bits. Set parameter lock as true to force use ECC mode.

Parameters

- base – OCOTP peripheral base address.
- address – The fuse address to be written.
- data – The value will be writen to fuse address.
- lock – Lock or unlock write fuse shadow register operation.

Return values

- kStatus_Success – Program and reload success.

- kStatus_OCOTP_Locked – The eFuse word is locked and cannot be programmed.
- kStatus_OCOTP_ProgramFail – eFuse word programming failed.
- kStatus_OCOTP_ReloadError – eFuse word programming success, but error happens during reload the values.
- kStatus_OCOTP_AccessError – Cannot access eFuse word.

static inline uint32_t OCOTP_GetVersion(OCOTP_Type *base)

Get the OCOTP controller version from the register.

Parameters

- base – OCOTP peripheral base address.

Return values

return – the version value.

OCOTP_READ_FUSE_DATA_COUNT

2.32 PDM: Microphone Interface

2.33 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_EnableDoze(PDM_Type *base, bool enable)

Enables/disables DOZE.

Parameters

- base – PDM base pointer
- enable – True means the module will enter Disable/Low Leakage mode when ipg_doze is asserted, false means the module will not enter Disable/Low Leakage mode when ipg_doze is asserted.

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_GetOutputStatus(PDM_Type *base)

Gets the PDM output status flag. Use the Status Mask in _pdm_output_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_ClearOutputStatus(PDM_Type *base, uint32_t mask)

Clears the PDM output status.

Parameters

- base – PDM base pointer
- mask – State mask. It can be a combination of the status in _pdm_output_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_SetHwvadConfig(PDM_Type *base, const *pdm_hwvad_config_t* *config)

Configure voice activity detector.

Parameters

- base – PDM base pointer
- config – Voice activity detector configure structure pointer .

static inline void PDM_ForceHwvadOutputDisable(PDM_Type *base, bool enable)

PDM hwvad force output disable.

Parameters

- base – PDM base pointer
- enable – true is output force disable, false is output not force.

static inline void PDM_ResetHwvad(PDM_Type *base)

PDM hwvad reset. It will reset VADNDATA register and will clean all internal buffers, should be called when the PDM isn't running.

Parameters

- base – PDM base pointer

static inline void PDM_EnableHwvad(PDM_Type *base, bool enable)

Enable/Disable Voice activity detector. Should be called when the PDM isn't running.

Parameters

- base – PDM base pointer.
- enable – True means enable voice activity detector, false means disable.

static inline void PDM_EnableHwvadInterrupts(PDM_Type *base, uint32_t mask)

Enables the PDM Voice Detector interrupt requests.

Parameters

- base – PDM base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kPDM_HWVADErrorInterruptEnable
  - kPDM_HWVADInterruptEnable

static inline void PDM_DisableHwvadInterrupts(PDM_Type *base, uint32_t mask)

Disables the PDM Voice Detector interrupt requests.

Parameters

- base – PDM base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kPDM_HWVADErrorInterruptEnable
  - kPDM_HWVADInterruptEnable

static inline void PDM_ClearHwvadInterruptStatusFlags(PDM_Type *base, uint32_t mask)

Clears the PDM voice activity detector status flags.

Parameters

- base – PDM base pointer
- mask – State mask,reference _pdm_hwvad_int_status.

static inline uint32_t PDM_GetHwvadInterruptStatusFlags(PDM_Type *base)

Clears the PDM voice activity detector status flags.

Parameters

- base – PDM base pointer

Returns

status, reference _pdm_hwvad_int_status

static inline uint32_t PDM_GetHwvadInitialFlag(PDM_Type *base)

Get the PDM voice activity detector initial flags.

Parameters

- base – PDM base pointer

Returns

initial flag.

static inline uint32_t PDM_GetHwvadVoiceDetectedFlag(PDM_Type *base)

Get the PDM voice activity detector voice detected flags. NOte: this flag is auto cleared when voice gone.

Parameters

- base – PDM base pointer

Returns

voice detected flag.

static inline void PDM_EnableHwvadSignalFilter(PDM_Type *base, bool enable)

Enables/disables voice activity detector signal filter.

Parameters

- base – PDM base pointer
- enable – True means enable signal filter, false means disable.

void PDM_SetHwvadSignalFilterConfig(PDM_Type *base, bool enableMaxBlock, uint32_t signalGain)

Configure voice activity detector signal filter.

Parameters

- base – PDM base pointer
- enableMaxBlock – If signal maximum block enabled.
- signalGain – Gain value for the signal energy.

void PDM_SetHwvadNoiseFilterConfig(PDM_Type *base, const *pdm_hwvad_noise_filter_t* *config)

Configure voice activity detector noise filter.

Parameters

- base – PDM base pointer
- config – Voice activity detector noise filter configure structure pointer .

static inline void PDM_EnableHwvadZeroCrossDetector(PDM_Type *base, bool enable)

Enables/disables voice activity detector zero cross detector.

Parameters

- base – PDM base pointer
- enable – True means enable zero cross detector, false means disable.

void PDM_SetHwvadZeroCrossDetectorConfig(PDM_Type *base, const *pdm_hwvad_zero_cross_detector_t* *config)

Configure voice activity detector zero cross detector.

Parameters

- base – PDM base pointer
- config – Voice activity detector zero cross detector configure structure pointer .

static inline uint16_t PDM_GetNoiseData(PDM_Type *base)

Reads noise data.

Parameters

- base – PDM base pointer.

Returns

Data in PDM noise data register.

static inline void PDM_SetHwvadInternalFilterStatus(PDM_Type *base, *pdm_hwvad_filter_status_t* status)

set hwvad internal filter status . Note: filter initial status should be asserted for two more cycles, then set it to normal operation.

Parameters

- base – PDM base pointer.
- status – internal filter status.

void PDM_SetHwvadInEnvelopeBasedMode(PDM_Type *base, const *pdm_hwvad_config_t* *hwvadConfig, const *pdm_hwvad_noise_filter_t* *noiseConfig, const *pdm_hwvad_zero_cross_detector_t* *zcdConfig, uint32_t signalGain)

set HWVAD in envelope based mode . Recommand configurations,

```
  static const pdm_hwvad_config_t hwvadConfig = {
    .channel           = 0,
    .initializeTime    = 10U,
    .cicOverSampleRate = 0U,
    .inputGain         = 0U,
    .frameTime         = 10U,
    .cutOffFreq        = kPDM_HwvadHpfBypassed,
    .enableFrameEnergy = false,
    .enablePreFilter   = true,
};

  static const pdm_hwvad_noise_filter_t noiseFilterConfig = {
    .enableAutoNoiseFilter = false,
    .enableNoiseMin        = true,
    .enableNoiseDecimation = true,
    .noiseFilterAdjustment = 0U,
    .noiseGain             = 7U,
    .enableNoiseDetectOR   = true,
  };
```

Parameters

- base – PDM base pointer.
- hwvadConfig – internal filter status.

- noiseConfig – Voice activity detector noise filter configure structure pointer.
- zcdConfig – Voice activity detector zero cross detector configure structure pointer .
- signalGain – signal gain value.

void PDM_SetHwvadInEnergyBasedMode(PDM_Type *base, const *pdm_hwvad_config_t* *hwvadConfig, const *pdm_hwvad_noise_filter_t* *noiseConfig, const *pdm_hwvad_zero_cross_detector_t* *zcdConfig, uint32_t signalGain)

brief set HWVAD in energy based mode . Recommand configurations, code static const pdm_hwvad_config_t hwvadConfig = { .channel = 0, .initializeTime = 10U, .cicOverSampleRate = 0U, .inputGain = 0U, .frameTime = 10U, .cutOffFreq = kPDM_HwvadHpfBypassed, .enableFrameEnergy = true, .enablePreFilter = true, };

static const pdm_hwvad_noise_filter_t noiseFilterConfig = { .enableAutoNoiseFilter = true, .enableNoiseMin = false, .enableNoiseDecimation = false, .noiseFilterAdjustment = 0U, .noiseGain = 7U, .enableNoiseDetectOR = false, }; code param base PDM base pointer. param hwvadConfig internal filter status. param noiseConfig Voice activity detector noise filter configure structure pointer. param zcdConfig Voice activity detector zero cross detector configure structure pointer . param signalGain signal gain value, signal gain value should be properly according to application.

void PDM_EnableHwvadInterruptCallback(PDM_Type *base, *pdm_hwvad_callback_t* vadCallback, void *userData, bool enable)

Enable/Disable hwvad callback.

This function enable/disable the hwvad interrupt for the selected PDM peripheral.

Parameters

- base – Base address of the PDM peripheral.
- vadCallback – callback Pointer to store callback function, should be NULL when disable.
- userData – user data.
- enable – true is enable, false is disable.

Return values

None. –

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 2.9.2

PDM return status.

Values:

enumerator kStatus_PDM_Busy
PDM is busy.

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

enumerator kStatus_PDM_HWVAD_VoiceDetected
PDM hwvad voice detected

enumerator kStatus_PDM_HWVAD_Error
PDM hwvad error

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_StatusFIRFilterReady
FIR filter data is ready

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

enumerator kPDM_StatusCh4FifoDataAvaliable
channel 4 fifo data reached watermark level

enumerator kPDM_StatusCh5FifoDataAvaliable
channel 5 fifo data reached watermark level

enumerator kPDM_StatusCh6FifoDataAvaliable
channel 6 fifo data reached watermark level

enumerator kPDM_StatusCh7FifoDataAvaliable
channel 7 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_EnableChannel4
channgel 4 enable mask

enumerator kPDM_EnableChannel5
channgel 5 enable mask

enumerator kPDM_EnableChannel6
channgel 6 enable mask

enumerator kPDM_EnableChannel7
channgel 7 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_FifoStatusUnderflowCh4
channel4 fifo status underflow

enumerator kPDM_FifoStatusUnderflowCh5
channel5 fifo status underflow

enumerator kPDM_FifoStatusUnderflowCh6
channel6 fifo status underflow

enumerator kPDM_FifoStatusUnderflowCh7
channel7 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

enumerator kPDM_FifoStatusOverflowCh4
channel4 fifo status overflow

enumerator kPDM_FifoStatusOverflowCh5
channel5 fifo status overflow

enumerator kPDM_FifoStatusOverflowCh6
channel6 fifo status overflow

enumerator kPDM_FifoStatusOverflowCh7
channel7 fifo status overflow

enum _pdm_output_status
The PDM output status.

Values:

enumerator kPDM_OutputStatusUnderFlowCh0
channel0 output status underflow

enumerator kPDM_OutputStatusUnderFlowCh1
channel1 output status underflow

enumerator kPDM_OutputStatusUnderFlowCh2
channel2 output status underflow

enumerator kPDM_OutputStatusUnderFlowCh3
channel3 output status underflow

enumerator kPDM_OutputStatusUnderFlowCh4
channel4 output status underflow

enumerator kPDM_OutputStatusUnderFlowCh5
channel5 output status underflow

enumerator kPDM_OutputStatusUnderFlowCh6
channel6 output status underflow

enumerator kPDM_OutputStatusUnderFlowCh7
channel7 output status underflow

enumerator kPDM_OutputStatusOverFlowCh0
channel0 output status overflow

enumerator kPDM_OutputStatusOverFlowCh1
channel1 output status overflow

enumerator kPDM_OutputStatusOverFlowCh2
channel2 output status overflow

enumerator kPDM_OutputStatusOverFlowCh3
channel3 output status overflow

enumerator kPDM_OutputStatusOverFlowCh4
channel4 output status overflow

enumerator kPDM_OutputStatusOverFlowCh5
channel5 output status overflow

enumerator kPDM_OutputStatusOverFlowCh6
channel6 output status overflow

enumerator kPDM_OutputStatusOverFlowCh7
channel7 output status overflow

enum _pdm_dc_remover
PDM DC remover configurations.

Values:

enumerator kPDM_DcRemoverCutOff21Hz
DC remover cut off 21HZ

enumerator kPDM_DcRemoverCutOff83Hz
DC remover cut off 83HZ

enumerator kPDM_DcRemoverCutOff152Hz
DC remover cut off 152HZ

enumerator kPDM_DcRemoverBypass
DC remover bypass

enum _pdm_df_quality_mode
PDM decimation filter quality mode.

Values:

enumerator kPDM_QualityModeMedium
quality mode memdium

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_DataWdith16
PDM data width 16bit

enum _pdm_hwvad_interrupt_enable
PDM voice activity detector interrupt type.

Values:

enumerator kPDM_HwvadErrorInterruptEnable
PDM channel HWVAD error interrupt enable.

enumerator kPDM_HwvadInterruptEnable
PDM channel HWVAD interrupt

enum _pdm_hwvad_int_status
The PDM hwvad interrupt status flag.

Values:

enumerator kPDM_HwvadStatusInputSaturation
HWVAD saturation condition

enumerator kPDM_HwvadStatusVoiceDetectFlag
HWVAD voice detect interrupt triggered

enum _pdm_hwvad_hpf_config
High pass filter configure cut-off frequency.

Values:

enumerator kPDM_HwvadHpfBypassed
High-pass filter bypass

enumerator kPDM_HwvadHpfCutOffFreq1750Hz
High-pass filter cut off frequency 1750HZ

enumerator kPDM_HwvadHpfCutOffFreq215Hz
High-pass filter cut off frequency 215HZ

enumerator kPDM_HwvadHpfCutOffFreq102Hz
High-pass filter cut off frequency 102HZ

enum _pdm_hwvad_filter_status
HWVAD internal filter status.

Values:

enumerator kPDM_HwvadInternalFilterNormalOperation
internal filter ready for normal operation

enumerator kPDM_HwvadInternalFilterInitial
interla filter are initial

enum _pdm_hwvad_zcd_result
PDM voice activity detector zero cross detector result.

Values:

enumerator kPDM_HwvadResultOREnergyBasedDetection
zero cross detector result will be OR with energy based detection

enumerator kPDM_HwvadResultANDEnergyBasedDetection
zero cross detector result will be AND with energy based detection

typedef enum *_pdm_dc_remover* pdm_dc_remover_t
PDM DC remover 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 enum *_pdm_hwvad_hpf_config* pdm_hwvad_hpf_config_t
High pass filter configure cut-off frequency.

typedef enum *_pdm_hwvad_filter_status* pdm_hwvad_filter_status_t
HWVAD internal filter status.

typedef struct *_pdm_hwvad_config* pdm_hwvad_config_t
PDM voice activity detector user configuration structure.

typedef struct *_pdm_hwvad_noise_filter* pdm_hwvad_noise_filter_t
PDM voice activity detector noise filter user configuration structure.

typedef enum *_pdm_hwvad_zcd_result* pdm_hwvad_zcd_result_t
PDM voice activity detector zero cross detector result.

typedef struct *_pdm_hwvad_zero_cross_detector* pdm_hwvad_zero_cross_detector_t
PDM voice activity detector zero cross detector 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.

typedef void (*pdm_hwvad_callback_t)(*status_t* status, void *userData)
PDM HWVAD callback prototype.

typedef struct *_pdm_hwvad_notification* pdm_hwvad_notification_t
PDM HWVAD notification structure.

PDM_XFER_QUEUE_SIZE
PDM XFER QUEUE SIZE.

struct __pdm_channel_config
#include <fsl_pdm.h> PDM channel configurations.

Public Members

pdm_dc_remover_t cutOffFreq
DC remover 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

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__hwvad__config
#include <fsl_pdm.h> PDM voice activity detector user configuration structure.

Public Members

uint8_t channel
Which channel uses voice activity detector

uint8_t initializeTime
Number of frames or samples to initialize voice activity detector.

uint8_t cicOverSampleRate
CIC filter over sampling rate

uint8_t inputGain
Voice activity detector input gain

uint32_t frameTime
Voice activity frame time

pdm_hwvad_hpf_config_t cutOffFreq
High pass filter cut off frequency

bool enableFrameEnergy
If frame energy enabled, true means enable

bool enablePreFilter
If pre-filter enabled

struct __pdm__hwvad__noise__filter
#include <fsl_pdm.h> PDM voice activity detector noise filter user configuration structure.

Public Members

bool enableAutoNoiseFilter
: If noise fileter automatically activated, true means enable

bool enableNoiseMin
: If Noise minimum block enabled, true means enabled

bool enableNoiseDecimation
: If enable noise input decimation

bool enableNoiseDetectOR
: Enables a OR logic in the output of minimum noise estimator block

uint32_t noiseFilterAdjustment
: The adjustment value of the noise filter

uint32_t noiseGain
: Gain value for the noise energy or envelope estimated

struct __pdm_hwvad_zero_cross_detector
: *#include <fsl_pdm.h>* PDM voice activity detector zero cross detector configuration structure.

Public Members

bool enableAutoThreshold
: If ZCD auto-threshold enabled, true means enabled.

pdm_hwvad_zcd_result_t zcdAnd
: Is ZCD result is AND'ed with energy-based detection, false means OR'ed

uint32_t threshold
: The adjustment value of the noise filter

uint32_t adjustmentThreshold
: Gain value for the noise energy or envelope estimated

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_hwvad_notification
: *#include <fsl_pdm.h>* PDM HWVAD notification structure.

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

2.34 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.

a. Scatter gather case: This functio support dynamic scatter gather and staic 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],
```

(continues on next page)

(continued from previous page)

```
    .dataSize = BUFFER_SIZE,
    .linkTransfer = &pdmXfer[0]
    },
};
```

b. 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

2.35 PDM SDMA Driver

void PDM_TransferCreateHandleSDMA(PDM_Type *base, *pdm_sdma_handle_t* *handle, *pdm_sdma_callback_t* callback, void *userData, *sdma_handle_t* *dmaHandle, uint32_t eventSource)

Initializes the PDM eDMA handle.

This function initializes the PDM 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.
- eventSource – PDM event source number.

status_t PDM_TransferReceiveSDMA(PDM_Type *base, *pdm_sdma_handle_t* *handle, *pdm_transfer_t* *xfer)

Performs a non-blocking PDM receive using eDMA.

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.

Parameters

- base – PDM base pointer
- handle – PDM eDMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a PDM eDMA receive successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_RxBusy – PDM is busy receiving data.

void PDM_TransferAbortReceiveSDMA(PDM_Type *base, *pdm_sdma_handle_t* *handle)

Aborts a PDM receive using eDMA.

Parameters

- base – PDM base pointer
- handle – PDM eDMA handle pointer.

void PDM_SetChannelConfigSDMA(PDM_Type *base, *pdm_sdma_handle_t* *handle, uint32_t channel, const *pdm_channel_config_t* *config)

PDM channel configurations.

Parameters

- base – PDM base pointer.
- handle – PDM eDMA handle pointer.
- channel – channel number.
- config – channel configurations.

void PDM_TransferTerminateReceiveSDMA(PDM_Type *base, *pdm_sdma_handle_t* *handle)

Terminate all the PDM sdma receive transfer.

Parameters

- base – PDM base pointer.
- handle – PDM SDMA handle pointer.

FSL_PDM_SDMA_DRIVER_VERSION

Version 2.7.1

typedef struct *_pdm_sdma_handle* pdm_sdma_handle_t

typedef void (*pdm_sdma_callback_t)(PDM_Type *base, *pdm_sdma_handle_t* *handle, *status_t* status, void *userData)

PDM eDMA transfer callback function for finish and error.

struct __pdm_sdma_handle

#include <fsl_pdm_sdma.h> PDM DMA transfer handle, users should not touch the content of the handle.

Public Members

sdma_handle_t *dmaHandle

DMA handler for PDM send

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint8_t fifoWidth

fifo width

uint8_t endChannel

The last enabled channel

uint8_t channelNums

total channel numbers

uint32_t count

The transfer data count in a DMA request

uint32_t state

Internal state for PDM eDMA transfer

uint32_t eventSource

PDM event source number

pdm_sdma_callback_t callback

Callback for users while transfer finish or error occurs

void *userData

User callback parameter

sdma_buffer_descriptor_t bdPool[(4U)]

BD pool for SDMA transfer.

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

2.36 PWM: Pulse Width Modulation Driver

status_t PWM_Init(PWM_Type *base, const *pwm_config_t* *config)

Ungates the PWM clock and configures the peripheral for basic operation.

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

Parameters

- base – PWM peripheral base address
- config – Pointer to user's PWM config structure.

Returns

kStatus_Success means success; else failed.

void PWM_Deinit(PWM_Type *base)

Gate the PWM submodule clock.

Parameters

- base – PWM peripheral base address

void PWM_GetDefaultConfig(*pwm_config_t* *config)

Fill in the PWM config struct with the default settings.

The default values are:

```
config->enableStopMode = false;
config->enableDozeMode = false;
config->enableWaitMode = false;
config->enableDozeMode = false;
config->clockSource = kPWM_LowFrequencyClock;
config->prescale = 0U;
config->outputConfig = kPWM_SetAtRolloverAndClearAtcomparison;
config->fifoWater = kPWM_FIFOWaterMark_2;
config->sampleRepeat = kPWM_EachSampleOnce;
config->byteSwap = kPWM_ByteNoSwap;
config->halfWordSwap = kPWM_HalfWordNoSwap;
```

Parameters

- config – Pointer to user's PWM config structure.

static inline void PWM_StartTimer(PWM_Type *base)

Starts the PWM counter when the PWM is enabled.

When the PWM is enabled, it begins a new period, the output pin is set to start a new period while the prescaler and counter are released and counting begins.

Parameters

- base – PWM peripheral base address

static inline void PWM_StopTimer(PWM_Type *base)

Stops the PWM counter when the pwm is disabled.

Parameters

- base – PWM peripheral base address

static inline void PWM_EnableInterrupts(PWM_Type *base, uint32_t mask)

Enables the selected PWM interrupts.

Parameters

- base – PWM peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration pwm_interrupt_enable_t

static inline void PWM_DisableInterrupts(PWM_Type *base, uint32_t mask)

Disables the selected PWM interrupts.

Parameters

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

static inline uint32_t PWM_GetEnabledInterrupts(PWM_Type *base)

Gets the enabled PWM interrupts.

Parameters

- base – PWM peripheral base address

Returns

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

static inline uint32_t PWM_GetStatusFlags(PWM_Type *base)

Gets the PWM status flags.

Parameters

- base – PWM peripheral base address

Returns

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

static inline void PWM_clearStatusFlags(PWM_Type *base, uint32_t mask)

Clears the PWM status flags.

Parameters

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

static inline uint32_t PWM_GetFIFOAvailable(PWM_Type *base)

Gets the PWM FIFO available.

Parameters

- base – PWM peripheral base address

Returns

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

static inline void PWM_SetSampleValue(PWM_Type *base, uint32_t value)

Sets the PWM sample value.

Parameters

- base – PWM peripheral base address
- value – The sample value. This is the input to the 4x16 FIFO. The value in this register denotes the value of the sample being currently used.

static inline uint32_t PWM_GetSampleValue(PWM_Type *base)

Gets the PWM sample value.

Parameters

- base – PWM peripheral base address

Returns

The sample value. It can be read only when the PWM is enable.

FSL_PWM_DRIVER_VERSION

enum _pwm_clock_source

PWM clock source select.

Values:

enumerator kPWM_PeripheralClock

The Peripheral clock is used as the clock

enumerator kPWM_HighFrequencyClock

High-frequency reference clock is used as the clock

enumerator kPWM_LowFrequencyClock

Low-frequency reference clock(32KHz) is used as the clock

enum _pwm_fifo_water_mark

PWM FIFO water mark select. Sets the data level at which the FIFO empty flag will be set.

Values:

enumerator kPWM_FIFOWaterMark_1

FIFO empty flag is set when there are more than or equal to 1 empty slots

enumerator kPWM_FIFOWaterMark_2

FIFO empty flag is set when there are more than or equal to 2 empty slots

enumerator kPWM_FIFOWaterMark_3

FIFO empty flag is set when there are more than or equal to 3 empty slots

enumerator kPWM_FIFOWaterMark_4

FIFO empty flag is set when there are more than or equal to 4 empty slots

enum _pwm_byte_data_swap

PWM byte data swap select. It determines the byte ordering of the 16-bit data when it goes into the FIFO from the sample register.

Values:

enumerator kPWM_ByteNoSwap

byte ordering remains the same

enumerator kPWM_ByteSwap

byte ordering is reversed

enum _pwm_half_word_data_swap

PWM half-word data swap select.

Values:

enumerator kPWM_HalfWordNoSwap

Half word swapping does not take place

enumerator kPWM_HalfWordSwap

Half word from write data bus are swapped

enum _pwm_output_configuration

PWM Output Configuration.

Values:

enumerator kPWM_SetAtRolloverAndClearAtcomparison

Output pin is set at rollover and cleared at comparison

enumerator kPWM_ClearAtRolloverAndSetAtcomparison

Output pin is cleared at rollover and set at comparison

enumerator kPWM_NoConfigure
PWM output is disconnected

enum _pwm_sample_repeat
PWM FIFO sample repeat It determines the number of times each sample from the FIFO is to be used.

Values:

enumerator kPWM_EachSampleOnce
Use each sample once

enumerator kPWM_EachSampletwice
Use each sample twice

enumerator kPWM_EachSampleFourTimes
Use each sample four times

enumerator kPWM_EachSampleEightTimes
Use each sample eight times

enum _pwm_interrupt_enable
List of PWM interrupt options.

Values:

enumerator kPWM_FIFOEmptyInterruptEnable
This bit controls the generation of the FIFO Empty interrupt.

enumerator kPWM_RolloverInterruptEnable
This bit controls the generation of the Rollover interrupt.

enumerator kPWM_CompareInterruptEnable
This bit controls the generation of the Compare interrupt

enum _pwm_status_flags
List of PWM status flags.

Values:

enumerator kPWM_FIFOEmptyFlag
This bit indicates the FIFO data level in comparison to the water level set by FWM field in the control register.

enumerator kPWM_RolloverFlag
This bit shows that a roll-over event has occurred.

enumerator kPWM_CompareFlag
This bit shows that a compare event has occurred.

enumerator kPWM_FIFOWriteErrorFlag
This bit shows that an attempt has been made to write FIFO when it is full.

enum _pwm_fifo_available
List of PWM FIFO available.

Values:

enumerator kPWM_NoDataInFIFOFlag
No data available

enumerator kPWM_OneWordInFIFOFlag
1 word of data in FIFO

enumerator kPWM_TwoWordsInFIFOFlag
2 word of data in FIFO

enumerator kPWM_ThreeWordsInFIFOFlag
3 word of data in FIFO

enumerator kPWM_FourWordsInFIFOFlag
4 word of data in FIFO

typedef enum *_pwm_clock_source* pwm_clock_source_t
PWM clock source select.

typedef enum *_pwm_fifo_water_mark* pwm_fifo_water_mark_t
PWM FIFO water mark select. Sets the data level at which the FIFO empty flag will be set.

typedef enum *_pwm_byte_data_swap* pwm_byte_data_swap_t
PWM byte data swap select. It determines the byte ordering of the 16-bit data when it goes into the FIFO from the sample register.

typedef enum *_pwm_half_word_data_swap* pwm_half_word_data_swap_t
PWM half-word data swap select.

typedef enum *_pwm_output_configuration* pwm_output_configuration_t
PWM Output Configuration.

typedef enum *_pwm_sample_repeat* pwm_sample_repeat_t
PWM FIFO sample repeat It determines the number of times each sample from the FIFO is to be used.

typedef enum *_pwm_interrupt_enable* pwm_interrupt_enable_t
List of PWM interrupt options.

typedef enum *_pwm_status_flags* pwm_status_flags_t
List of PWM status flags.

typedef enum *_pwm_fifo_available* pwm_fifo_available_t
List of PWM FIFO available.

typedef struct *_pwm_config* pwm_config_t

static inline void PWM_SoftwareReset(PWM_Type *base)
Sofrware reset.

PWM is reset when this bit is set to 1. It is a self clearing bit. Setting this bit resets all the registers to their reset values except for the STOPEN, DOZEN, WAITEN, and DBGEN bits in this control register.

Parameters

- base – PWM peripheral base address

static inline void PWM_SetPeriodValue(PWM_Type *base, uint32_t value)
Sets the PWM period value.

Parameters

- base – PWM peripheral base address
- value – The period value. The PWM period register (PWM_PWMPR) determines the period of the PWM output signal. Writing 0xFFFF to this register will achieve the same result as writing 0xFFFE. PWMO (Hz) = PCLK(Hz) / (period +2)

static inline uint32_t PWM_GetPeriodValue(PWM_Type *base)

Gets the PWM period value.

Parameters

- base – PWM peripheral base address

Returns

The period value. The PWM period register (PWM_PWMPR) determines the period of the PWM output signal.

static inline uint32_t PWM_GetCounterValue(PWM_Type *base)

Gets the PWM counter value.

Parameters

- base – PWM peripheral base address

Returns

The counter value. The current count value.

struct __pwm_config

#include <fsl_pwm.h>

Public Members

bool enableStopMode

True: PWM continues to run in stop mode; False: PWM is paused in stop mode.

bool enableDozeMode

True: PWM continues to run in doze mode; False: PWM is paused in doze mode.

bool enableWaitMode

True: PWM continues to run in wait mode; False: PWM is paused in wait mode.

bool enableDebugMode

True: PWM continues to run in debug mode; False: PWM is paused in debug mode.

uint16_t prescale

Pre-scaler to divide down the clock The prescaler value is not more than 0xFFF. Divide by (value + 1)

pwm_clock_source_t clockSource

Clock source for the counter

pwm_output_configuration_t outputConfig

Set the mode of the PWM output on the output pin.

pwm_fifo_water_mark_t fifoWater

Set the data level for FIFO.

pwm_sample_repeat_t sampleRepeat

The number of times each sample from the FIFO is to be used.

pwm_byte_data_swap_t byteSwap

It determines the byte ordering of the 16-bit data when it goes into the FIFO from the sample register.

pwm_half_word_data_swap_t halfWordSwap

It determines which half word data from the 32-bit IP Bus interface is written into the lower 16 bits of the sample register.

2.37 RDC: Resource Domain Controller

enum _rdc_interrupts

RDC interrupts.

Values:

enumerator kRDC_RestoreCompleteInterrupt

Interrupt generated when the RDC has completed restoring state to a recently re-powered memory regions.

enum _rdc_flags

RDC status.

Values:

enumerator kRDC_PowerDownDomainOn

Power down domain is ON.

enum _rdc_access_policy

Access permission policy.

Values:

enumerator kRDC_NoAccess

Could not read or write.

enumerator kRDC_WriteOnly

Write only.

enumerator kRDC_ReadOnly

Read only.

enumerator kRDC_ReadWrite

Read and write.

typedef struct *_rdc_hardware_config* rdc_hardware_config_t

RDC hardware configuration.

typedef struct *_rdc_domain_assignment* rdc_domain_assignment_t

Master domain assignment.

typedef struct *_rdc_periph_access_config* rdc_periph_access_config_t

Peripheral domain access permission configuration.

typedef struct *_rdc_mem_access_config* rdc_mem_access_config_t

Memory region domain access control configuration.

Note that when setting the rdc_mem_access_config_t::baseAddress and rdc_mem_access_config_t::endAddress, should be aligned to the region resolution, see rdc_mem_t definitions.

typedef struct *_rdc_mem_status* rdc_mem_status_t

Memory region access violation status.

void RDC_Init(RDC_Type *base)

Initializes the RDC module.

This function enables the RDC clock.

Parameters

- base – RDC peripheral base address.

void RDC_Deinit(RDC_Type *base)

De-initializes the RDC module.

This function disables the RDC clock.

Parameters

- base – RDC peripheral base address.

void RDC_GetHardwareConfig(RDC_Type *base, *rdc_hardware_config_t* *config)

Gets the RDC hardware configuration.

This function gets the RDC hardware configurations, including number of bus masters, number of domains, number of memory regions and number of peripherals.

Parameters

- base – RDC peripheral base address.
- config – Pointer to the structure to get the configuration.

static inline void RDC_EnableInterrupts(RDC_Type *base, uint32_t mask)

Enable interrupts.

Parameters

- base – RDC peripheral base address.
- mask – Interrupts to enable, it is OR'ed value of enum _rdc_interrupts.

static inline void RDC_DisableInterrupts(RDC_Type *base, uint32_t mask)

Disable interrupts.

Parameters

- base – RDC peripheral base address.
- mask – Interrupts to disable, it is OR'ed value of enum _rdc_interrupts.

static inline uint32_t RDC_GetInterruptStatus(RDC_Type *base)

Get the interrupt pending status.

Parameters

- base – RDC peripheral base address.

Returns

Interrupts pending status, it is OR'ed value of enum _rdc_interrupts.

static inline void RDC_ClearInterruptStatus(RDC_Type *base, uint32_t mask)

Clear interrupt pending status.

Parameters

- base – RDC peripheral base address.
- mask – Status to clear, it is OR'ed value of enum _rdc_interrupts.

static inline uint32_t RDC_GetStatus(RDC_Type *base)

Get RDC status.

Parameters

- base – RDC peripheral base address.

Returns

mask RDC status, it is OR'ed value of enum _rdc_flags.

static inline void RDC_ClearStatus(RDC_Type *base, uint32_t mask)

Clear RDC status.

Parameters

- base – RDC peripheral base address.
- mask – RDC status to clear, it is OR'ed value of enum _rdc_flags.

void RDC_SetMasterDomainAssignment(RDC_Type *base, rdc_master_t master, const *rdc_domain_assignment_t* *domainAssignment)

Set master domain assignment.

Parameters

- base – RDC peripheral base address.
- master – Which master to set.
- domainAssignment – Pointer to the assignment.

void RDC_GetDefaultMasterDomainAssignment(*rdc_domain_assignment_t* *domainAssignment)

Get default master domain assignment.

The default configuration is:

```
assignment->domainId = 0U;
assignment->lock = 0U;
```

Parameters

- domainAssignment – Pointer to the assignment.

static inline void RDC_LockMasterDomainAssignment(RDC_Type *base, rdc_master_t master)

Lock master domain assignment.

Once locked, it could not be unlocked until next reset.

Parameters

- base – RDC peripheral base address.
- master – Which master to lock.

void RDC_SetPeriphAccessConfig(RDC_Type *base, const *rdc_periph_access_config_t* *config)

Set peripheral access policy.

Parameters

- base – RDC peripheral base address.
- config – Pointer to the policy configuration.

void RDC_GetDefaultPeriphAccessConfig(*rdc_periph_access_config_t* *config)

Get default peripheral access policy.

The default configuration is:

```
config->lock = false;
config->enableSema = false;
config->policy = RDC_ACCESS_POLICY(0, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(1, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(2, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(3, kRDC_ReadWrite);
```

Parameters

- config – Pointer to the policy configuration.

static inline void RDC_LockPeriphAccessConfig(RDC_Type *base, rdc_periph_t periph)

Lock peripheral access policy configuration.

Once locked, it could not be unlocked until reset.

Parameters

- base – RDC peripheral base address.
- periph – Which peripheral to lock.

static inline uint8_t RDC_GetPeriphAccessPolicy(RDC_Type *base, rdc_periph_t periph, uint8_t domainId)

Get the peripheral access policy for specific domain.

Parameters

- base – RDC peripheral base address.
- periph – Which peripheral to get.
- domainId – Get policy for which domain.

Returns

Access policy, see _rdc_access_policy.

void RDC_SetMemAccessConfig(RDC_Type *base, const *rdc_mem_access_config_t* *config)

Set memory region access policy.

Note that when setting the baseAddress and endAddress in config, should be aligned to the region resolution, see rdc_mem_t definitions.

Parameters

- base – RDC peripheral base address.
- config – Pointer to the policy configuration.

void RDC_GetDefaultMemAccessConfig(*rdc_mem_access_config_t* *config)

Get default memory region access policy.

The default configuration is:

```
config->lock = false;
config->baseAddress = 0;
config->endAddress = 0;
config->policy = RDC_ACCESS_POLICY(0, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(1, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(2, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(3, kRDC_ReadWrite);
```

Parameters

- config – Pointer to the policy configuration.

static inline void RDC_LockMemAccessConfig(RDC_Type *base, rdc_mem_t mem)

Lock memory access policy configuration.

Once locked, it could not be unlocked until reset. After locked, you can only call RDC_SetMemAccessValid to enable the configuration, but can not disable it or change other settings.

Parameters

- base – RDC peripheral base address.
- mem – Which memory region to lock.

static inline void RDC_SetMemAccessValid(RDC_Type *base, rdc_mem_t mem, bool valid)

Enable or disable memory access policy configuration.

Parameters

- base – RDC peripheral base address.
- mem – Which memory region to operate.
- valid – Pass in true to valid, false to invalid.

void RDC_GetMemViolationStatus(RDC_Type *base, rdc_mem_t mem, *rdc_mem_status_t* *status)

Get the memory region violation status.

The first access violation is captured. Subsequent violations are ignored until the status register is cleared. Contents are cleared upon reading the register. Clearing of contents occurs only when the status is read by the memory region's associated domain ID(s).

Parameters

- base – RDC peripheral base address.
- mem – Which memory region to get.
- status – The returned status.

static inline void RDC_ClearMemViolationFlag(RDC_Type *base, rdc_mem_t mem)

Clear the memory region violation flag.

Parameters

- base – RDC peripheral base address.
- mem – Which memory region to clear.

static inline uint8_t RDC_GetMemAccessPolicy(RDC_Type *base, rdc_mem_t mem, uint8_t domainId)

Get the memory region access policy for specific domain.

Parameters

- base – RDC peripheral base address.
- mem – Which memory region to get.
- domainId – Get policy for which domain.

Returns

Access policy, see _rdc_access_policy.

static inline uint8_t RDC_GetCurrentMasterDomainId(RDC_Type *base)

Gets the domain ID of the current bus master.

This function returns the domain ID of the current bus master.

Parameters

- base – RDC peripheral base address.

Returns

Domain ID of current bus master.

FSL_RDC_DRIVER_VERSION

RDC_ACCESS_POLICY(domainID, policy)

struct __rdc_hardware_config

#include <fsl_rdc.h> RDC hardware configuration.

Public Members

uint32_t domainNumber
: Number of domains.

uint32_t masterNumber
: Number of bus masters.

uint32_t periphNumber
: Number of peripherals.

uint32_t memNumber
: Number of memory regions.

struct __rdc_domain_assignment
: *#include <fsl_rdc.h>* Master domain assignment.

Public Members

uint32_t domainId
: Domain ID.

uint32_t ___pad0___
: Reserved.

uint32_t lock
: Lock the domain assignment.

struct __rdc_periph_access_config
: *#include <fsl_rdc.h>* Peripheral domain access permission configuration.

Public Members

rdc_periph_t periph
: Peripheral name.

bool lock
: Lock the permission until reset.

bool enableSema
: Enable semaphore or not, when enabled, master should call RDC_SEMA42_Lock to lock the semaphore gate accordingly before access the peripheral.

uint16_t policy
: Access policy.

struct __rdc_mem_access_config
: *#include <fsl_rdc.h>* Memory region domain access control configuration.

  Note that when setting the rdc_mem_access_config_t::baseAddress and rdc_mem_access_config_t::endAddress, should be aligned to the region resolution, see rdc_mem_t definitions.

Public Members

rdc_mem_t mem
: Memory region descriptor name.

bool lock
: Lock the configuration.

uint64_t baseAddress
: Start address of the memory region.

uint64_t endAddress
: End address of the memory region.

uint16_t policy
: Access policy.

struct _rdc_mem_status
: *#include <fsl_rdc.h>* Memory region access violation status.

Public Members

bool hasViolation
: Violating happens or not.

uint8_t domainID
: Violating Domain ID.

uint64_t address
: Violating Address.

2.38 RDC_SEMA42: Hardware Semaphores Driver

FSL_RDC_SEMA42_DRIVER_VERSION
: RDC_SEMA42 driver version.

void RDC_SEMA42_Init(RDC_SEMAPHORE_Type *base)
: Initializes the RDC_SEMA42 module.

  This function initializes the RDC_SEMA42 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either RDC_SEMA42_ResetGate or RDC_SEMA42_ResetAllGates function.

  Parameters

  - base – RDC_SEMA42 peripheral base address.

void RDC_SEMA42_Deinit(RDC_SEMAPHORE_Type *base)
: De-initializes the RDC_SEMA42 module.

  This function de-initializes the RDC_SEMA42 module. It only disables the clock.

  Parameters

  - base – RDC_SEMA42 peripheral base address.

status_t RDC_SEMA42_TryLock(RDC_SEMAPHORE_Type *base, uint8_t gateNum, uint8_t masterIndex, uint8_t domainId)
: Tries to lock the RDC_SEMA42 gate.

  This function tries to lock the specific RDC_SEMA42 gate. If the gate has been locked by another processor, this function returns an error code.

  Parameters

  - base – RDC_SEMA42 peripheral base address.

- gateNum – Gate number to lock.
- masterIndex – Current processor master index.
- domainId – Current processor domain ID.

Return values

- kStatus_Success – Lock the sema42 gate successfully.
- kStatus_Failed – Sema42 gate has been locked by another processor.

void RDC_SEMA42_Lock(RDC_SEMAPHORE_Type *base, uint8_t gateNum, uint8_t masterIndex, uint8_t domainId)

Locks the RDC_SEMA42 gate.

This function locks the specific RDC_SEMA42 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

Parameters

- base – RDC_SEMA42 peripheral base address.
- gateNum – Gate number to lock.
- masterIndex – Current processor master index.
- domainId – Current processor domain ID.

static inline void RDC_SEMA42_Unlock(RDC_SEMAPHORE_Type *base, uint8_t gateNum)

Unlocks the RDC_SEMA42 gate.

This function unlocks the specific RDC_SEMA42 gate. It only writes unlock value to the RDC_SEMA42 gate register. However, it does not check whether the RDC_SEMA42 gate is locked by the current processor or not. As a result, if the RDC_SEMA42 gate is not locked by the current processor, this function has no effect.

Parameters

- base – RDC_SEMA42 peripheral base address.
- gateNum – Gate number to unlock.

static inline int32_t RDC_SEMA42_GetLockMasterIndex(RDC_SEMAPHORE_Type *base, uint8_t gateNum)

Gets which master has currently locked the gate.

Parameters

- base – RDC_SEMA42 peripheral base address.
- gateNum – Gate number.

Returns

Return -1 if the gate is not locked by any master, otherwise return the master index.

int32_t RDC_SEMA42_GetLockDomainID(RDC_SEMAPHORE_Type *base, uint8_t gateNum)

Gets which domain has currently locked the gate.

Parameters

- base – RDC_SEMA42 peripheral base address.
- gateNum – Gate number.

Returns

Return -1 if the gate is not locked by any domain, otherwise return the domain ID.

status_t RDC_SEMA42_ResetGate(RDC_SEMAPHORE_Type *base, uint8_t gateNum)

Resets the RDC_SEMA42 gate to an unlocked status.

This function resets a RDC_SEMA42 gate to an unlocked status.

Parameters

- base – RDC_SEMA42 peripheral base address.
- gateNum – Gate number.

Return values

- kStatus_Success – RDC_SEMA42 gate is reset successfully.
- kStatus_Failed – Some other reset process is ongoing.

static inline *status_t* RDC_SEMA42_ResetAllGates(RDC_SEMAPHORE_Type *base)

Resets all RDC_SEMA42 gates to an unlocked status.

This function resets all RDC_SEMA42 gate to an unlocked status.

Parameters

- base – RDC_SEMA42 peripheral base address.

Return values

- kStatus_Success – RDC_SEMA42 is reset successfully.
- kStatus_RDC_SEMA42_Reseting – Some other reset process is ongoing.

RDC_SEMA42_GATE_NUM_RESET_ALL

The number to reset all RDC_SEMA42 gates.

RDC_SEMA42_GATEn(base, n)

RDC_SEMA42 gate n register address.

RDC_SEMA42_GATE_COUNT

RDC_SEMA42 gate count.

RDC_SEMAPHORE_GATE_GTFSM_MASK

2.39 SAI: Serial Audio Interface

2.40 SAI Driver

void SAI_Init(I2S_Type *base)

Initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless SAI_Init is called to enable the clock.

Parameters

- base – SAI base pointer.

void SAI_Deinit(I2S_Type *base)

De-initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless SAI_TxInit or SAI_RxInit is called to enable the clock.

Parameters

- base – SAI base pointer.

void SAI_TxReset(I2S_Type *base)

Resets the SAI Tx.

This function enables the software reset and FIFO reset of SAI Tx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

void SAI_RxReset(I2S_Type *base)

Resets the SAI Rx.

This function enables the software reset and FIFO reset of SAI Rx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

void SAI_TxEnable(I2S_Type *base, bool enable)

Enables/disables the SAI Tx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Tx, false means disable.

void SAI_RxEnable(I2S_Type *base, bool enable)

Enables/disables the SAI Rx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Rx, false means disable.

static inline void SAI_TxSetBitClockDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_RxSetBitClockDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_RxSetFrameSyncDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_TxSetFrameSyncDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Tx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

void SAI_TxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)

Transmitter bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.
- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

void SAI_RxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)

Receiver bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.
- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

void SAI_TxSetBitclockConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_bit_clock_t* *config)

Transmitter Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

void SAI_RxSetBitclockConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_bit_clock_t* *config)

Receiver Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

void SAI_TxSetFrameSyncConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_frame_sync_t* *config)

SAI transmitter Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

void SAI_RxSetFrameSyncConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_frame_sync_t* *config)

SAI receiver Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

void SAI_TxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI transmitter Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_RxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI receiver Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_TxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI transmitter configurations.

Parameters

- base – SAI base pointer.
- config – transmitter configurations.

void SAI_RxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI receiver configurations.

Parameters

- base – SAI base pointer.
- config – receiver configurations.

void SAI_GetClassicI2SConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get classic I2S mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetLeftJustifiedConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get left justified mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetRightJustifiedConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get right justified mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetTDMConfig(*sai_transceiver_t* *config, *sai_frame_sync_len_t* frameSyncWidth, *sai_word_width_t* bitWidth, uint32_t dataWordNum, uint32_t saiChannelMask)

Get TDM mode configurations.

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data word width.
- dataWordNum – word number in one frame.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetDSPConfig(*sai_transceiver_t* *config, *sai_frame_sync_len_t* frameSyncWidth, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get DSP mode configurations.

DSP/PCM MODE B configuration flow for TX. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
SAI_TxSetConfig(base, config)
```

Note: DSP mode is also called PCM mode which support MODE A and MODE B, DSP/PCM MODE A configuration flow. RX is 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 –

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 – tranmitter 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 2.4.7

_sai_status_t, SAI return status.

Values:

enumerator kStatus_SAI_TxBusy
SAI Tx is busy.

enumerator kStatus_SAI_RxBusy
SAI Rx is busy.

enumerator kStatus_SAI_TxError
SAI Tx FIFO error.

enumerator kStatus_SAI_RxError
SAI Rx FIFO error.

enumerator kStatus_SAI_QueueFull
SAI transfer queue is full.

enumerator kStatus_SAI_TxIdle
SAI Tx is idle

enumerator kStatus_SAI_RxIdle
SAI Rx is idle

_sai_channel_mask,.sai channel mask value, actual channel numbers is depend soc specific

Values:

enumerator kSAI_Channel0Mask
channel 0 mask value

enumerator kSAI_Channel1Mask
channel 1 mask value

enumerator kSAI_Channel2Mask
channel 2 mask value

enumerator kSAI_Channel3Mask
channel 3 mask value

enumerator kSAI_Channel4Mask
channel 4 mask value

enumerator kSAI_Channel5Mask
channel 5 mask value

enumerator kSAI_Channel6Mask
channel 6 mask value

enumerator kSAI_Channel7Mask
channel 7 mask value

enum _sai_protocol
Define the SAI bus type.

Values:

enumerator kSAI_BusLeftJustified
Uses left justified format.

enumerator kSAI_BusRightJustified
Uses right justified format.

enumerator kSAI_BusI2S
Uses I2S format.

enumerator kSAI_BusPCMA
Uses I2S PCM A format.

enumerator kSAI_BusPCMB
Uses I2S PCM B format.

enum _sai_master_slave
Master or slave mode.

Values:

enumerator kSAI_Master
Master mode include bclk and frame sync

enumerator kSAI_Slave
Slave mode include bclk and frame sync

enumerator kSAI_Bclk_Master_FrameSync_Slave
bclk in master mode, frame sync in slave mode

enumerator kSAI_Bclk_Slave_FrameSync_Master
bclk in slave mode, frame sync in master mode

enum _sai_mono_stereo
Mono or stereo audio format.

Values:

enumerator kSAI_Stereo
Stereo sound.

enumerator kSAI_MonoRight
Only Right channel have sound.

enumerator kSAI_MonoLeft
Only left channel have sound.

enum _sai_data_order
SAI data order, MSB or LSB.

Values:

enumerator kSAI_DataLSB
LSB bit transferred first

enumerator kSAI_DataMSB
MSB bit transferred first

enum _sai_clock_polarity

SAI clock polarity, active high or low.

Values:

enumerator kSAI_PolarityActiveHigh

Drive outputs on rising edge

enumerator kSAI_PolarityActiveLow

Drive outputs on falling edge

enumerator kSAI_SampleOnFallingEdge

Sample inputs on falling edge

enumerator kSAI_SampleOnRisingEdge

Sample inputs on rising edge

enum _sai_sync_mode

Synchronous or asynchronous mode.

Values:

enumerator kSAI_ModeAsync

Asynchronous mode

enumerator kSAI_ModeSync

Synchronous mode (with receiver or transmit)

enum _sai_mclk_source

Mater clock source.

Values:

enumerator kSAI_MclkSourceSysclk

Master clock from the system clock

enumerator kSAI_MclkSourceSelect1

Master clock from source 1

enumerator kSAI_MclkSourceSelect2

Master clock from source 2

enumerator kSAI_MclkSourceSelect3

Master clock from source 3

enum _sai_bclk_source

Bit clock source.

Values:

enumerator kSAI_BclkSourceBusclk

Bit clock using bus clock

enumerator kSAI_BclkSourceMclkOption1

Bit clock MCLK option 1

enumerator kSAI_BclkSourceMclkOption2

Bit clock MCLK option2

enumerator kSAI_BclkSourceMclkOption3

Bit clock MCLK option3

enumerator kSAI_BclkSourceMclkDiv

Bit clock using master clock divider

enumerator kSAI_BclkSourceOtherSai0
Bit clock from other SAI device

enumerator kSAI_BclkSourceOtherSai1
Bit clock from other SAI device

_sai_interrupt_enable_t, The SAI interrupt enable flag

Values:

enumerator kSAI_WordStartInterruptEnable
Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorInterruptEnable
Sync error flag, means the sync error is detected

enumerator kSAI_FIFOWarningInterruptEnable
FIFO warning flag, means the FIFO is empty

enumerator kSAI_FIFOErrorInterruptEnable
FIFO error flag

_sai_dma_enable_t, The DMA request sources

Values:

enumerator kSAI_FIFOWarningDMAEnable
FIFO warning caused by the DMA request

_sai_flags, The SAI status flag

Values:

enumerator kSAI_WordStartFlag
Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorFlag
Sync error flag, means the sync error is detected

enumerator kSAI_FIFOErrorFlag
FIFO error flag

enumerator kSAI_FIFOWarningFlag
FIFO warning flag

enum _sai_reset_type
The reset type.

Values:

enumerator kSAI_ResetTypeSoftware
Software reset, reset the logic state

enumerator kSAI_ResetTypeFIFO
FIFO reset, reset the FIFO read and write pointer

enumerator kSAI_ResetAll
All reset.

enum _sai_sample_rate

Audio sample rate.

Values:

enumerator kSAI_SampleRate8KHz
Sample rate 8000 Hz

enumerator kSAI_SampleRate11025Hz
Sample rate 11025 Hz

enumerator kSAI_SampleRate12KHz
Sample rate 12000 Hz

enumerator kSAI_SampleRate16KHz
Sample rate 16000 Hz

enumerator kSAI_SampleRate22050Hz
Sample rate 22050 Hz

enumerator kSAI_SampleRate24KHz
Sample rate 24000 Hz

enumerator kSAI_SampleRate32KHz
Sample rate 32000 Hz

enumerator kSAI_SampleRate44100Hz
Sample rate 44100 Hz

enumerator kSAI_SampleRate48KHz
Sample rate 48000 Hz

enumerator kSAI_SampleRate96KHz
Sample rate 96000 Hz

enumerator kSAI_SampleRate192KHz
Sample rate 192000 Hz

enumerator kSAI_SampleRate384KHz
Sample rate 384000 Hz

enum _sai_word_width

Audio word width.

Values:

enumerator kSAI_WordWidth8bits
Audio data width 8 bits

enumerator kSAI_WordWidth16bits
Audio data width 16 bits

enumerator kSAI_WordWidth24bits
Audio data width 24 bits

enumerator kSAI_WordWidth32bits
Audio data width 32 bits

enum _sai_transceiver_type

sai transceiver type

Values:

enumerator kSAI_Transmitter
sai transmitter

enumerator kSAI_Receiver
sai receiver

enum _sai_frame_sync_len
sai frame sync len

Values:

enumerator kSAI_FrameSyncLenOneBitClk
1 bit clock frame sync len for DSP mode

enumerator kSAI_FrameSyncLenPerWordWidth
Frame sync length decided by word width

typedef enum *_sai_protocol* sai_protocol_t
Define the SAI bus type.

typedef enum *_sai_master_slave* sai_master_slave_t
Master or slave mode.

typedef enum *_sai_mono_stereo* sai_mono_stereo_t
Mono or stereo audio format.

typedef enum *_sai_data_order* sai_data_order_t
SAI data order, MSB or LSB.

typedef enum *_sai_clock_polarity* sai_clock_polarity_t
SAI clock polarity, active high or low.

typedef enum *_sai_sync_mode* sai_sync_mode_t
Synchronous or asynchronous mode.

typedef enum *_sai_mclk_source* sai_mclk_source_t
Mater clock source.

typedef enum *_sai_bclk_source* sai_bclk_source_t
Bit clock source.

typedef enum *_sai_reset_type* sai_reset_type_t
The reset type.

typedef struct *_sai_config* sai_config_t
SAI user configuration structure.

typedef enum *_sai_sample_rate* sai_sample_rate_t
Audio sample rate.

typedef enum *_sai_word_width* sai_word_width_t
Audio word width.

typedef enum *_sai_transceiver_type* sai_transceiver_type_t
sai transceiver type

typedef enum *_sai_frame_sync_len* sai_frame_sync_len_t
sai frame sync len

typedef struct *_sai_transfer_format* sai_transfer_format_t
sai transfer format

typedef struct *_sai_bit_clock* sai_bit_clock_t
sai bit clock configurations

typedef struct *_sai_frame_sync* sai_frame_sync_t
sai frame sync configurations

typedef struct *_sai_serial_data* sai_serial_data_t
sai serial data configurations

typedef struct *_sai_transceiver* sai_transceiver_t
sai transceiver configurations

typedef struct *_sai_transfer* sai_transfer_t
SAI transfer structure.

typedef struct *_sai_handle* sai_handle_t

typedef void (*sai_transfer_callback_t)(I2S_Type *base, *sai_handle_t* *handle, *status_t* status, void *userData)
SAI transfer callback prototype.

SAI_XFER_QUEUE_SIZE
SAI transfer queue size, user can refine it according to use case.

FSL_SAI_HAS_FIFO_EXTEND_FEATURE
sai fifo feature

struct sai_config
#include <fsl_sai.h> SAI user configuration structure.

Public Members

sai_protocol_t protocol
Audio bus protocol in SAI

sai_sync_mode_t syncMode
SAI sync mode, control Tx/Rx clock sync

sai_bclk_source_t bclkSource
Bit Clock source

sai_master_slave_t masterSlave
Master or slave

struct sai_transfer_format
#include <fsl_sai.h> sai transfer format

Public Members

uint32_t sampleRate_Hz
Sample rate of audio data

uint32_t bitWidth
Data length of audio data, usually 8/16/24/32 bits

sai_mono_stereo_t stereo
Mono or stereo

uint8_t channel
Transfer start channel

uint8_t channelMask
: enabled channel mask value, reference _sai_channel_mask

uint8_t endChannel
: end channel number

uint8_t channelNums
: Total enabled channel numbers

sai_protocol_t protocol
: Which audio protocol used

bool isFrameSyncCompact
: True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.

struct _sai_bit_clock
: *#include <fsl_sai.h>* sai bit clock configurations

Public Members

bool bclkInputDelay
: bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time .

sai_clock_polarity_t bclkPolarity
: bit clock polarity

sai_bclk_source_t bclkSource
: bit Clock source

struct _sai_frame_sync
: *#include <fsl_sai.h>* sai frame sync configurations

Public Members

uint8_t frameSyncWidth
: frame sync width in number of bit clocks

bool frameSyncEarly
: TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame

sai_clock_polarity_t frameSyncPolarity
: frame sync polarity

struct _sai_serial_data
: *#include <fsl_sai.h>* sai serial data configurations

Public Members

sai_data_order_t dataOrder
: configure whether the LSB or MSB is transmitted first

uint8_t dataWord0Length
: configure the number of bits in the first word in each frame

uint8_t dataWordNLength
: configure the number of bits in the each word in each frame, except the first word

uint8_t dataWordLength
: used to record the data length for dma transfer

uint8_t dataFirstBitShifted
: Configure the bit index for the first bit transmitted for each word in the frame

uint8_t dataWordNum
: configure the number of words in each frame

uint32_t dataMaskedWord
: configure whether the transmit word is masked

struct __sai__transceiver
: *#include <fsl_sai.h>* sai transceiver configurations

Public Members

sai_serial_data_t serialData
: serial data configurations

sai_frame_sync_t frameSync
: ws configurations

sai_bit_clock_t bitClock
: bit clock configurations

sai_master_slave_t masterSlave
: transceiver is master or slave

sai_sync_mode_t syncMode
: transceiver sync mode

uint8_t startChannel
: Transfer start channel

uint8_t channelMask
: enabled channel mask value, reference _sai_channel_mask

uint8_t endChannel
: end channel number

uint8_t channelNums
: Total enabled channel numbers

struct __sai__transfer
: *#include <fsl_sai.h>* SAI transfer structure.

Public Members

uint8_t *data
: Data start address to transfer.

size_t dataSize
: Transfer size.

struct __sai__handle
: *#include <fsl_sai.h>* SAI handle structure.

Public Members

I2S_Type *base
: base address

uint32_t state
: Transfer status

sai_transfer_callback_t callback
: Callback function called at transfer event

void *userData
: Callback parameter passed to callback function

uint8_t bitWidth
: Bit width for transfer, 8/16/24/32 bits

uint8_t channel
: Transfer start channel

uint8_t channelMask
: enabled channel mask value, refernece _sai_channel_mask

uint8_t endChannel
: end channel number

uint8_t channelNums
: Total enabled channel numbers

sai_transfer_t saiQueue[(4U)]
: Transfer queue storing queued transfer

size_t transferSize[(4U)]
: Data bytes need to transfer

volatile uint8_t queueUser
: Index for user to queue transfer

volatile uint8_t queueDriver
: Index for driver to get the transfer data and size

2.41 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

2.42 SAI SDMA Driver

void SAI_TransferTxCreateHandleSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle, *sai_sdma_callback_t* callback, void *userData, *sdma_handle_t* *dmaHandle, uint32_t eventSource)

Initializes the SAI SDMA handle.

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

Parameters

- base – SAI base pointer.
- handle – SAI SDMA handle pointer.
- base – SAI peripheral base address.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- dmaHandle – SDMA handle pointer, this handle shall be static allocated by users.
- eventSource – SAI event source number.

void SAI_TransferRxCreateHandleSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle, *sai_sdma_callback_t* callback, void *userData, *sdma_handle_t* *dmaHandle, uint32_t eventSource)

Initializes the SAI Rx SDMA handle.

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

Parameters

- base – SAI base pointer.
- handle – SAI SDMA handle pointer.
- base – SAI peripheral base address.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.

- dmaHandle – SDMA handle pointer, this handle shall be static allocated by users.
- eventSource – SAI event source number.

status_t SAI_TransferSendSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle, *sai_transfer_t* *xfer)

Performs a non-blocking SAI transfer using DMA.

Note: This interface returns immediately after the transfer initiates. Call SAI_GetTransferStatus to poll the transfer status and check whether the SAI transfer is finished.

Parameters

- base – SAI base pointer.
- handle – SAI SDMA handle pointer.
- xfer – Pointer to the DMA transfer structure.

Return values

- kStatus_Success – Start a SAI SDMA send successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_TxBusy – SAI is busy sending data.

status_t SAI_TransferReceiveSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle, *sai_transfer_t* *xfer)

Performs a non-blocking SAI receive using SDMA.

Note: This interface returns immediately after the transfer initiates. Call the SAI_GetReceiveRemainingBytes to poll the transfer status and check whether the SAI transfer is finished.

Parameters

- base – SAI base pointer
- handle – SAI SDMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a SAI SDMA receive successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_RxBusy – SAI is busy receiving data.

void SAI_TransferAbortSendSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle)

Aborts a SAI transfer using SDMA.

Parameters

- base – SAI base pointer.
- handle – SAI SDMA handle pointer.

void SAI_TransferAbortReceiveSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle)

Aborts a SAI receive using SDMA.

Parameters

- base – SAI base pointer
- handle – SAI SDMA handle pointer.

void SAI_TransferTerminateReceiveSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle)

Terminate all the SAI sdma receive transfer.

Parameters

- base – SAI base pointer.
- handle – SAI SDMA handle pointer.

void SAI_TransferTerminateSendSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle)

Terminate all the SAI sdma send transfer.

Parameters

- base – SAI base pointer.
- handle – SAI SDMA handle pointer.

void SAI_TransferRxSetConfigSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle,
sai_transceiver_t *saiConfig)

brief Configures the SAI RX.

param base SAI base pointer. param handle SAI SDMA handle pointer. param saiConig sai configurations.

void SAI_TransferTxSetConfigSDMA(I2S_Type *base, *sai_sdma_handle_t* *handle,
sai_transceiver_t *saiConfig)

brief Configures the SAI Tx.

param base SAI base pointer. param handle SAI SDMA handle pointer. param saiConig sai configurations.

FSL_SAI_SDMA_DRIVER_VERSION

Version 2.6.0

typedef struct *_sai_sdma_handle* sai_sdma_handle_t

typedef void (*sai_sdma_callback_t)(I2S_Type *base, *sai_sdma_handle_t* *handle, *status_t* status, void *userData)

SAI SDMA transfer callback function for finish and error.

struct _sai_sdma_handle

#include <fsl_sai_sdma.h> SAI DMA transfer handle, users should not touch the content of the handle.

Public Members

sdma_handle_t *dmaHandle

DMA handler for SAI send

uint8_t bytesPerFrame

Bytes in a frame

uint8_t channel

start data channel

uint8_t channelNums
: total transfer channel numbers, used for multififo

uint8_t channelMask
: enabled channel mask value, refernece _sai_channel_mask

uint8_t fifoOffset
: fifo address offset between multifo

uint32_t count
: The transfer data count in a DMA request

uint32_t state
: Internal state for SAI SDMA transfer

uint32_t eventSource
: SAI event source number

sai_sdma_callback_t callback
: Callback for users while transfer finish or error occurs

void *userData
: User callback parameter

sdma_buffer_descriptor_t bdPool[(4U)]
: BD pool for SDMA transfer.

sai_transfer_t saiQueue[(4U)]
: Transfer queue storing queued transfer.

size_t transferSize[(4U)]
: Data bytes need to transfer

volatile uint8_t queueUser
: Index for user to queue transfer.

volatile uint8_t queueDriver
: Index for driver to get the transfer data and size

2.43 SDMA: Smart Direct Memory Access (SDMA) Controller Driver

void SDMA_Init(SDMAARM_Type *base, const *sdma_config_t* *config)
: Initializes the SDMA peripheral.

  This function ungates the SDMA clock and configures the SDMA peripheral according to the configuration structure.

  Note: This function enables the minor loop map feature.

  Parameters

  - base – SDMA peripheral base address.
  - config – A pointer to the configuration structure, see “sdma_config_t”.

void SDMA_Deinit(SDMAARM_Type *base)

Deinitializes the SDMA peripheral.

This function gates the SDMA clock.

Parameters

- base – SDMA peripheral base address.

void SDMA_GetDefaultConfig(*sdma_config_t* *config)

Gets the SDMA default configuration structure.

This function sets the configuration structure to default values. The default configuration is set to the following values.

```
config.enableRealTimeDebugPin = false;
config.isSoftwareResetClearLock = true;
config.ratio = kSDMA_HalfARMClockFreq;
```

Parameters

- config – A pointer to the SDMA configuration structure.

void SDMA_ResetModule(SDMAARM_Type *base)

Sets all SDMA core register to reset status.

If only reset ARM core, SDMA register cannot return to reset value, shall call this function to reset all SDMA register to reset value. But the internal status cannot be reset.

Parameters

- base – SDMA peripheral base address.

static inline void SDMA_EnableChannelErrorInterrupts(SDMAARM_Type *base, uint32_t channel)

Enables the interrupt source for the SDMA error.

Enable this will trigger an interrupt while SDMA occurs error while executing scripts.

Parameters

- base – SDMA peripheral base address.
- channel – SDMA channel number.

static inline void SDMA_DisableChannelErrorInterrupts(SDMAARM_Type *base, uint32_t channel)

Disables the interrupt source for the SDMA error.

Parameters

- base – SDMA peripheral base address.
- channel – SDMA channel number.

void SDMA_ConfigBufferDescriptor(*sdma_buffer_descriptor_t* *bd, uint32_t srcAddr, uint32_t destAddr, *sdma_transfer_size_t* busWidth, size_t bufferSize, bool isLast, bool enableInterrupt, bool isWrap, *sdma_transfer_type_t* type)

Sets buffer descriptor contents.

This function sets the descriptor contents such as source, dest address and status bits.

Parameters

- bd – Pointer to the buffer descriptor structure.
- srcAddr – Source address for the buffer descriptor.

- destAddr – Destination address for the buffer descriptor.
- busWidth – The transfer width, it only can be a member of sdma_transfer_size_t.
- bufferSize – Buffer size for this descriptor, this number shall less than 0xFFFF. If need to transfer a big size, shall divide into several buffer descriptors.
- isLast – Is the buffer descriptor the last one for the channel to transfer. If only one descriptor used for the channel, this bit shall set to TRUE.
- enableInterrupt – If trigger an interrupt while this buffer descriptor transfer finished.
- isWrap – Is the buffer descriptor need to be wrapped. While this bit set to true, it will automatically wrap to the first buffer descrtiptor to do transfer.
- type – Transfer type, memory to memory, peripheral to memory or memory to peripheral.

static inline void SDMA_SetChannelPriority(SDMAARM_Type *base, uint32_t channel, uint8_t priority)

Set SDMA channel priority.

This function sets the channel priority. The default value is 0 for all channels, priority 0 will prevents channel from starting, so the priority must be set before start a channel.

Parameters

- base – SDMA peripheral base address.
- channel – SDMA channel number.
- priority – SDMA channel priority.

static inline void SDMA_SetSourceChannel(SDMAARM_Type *base, uint32_t source, uint32_t channelMask)

Set SDMA request source mapping channel.

This function sets which channel will be triggered by the dma request source.

Parameters

- base – SDMA peripheral base address.
- source – SDMA dma request source number.
- channelMask – SDMA channel mask. 1 means channel 0, 2 means channel 1, 4 means channel 3. SDMA supports an event trigger multi-channel. A channel can also be triggered by several source events.

static inline void SDMA_StartChannelSoftware(SDMAARM_Type *base, uint32_t channel)

Start a SDMA channel by software trigger.

This function start a channel.

Parameters

- base – SDMA peripheral base address.
- channel – SDMA channel number.

static inline void SDMA_StartChannelEvents(SDMAARM_Type *base, uint32_t channel)

Start a SDMA channel by hardware events.

This function start a channel.

Parameters

- base – SDMA peripheral base address.
- channel – SDMA channel number.

static inline void SDMA_StopChannel(SDMAARM_Type *base, uint32_t channel)

Stop a SDMA channel.

This function stops a channel.

Parameters

- base – SDMA peripheral base address.
- channel – SDMA channel number.

void SDMA_SetContextSwitchMode(SDMAARM_Type *base, *sdma_context_switch_mode_t* mode)

Set the SDMA context switch mode.

Parameters

- base – SDMA peripheral base address.
- mode – SDMA context switch mode.

static inline uint32_t SDMA_GetChannelInterruptStatus(SDMAARM_Type *base)

Gets the SDMA interrupt status of all channels.

Parameters

- base – SDMA peripheral base address.

Returns

The interrupt status for all channels. Check the relevant bits for specific channel.

static inline void SDMA_ClearChannelInterruptStatus(SDMAARM_Type *base, uint32_t mask)

Clear the SDMA channel interrupt status of specific channels.

Parameters

- base – SDMA peripheral base address.
- mask – The interrupt status need to be cleared.

static inline uint32_t SDMA_GetChannelStopStatus(SDMAARM_Type *base)

Gets the SDMA stop status of all channels.

Parameters

- base – SDMA peripheral base address.

Returns

The stop status for all channels. Check the relevant bits for specific channel.

static inline void SDMA_ClearChannelStopStatus(SDMAARM_Type *base, uint32_t mask)

Clear the SDMA channel stop status of specific channels.

Parameters

- base – SDMA peripheral base address.
- mask – The stop status need to be cleared.

static inline uint32_t SDMA_GetChannelPendStatus(SDMAARM_Type *base)

Gets the SDMA channel pending status of all channels.

Parameters

- base – SDMA peripheral base address.

Returns
The pending status for all channels. Check the relevant bits for specific channel.

static inline void SDMA_ClearChannelPendStatus(SDMAARM_Type *base, uint32_t mask)

Clear the SDMA channel pending status of specific channels.

Parameters

- base – SDMA peripheral base address.
- mask – The pending status need to be cleared.

static inline uint32_t SDMA_GetErrorStatus(SDMAARM_Type *base)

Gets the SDMA channel error status.

SDMA channel error flag is asserted while an incoming DMA request was detected and it triggers a channel that is already pending or being serviced. This probably means there is an overflow of data for that channel.

Parameters

- base – SDMA peripheral base address.

Returns
The error status for all channels. Check the relevant bits for specific channel.

bool SDMA_GetRequestSourceStatus(SDMAARM_Type *base, uint32_t source)

Gets the SDMA request source pending status.

Parameters

- base – SDMA peripheral base address.
- source – DMA request source number.

Returns
True means the request source is pending, otherwise not pending.

void SDMA_CreateHandle(*sdma_handle_t* *handle, SDMAARM_Type *base, uint32_t channel, *sdma_context_data_t* *context)

Creates the SDMA handle.

This function is called if using the transactional API for SDMA. This function initializes the internal state of the SDMA handle.

Parameters

- handle – SDMA handle pointer. The SDMA handle stores callback function and parameters.
- base – SDMA peripheral base address.
- channel – SDMA channel number.
- context – Context structure for the channel to download into SDMA. Users shall make sure the context located in a non-cacheable memory, or it will cause SDMA run fail. Users shall not touch the context contents, it only be filled by SDMA driver in SDMA_SubmitTransfer function.

void SDMA_InstallBDMemory(*sdma_handle_t* *handle, *sdma_buffer_descriptor_t* *BDPool, uint32_t BDCount)

Installs the BDs memory pool into the SDMA handle.

This function is called after the SDMA_CreateHandle to use multi-buffer feature.

Parameters

- handle – SDMA handle pointer.

- BDPool – A memory pool to store BDs. It must be located in non-cacheable address.
- BDCount – The number of BD slots.

void SDMA_SetCallback(*sdma_handle_t* *handle, *sdma_callback* callback, void *userData)

Installs a callback function for the SDMA transfer.

This callback is called in the SDMA IRQ handler. Use the callback to do something after the current major loop transfer completes.

Parameters

- handle – SDMA handle pointer.
- callback – SDMA callback function pointer.
- userData – A parameter for the callback function.

void SDMA_SetMultiFifoConfig(*sdma_transfer_config_t* *config, uint32_t fifoNums, uint32_t fifoOffset)

multi fifo configurations.

This api is used to support multi fifo for SDMA, if user want to get multi fifo data, then this api shoule be called before submit transfer.

Parameters

- config – transfer configurations.
- fifoNums – fifo numbers that multi fifo operation perform, support up to 15 fifo numbers.
- fifoOffset – fifoOffset = fifo address offset / sizeof(uint32_t) - 1.

void SDMA_EnableSwDone(SDMAARM_Type *base, *sdma_transfer_config_t* *config, uint8_t sel, *sdma_peripheral_t* type)

enable sdma sw done feature.

Deprecated:

Do not use this function. It has been superceded by SDMA_SetDoneConfig.

Parameters

- base – SDMA base.
- config – transfer configurations.
- sel – sw done selector.
- type – peripheral type is used to determine the corresponding peripheral sw done selector bit.

void SDMA_SetDoneConfig(SDMAARM_Type *base, *sdma_transfer_config_t* *config, *sdma_peripheral_t* type, *sdma_done_src_t* doneSrc)

sdma channel done configurations.

Parameters

- base – SDMA base.
- config – transfer configurations.
- type – peripheral type.
- doneSrc – reference sdma_done_src_t.

void SDMA_LoadScript(SDMAARM_Type *base, uint32_t destAddr, void *srcAddr, size_t bufferSizeBytes)

load script to sdma program memory.

Parameters

- base – SDMA base.
- destAddr – dest script address, should be SDMA program memory address.
- srcAddr – source address of target script.
- bufferSizeBytes – bytes size of script.

void SDMA_DumpScript(SDMAARM_Type *base, uint32_t srcAddr, void *destAddr, size_t bufferSizeBytes)

dump script from sdma program memory.

Parameters

- base – SDMA base.
- srcAddr – should be SDMA program memory address.
- destAddr – address to store scripts.
- bufferSizeBytes – bytes size of script.

static inline const char *SDMA_GetRamScriptVersion(SDMAARM_Type *base)

Get RAM script version.

Parameters

- base – SDMA base.

Returns

The script version of RAM.

void SDMA_PrepareTransfer(*sdma_transfer_config_t* *config, uint32_t srcAddr, uint32_t destAddr, uint32_t srcWidth, uint32_t destWidth, uint32_t bytesEachRequest, uint32_t transferSize, uint32_t eventSource, *sdma_peripheral_t* peripheral, *sdma_transfer_type_t* type)

Prepares the SDMA transfer structure.

This function prepares the transfer configuration structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error.

Parameters

- config – The user configuration structure of type sdma_transfer_t.
- srcAddr – SDMA transfer source address.
- destAddr – SDMA transfer destination address.
- srcWidth – SDMA transfer source address width(bytes).
- destWidth – SDMA transfer destination address width(bytes).
- bytesEachRequest – SDMA transfer bytes per channel request.
- transferSize – SDMA transfer bytes to be transferred.
- eventSource – Event source number for the transfer, if use software trigger, just write 0.

- peripheral – Peripheral type, used to decide if need to use some special scripts.
- type – SDMA transfer type. Used to decide the correct SDMA script address in SDMA ROM.

void SDMA_PrepareP2PTransfer(*sdma_transfer_config_t* *config, uint32_t srcAddr, uint32_t destAddr, uint32_t srcWidth, uint32_t destWidth, uint32_t bytesEachRequest, uint32_t transferSize, uint32_t eventSource, uint32_t eventSource1, *sdma_peripheral_t* peripheral, *sdma_p2p_config_t* *p2p)

Prepares the SDMA P2P transfer structure.

This function prepares the transfer configuration structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error.

Parameters

- config – The user configuration structure of type sdma_transfer_t.
- srcAddr – SDMA transfer source address.
- destAddr – SDMA transfer destination address.
- srcWidth – SDMA transfer source address width(bytes).
- destWidth – SDMA transfer destination address width(bytes).
- bytesEachRequest – SDMA transfer bytes per channel request.
- transferSize – SDMA transfer bytes to be transferred.
- eventSource – Event source number for the transfer.
- eventSource1 – Event source1 number for the transfer.
- peripheral – Peripheral type, used to decide if need to use some special scripts.
- p2p – sdma p2p configuration pointer.

void SDMA_SubmitTransfer(*sdma_handle_t* *handle, const *sdma_transfer_config_t* *config)

Submits the SDMA transfer request.

This function submits the SDMA transfer request according to the transfer configuration structure.

Parameters

- handle – SDMA handle pointer.
- config – Pointer to SDMA transfer configuration structure.

void SDMA_StartTransfer(*sdma_handle_t* *handle)

SDMA starts transfer.

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

Parameters

- handle – SDMA handle pointer.

void SDMA_StopTransfer(*sdma_handle_t* *handle)

SDMA stops transfer.

This function disables the channel request to pause the transfer. Users can call SDMA_StartTransfer() again to resume the transfer.

Parameters

- handle – SDMA handle pointer.

void SDMA_AbortTransfer(*sdma_handle_t* *handle)

SDMA aborts transfer.

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

Parameters

- handle – DMA handle pointer.

uint32_t SDMA_GetTransferredBytes(*sdma_handle_t* *handle)

Get transferred bytes while not using BD pools.

This function returns the buffer descriptor count value if not using buffer descriptor. While do a simple transfer, which only uses one descriptor, the SDMA driver inside handle the buffer descriptor. In uart receive case, it can tell users how many data already received, also it can tells users how many data transfferd while error occurred. Notice, the count would not change while transfer is on-going using default SDMA script.

Parameters

- handle – DMA handle pointer.

Returns

Transferred bytes.

void SDMA_HandleIRQ(*sdma_handle_t* *handle)

SDMA IRQ handler for complete a buffer descriptor transfer.

This function clears the interrupt flags and also handle the CCB for the channel.

Parameters

- handle – SDMA handle pointer.

FSL_SDMA_DRIVER_VERSION

SDMA driver version.

Version 2.4.2.

enum _sdma_transfer_size

SDMA transfer configuration.

Values:

enumerator kSDMA_TransferSize1Bytes

Source/Destination data transfer size is 1 byte every time

enumerator kSDMA_TransferSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kSDMA_TransferSize3Bytes

Source/Destination data transfer size is 3 bytes every time

enumerator kSDMA_TransferSize4Bytes

Source/Destination data transfer size is 4 bytes every time

enum _sdma_bd_status

SDMA buffer descriptor status.

Values:

enumerator kSDMA_BDStatusDone

BD ownership, 0 means ARM core owns the BD, while 1 means SDMA owns BD.

enumerator kSDMA_BDStatusWrap

While this BD is last one, the next BD will be the first one

enumerator kSDMA_BDStatusContinuous

Buffer is allowed to transfer/receive to/from multiple buffers

enumerator kSDMA_BDStatusInterrupt

While this BD finished, send an interrupt.

enumerator kSDMA_BDStatusError

Error occurred on buffer descriptor command.

enumerator kSDMA_BDStatusLast

This BD is the last BD in this array. It means the transfer ended after this buffer

enumerator kSDMA_BDStatusExtend

Buffer descriptor extend status for SDMA scripts

enum _sdma_bd_command

SDMA buffer descriptor command.

Values:

enumerator kSDMA_BDCommandSETDM

Load SDMA data memory from ARM core memory buffer.

enumerator kSDMA_BDCommandGETDM

Copy SDMA data memory to ARM core memory buffer.

enumerator kSDMA_BDCommandSETPM

Load SDMA program memory from ARM core memory buffer.

enumerator kSDMA_BDCommandGETPM

Copy SDMA program memory to ARM core memory buffer.

enumerator kSDMA_BDCommandSETCTX

Load context for one channel into SDMA RAM from ARM platform memory buffer.

enumerator kSDMA_BDCommandGETCTX

Copy context for one channel from SDMA RAM to ARM platform memory buffer.

enum _sdma_context_switch_mode

SDMA context switch mode.

Values:

enumerator kSDMA_ContextSwitchModeStatic

SDMA context switch mode static

enumerator kSDMA_ContextSwitchModeDynamicLowPower

SDMA context switch mode dynamic with low power

enumerator kSDMA_ContextSwitchModeDynamicWithNoLoop

SDMA context switch mode dynamic with no loop

enumerator kSDMA_ContextSwitchModeDynamic
SDMA context switch mode dynamic

enum _sdma_clock_ratio
SDMA core clock frequency ratio to the ARM DMA interface.

Values:

enumerator kSDMA_HalfARMClockFreq
SDMA core clock frequency half of ARM platform

enumerator kSDMA_ARMClockFreq
SDMA core clock frequency equals to ARM platform

enum _sdma_transfer_type
SDMA transfer type.

Values:

enumerator kSDMA_MemoryToMemory
Transfer from memory to memory

enumerator kSDMA_PeripheralToMemory
Transfer from peripheral to memory

enumerator kSDMA_MemoryToPeripheral
Transfer from memory to peripheral

enumerator kSDMA_PeripheralToPeripheral
Transfer from peripheral to peripheral

enum sdma_peripheral
Peripheral type use SDMA.

Values:

enumerator kSDMA_PeripheralTypeMemory
Peripheral DDR memory

enumerator kSDMA_PeripheralTypeUART
UART use SDMA

enumerator kSDMA_PeripheralTypeUART_SP
UART instance in SPBA use SDMA

enumerator kSDMA_PeripheralTypeSPDIF
SPDIF use SDMA

enumerator kSDMA_PeripheralNormal
Normal peripheral use SDMA

enumerator kSDMA_PeripheralNormal_SP
Normal peripheral in SPBA use SDMA

enumerator kSDMA_PeripheralMultiFifoPDM
multi fifo PDM

enumerator kSDMA_PeripheralMultiFifoSaiRX
multi fifo sai rx use SDMA

enumerator kSDMA_PeripheralMultiFifoSaiTX
multi fifo sai tx use SDMA

enumerator kSDMA_PeripheralASRCM2P
asrc m2p

enumerator kSDMA_PeripheralASRCP2M
asrc p2m

enumerator kSDMA_PeripheralASRCP2P
asrc p2p

_sdma_transfer_status SDMA transfer status

Values:

enumerator kStatus_SDMA_ERROR
SDMA context error.

enumerator kStatus_SDMA_Busy
Channel is busy and can't handle the transfer request.

_sdma_multi_fifo_mask SDMA multi fifo mask

Values:

enumerator kSDMA_MultiFifoWatermarkLevelMask
multi fifo watermark level mask

enumerator kSDMA_MultiFifoNumsMask
multi fifo nums mask

enumerator kSDMA_MultiFifoOffsetMask
multi fifo offset mask

enumerator kSDMA_MultiFifoSwDoneMask
multi fifo sw done mask

enumerator kSDMA_MultiFifoSwDoneSelectorMask
multi fifo sw done selector mask

_sdma_multi_fifo_shift SDMA multi fifo shift

Values:

enumerator kSDMA_MultiFifoWatermarkLevelShift
multi fifo watermark level shift

enumerator kSDMA_MultiFifoNumsShift
multi fifo nums shift

enumerator kSDMA_MultiFifoOffsetShift
multi fifo offset shift

enumerator kSDMA_MultiFifoSwDoneShift
multi fifo sw done shift

enumerator kSDMA_MultiFifoSwDoneSelectorShift
multi fifo sw done selector shift

_sdma_done_channel SDMA done channel

Values:

enumerator kSDMA_DoneChannel0
SDMA done channel 0

enumerator kSDMA_DoneChannel1
SDMA done channel 1

enumerator kSDMA_DoneChannel2
SDMA done channel 2

enumerator kSDMA_DoneChannel3
SDMA done channel 3

enumerator kSDMA_DoneChannel4
SDMA done channel 4

enumerator kSDMA_DoneChannel5
SDMA done channel 5

enumerator kSDMA_DoneChannel6
SDMA done channel 6

enumerator kSDMA_DoneChannel7
SDMA done channel 7

enum _sdma_done_src
SDMA done source.

Values:

enumerator kSDMA_DoneSrcSW
software done

enumerator kSDMA_DoneSrcHwEvent0U
HW event 0 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent1U
HW event 1 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent2U
HW event 2 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent3U
HW event 3 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent4U
HW event 4 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent5U
HW event 5 is used for DONE event

enumerator kSDMA_DoneSrCHwEvent6U
HW event 6 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent7U
HW event 7 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent8U
HW event 8 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent9U
HW event 9 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent10U
HW event 10 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent11U
HW event 11 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent12U
HW event 12 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent13U
HW event 13 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent14U
HW event 14 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent15U
HW event 15 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent16U
HW event 16 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent17U
HW event 17 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent18U
HW event 18 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent19U
HW event 19 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent20U
HW event 20 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent21U
HW event 21 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent22U
HW event 22 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent23U
HW event 23 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent24U
HW event 24 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent25U
HW event 25 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent26U
HW event 26 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent27U
HW event 27 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent28U
HW event 28 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent29U
HW event 29 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent30U
HW event 30 is used for DONE event

enumerator kSDMA_DoneSrcHwEvent31U
HW event 31 is used for DONE event

typedef enum *_sdma_transfer_size* sdma_transfer_size_t
SDMA transfer configuration.

typedef enum *_sdma_bd_status* sdma_bd_status_t
SDMA buffer descriptor status.

typedef enum *_sdma_bd_command* sdma_bd_command_t
SDMA buffer descriptor command.

typedef enum *_sdma_context_switch_mode* sdma_context_switch_mode_t
SDMA context switch mode.

typedef enum *_sdma_clock_ratio* sdma_clock_ratio_t
SDMA core clock frequency ratio to the ARM DMA interface.

typedef enum *_sdma_transfer_type* sdma_transfer_type_t
SDMA transfer type.

typedef enum *sdma_peripheral* sdma_peripheral_t
Peripheral type use SDMA.

typedef enum *_sdma_done_src* sdma_done_src_t
SDMA done source.

typedef struct *_sdma_config* sdma_config_t
SDMA global configuration structure.

typedef struct *_sdma_multi_fifo_config* sdma_multi_fifo_config_t
SDMA multi fifo configurations.

typedef struct *_sdma_sw_done_config* sdma_sw_done_config_t
SDMA sw done configurations.

typedef struct *_sdma_p2p_config* sdma_p2p_config_t
SDMA peripheral to peripheral R7 config.

typedef struct *_sdma_transfer_config* sdma_transfer_config_t
SDMA transfer configuration.

This structure configures the source/destination transfer attribute.

typedef struct *_sdma_buffer_descriptor* sdma_buffer_descriptor_t
SDMA buffer descriptor structure.

This structure is a buffer descriptor, this structure describes the buffer start address and other options

typedef struct *_sdma_channel_control* sdma_channel_control_t
SDMA channel control descriptor structure.

typedef struct *_sdma_context_data* sdma_context_data_t
SDMA context structure for each channel. This structure can be load into SDMA core, with this structure, SDMA scripts can start work.

typedef void (*sdma_callback)(struct *_sdma_handle* *handle, void *userData, bool transferDone, uint32_t bdIndex)
Define callback function for SDMA.

typedef struct *_sdma_handle* sdma_handle_t
SDMA transfer handle structure.

SDMA_DRIVER_LOAD_RAM_SCRIPT

struct _sdma_config

#include <fsl_sdma.h> SDMA global configuration structure.

Public Members

bool enableRealTimeDebugPin

If enable real-time debug pin, default is closed to reduce power consumption.

bool isSoftwareResetClearLock

If software reset clears the LOCK bit which prevent writing SDMA scripts into SDMA.

sdma_clock_ratio_t ratio

SDMA core clock ratio to ARM platform DMA interface

struct _sdma_multi_fifo_config

#include <fsl_sdma.h> SDMA multi fifo configurations.

Public Members

uint8_t fifoNums

fifo numbers

uint8_t fifoOffset

offset between multi fifo data register address

struct _sdma_sw_done_config

#include <fsl_sdma.h> SDMA sw done configurations.

Public Members

bool enableSwDone

true is enable sw done, false is disable

uint8_t swDoneSel

sw done channel number per peripheral type

struct _sdma_p2p_config

#include <fsl_sdma.h> SDMA peripheral to peripheral R7 config.

Public Members

uint8_t sourceWatermark

lower watermark value

uint8_t destWatermark

higher water makr value

bool continuousTransfer

0: the amount of samples to be transferred is equal to the cont field of mode word 1: the amount of samples to be transferred is unknown and script will keep on transferring as long as both events are detected and script must be stopped by application.

struct _sdma_transfer_config

#include <fsl_sdma.h> SDMA transfer configuration.

This structure configures the source/destination transfer attribute.

Public Members

uint32_t srcAddr
: Source address of the transfer

uint32_t destAddr
: Destination address of the transfer

sdma_transfer_size_t srcTransferSize
: Source data transfer size.

sdma_transfer_size_t destTransferSize
: Destination data transfer size.

uint32_t bytesPerRequest
: Bytes to transfer in a minor loop

uint32_t transferSzie
: Bytes to transfer for this descriptor

uint32_t scriptAddr
: SDMA script address located in SDMA ROM.

uint32_t eventSource
: Event source number for the channel. 0 means no event, use software trigger

uint32_t eventSource1
: event source 1

bool isEventIgnore
: True means software trigger, false means hardware trigger

bool isSoftTriggerIgnore
: If ignore the HE bit, 1 means use hardware events trigger, 0 means software trigger

sdma_transfer_type_t type
: Transfer type, transfer type used to decide the SDMA script.

sdma_multi_fifo_config_t multiFifo
: multi fifo configurations

sdma_sw_done_config_t swDone
: sw done selector

uint32_t watermarkLevel
: watermark level

uint32_t eventMask0
: event mask 0

uint32_t eventMask1
: event mask 1

struct __sdma__buffer__descriptor
: *#include <fsl_sdma.h>* SDMA buffer descriptor structure.

  This structure is a buffer descriptor, this structure describes the buffer start address and other options

Public Members

uint32_t count
: Bytes of the buffer length for this buffer descriptor.

uint32_t status
: E,R,I,C,W,D status bits stored here

uint32_t command
: command mostlky used for channel 0

uint32_t bufferAddr
: Buffer start address for this descriptor.

uint32_t extendBufferAddr
: External buffer start address, this is an optional for a transfer.

struct _sdma_channel_control
: *#include <fsl_sdma.h>* SDMA channel control descriptor structure.

Public Members

uint32_t currentBDAddr
: Address of current buffer descriptor processed

uint32_t baseBDAddr
: The start address of the buffer descriptor array

uint32_t channelDesc
: Optional for transfer

uint32_t status
: Channel status

struct _sdma_context_data
: *#include <fsl_sdma.h>* SDMA context structure for each channel. This structure can be load into SDMA core, with this structure, SDMA scripts can start work.

Public Members

uint32_t GeneralReg[8]
: 8 general regsiters used for SDMA RISC core

struct _sdma_handle
: *#include <fsl_sdma.h>* SDMA transfer handle structure.

Public Members

sdma_callback callback
: Callback function for major count exhausted.

void *userData
: Callback function parameter.

SDMAARM_Type *base
: SDMA peripheral base address.

sdma_buffer_descriptor_t *BDPool
: Pointer to memory stored BD arrays.

uint32_t bdCount
How many buffer descriptor

uint32_t bdIndex
How many buffer descriptor

uint32_t eventSource
Event source count for the channel

uint32_t eventSource1
Event source 1 count for the channel

sdma_context_data_t *context
Channel context to exectute in SDMA

uint8_t channel
SDMA channel number.

uint8_t priority
SDMA channel priority

uint8_t flags
The status of the current channel.

2.44 SEMA4: Hardware Semaphores Driver

FSL_SEMA4_DRIVER_VERSION
SEMA4 driver version.

void SEMA4_Init(SEMA4_Type *base)
Initializes the SEMA4 module.

This function initializes the SEMA4 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either SEMA4_ResetGate or SEMA4_ResetAllGates function.

Parameters

- base – SEMA4 peripheral base address.

void SEMA4_Deinit(SEMA4_Type *base)
De-initializes the SEMA4 module.

This function de-initializes the SEMA4 module. It only disables the clock.

Parameters

- base – SEMA4 peripheral base address.

status_t SEMA4_TryLock(SEMA4_Type *base, uint8_t gateNum, uint8_t procNum)
Tries to lock the SEMA4 gate.

This function tries to lock the specific SEMA4 gate. If the gate has been locked by another processor, this function returns an error code.

Parameters

- base – SEMA4 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

Return values

- kStatus_Success – Lock the sema4 gate successfully.
- kStatus_Fail – Sema4 gate has been locked by another processor.

status_t SEMA4_Lock(SEMA4_Type *base, uint8_t gateNum, uint8_t procNum)

Locks the SEMA4 gate.

This function locks the specific SEMA4 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

If SEMA4_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

Parameters

- base – SEMA4 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

Return values

- kStatus_Success – The gate was successfully locked.
- kStatus_Timeout – Timeout occurred while waiting for the gate to be unlocked.

Returns

status_t

static inline void SEMA4_Unlock(SEMA4_Type *base, uint8_t gateNum)

Unlocks the SEMA4 gate.

This function unlocks the specific SEMA4 gate. It only writes unlock value to the SEMA4 gate register. However, it does not check whether the SEMA4 gate is locked by the current processor or not. As a result, if the SEMA4 gate is not locked by the current processor, this function has no effect.

Parameters

- base – SEMA4 peripheral base address.
- gateNum – Gate number to unlock.

static inline int32_t SEMA4_GetLockProc(SEMA4_Type *base, uint8_t gateNum)

Gets the status of the SEMA4 gate.

This function checks the lock status of a specific SEMA4 gate.

Parameters

- base – SEMA4 peripheral base address.
- gateNum – Gate number.

Returns

Return -1 if the gate is unlocked, otherwise return the processor number which has locked the gate.

status_t SEMA4_ResetGate(SEMA4_Type *base, uint8_t gateNum)

Resets the SEMA4 gate to an unlocked status.

This function resets a SEMA4 gate to an unlocked status.

Parameters

- base – SEMA4 peripheral base address.
- gateNum – Gate number.

Return values

- kStatus_Success – SEMA4 gate is reset successfully.
- kStatus_Fail – Some other reset process is ongoing.

static inline *status_t* SEMA4_ResetAllGates(SEMA4_Type *base)

Resets all SEMA4 gates to an unlocked status.

This function resets all SEMA4 gate to an unlocked status.

Parameters

- base – SEMA4 peripheral base address.

Return values

- kStatus_Success – SEMA4 is reset successfully.
- kStatus_Fail – Some other reset process is ongoing.

static inline void SEMA4_EnableGateNotifyInterrupt(SEMA4_Type *base, uint8_t procNum, uint16_t mask)

Enable the gate notification interrupt.

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle.

Parameters

- base – SEMA4 peripheral base address.
- procNum – Current processor number.
- mask – OR'ed value of the gate index, for example: (1«0) | (1«1) means gate 0 and gate 1.

static inline void SEMA4_DisableGateNotifyInterrupt(SEMA4_Type *base, uint8_t procNum, uint16_t mask)

Disable the gate notification interrupt.

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle.

Parameters

- base – SEMA4 peripheral base address.
- procNum – Current processor number.
- mask – OR'ed value of the gate index, for example: (1«0) | (1«1) means gate 0 and gate 1.

static inline uint32_t SEMA4_GetGateNotifyStatus(SEMA4_Type *base, uint8_t procNum)

Get the gate notification flags.

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle. The status flags are cleared automatically when the gate is locked by current core or locked again before the other core.

Parameters

- base – SEMA4 peripheral base address.
- procNum – Current processor number.

Returns

OR'ed value of the gate index, for example: (1«0) | (1«1) means gate 0 and gate 1 flags are pending.

status_t SEMA4_ResetGateNotify(SEMA4_Type *base, uint8_t gateNum)

Resets the SEMA4 gate IRQ notification.

This function resets a SEMA4 gate IRQ notification.

Parameters

- base – SEMA4 peripheral base address.
- gateNum – Gate number.

Return values

- kStatus_Success – Reset successfully.
- kStatus_Fail – Some other reset process is ongoing.

static inline *status_t* SEMA4_ResetAllGateNotify(SEMA4_Type *base)

Resets all SEMA4 gates IRQ notification.

This function resets all SEMA4 gate IRQ notifications.

Parameters

- base – SEMA4 peripheral base address.

Return values

- kStatus_Success – Reset successfully.
- kStatus_Fail – Some other reset process is ongoing.

SEMA4_GATE_NUM_RESET_ALL

The number to reset all SEMA4 gates.

SEMA4_GATEn(base, n)

SEMA4 gate n register address.

SEMA4_BUSY_POLL_COUNT

Maximum polling iterations for SEMA4 waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the SEMA4 driver code before timing out and returning an error.

It applies to all waiting loops in SEMA4 driver, such as waiting for a gate to be unlocked, waiting for a reset to complete, or waiting for a resource to become available.

This is a count of loop iterations, not a time-based value.

If defined as 0, polling loops will continue indefinitely until their exit condition is met, which could potentially cause the system to hang if hardware doesn't respond or if a resource is never released.

2.45 SNVS: Secure Non-Volatile Storage

2.46 Secure Non-Volatile Storage High-Power

void SNVS_HP_Init(SNVS_Type *base)

Initialize the SNVS.

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

Parameters

- base – SNVS peripheral base address

void SNVS_HP_Deinit(SNVS_Type *base)

Deinitialize the SNVS.

Parameters

- base – SNVS peripheral base address

void SNVS_HP_RTC_Init(SNVS_Type *base, const *snvs_hp_rtc_config_t* *config)

Ungates the SNVS clock and configures the peripheral for basic operation.

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

Parameters

- base – SNVS peripheral base address
- config – Pointer to the user's SNVS configuration structure.

void SNVS_HP_RTC_Deinit(SNVS_Type *base)

Stops the RTC and SRTC timers.

Parameters

- base – SNVS peripheral base address

void SNVS_HP_RTC_GetDefaultConfig(*snvs_hp_rtc_config_t* *config)

Fills in the SNVS config struct with the default settings.

The default values are as follows.

```
config->rtccalenable = false;
config->rtccalvalue = 0U;
config->PIFreq = 0U;
```

Parameters

- config – Pointer to the user's SNVS configuration structure.

status_t SNVS_HP_RTC_SetDatetime(SNVS_Type *base, const *snvs_hp_rtc_datetime_t* *datetime)

Sets the SNVS RTC date and time according to the given time structure.

Parameters

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

Returns

kStatus_Success: Success in setting the time and starting the SNVS RTC kStatus_InvalidArgument: Error because the datetime format is incorrect

void SNVS_HP_RTC_GetDatetime(SNVS_Type *base, *snvs_hp_rtc_datetime_t* *datetime)

Gets the SNVS RTC time and stores it in the given time structure.

Parameters

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

status_t SNVS_HP_RTC_SetAlarm(SNVS_Type *base, const *snvs_hp_rtc_datetime_t* *alarmTime)

Sets the SNVS RTC alarm time.

The function sets the RTC alarm. It also checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

Parameters

- base – SNVS peripheral base address
- alarmTime – Pointer to the structure where the alarm time is stored.

Returns

kStatus_Success: success in setting the SNVS RTC alarm kStatus_InvalidArgument: Error because the alarm datetime format is incorrect kStatus_Fail: Error because the alarm time has already passed

void SNVS_HP_RTC_GetAlarm(SNVS_Type *base, *snvs_hp_rtc_datetime_t* *datetime)

Returns the SNVS RTC alarm time.

Parameters

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the alarm date and time details are stored.

static inline void SNVS_HP_RTC_EnableInterrupts(SNVS_Type *base, uint32_t mask)

Enables the selected SNVS interrupts.

Parameters

- base – SNVS peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration :: _snvs_hp_interrupts_t

static inline void SNVS_HP_RTC_DisableInterrupts(SNVS_Type *base, uint32_t mask)

Disables the selected SNVS interrupts.

Parameters

- base – SNVS peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration :: _snvs_hp_interrupts_t

uint32_t SNVS_HP_RTC_GetEnabledInterrupts(SNVS_Type *base)

Gets the enabled SNVS interrupts.

Parameters

- base – SNVS peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration :: _snvs_hp_interrupts_t

uint32_t SNVS_HP_RTC_GetStatusFlags(SNVS_Type *base)

Gets the SNVS status flags.

Parameters

- base – SNVS peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration :: _snvs_hp_status_flags_t

static inline void SNVS_HP_RTC_ClearStatusFlags(SNVS_Type *base, uint32_t mask)

Clears the SNVS status flags.

Parameters

- base – SNVS peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration :: _snvs_hp_status_flags_t

static inline void SNVS_HP_RTC_StartTimer(SNVS_Type *base)

Starts the SNVS RTC time counter.

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_RTC_StopTimer(SNVS_Type *base)

Stops the SNVS RTC time counter.

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_EnableHighAssuranceCounter(SNVS_Type *base, bool enable)

Enable or disable the High Assurance Counter (HAC)

Parameters

- base – SNVS peripheral base address
- enable – Pass true to enable, false to disable.

static inline void SNVS_HP_StartHighAssuranceCounter(SNVS_Type *base, bool start)

Start or stop the High Assurance Counter (HAC)

Parameters

- base – SNVS peripheral base address
- start – Pass true to start, false to stop.

static inline void SNVS_HP_SetHighAssuranceCounterInitialValue(SNVS_Type *base, uint32_t value)

Set the High Assurance Counter (HAC) initialize value.

Parameters

- base – SNVS peripheral base address
- value – The initial value to set.

static inline void SNVS_HP_LoadHighAssuranceCounter(SNVS_Type *base)

Load the High Assurance Counter (HAC)

This function loads the HAC initialize value to counter register.

Parameters

- base – SNVS peripheral base address

static inline uint32_t SNVS_HP_GetHighAssuranceCounter(SNVS_Type *base)

Get the current High Assurance Counter (HAC) value.

Parameters

- base – SNVS peripheral base address

Returns

HAC currnet value.

static inline void SNVS_HP_ClearHighAssuranceCounter(SNVS_Type *base)

Clear the High Assurance Counter (HAC)

This function can be called in a functional or soft fail state. When the HAC is enabled:

- If the HAC is cleared in the soft fail state, the SSM transitions to the hard fail state immediately;
- If the HAC is cleared in functional state, the SSM will transition to hard fail immediately after transitioning to soft fail.

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_LockHighAssuranceCounter(SNVS_Type *base)

Lock the High Assurance Counter (HAC)

Once locked, the HAC initialize value could not be changed, the HAC enable status could not be changed. This could only be unlocked by system reset.

Parameters

- base – SNVS peripheral base address

FSL_SNVS_HP_DRIVER_VERSION

Version 2.3.2

enum _snvs_hp_interrupts

List of SNVS interrupts.

Values:

enumerator kSNVS_RTC_AlarmInterrupt

RTC time alarm

enumerator kSNVS_RTC_PeriodicInterrupt

RTC periodic interrupt

enum _snvs_hp_status_flags

List of SNVS flags.

Values:

enumerator kSNVS_RTC_AlarmInterruptFlag

RTC time alarm flag

enumerator kSNVS_RTC_PeriodicInterruptFlag

RTC periodic interrupt flag

enumerator kSNVS_ZMK_ZeroFlag

The ZMK is zero

enumerator kSNVS_OTPMK_ZeroFlag

The OTPMK is zero

enum _snvs_hp_sv_status_flags

List of SNVS security violation flags.

Values:

enumerator kSNVS_LP_ViolationFlag

Low Power section Security Violation

enumerator kSNVS_ZMK_EccFailFlag

Zeroizable Master Key Error Correcting Code Check Failure

enumerator kSNVS_LP_SoftwareViolationFlag
LP Software Security Violation

enumerator kSNVS_FatalSoftwareViolationFlag
Software Fatal Security Violation

enumerator kSNVS_SoftwareViolationFlag
Software Security Violation

enumerator kSNVS_Violation0Flag
Security Violation 0

enumerator kSNVS_Violation1Flag
Security Violation 1

enumerator kSNVS_Violation2Flag
Security Violation 2

enumerator kSNVS_Violation4Flag
Security Violation 4

enumerator kSNVS_Violation5Flag
Security Violation 5

enum _snvs_hp_ssm_state
List of SNVS Security State Machine State.

Values:

enumerator kSNVS_SSMInit
Init

enumerator kSNVS_SSMHardFail
Hard Fail

enumerator kSNVS_SSMSoftFail
Soft Fail

enumerator kSNVS_SSMInitInter
Init Intermediate (transition state between Init and Check)

enumerator kSNVS_SSMCheck
Check

enumerator kSNVS_SSMNonSecure
Non-Secure

enumerator kSNVS_SSMTrusted
Trusted

enumerator kSNVS_SSMSecure
Secure

typedef enum *_snvs_hp_interrupts* snvs_hp_interrupts_t
List of SNVS interrupts.

typedef enum *_snvs_hp_status_flags* snvs_hp_status_flags_t
List of SNVS flags.

typedef enum *_snvs_hp_sv_status_flags* snvs_hp_sv_status_flags_t
List of SNVS security violation flags.

typedef struct *_snvs_hp_rtc_datetime* snvs_hp_rtc_datetime_t

Structure is used to hold the date and time.

typedef struct *_snvs_hp_rtc_config* snvs_hp_rtc_config_t

SNVS config structure.

This structure holds the configuration settings for the SNVS peripheral. To initialize this structure to reasonable defaults, call the SNVS_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

typedef enum *_snvs_hp_ssm_state* snvs_hp_ssm_state_t

List of SNVS Security State Machine State.

static inline void SNVS_HP_EnableMasterKeySelection(SNVS_Type *base, bool enable)

Enable or disable master key selection.

Parameters

- base – SNVS peripheral base address
- enable – Pass true to enable, false to disable.

static inline void SNVS_HP_ProgramZeroizableMasterKey(SNVS_Type *base)

Trigger to program Zeroizable Master Key.

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_ChangeSSMState(SNVS_Type *base)

Trigger SSM State Transition.

Trigger state transition of the system security monitor (SSM). It results only the following transitions of the SSM:

- Check State -> Non-Secure (when Non-Secure Boot and not in Fab Configuration)
- Check State –> Trusted (when Secure Boot or in Fab Configuration)
- Trusted State –> Secure
- Secure State –> Trusted
- Soft Fail –> Non-Secure

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_SetSoftwareFatalSecurityViolation(SNVS_Type *base)

Trigger Software Fatal Security Violation.

The result SSM state transition is:

- Check State -> Soft Fail
- Non-Secure State -> Soft Fail
- Trusted State -> Soft Fail
- Secure State -> Soft Fail

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_SetSoftwareSecurityViolation(SNVS_Type *base)

Trigger Software Security Violation.

The result SSM state transition is:

- Check -> Non-Secure
- Trusted -> Soft Fail
- Secure -> Soft Fail

Parameters

- base – SNVS peripheral base address

static inline *snvs_hp_ssm_state_t* SNVS_HP_GetSSMState(SNVS_Type *base)

Get current SSM State.

Parameters

- base – SNVS peripheral base address

Returns

Current SSM state

static inline void SNVS_HP_ResetLP(SNVS_Type *base)

Reset the SNVS LP section.

Reset the LP section except SRTC and Time alarm.

Parameters

- base – SNVS peripheral base address

static inline uint32_t SNVS_HP_GetStatusFlags(SNVS_Type *base)

Get the SNVS HP status flags.

The flags are returned as the OR'ed value f the enumeration :: _snvs_hp_status_flags_t.

Parameters

- base – SNVS peripheral base address

Returns

The OR'ed value of status flags.

static inline void SNVS_HP_ClearStatusFlags(SNVS_Type *base, uint32_t mask)

Clear the SNVS HP status flags.

The flags to clear are passed in as the OR'ed value of the enumeration :: _snvs_hp_status_flags_t. Only these flags could be cleared using this API.

- kSNVS_RTC_PeriodicInterruptFlag
- kSNVS_RTC_AlarmInterruptFlag

Parameters

- base – SNVS peripheral base address
- mask – OR'ed value of the flags to clear.

static inline uint32_t SNVS_HP_GetSecurityViolationStatusFlags(SNVS_Type *base)

Get the SNVS HP security violation status flags.

The flags are returned as the OR'ed value of the enumeration :: _snvs_hp_sv_status_flags_t.

Parameters

- base – SNVS peripheral base address

Returns
The OR'ed value of security violation status flags.

static inline void SNVS_HP_ClearSecurityViolationStatusFlags(SNVS_Type *base, uint32_t mask)

Clear the SNVS HP security violation status flags.

The flags to clear are passed in as the OR'ed value of the enumeration :: _snvs_hp_sv_status_flags_t. Only these flags could be cleared using this API.

- kSNVS_ZMK_EccFailFlag
- kSNVS_Violation0Flag
- kSNVS_Violation1Flag
- kSNVS_Violation2Flag
- kSNVS_Violation3Flag
- kSNVS_Violation4Flag
- kSNVS_Violation5Flag

Parameters

- base – SNVS peripheral base address
- mask – OR'ed value of the flags to clear.

SNVS_HPSVSR_SV0_MASK

SNVS_HPSVSR_SV1_MASK

SNVS_HPSVSR_SV2_MASK

SNVS_HPSVSR_SV4_MASK

SNVS_HPSVSR_SV5_MASK

SNVS_MAKE_HP_SV_FLAG(x)

Macro to make security violation flag.

Macro help to make security violation flag kSNVS_Violation0Flag to kSNVS_Violation5Flag, For example, SNVS_MAKE_HP_SV_FLAG(0) is kSNVS_Violation0Flag.

struct _snvs_hp_rtc_datetime

#include <fsl_snvs_hp.h> Structure is used to hold the date and time.

Public Members

uint16_t year

Range from 1970 to 2099.

uint8_t month

Range from 1 to 12.

uint8_t day

Range from 1 to 31 (depending on month).

uint8_t hour

Range from 0 to 23.

uint8_t minute

Range from 0 to 59.

uint8_t second

Range from 0 to 59.

struct _snvs_hp_rtc_config

#include <fsl_snvs_hp.h> SNVS config structure.

This structure holds the configuration settings for the SNVS peripheral. To initialize this structure to reasonable defaults, call the SNVS_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

Public Members

bool rtcCalEnable

true: RTC calibration mechanism is enabled; false:No calibration is used

uint32_t rtcCalValue

Defines signed calibration value for nonsecure RTC; This is a 5-bit 2's complement value, range from -16 to +15

uint32_t periodicInterruptFreq

Defines frequency of the periodic interrupt; Range from 0 to 15

2.47 Secure Non-Volatile Storage Low-Power

void SNVS_LP_Init(SNVS_Type *base)

Ungates the SNVS clock and configures the peripheral for basic operation.

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

Parameters

- base – SNVS peripheral base address

void SNVS_LP_Deinit(SNVS_Type *base)

Deinit the SNVS LP section.

Parameters

- base – SNVS peripheral base address

status_t SNVS_LP_SRTC_SetDatetime(SNVS_Type *base, const *snvs_lp_srtc_datetime_t* *datetime)

Sets the SNVS SRTC date and time according to the given time structure.

Parameters

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

Returns

kStatus_Success: Success in setting the time and starting the SNVS SRTC kStatus_InvalidArgument: Error because the datetime format is incorrect

void SNVS_LP_SRTC_GetDatetime(SNVS_Type *base, *snvs_lp_srtc_datetime_t* *datetime)

Gets the SNVS SRTC time and stores it in the given time structure.

Parameters

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

status_t SNVS_LP_SRTC_SetAlarm(SNVS_Type *base, const *snvs_lp_srtc_datetime_t* *alarmTime)

Sets the SNVS SRTC alarm time.

The function sets the SRTC alarm. It also checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error. Please note, that SRTC alarm has limited resolution because only 32 most significant bits of SRTC counter are compared to SRTC Alarm register. If the alarm time is beyond SRTC resolution, the function does not set the alarm and returns an error.

Parameters

- base – SNVS peripheral base address
- alarmTime – Pointer to the structure where the alarm time is stored.

Returns

kStatus_Success: success in setting the SNVS SRTC alarm kStatus_InvalidArgument: Error because the alarm datetime format is incorrect kStatus_Fail: Error because the alarm time has already passed or is beyond resolution

void SNVS_LP_SRTC_GetAlarm(SNVS_Type *base, *snvs_lp_srtc_datetime_t* *datetime)

Returns the SNVS SRTC alarm time.

Parameters

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the alarm date and time details are stored.

static inline void SNVS_LP_SRTC_EnableInterrupts(SNVS_Type *base, uint32_t mask)

Enables the selected SNVS interrupts.

Parameters

- base – SNVS peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration :: _snvs_lp_srtc_interrupts

static inline void SNVS_LP_SRTC_DisableInterrupts(SNVS_Type *base, uint32_t mask)

Disables the selected SNVS interrupts.

Parameters

- base – SNVS peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration :: _snvs_lp_srtc_interrupts

uint32_t SNVS_LP_SRTC_GetEnabledInterrupts(SNVS_Type *base)

Gets the enabled SNVS interrupts.

Parameters

- base – SNVS peripheral base address

Returns
The enabled interrupts. This is the logical OR of members of the enumeration :: _snvs_lp_srtc_interrupts

uint32_t SNVS_LP_SRTC_GetStatusFlags(SNVS_Type *base)

Gets the SNVS status flags.

Parameters

- base – SNVS peripheral base address

Returns
The status flags. This is the logical OR of members of the enumeration :: _snvs_lp_srtc_status_flags

static inline void SNVS_LP_SRTC_ClearStatusFlags(SNVS_Type *base, uint32_t mask)

Clears the SNVS status flags.

Parameters

- base – SNVS peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration :: _snvs_lp_srtc_status_flags

static inline void SNVS_LP_SRTC_StartTimer(SNVS_Type *base)

Starts the SNVS SRTC time counter.

Parameters

- base – SNVS peripheral base address

static inline void SNVS_LP_SRTC_StopTimer(SNVS_Type *base)

Stops the SNVS SRTC time counter.

Parameters

- base – SNVS peripheral base address

void SNVS_LP_PassiveTamperPin_GetDefaultConfig(*snvs_lp_passive_tamper_t* *config)

Fills in the SNVS tamper pin config struct with the default settings.

The default values are as follows. code config->polarity = 0U; config->filterenable = 0U; if available on SoC config->filter = 0U; if available on SoC endcode

Parameters

- config – Pointer to the user's SNVS configuration structure.

static inline void SNVS_LP_EnableMonotonicCounter(SNVS_Type *base, bool enable)

Enable or disable the Monotonic Counter.

Parameters

- base – SNVS peripheral base address
- enable – Pass true to enable, false to disable.

uint64_t SNVS_LP_GetMonotonicCounter(SNVS_Type *base)

Get the current Monotonic Counter.

Parameters

- base – SNVS peripheral base address

Returns
Current Monotonic Counter value.

static inline void SNVS_LP_IncreaseMonotonicCounter(SNVS_Type *base)

Increase the Monotonic Counter.

Increase the Monotonic Counter by 1.

Parameters

- base – SNVS peripheral base address

void SNVS_LP_WriteZeroizableMasterKey(SNVS_Type *base, uint32_t ZMKey[8U])

Write Zeroizable Master Key (ZMK) to the SNVS registers.

Parameters

- base – SNVS peripheral base address
- ZMKey – The ZMK write to the SNVS register.

static inline void SNVS_LP_SetZeroizableMasterKeyValid(SNVS_Type *base, bool valid)

Set Zeroizable Master Key valid.

This API could only be called when using software programming mode. After writing ZMK using SNVS_LP_WriteZeroizableMasterKey, call this API to make the ZMK valid.

Parameters

- base – SNVS peripheral base address
- valid – Pass true to set valid, false to set invalid.

static inline bool SNVS_LP_GetZeroizableMasterKeyValid(SNVS_Type *base)

Get Zeroizable Master Key valid status.

In hardware programming mode, call this API to check whether the ZMK is valid.

Parameters

- base – SNVS peripheral base address

Returns

true if valid, false if invalid.

static inline void SNVS_LP_SetZeroizableMasterKeyProgramMode(SNVS_Type *base, *snvs_lp_zmk_program_mode_t* mode)

Set Zeroizable Master Key programming mode.

Parameters

- base – SNVS peripheral base address
- mode – ZMK programming mode.

static inline void SNVS_LP_EnableZeroizableMasterKeyECC(SNVS_Type *base, bool enable)

Enable or disable Zeroizable Master Key ECC.

Parameters

- base – SNVS peripheral base address
- enable – Pass true to enable, false to disable.

static inline void SNVS_LP_SetMasterKeyMode(SNVS_Type *base, *snvs_lp_master_key_mode_t* mode)

Set SNVS Master Key mode.

Note: When kSNVS_ZMK or kSNVS_CMK used, the SNVS_HP must be configured to enable the master key selection.

Parameters

- base – SNVS peripheral base address
- mode – Master Key mode.

FSL_SNVS_LP_DRIVER_VERSION
Version 2.4.6

enum _snvs_lp_srtc_interrupts
List of SNVS_LP interrupts.

Values:

enumerator kSNVS_SRTC_AlarmInterrupt
SRTC time alarm.

enum _snvs_lp_srtc_status_flags
List of SNVS_LP flags.

Values:

enumerator kSNVS_SRTC_AlarmInterruptFlag
SRTC time alarm flag

enum _snvs_lp_external_tamper_status
List of SNVS_LP external tampers status.

Values:

enumerator kSNVS_TamperNotDetected

enumerator kSNVS_TamperDetected

enum _snvs_lp_external_tamper_polarity
SNVS_LP external tamper polarity.

Values:

enumerator kSNVS_ExternalTamperActiveLow

enumerator kSNVS_ExternalTamperActiveHigh

enum _snvs_lp_zmk_program_mode
SNVS_LP Zeroizable Master Key programming mode.

Values:

enumerator kSNVS_ZMKSoftwareProgram
Software programming mode.

enumerator kSNVS_ZMKHardwareProgram
Hardware programming mode.

enum _snvs_lp_master_key_mode
SNVS_LP Master Key mode.

Values:

enumerator kSNVS_OTPMK
One Time Programmable Master Key.

enumerator kSNVS_ZMK
Zeroizable Master Key.

enumerator kSNVS_CMK
Combined Master Key, it is XOR of OPTMK and ZMK.

typedef enum *_snvs_lp_srtc_interrupts* snvs_lp_srtc_interrupts_t
List of SNVS_LP interrupts.

typedef enum *_snvs_lp_srtc_status_flags* snvs_lp_srtc_status_flags_t
List of SNVS_LP flags.

typedef enum *_snvs_lp_external_tamper_status* snvs_lp_external_tamper_status_t
List of SNVS_LP external tampers status.

typedef enum *_snvs_lp_external_tamper_polarity* snvs_lp_external_tamper_polarity_t
SNVS_LP external tamper polarity.

typedef struct *_snvs_lp_srtc_datetime* snvs_lp_srtc_datetime_t
Structure is used to hold the date and time.

typedef struct *_snvs_lp_srtc_config* snvs_lp_srtc_config_t
SNVS_LP config structure.

This structure holds the configuration settings for the SNVS_LP peripheral. To initialize this structure to reasonable defaults, call the SNVS_LP_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

typedef enum *_snvs_lp_zmk_program_mode* snvs_lp_zmk_program_mode_t
SNVS_LP Zeroizable Master Key programming mode.

typedef enum *_snvs_lp_master_key_mode* snvs_lp_master_key_mode_t
SNVS_LP Master Key mode.

void SNVS_LP_SRTC_Init(SNVS_Type *base, const *snvs_lp_srtc_config_t* *config)
Ungates the SNVS clock and configures the peripheral for basic operation.

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

Parameters

- base – SNVS peripheral base address
- config – Pointer to the user's SNVS configuration structure.

void SNVS_LP_SRTC_Deinit(SNVS_Type *base)
Stops the SRTC timer.

Parameters

- base – SNVS peripheral base address

void SNVS_LP_SRTC_GetDefaultConfig(*snvs_lp_srtc_config_t* *config)
Fills in the SNVS_LP config struct with the default settings.

The default values are as follows.

```
config->srtccalenable = false;
config->srtccalvalue = 0U;
```

Parameters

- config – Pointer to the user's SNVS configuration structure.

SNVS_ZMK_REG_COUNT

Define of SNVS_LP Zeroizable Master Key registers.

SNVS_LP_MAX_TAMPER

Define of SNVS_LP Max possible tamper.

struct snvs_lp_passive_tamper_t

#include <fsl_snvs_lp.h> Structure is used to configure SNVS LP passive tamper pins.

struct _snvs_lp_srtc_datetime

#include <fsl_snvs_lp.h> Structure is used to hold the date and time.

Public Members

uint16_t year

Range from 1970 to 2099.

uint8_t month

Range from 1 to 12.

uint8_t day

Range from 1 to 31 (depending on month).

uint8_t hour

Range from 0 to 23.

uint8_t minute

Range from 0 to 59.

uint8_t second

Range from 0 to 59.

struct _snvs_lp_srtc_config

#include <fsl_snvs_lp.h> SNVS_LP config structure.

This structure holds the configuration settings for the SNVS_LP peripheral. To initialize this structure to reasonable defaults, call the SNVS_LP_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

Public Members

bool srtcCalEnable

true: SRTC calibration mechanism is enabled; false: No calibration is used

uint32_t srtcCalValue

Defines signed calibration value for SRTC; This is a 5-bit 2's complement value, range from -16 to +15

2.48 SRC: System Reset Controller Driver

FSL_SRC_DRIVER_VERSION

SRC driver version 2.0.1.

enum _src_reset_status_flags

SRC reset status flags.

Values:

enumerator kSRC_WarmBootIndicationFlag

WARM boot indication shows that WARM boot was initiated by software.

enumerator kSRC_TemperatureSensorResetFlag

Indicates whether the reset was the result of software reset from on-chip Temperature Sensor. Temperature Sensor Interrupt needs to be served before this bit can be cleaned.

enumerator kSRC_JTAGSoftwareResetFlag

Indicates whether the reset was the result of setting SJC_GPCCR bit 31.

enumerator kSRC_JTAGGeneratedResetFlag

Indicates a reset has been caused by JTAG selection of certain IR codes: EXTEST or HIGHZ.

enumerator kSRC_WatchdogResetFlag

Indicates a reset has been caused by the watchdog timer timing out. This reset source can be blocked by disabling the watchdog.

enum _src_warm_reset_bypass_count

Selection of WARM reset bypass count.

This type defines the 32KHz clock cycles to count before bypassing the MMDC acknowledge for WARM reset. If the MMDC acknowledge is not asserted before this counter is elapsed, a COLD reset will be initiated.

Values:

enumerator kSRC_WarmResetWaitAlways

System will wait until MMDC acknowledge is asserted.

enumerator kSRC_WarmResetWaitClk16

Wait 16 32KHz clock cycles before switching the reset.

enumerator kSRC_WarmResetWaitClk32

Wait 32 32KHz clock cycles before switching the reset.

enumerator kSRC_WarmResetWaitClk64

Wait 64 32KHz clock cycles before switching the reset.

typedef enum *_src_warm_reset_bypass_count* src_warm_reset_bypass_count_t

Selection of WARM reset bypass count.

This type defines the 32KHz clock cycles to count before bypassing the MMDC acknowledge for WARM reset. If the MMDC acknowledge is not asserted before this counter is elapsed, a COLD reset will be initiated.

static inline void SRC_EnableWDOGReset(SRC_Type *base, bool enable)

Enable the WDOG Reset in SRC.

WDOG Reset is enabled in SRC by default. If the WDOG event to SRC is masked, it would not create a reset to the chip. During the time the WDOG event is masked, when the WDOG event flag is asserted, it would remain asserted regardless of servicing the WDOG module. The only way to clear that bit is the hardware reset.

Parameters

- base – SRC peripheral base address.
- enable – Enable the reset or not.

static inline void SRC_SetWarmResetBypassCount(SRC_Type *base,
src_warm_reset_bypass_count_t option)

Set the delay count of waiting MMDC's acknowledge.

This function would define the 32KHz clock cycles to count before bypassing the MMDC acknowledge for WARM reset. If the MMDC acknowledge is not asserted before this counter is elapsed, a COLD reset will be initiated.

Parameters

- base – SRC peripheral base address.
- option – The option of setting mode, see to src_warm_reset_bypass_count_t.

static inline void SRC_EnableWarmReset(SRC_Type *base, bool enable)

Enable the WARM reset.

Only when the WARM reset is enabled, the WARM reset requests would be served by WARM reset. Otherwise, all the WARM reset sources would generate COLD reset.

Parameters

- base – SRC peripheral base address.
- enable – Enable the WARM reset or not.

static inline uint32_t SRC_GetBootModeWord1(SRC_Type *base)

Get the boot mode register 1 value.

The Boot Mode register contains bits that reflect the status of BOOT_CFGx pins of the chip. See to chip-specific document for detail information about value.

Parameters

- base – SRC peripheral base address.

Returns

status of BOOT_CFGx pins of the chip.

static inline uint32_t SRC_GetBootModeWord2(SRC_Type *base)

Get the boot mode register 2 value.

The Boot Mode register contains bits that reflect the status of BOOT_MODEx Pins and fuse values that controls boot of the chip. See to chip-specific document for detail information about value.

Parameters

- base – SRC peripheral base address.

Returns

status of BOOT_MODEx Pins and fuse values that controls boot of the chip.

static inline void SRC_SetWarmBootIndication(SRC_Type *base, bool enable)

Set the warm boot indication flag.

WARM boot indication shows that WARM boot was initiated by software. This indicates to the software that it saved the needed information in the memory before initiating the WARM reset. In this case, software will set this bit to '1', before initiating the WARM reset. The warm_boot bit should be used as indication only after a warm_reset sequence. Software should clear this bit after warm_reset to indicate that the next warm_reset is not performed with warm_boot.

Parameters

- base – SRC peripheral base address.
- enable – Assert the flag or not.

static inline uint32_t SRC_GetResetStatusFlags(SRC_Type *base)

Get the status flags of SRC.

Parameters

- base – SRC peripheral base address.

Returns

Mask value of status flags, see to _src_reset_status_flags.

void SRC_ClearResetStatusFlags(SRC_Type *base, uint32_t flags)

Clear the status flags of SRC.

Parameters

- base – SRC peripheral base address.
- flags – value of status flags to be cleared, see to _src_reset_status_flags.

static inline void SRC_SetGeneralPurposeRegister(SRC_Type *base, uint32_t index, uint32_t value)

Set value to general purpose registers.

General purpose registers (GPRx) would hold the value during reset process. Wakeup function could be kept in these register. For example, the GPR1 holds the entry function for waking-up from Partial SLEEP mode while the GPR2 holds the argument. Other GPRx register would store the arbitray values.

Parameters

- base – SRC peripheral base address.
- index – The index of GPRx register array. Note index 0 reponses the GPR1 register.
- value – Setting value for GPRx register.

static inline uint32_t SRC_GetGeneralPurposeRegister(SRC_Type *base, uint32_t index)

Get the value from general purpose registers.

Parameters

- base – SRC peripheral base address.
- index – The index of GPRx register array. Note index 0 reponses the GPR1 register.

Returns

The setting value for GPRx register.

2.49 TMU: Thermal Management Unit Driver

TMU interrupt enable, _tmu_interrupt_enable.

Values:

enumerator kTMU_ImmediateTemperature0InterruptEnable

Immediate temperature threshold exceeded interrupt enable of probe0.

enumerator kTMU_AverageTemperature0InterruptEnable

Average temperature threshold exceeded interrupt enable of probe0.

enumerator kTMU_AverageTemperature0CriticalInterruptEnable

Average temperature critical threshold exceeded interrupt enable of probe0.

enumerator kTMU_ImmediateTemperature1Interrupt1Enable
Immediate temperature threshold exceeded interrupt enable of probe1.

enumerator kTMU_AverageTemperature1Interrupt1Enable
Average temperature threshold exceeded interrupt enable of probe1.

enumerator kTMU_AverageTemperature1CriticalInterrupt1Enable
Average temperature critical threshold exceeded interrupt enable of probe1.

TMU interrupt enable, _tmu_interrupt_status_flags.

Values:

enumerator kTMU_ImmediateTemperature0InterruptStausFlags
Immediate temperature threshold exceeded interrupt status of probe0.

enumerator kTMU_AverageTemperature0InterruptStausFlags
Average temperature threshold exceeded interrupt status of probe0.

enumerator kTMU_AverageTemperature0CriticalInterruptStausFlags
Average temperature critical threshold exceeded interrupt status of probe0.

enumerator kTMU_ImmediateTemperature1Interrupt1StausFlags
Immediate temperature threshold exceeded interrupt status of probe1.

enumerator kTMU_AverageTemperature1Interrupt1StausFlags
Average temperature threshold exceeded interrupt status of probe1.

enumerator kTMU_AverageTemperature1CriticalInterrupt1StausFlags
Average temperature critical threshold exceeded interrupt status of probe1.

enum _tmu_probe_select
Probe selection.

Values:

enumerator kTMU_ProbeSelectMainProbe
Select the main probe only.

enumerator kTMU_ProbeSelectRemoteProbe
Select the remote probe(near A53) only.

enumerator kTMU_ProbeSelectBothProbes
Select both 2 probes.

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 enum *_tmu_probe_select* tmu_probe_select_t
Probe selection.

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_Enable(TMU_Type *base, bool enable)
Enable/disable TMU module.

Parameters

- base – TMU peripheral base address.
- enable – enable or disable TMU.

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.averageLPF                                = kTMU_AverageLowPassFilter0_5;
config.probeSelect                               = kTMU_ProbeSelectMainProbe;
config.thresholdConfig.immediateThresholdEnable  = false;
config.thresholdConfig.immediateThresholdValue   = DEMO_TMU_IMMEDIATE_THRESOLD;
config.thresholdConfig.AverageThresholdEnable    = true;
config.thresholdConfig.averageThresholdValue     = DEMO_TMU_AVERAGE_THRESOLD;
config.thresholdConfig.AverageCriticalThresholdEnable = false;
config.thresholdConfig.averageCriticalThresholdValue  = DEMO_TMU_AVERAGE_CRITICAL_
↪THRESOLD;
```

Parameters

- config – Pointer to TMU configuration structure.

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.

static inline uint32_t TMU_GetInterruptStatusFlags(TMU_Type *base)

Get interrupt status flags.

Parameters

- base – TMU peripheral base address.

Return values

The – current interrupt status.

static inline void TMU_ClearInterruptStatusFlags(TMU_Type *base, uint32_t mask)

Clear interrupt status flags.

Parameters

- base – TMU peripheral base address.
- mask – The mask of interrupt status flags. Refer to "_tmu_interrupt_status_flags" enumeration.

status_t TMU_GetImmediateTemperature(TMU_Type *base, *tmu_probe_select_t* probe, int8_t *temperature)

Get the last immediate temperature at site.

Parameters

- base – TMU peripheral base address.
- probe – probe selection, if select both 2 probes, return main probe temeperature by default. Refer to "tmu_probe_select_t" structure.
- temperature – Last immediate temperature reading at site when V=1. besides, Bit-8 is sign bit: 1 means nagetive and 0 means positive.

Return values

get – immediate temperature status.

status_t TMU_GetAverageTemperature(TMU_Type *base, *tmu_probe_select_t* probe, int8_t *temperature)

Get the last average temperature at site.

Parameters

- base – TMU peripheral base address.
- probe – probe selection, if select both 2 probes, return main probe temeperature by default. Refer to "tmu_probe_select_t" structure.
- temperature – Last average temperature reading at site; besides, Bit-8 is sign bit: 1 means nagetive and 0 means positive.

Return values

get – average temperature status.

void TMU_UpdateHighTemperatureThreshold(TMU_Type *base, *tmu_probe_select_t* probe, const *tmu_threshold_config_t* *thresholdConfig)

Update the high temperature threshold value.

Parameters

- base – TMU peripheral base address.
- probe – probe selection, if select both 2 probes, set main probe path by default. Refer to “tmu_probe_select_t” structure.
- thresholdConfig – threshold configuration. Refer to “tmu_threshold_config_t” structure.

FSL_TMU_DRIVER_VERSION

TMU driver version.

Version 2.0.0.

struct tmu_threshold_config

#include <fsl_tmu.h> configuration for TMU threshold.

Public Members

bool immediateThresholdEnable

Enable high temperature immediate threshold.

bool AverageThresholdEnable

Enable high temperature average threshold.

bool AverageCriticalThresholdEnable

Enable high temperature average critical threshold.

uint8_t immediateThresholdValueOfMainProbe

Range:-40~125. Valid when corresponding threshold is enabled. High temperature immediate threshold value of main probe. Besides, bit-8 is sign bit: 1 means nagetive and 0 means positive.

uint8_t averageThresholdValueOfMainProbe

Range:-40~125. Valid when corresponding threshold is enabled. High temperature average threshold value of main probe. Besides, bit-8 is sign bit: 1 means nagetive and 0 means positive.

uint8_t averageCriticalThresholdValueOfMainProbe

Range:-40~125. Valid when corresponding threshold is enabled. High temperature average critical threshold value of main probe. Besides, bit-8 is sign bit: 1 means nagetive and 0 means positive.

uint8_t immediateThresholdValueOfRemoteProbe

Range:-40~125. Valid when corresponding threshold is enabled. High temperature immediate threshold value of remote probe. Besides, bit-8 is sign bit: 1 means nagetive and 0 means positive.

uint8_t averageThresholdValueOfRemoteProbe

Range:-40~125. Valid when corresponding threshold is enabled. High temperature average threshold value of remote probe. Besides, bit-8 is sign bit: 1 means nagetive and 0 means positive.

uint8_t averageCriticalThresholdValueOfRemoteProbe

Range:-40~125. Valid when corresponding threshold is enabled. High temperature average critical threshold value of remote probe. Besides, bit-8 is sign bit: 1 means nagetive and 0 means positive.

struct _tmu_config

#include <fsl_tmu.h> Configuration for TMU module.

Public Members

tmu_probe_select_t probeSelect

The temperature monitor probe select.

tmu_average_low_pass_filter_t averageLPF

The average temperature is calculated as: ALPF x Current_Temp + (1 - ALPF) x Average_Temp. For proper operation, this field should only change when monitoring is disabled.

tmu_threshold_config_t thresholdConfig

The high temperature threshold configuration.

2.50 UART: Universal Asynchronous Receiver/Transmitter Driver

2.51 UART Driver

static inline void UART_SoftwareReset(UART_Type *base)

Resets the UART using software.

This function resets the transmit and receive state machines, all FIFOs and register USR1, USR2, UBIR, UBMR, UBRC , URXD, UTXD and UTS[6-3]

Parameters

- base – UART peripheral base address.

status_t UART_Init(UART_Type *base, const *uart_config_t* *config, uint32_t srcClock_Hz)

Initializes an UART instance with the user configuration structure and the peripheral clock.

This function configures the UART module with user-defined settings. Call the UART_GetDefaultConfig() function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the UART.

```
uart_config_t uartConfig;
uartConfig.baudRate_Bps = 115200U;
uartConfig.parityMode = kUART_ParityDisabled;
uartConfig.dataBitsCount = kUART_EightDataBits;
uartConfig.stopBitCount = kUART_OneStopBit;
uartConfig.txFifoWatermark = 2;
uartConfig.rxFifoWatermark = 1;
uartConfig.enableAutoBaudrate = false;
uartConfig.enableTx = true;
uartConfig.enableRx = true;
UART_Init(UART1, &uartConfig, 24000000U);
```

Parameters

- base – UART peripheral base address.
- config – Pointer to a user-defined configuration structure.
- srcClock_Hz – UART clock source frequency in HZ.

Return values

kStatus_Success – UART initialize succeed

void UART_Deinit(UART_Type *base)

Deinitializes a UART instance.

This function waits for transmit to complete, disables TX and RX, and disables the UART clock.

Parameters

- base – UART peripheral base address.

void UART_GetDefaultConfig(*uart_config_t* *config)

Gets the default configuration structure.

l

This function initializes the UART configuration structure to a default value. The default values are: uartConfig->baudRate_Bps = 115200U; uartConfig->parityMode = kUART_ParityDisabled; uartConfig->dataBitsCount = kUART_EightDataBits; uartConfig->stopBitCount = kUART_OneStopBit; uartConfig->txFifoWatermark = 2; uartConfig->rxFifoWatermark = 1; uartConfig->enableAutoBaudrate = flase; uartConfig->enableTx = false; uartConfig->enableRx = false;

Parameters

- config – Pointer to a configuration structure.

status_t UART_SetBaudRate(UART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the UART instance baud rate.

This function configures the UART module baud rate. This function is used to update the UART module baud rate after the UART module is initialized by the UART_Init.

```
UART_SetBaudRate(UART1, 115200U, 20000000U);
```

Parameters

- base – UART peripheral base address.
- baudRate_Bps – UART baudrate to be set.
- srcClock_Hz – UART clock source frequency in Hz.

Return values

- kStatus_UART_BaudrateNotSupport – Baudrate is not support in the current clock source.
- kStatus_Success – Set baudrate succeeded.

static inline void UART_Enable(UART_Type *base)

This function is used to Enable the UART Module.

Parameters

- base – UART base pointer.

static inline void UART_SetIdleCondition(UART_Type *base, *uart_idle_condition_t* condition)

This function is used to configure the IDLE line condition.

Parameters

- base – UART base pointer.
- condition – IDLE line detect condition of the enumerators in uart_idle_condition_t.

static inline void UART_Disable(UART_Type *base)

This function is used to Disable the UART Module.

Parameters

- base – UART base pointer.

bool UART_GetStatusFlag(UART_Type *base, uint32_t flag)

This function is used to get the current status of specific UART status flag(including interrupt flag). The available status flag can be select from uart_status_flag_t enumeration.

Parameters

- base – UART base pointer.
- flag – Status flag to check.

Return values

current – state of corresponding status flag.

void UART_ClearStatusFlag(UART_Type *base, uint32_t flag)

This function is used to clear the current status of specific UART status flag. The available status flag can be select from uart_status_flag_t enumeration.

Parameters

- base – UART base pointer.
- flag – Status flag to clear.

void UART_EnableInterrupts(UART_Type *base, uint32_t mask)

Enables UART interrupts according to the provided mask.

This function enables the UART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See _uart_interrupt_enable. For example, to enable TX empty interrupt and RX data ready interrupt, do the following.

```
UART_EnableInterrupts(UART1,kUART_TxEmptyEnable | kUART_RxDataReadyEnable);
```

Parameters

- base – UART peripheral base address.
- mask – The interrupts to enable. Logical OR of _uart_interrupt_enable.

void UART_DisableInterrupts(UART_Type *base, uint32_t mask)

Disables the UART interrupts according to the provided mask.

This function disables the UART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See _uart_interrupt_enable. For example, to disable TX empty interrupt and RX data ready interrupt do the following.

```
UART_EnableInterrupts(UART1,kUART_TxEmptyEnable | kUART_RxDataReadyEnable);
```

Parameters

- base – UART peripheral base address.
- mask – The interrupts to disable. Logical OR of _uart_interrupt_enable.

uint32_t UART_GetEnabledInterrupts(UART_Type *base)

Gets enabled UART interrupts.

This function gets the enabled UART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators _uart_interrupt_enable. To check a specific interrupt enable status, compare the return value with enumerators in _uart_interrupt_enable. For example, to check whether the TX empty interrupt is enabled:

```
uint32_t enabledInterrupts = UART_GetEnabledInterrupts(UART1);

if (kUART_TxEmptyEnable & enabledInterrupts)
{
    ...
}
```

Parameters

- base – UART peripheral base address.

Returns

UART interrupt flags which are logical OR of the enumerators in _uart_interrupt_enable.

static inline void UART_EnableTx(UART_Type *base, bool enable)

Enables or disables the UART transmitter.

This function enables or disables the UART transmitter.

Parameters

- base – UART peripheral base address.
- enable – True to enable, false to disable.

static inline void UART_EnableRx(UART_Type *base, bool enable)

Enables or disables the UART receiver.

This function enables or disables the UART receiver.

Parameters

- base – UART peripheral base address.
- enable – True to enable, false to disable.

static inline void UART_WriteByte(UART_Type *base, uint8_t data)

Writes to the transmitter register.

This function is used to write data to transmitter register. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

- base – UART peripheral base address.
- data – Data write to the TX register.

static inline uint8_t UART_ReadByte(UART_Type *base)

Reads the receiver register.

This function is used to read data from receiver register. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

- base – UART peripheral base address.

Returns

Data read from data register.

status_t UART_WriteBlocking(UART_Type *base, const uint8_t *data, size_t length)

Writes to the TX register using a blocking method.

This function polls the TX register, waits for the TX register to be empty or for the TX FIFO to have room and writes data to the TX buffer.

Parameters

- base – UART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t UART_ReadBlocking(UART_Type *base, uint8_t *data, size_t length)

Read RX data register using a blocking method.

This function polls the RX register, waits for the RX register to be full or for RX FIFO to have data, and reads data from the TX register.

Parameters

- base – UART peripheral base address.
- data – Start address of the buffer to store the received data.
- length – Size of the buffer.

Return values

- kStatus_UART_RxHardwareOverrun – Receiver overrun occurred while receiving data.
- kStatus_UART_NoiseError – A noise error occurred while receiving data.
- kStatus_UART_FramingError – A framing error occurred while receiving data.
- kStatus_UART_ParityError – A parity error occurred while receiving data.
- kStatus_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

void UART_TransferCreateHandle(UART_Type *base, *uart_handle_t* *handle, *uart_transfer_callback_t* callback, void *userData)

Initializes the UART handle.

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

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.
- callback – The callback function.
- userData – The parameter of the callback function.

void UART_TransferStartRingBuffer(UART_Type *base, *uart_handle_t* *handle, uint8_t *ringBuffer, size_t ringBufferSize)

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received are stored into the ring buffer even when the user doesn't call the UART_TransferReceiveNonBlocking() API. If data is already received in the ring buffer, the user can get the received data from the ring buffer directly.

Note: When using the RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, only 31 bytes are used for saving data.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.
- ringBuffer – Start address of the ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – Size of the ring buffer.

void UART_TransferStopRingBuffer(UART_Type *base, *uart_handle_t* *handle)

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.

size_t UART_TransferGetRxRingBufferLength(*uart_handle_t* *handle)

Get the length of received data in RX ring buffer.

Parameters

- handle – UART handle pointer.

Returns

Length of received data in RX ring buffer.

status_t UART_TransferSendNonBlocking(UART_Type *base, *uart_handle_t* *handle, *uart_transfer_t* *xfer)

Transmits a buffer of data using the interrupt method.

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in the ISR, the UART driver calls the callback function and passes the kStatus_UART_TxIdle as status parameter.

Note: The kStatus_UART_TxIdle is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out. Before disabling the TX, check the kUART_TransmissionCompleteFlag to ensure that the TX is finished.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.
- xfer – UART transfer structure. See uart_transfer_t.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_UART_TxBusy – Previous transmission still not finished; data not all written to TX register yet.
- kStatus_InvalidArgument – Invalid argument.

void UART_TransferAbortSend(UART_Type *base, *uart_handle_t* *handle)

Aborts the interrupt-driven data transmit.

This function aborts the interrupt-driven data sending. The user can get the remainBytes to find out how many bytes are not sent out.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.

status_t UART_TransferGetSendCount(UART_Type *base, *uart_handle_t* *handle, uint32_t *count)

Gets the number of bytes written to the UART TX register.

This function gets the number of bytes written to the UART TX register by using the interrupt method.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.
- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – The parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

status_t UART_TransferReceiveNonBlocking(UART_Type *base, *uart_handle_t* *handle, *uart_transfer_t* *xfer, size_t *receivedBytes)

Receives a buffer of data using an interrupt method.

This function receives data using an interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter receivedBytes shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough to read, the receive request is saved by the UART driver. When the new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter kStatus_UART_RxIdle. For example, the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer. The 5 bytes are copied to the xfer->data and this function returns with the parameter receivedBytes set to 5. For the left 5 bytes, newly arrived data is saved from the xfer->data[5]. When 5 bytes are received, the UART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to the xfer->data. When all data is received, the upper layer is notified.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.
- xfer – UART transfer structure, see uart_transfer_t.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into transmit queue.
- kStatus_UART_RxBusy – Previous receive request is not finished.
- kStatus_InvalidArgument – Invalid argument.

void UART_TransferAbortReceive(UART_Type *base, *uart_handle_t* *handle)

Aborts the interrupt-driven data receiving.

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to know how many bytes are not received yet.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.

status_t UART_TransferGetReceiveCount(UART_Type *base, *uart_handle_t* *handle, uint32_t *count)

Gets the number of bytes that have been received.

This function gets the number of bytes that have been received.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.
- count – Receive bytes count.

Return values

- kStatus_NoTransferInProgress – No receive in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

void UART_TransferHandleIRQ(UART_Type *base, void *irqHandle)

UART IRQ handle function.

This function handles the UART transmit and receive IRQ request.

Parameters

- base – UART peripheral base address.
- irqHandle – UART handle pointer.

static inline void UART_EnableTxDMA(UART_Type *base, bool enable)

Enables or disables the UART transmitter DMA request.

This function enables or disables the transmit request when the transmitter has one or more slots available in the TxFIFO. The fill level in the TxFIFO that generates the DMA request is controlled by the TXTL bits.

Parameters

- base – UART peripheral base address.
- enable – True to enable, false to disable.

static inline void UART_EnableRxDMA(UART_Type *base, bool enable)

Enables or disables the UART receiver DMA request.

This function enables or disables the receive request when the receiver has data in the RxFIFO. The fill level in the RxFIFO at which a DMA request is generated is controlled by the RXTL bits .

Parameters

- base – UART peripheral base address.
- enable – True to enable, false to disable.

static inline void UART_SetTxFifoWatermark(UART_Type *base, uint8_t watermark)

This function is used to set the watermark of UART Tx FIFO. A maskable interrupt is generated whenever the data level in the TxFIFO falls below the Tx FIFO watermark.

Parameters

- base – UART base pointer.
- watermark – The Tx FIFO watermark.

static inline void UART_SetRxRTSWatermark(UART_Type *base, uint8_t watermark)

This function is used to set the watermark of UART RTS deassertion.

The RTS signal deasserts whenever the data count in RxFIFO reaches the Rx RTS watermark.

Parameters

- base – UART base pointer.
- watermark – The Rx RTS watermark.

static inline void UART_SetRxFifoWatermark(UART_Type *base, uint8_t watermark)

This function is used to set the watermark of UART Rx FIFO. A maskable interrupt is generated whenever the data level in the RxFIFO reaches the Rx FIFO watermark.

Parameters

- base – UART base pointer.
- watermark – The Rx FIFO watermark.

static inline void UART_EnableAutoBaudRate(UART_Type *base, bool enable)

This function is used to set the enable condition of Automatic Baud Rate Detection feature.

Parameters

- base – UART base pointer.
- enable – Enable/Disable Automatic Baud Rate Detection feature.
  - true: Enable Automatic Baud Rate Detection feature.
  - false: Disable Automatic Baud Rate Detection feature.

static inline bool UART_IsAutoBaudRateComplete(UART_Type *base)

This function is used to read if the automatic baud rate detection has finished.

Parameters

- base – UART base pointer.

Returns

- true: Automatic baud rate detection has finished.
- false: Automatic baud rate detection has not finished.

FSL_UART_DRIVER_VERSION

UART driver version.

Error codes for the UART driver.

Values:

enumerator kStatus_UART_TxBusy

Transmitter is busy.

enumerator kStatus_UART_RxBusy

Receiver is busy.

enumerator kStatus_UART_TxIdle
UART transmitter is idle.

enumerator kStatus_UART_RxIdle
UART receiver is idle.

enumerator kStatus_UART_TxWatermarkTooLarge
TX FIFO watermark too large

enumerator kStatus_UART_RxWatermarkTooLarge
RX FIFO watermark too large

enumerator kStatus_UART_FlagCannotClearManually
UART flag can't be manually cleared.

enumerator kStatus_UART_Error
Error happens on UART.

enumerator kStatus_UART_RxRingBufferOverrun
UART RX software ring buffer overrun.

enumerator kStatus_UART_RxHardwareOverrun
UART RX receiver overrun.

enumerator kStatus_UART_NoiseError
UART noise error.

enumerator kStatus_UART_FramingError
UART framing error.

enumerator kStatus_UART_ParityError
UART parity error.

enumerator kStatus_UART_BaudrateNotSupport
Baudrate is not support in current clock source

enumerator kStatus_UART_BreakDetect
Receiver detect BREAK signal

enumerator kStatus_UART_Timeout
UART times out.

enum _uart_data_bits
UART data bits count.

Values:

enumerator kUART_SevenDataBits
Seven data bit

enumerator kUART_EightDataBits
Eight data bit

enum _uart_parity_mode
UART parity mode.

Values:

enumerator kUART_ParityDisabled
Parity disabled

enumerator kUART_ParityEven
Even error check is selected

enumerator kUART_ParityOdd
Odd error check is selected

enum _uart_stop_bit_count
UART stop bit count.

Values:

enumerator kUART_OneStopBit
One stop bit

enumerator kUART_TwoStopBit
Two stop bits

enum _uart_idle_condition
UART idle condition detect.

Values:

enumerator kUART_IdleFor4Frames
Idle for more than 4 frames

enumerator kUART_IdleFor8Frames
Idle for more than 8 frames

enumerator kUART_IdleFor16Frames
Idle for more than 16 frames

enumerator kUART_IdleFor32Frames
Idle for more than 32 frames

enum _uart_interrupt_enable
This structure contains the settings for all of the UART interrupt configurations.

Values:

enumerator kUART_AutoBaudEnable

enumerator kUART_TxReadyEnable

enumerator kUART_IdleEnable

enumerator kUART_RxReadyEnable

enumerator kUART_TxEmptyEnable

enumerator kUART_RtsDeltaEnable

enumerator kUART_EscapeEnable

enumerator kUART_RtsEnable

enumerator kUART_AgingTimerEnable

enumerator kUART_DtrEnable

enumerator kUART_ParityErrorEnable

enumerator kUART_FrameErrorEnable

enumerator kUART_DcdEnable

enumerator kUART_RiEnable

enumerator kUART_RxDsEnable

enumerator kUART_tAirWakeEnable

enumerator kUART_AwakeEnable

enumerator kUART_DtrDeltaEnable

enumerator kUART_AutoBaudCntEnable

enumerator kUART_IrEnable

enumerator kUART_WakeEnable

enumerator kUART_TxCompleteEnable

enumerator kUART_BreakDetectEnable

enumerator kUART_RxOverrunEnable

enumerator kUART_RxDataReadyEnable

enumerator kUART_RxDmaIdleEnable

enumerator kUART_AllInterruptsEnable

UART status flags.

This provides constants for the UART status flags for use in the UART functions.

Values:

enumerator kUART_RxCharReadyFlag
Rx Character Ready Flag.

enumerator kUART_RxErrorFlag
Rx Error Detect Flag.

enumerator kUART_RxOverrunErrorFlag
Rx Overrun Flag.

enumerator kUART_RxFrameErrorFlag
Rx Frame Error Flag.

enumerator kUART_RxBreakDetectFlag
Rx Break Detect Flag.

enumerator kUART_RxParityErrorFlag
Rx Parity Error Flag.

enumerator kUART_ParityErrorFlag
Parity Error Interrupt Flag.

enumerator kUART_RtsStatusFlag
RTS_B Pin Status Flag.

enumerator kUART_TxReadyFlag
Transmitter Ready Interrupt/DMA Flag.

enumerator kUART_RtsDeltaFlag
RTS Delta Flag.

enumerator kUART_EscapeFlag
Escape Sequence Interrupt Flag.

enumerator kUART_FrameErrorFlag
Frame Error Interrupt Flag.

enumerator kUART_RxReadyFlag
Receiver Ready Interrupt/DMA Flag.

enumerator kUART_AgingTimerFlag
Aging Timer Interrupt Flag.

enumerator kUART_DtrDeltaFlag
DTR Delta Flag.

enumerator kUART_RxDsFlag
Receiver IDLE Interrupt Flag.

enumerator kUART_tAirWakeFlag
Asynchronous IR WAKE Interrupt Flag.

enumerator kUART_AwakeFlag
Asynchronous WAKE Interrupt Flag.

enumerator kUART_Rs485SlaveAddrMatchFlag
RS-485 Slave Address Detected Interrupt Flag.

enumerator kUART_AutoBaudFlag
Automatic Baud Rate Detect Complete Flag.

enumerator kUART_TxEmptyFlag
Transmit Buffer FIFO Empty.

enumerator kUART_DtrFlag
DTR edge triggered interrupt flag.

enumerator kUART_IdleFlag
Idle Condition Flag.

enumerator kUART_AutoBaudCntStopFlag
Auto-baud Counter Stopped Flag.

enumerator kUART_RiDeltaFlag
Ring Indicator Delta Flag.

enumerator kUART_RiFlag
Ring Indicator Input Flag.

enumerator kUART_IrFlag
Serial Infrared Interrupt Flag.

enumerator kUART_WakeFlag
Wake Flag.

enumerator kUART_DcdDeltaFlag
Data Carrier Detect Delta Flag.

enumerator kUART_DcdFlag
Data Carrier Detect Input Flag.

enumerator kUART_RtsFlag
RTS Edge Triggered Interrupt Flag.

enumerator kUART_TxCompleteFlag
Transmitter Complete Flag.

enumerator kUART_BreakDetectFlag
BREAK Condition Detected Flag.

enumerator kUART_RxOverrunFlag
Overrun Error Flag.

enumerator kUART_RxDataReadyFlag
Receive Data Ready Flag.

typedef enum *_uart_data_bits* uart_data_bits_t
UART data bits count.

typedef enum *_uart_parity_mode* uart_parity_mode_t
UART parity mode.

typedef enum *_uart_stop_bit_count* uart_stop_bit_count_t
UART stop bit count.

typedef enum *_uart_idle_condition* uart_idle_condition_t
UART idle condition detect.

typedef struct *_uart_config* uart_config_t
UART configuration structure.

typedef struct *_uart_transfer* uart_transfer_t
UART transfer structure.

typedef struct *_uart_handle* uart_handle_t
Forward declaration of the handle typedef.

typedef void (*uart_transfer_callback_t)(UART_Type *base, *uart_handle_t* *handle, *status_t* status, void *userData)
UART transfer callback function.

typedef void (*uart_isr_t)(UART_Type *base, void *handle)

const IRQn_Type s_uartIRQ[]

uart_isr_t s_uartIsr

void *s_uartHandle[]
Pointers to uart handles for each instance.

uint32_t UART_GetInstance(UART_Type *base)
Get the UART instance from peripheral base address.

Parameters

- base – UART peripheral base address.

Returns
UART instance.

UART_RETRY_TIMES
Retry times for waiting flag.

struct __uart_config
#include <fsl_uart.h> UART configuration structure.

Public Members

uint32_t baudRate_Bps
: UART baud rate.

uart_parity_mode_t parityMode
: Parity error check mode of this module.

uart_data_bits_t dataBitsCount
: Data bits count, eight (default), seven

uart_stop_bit_count_t stopBitCount
: Number of stop bits in one frame.

uint8_t txFifoWatermark
: TX FIFO watermark

uint8_t rxFifoWatermark
: RX FIFO watermark

uint8_t rxRTSWatermark
: RX RTS watermark, RX FIFO data count being larger than this triggers RTS deassertion

bool enableAutoBaudRate
: Enable automatic baud rate detection

bool enableTx
: Enable TX

bool enableRx
: Enable RX

bool enableRxRTS
: RX RTS enable

bool enableTxCTS
: TX CTS enable

struct __uart__transfer
: *#include <fsl_uart.h>* UART transfer structure.

Public Members

size_t dataSize
: The byte count to be transfer.

struct __uart__handle
: *#include <fsl_uart.h>* UART handle structure.

Public Members

const uint8_t *volatile txData
: Address of remaining data to send.

volatile size_t txDataSize
: Size of the remaining data to send.

size_t txDataSizeAll
: Size of the data to send out.

uint8_t *volatile rxData
: Address of remaining data to receive.

volatile size_t rxDataSize
: Size of the remaining data to receive.

size_t rxDataSizeAll
: Size of the data to receive.

uint8_t *rxRingBuffer
: Start address of the receiver ring buffer.

size_t rxRingBufferSize
: Size of the ring buffer.

volatile uint16_t rxRingBufferHead
: Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail
: Index for the user to get data from the ring buffer.

uart_transfer_callback_t callback
: Callback function.

void *userData
: UART callback function parameter.

volatile uint8_t txState
: TX transfer state.

volatile uint8_t rxState
: RX transfer state

union ___unnamed41___

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.

2.52 UART FreeRTOS Driver

2.53 UART SDMA Driver

void UART_TransferCreateHandleSDMA(UART_Type *base, *uart_sdma_handle_t* *handle, *uart_sdma_transfer_callback_t* callback, void *userData, *sdma_handle_t* *txSdmaHandle, *sdma_handle_t* *rxSdmaHandle, uint32_t eventSourceTx, uint32_t eventSourceRx)

Initializes the UART handle which is used in transactional functions.

Parameters

- base – UART peripheral base address.
- handle – Pointer to the uart_sdma_handle_t structure.
- callback – UART callback, NULL means no callback.
- userData – User callback function data.
- rxSdmaHandle – User-requested DMA handle for RX DMA transfer.
- txSdmaHandle – User-requested DMA handle for TX DMA transfer.
- eventSourceTx – Eventsource for TX DMA transfer.
- eventSourceRx – Eventsource for RX DMA transfer.

status_t UART_SendSDMA(UART_Type *base, *uart_sdma_handle_t* *handle, *uart_transfer_t* *xfer)

Sends data using sDMA.

This function sends data using sDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.
- xfer – UART sDMA transfer structure. See uart_transfer_t.

Return values

- kStatus_Success – if succeeded; otherwise failed.
- kStatus_UART_TxBusy – Previous transfer ongoing.
- kStatus_InvalidArgument – Invalid argument.

status_t UART_ReceiveSDMA(UART_Type *base, *uart_sdma_handle_t* *handle, *uart_transfer_t* *xfer)

Receives data using sDMA.

This function receives data using sDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – UART peripheral base address.
- handle – Pointer to the uart_sdma_handle_t structure.
- xfer – UART sDMA transfer structure. See uart_transfer_t.

Return values

- kStatus_Success – if succeeded; otherwise failed.
- kStatus_UART_RxBusy – Previous transfer ongoing.
- kStatus_InvalidArgument – Invalid argument.

void UART_TransferAbortSendSDMA(UART_Type *base, *uart_sdma_handle_t* *handle)

Aborts the sent data using sDMA.

This function aborts sent data using sDMA.

Parameters

- base – UART peripheral base address.
- handle – Pointer to the uart_sdma_handle_t structure.

void UART_TransferAbortReceiveSDMA(UART_Type *base, *uart_sdma_handle_t* *handle)

Aborts the receive data using sDMA.

This function aborts receive data using sDMA.

Parameters

- base – UART peripheral base address.
- handle – Pointer to the uart_sdma_handle_t structure.

void UART_TransferSdmaHandleIRQ(UART_Type *base, void *uartSdmaHandle)

UART IRQ handle function.

This function handles the UART transmit complete IRQ request and invoke user callback.

Parameters

- base – UART peripheral base address.
- uartSdmaHandle – UART handle pointer.

FSL_UART_SDMA_DRIVER_VERSION

UART SDMA driver version.

typedef struct *_uart_sdma_handle* uart_sdma_handle_t

typedef void (*uart_sdma_transfer_callback_t)(UART_Type *base, *uart_sdma_handle_t* *handle, *status_t* status, void *userData)

UART transfer callback function.

struct __uart_sdma_handle

#include <fsl_uart_sdma.h> UART sDMA handle.

Public Members

uart_sdma_transfer_callback_t callback

Callback function.

void *userData

UART callback function parameter.

size_t rxDataSizeAll

Size of the data to receive.

size_t txDataSizeAll

Size of the data to send out.

sdma_handle_t *txSdmaHandle

The sDMA TX channel used.

sdma_handle_t *rxSdmaHandle

The sDMA RX channel used.

volatile uint8_t txState

TX transfer state.

volatile uint8_t rxState

RX transfer state

2.54 USDHC: Ultra Secured Digital Host Controller Driver

void USDHC_Init(USDHC_Type *base, const *usdhc_config_t* *config)

USDHC module initialization function.

Configures the USDHC according to the user configuration.

Example:

```
usdhc_config_t config;
config.cardDetectDat3 = false;
config.endianMode = kUSDHC_EndianModeLittle;
config.dmaMode = kUSDHC_DmaModeAdma2;
config.readWatermarkLevel = 128U;
config.writeWatermarkLevel = 128U;
USDHC_Init(USDHC, &config);
```

Parameters

- base – USDHC peripheral base address.
- config – USDHC configuration information.

Return values

kStatus_Success – Operate successfully.

void USDHC_Deinit(USDHC_Type *base)

Deinitializes the USDHC.

Parameters

- base – USDHC peripheral base address.

bool USDHC_Reset(USDHC_Type *base, uint32_t mask, uint32_t timeout)

Resets the USDHC.

Parameters

- base – USDHC peripheral base address.
- mask – The reset type mask(_usdhc_reset).
- timeout – Timeout for reset.

Return values

- true – Reset successfully.
- false – Reset failed.

status_t USDHC_SetAdmaTableConfig(USDHC_Type *base, *usdhc_adma_config_t* *dmaConfig, *usdhc_data_t* *dataConfig, uint32_t flags)

Sets the DMA descriptor table configuration. A high level DMA descriptor configuration function.

Parameters

- base – USDHC peripheral base address.
- dmaConfig – ADMA configuration
- dataConfig – Data descriptor
- flags – ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum _usdhc_adma_flag.

Return values

- kStatus_OutOfRange – ADMA descriptor table length isn't enough to describe data.
- kStatus_Success – Operate successfully.

status_t USDHC_SetInternalDmaConfig(USDHC_Type *base, *usdhc_adma_config_t* *dmaConfig, const uint32_t *dataAddr, bool enAutoCmd23)

Internal DMA configuration. This function is used to config the USDHC DMA related registers.

Parameters

- base – USDHC peripheral base address.
- dmaConfig – ADMA configuration.
- dataAddr – Transfer data address, a simple DMA parameter, if ADMA is used, leave it to NULL.
- enAutoCmd23 – Flag to indicate Auto CMD23 is enable or not, a simple DMA parameter, if ADMA is used, leave it to false.

Return values

- kStatus_OutOfRange – ADMA descriptor table length isn't enough to describe data.
- kStatus_Success – Operate successfully.

status_t USDHC_SetADMA2Descriptor(uint32_t *admaTable, uint32_t admaTableWords, const uint32_t *dataBufferAddr, uint32_t dataBytes, uint32_t flags)

Sets the ADMA2 descriptor table configuration.

Parameters

- admaTable – ADMA table address.
- admaTableWords – ADMA table length.
- dataBufferAddr – Data buffer address.
- dataBytes – Data Data length.
- flags – ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum _usdhc_adma_flag.

Return values

- kStatus_OutOfRange – ADMA descriptor table length isn't enough to describe data.
- kStatus_Success – Operate successfully.

status_t USDHC_SetADMA1Descriptor(uint32_t *admaTable, uint32_t admaTableWords, const uint32_t *dataBufferAddr, uint32_t dataBytes, uint32_t flags)

Sets the ADMA1 descriptor table configuration.

Parameters

- admaTable – ADMA table address.
- admaTableWords – ADMA table length.
- dataBufferAddr – Data buffer address.
- dataBytes – Data length.
- flags – ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum _usdhc_adma_flag.

Return values

- kStatus_OutOfRange – ADMA descriptor table length isn't enough to describe data.
- kStatus_Success – Operate successfully.

static inline void USDHC_EnableInternalDMA(USDHC_Type *base, bool enable)

Enables internal DMA.

Parameters

- base – USDHC peripheral base address.
- enable – enable or disable flag

static inline void USDHC_EnableInterruptStatus(USDHC_Type *base, uint32_t mask)

Enables the interrupt status.

Parameters

- base – USDHC peripheral base address.
- mask – Interrupt status flags mask(_usdhc_interrupt_status_flag).

static inline void USDHC_DisableInterruptStatus(USDHC_Type *base, uint32_t mask)

Disables the interrupt status.

Parameters

- base – USDHC peripheral base address.
- mask – The interrupt status flags mask(_usdhc_interrupt_status_flag).

static inline void USDHC_EnableInterruptSignal(USDHC_Type *base, uint32_t mask)

Enables the interrupt signal corresponding to the interrupt status flag.

Parameters

- base – USDHC peripheral base address.
- mask – The interrupt status flags mask(_usdhc_interrupt_status_flag).

static inline void USDHC_DisableInterruptSignal(USDHC_Type *base, uint32_t mask)

Disables the interrupt signal corresponding to the interrupt status flag.

Parameters

- base – USDHC peripheral base address.
- mask – The interrupt status flags mask(_usdhc_interrupt_status_flag).

static inline uint32_t USDHC_GetEnabledInterruptStatusFlags(USDHC_Type *base)

Gets the enabled interrupt status.

Parameters

- base – USDHC peripheral base address.

Returns

Current interrupt status flags mask(_usdhc_interrupt_status_flag).

static inline uint32_t USDHC_GetInterruptStatusFlags(USDHC_Type *base)

Gets the current interrupt status.

Parameters

- base – USDHC peripheral base address.

Returns

Current interrupt status flags mask(_usdhc_interrupt_status_flag).

static inline void USDHC_ClearInterruptStatusFlags(USDHC_Type *base, uint32_t mask)

Clears a specified interrupt status. write 1 clears.

Parameters

- base – USDHC peripheral base address.
- mask – The interrupt status flags mask(_usdhc_interrupt_status_flag).

static inline uint32_t USDHC_GetAutoCommand12ErrorStatusFlags(USDHC_Type *base)

Gets the status of auto command 12 error.

Parameters

- base – USDHC peripheral base address.

Returns

Auto command 12 error status flags mask(_usdhc_auto_command12_error_status_flag).

static inline uint32_t USDHC_GetAdmaErrorStatusFlags(USDHC_Type *base)

Gets the status of the ADMA error.

Parameters

- base – USDHC peripheral base address.

Returns

ADMA error status flags mask(_usdhc_adma_error_status_flag).

static inline uint32_t USDHC_GetPresentStatusFlags(USDHC_Type *base)

Gets a present status.

This function gets the present USDHC's status except for an interrupt status and an error status.

Parameters

- base – USDHC peripheral base address.

Returns

Present USDHC's status flags mask(_usdhc_present_status_flag).

void USDHC_GetCapability(USDHC_Type *base, *usdhc_capability_t* *capability)

Gets the capability information.

Parameters

- base – USDHC peripheral base address.
- capability – Structure to save capability information.

static inline void USDHC_ForceClockOn(USDHC_Type *base, bool enable)

Forces the card clock on.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

uint32_t USDHC_SetSdClock(USDHC_Type *base, uint32_t srcClock_Hz, uint32_t busClock_Hz)

Sets the SD bus clock frequency.

Parameters

- base – USDHC peripheral base address.
- srcClock_Hz – USDHC source clock frequency united in Hz.
- busClock_Hz – SD bus clock frequency united in Hz.

Returns
The nearest frequency of busClock_Hz configured for SD bus.

bool USDHC_SetCardActive(USDHC_Type *base, uint32_t timeout)

Sends 80 clocks to the card to set it to the active state.

This function must be called each time the card is inserted to ensure that the card can receive the command correctly.

Parameters

- base – USDHC peripheral base address.
- timeout – Timeout to initialize card.

Return values

- true – Set card active successfully.
- false – Set card active failed.

static inline void USDHC_AssertHardwareReset(USDHC_Type *base, bool high)

Triggers a hardware reset.

Parameters

- base – USDHC peripheral base address.
- high – 1 or 0 level

static inline void USDHC_SetDataBusWidth(USDHC_Type *base, *usdhc_data_bus_width_t* width)

Sets the data transfer width.

Parameters

- base – USDHC peripheral base address.
- width – Data transfer width.

static inline void USDHC_WriteData(USDHC_Type *base, uint32_t data)

Fills the data port.

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

- base – USDHC peripheral base address.
- data – The data about to be sent.

static inline uint32_t USDHC_ReadData(USDHC_Type *base)

Retrieves the data from the data port.

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

- base – USDHC peripheral base address.

Returns
The data has been read.

void USDHC_SendCommand(USDHC_Type *base, *usdhc_command_t* *command)

Sends command function.

Parameters

- base – USDHC peripheral base address.
- command – configuration

static inline void USDHC_EnableWakeupEvent(USDHC_Type *base, uint32_t mask, bool enable)

Enables or disables a wakeup event in low-power mode.

Parameters

- base – USDHC peripheral base address.
- mask – Wakeup events mask(_usdhc_wakeup_event).
- enable – True to enable, false to disable.

static inline void USDHC_CardDetectByData3(USDHC_Type *base, bool enable)

Detects card insert status.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

static inline bool USDHC_DetectCardInsert(USDHC_Type *base)

Detects card insert status.

Parameters

- base – USDHC peripheral base address.

static inline void USDHC_EnableSdioControl(USDHC_Type *base, uint32_t mask, bool enable)

Enables or disables the SDIO card control.

Parameters

- base – USDHC peripheral base address.
- mask – SDIO card control flags mask(_usdhc_sdio_control_flag).
- enable – True to enable, false to disable.

static inline void USDHC_SetContinueRequest(USDHC_Type *base)

Restarts a transaction which has stopped at the block GAP for the SDIO card.

Parameters

- base – USDHC peripheral base address.

static inline void USDHC_RequestStopAtBlockGap(USDHC_Type *base, bool enable)

Request stop at block gap function.

Parameters

- base – USDHC peripheral base address.
- enable – True to stop at block gap, false to normal transfer.

void USDHC_SetMmcBootConfig(USDHC_Type *base, const *usdhc_boot_config_t* *config)

Configures the MMC boot feature.

Example:

```
usdhc_boot_config_t config;
config.ackTimeoutCount = 4;
config.bootMode = kUSDHC_BootModeNormal;
config.blockCount = 5;
config.enableBootAck = true;
config.enableBoot = true;
config.enableAutoStopAtBlockGap = true;
USDHC_SetMmcBootConfig(USDHC, &config);
```

Parameters

- base – USDHC peripheral base address.
- config – The MMC boot configuration information.

static inline void USDHC_EnableMmcBoot(USDHC_Type *base, bool enable)

Enables or disables the mmc boot mode.

Parameters

- base – USDHC peripheral base address.
- enable – True to enable, false to disable.

static inline void USDHC_SetForceEvent(USDHC_Type *base, uint32_t mask)

Forces generating events according to the given mask.

Parameters

- base – USDHC peripheral base address.
- mask – The force events bit posistion (_usdhc_force_event).

static inline void UDSHC_SelectVoltage(USDHC_Type *base, bool en18v)

Selects the USDHC output voltage.

Parameters

- base – USDHC peripheral base address.
- en18v – True means 1.8V, false means 3.0V.

void USDHC_EnableDDRMode(USDHC_Type *base, bool enable, uint32_t nibblePos)

The enable/disable DDR mode.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag
- nibblePos – nibble position

void USDHC_SetDataConfig(USDHC_Type *base, *usdhc_transfer_direction_t* dataDirection, uint32_t blockCount, uint32_t blockSize)

USDHC data configuration.

Parameters

- base – USDHC peripheral base address.
- dataDirection – Data direction, tx or rx.
- blockCount – Data block count.
- blockSize – Data block size.

void USDHC_TransferCreateHandle(USDHC_Type *base, *usdhc_handle_t* *handle, const *usdhc_transfer_callback_t* *callback, void *userData)

Creates the USDHC handle.

Parameters

- base – USDHC peripheral base address.
- handle – USDHC handle pointer.
- callback – Structure pointer to contain all callback functions.
- userData – Callback function parameter.

status_t USDHC_TransferNonBlocking(USDHC_Type *base, *usdhc_handle_t* *handle, *usdhc_adma_config_t* *dmaConfig, *usdhc_transfer_t* *transfer)

Transfers the command/data using an interrupt and an asynchronous method.

This function sends a command and data and returns immediately. It doesn't wait for the transfer to complete or to encounter an error. The application must not call this API in multiple threads at the same time. Because of that this API doesn't support the re-entry mechanism.

Note: Call API USDHC_TransferCreateHandle when calling this API.

Parameters

- base – USDHC peripheral base address.
- handle – USDHC handle.
- dmaConfig – ADMA configuration.
- transfer – Transfer content.

Return values

- kStatus_InvalidArgument – Argument is invalid.
- kStatus_USDHC_BusyTransferring – Busy transferring.
- kStatus_USDHC_PrepareAdmaDescriptorFailed – Prepare ADMA descriptor failed.
- kStatus_Success – Operate successfully.

status_t USDHC_TransferBlocking(USDHC_Type *base, *usdhc_adma_config_t* *dmaConfig, *usdhc_transfer_t* *transfer)

Transfers the command/data using a blocking method.

This function waits until the command response/data is received or the USDHC encounters an error by polling the status flag.

The application must not call this API in multiple threads at the same time. Because this API doesn't support the re-entry mechanism.

Note: There is no need to call API USDHC_TransferCreateHandle when calling this API.

Parameters

- base – USDHC peripheral base address.
- dmaConfig – adma configuration
- transfer – Transfer content.

Return values

- kStatus_InvalidArgument – Argument is invalid.
- kStatus_USDHC_PrepareAdmaDescriptorFailed – Prepare ADMA descriptor failed.
- kStatus_USDHC_SendCommandFailed – Send command failed.
- kStatus_USDHC_TransferDataFailed – Transfer data failed.

- kStatus_Success – Operate successfully.

void USDHC_TransferHandleIRQ(USDHC_Type *base, *usdhc_handle_t* *handle)

IRQ handler for the USDHC.

This function deals with the IRQs on the given host controller.

Parameters

- base – USDHC peripheral base address.
- handle – USDHC handle.

FSL_USDHC_DRIVER_VERSION

Driver version 2.8.5.

Enum _usdhc_status. USDHC status.

Values:

enumerator kStatus_USDHC_BusyTransferring
Transfer is on-going.

enumerator kStatus_USDHC_PrepareAdmaDescriptorFailed
Set DMA descriptor failed.

enumerator kStatus_USDHC_SendCommandFailed
Send command failed.

enumerator kStatus_USDHC_TransferDataFailed
Transfer data failed.

enumerator kStatus_USDHC_DMADataAddrNotAlign
Data address not aligned.

enumerator kStatus_USDHC_ReTuningRequest
Re-tuning request.

enumerator kStatus_USDHC_TuningError
Tuning error.

enumerator kStatus_USDHC_NotSupport
Not support.

enumerator kStatus_USDHC_TransferDataComplete
Transfer data complete.

enumerator kStatus_USDHC_SendCommandSuccess
Transfer command complete.

enumerator kStatus_USDHC_TransferDMAComplete
Transfer DMA complete.

Enum _usdhc_capability_flag. Host controller capabilities flag mask. .

Values:

enumerator kUSDHC_SupportAdmaFlag
Support ADMA.

enumerator kUSDHC_SupportHighSpeedFlag
Support high-speed.

enumerator kUSDHC_SupportDmaFlag
Support DMA.

enumerator kUSDHC_SupportSuspendResumeFlag
Support suspend/resume.

enumerator kUSDHC_SupportV330Flag
Support voltage 3.3V.

enumerator kUSDHC_SupportV300Flag
Support voltage 3.0V.

enumerator kUSDHC_SupportV180Flag
Support voltage 1.8V.

enumerator kUSDHC_Support4BitFlag
Flag in HTCAPBLT_MBL's position, supporting 4-bit mode.

enumerator kUSDHC_Support8BitFlag
Flag in HTCAPBLT_MBL's position, supporting 8-bit mode.

enumerator kUSDHC_SupportDDR50Flag
SD version 3.0 new feature, supporting DDR50 mode.

enumerator kUSDHC_SupportSDR104Flag
Support SDR104 mode.

enumerator kUSDHC_SupportSDR50Flag
Support SDR50 mode.

Enum _usdhc_wakeup_event. Wakeup event mask. .

Values:

enumerator kUSDHC_WakeupEventOnCardInt
Wakeup on card interrupt.

enumerator kUSDHC_WakeupEventOnCardInsert
Wakeup on card insertion.

enumerator kUSDHC_WakeupEventOnCardRemove
Wakeup on card removal.

enumerator kUSDHC_WakeupEventsAll
All wakeup events

Enum _usdhc_reset. Reset type mask. .

Values:

enumerator kUSDHC_ResetAll
Reset all except card detection.

enumerator kUSDHC_ResetCommand
Reset command line.

enumerator kUSDHC_ResetData
Reset data line.

enumerator kUSDHC_ResetTuning
Reset tuning circuit.

enumerator kUSDHC_ResetsAll
All reset types

Enum _usdhc_transfer_flag. Transfer flag mask.

Values:

enumerator kUSDHC_EnableDmaFlag
Enable DMA.

enumerator kUSDHC_CommandTypeSuspendFlag
Suspend command.

enumerator kUSDHC_CommandTypeResumeFlag
Resume command.

enumerator kUSDHC_CommandTypeAbortFlag
Abort command.

enumerator kUSDHC_EnableBlockCountFlag
Enable block count.

enumerator kUSDHC_EnableAutoCommand12Flag
Enable auto CMD12.

enumerator kUSDHC_DataReadFlag
Enable data read.

enumerator kUSDHC_MultipleBlockFlag
Multiple block data read/write.

enumerator kUSDHC_EnableAutoCommand23Flag
Enable auto CMD23.

enumerator kUSDHC_ResponseLength136Flag
136-bit response length.

enumerator kUSDHC_ResponseLength48Flag
48-bit response length.

enumerator kUSDHC_ResponseLength48BusyFlag
48-bit response length with busy status.

enumerator kUSDHC_EnableCrcCheckFlag
Enable CRC check.

enumerator kUSDHC_EnableIndexCheckFlag
Enable index check.

enumerator kUSDHC_DataPresentFlag
Data present flag.

Enum _usdhc_present_status_flag. Present status flag mask. .

Values:

enumerator kUSDHC_CommandInhibitFlag
Command inhibit.

enumerator kUSDHC_DataInhibitFlag
Data inhibit.

enumerator kUSDHC_DataLineActiveFlag
Data line active.

enumerator kUSDHC_SdClockStableFlag
SD bus clock stable.

enumerator kUSDHC_WriteTransferActiveFlag
Write transfer active.

enumerator kUSDHC_ReadTransferActiveFlag
Read transfer active.

enumerator kUSDHC_BufferWriteEnableFlag
Buffer write enable.

enumerator kUSDHC_BufferReadEnableFlag
Buffer read enable.

enumerator kUSDHC_ReTuningRequestFlag
Re-tuning request flag, only used for SDR104 mode.

enumerator kUSDHC_DelaySettingFinishedFlag
Delay setting finished flag.

enumerator kUSDHC_CardInsertedFlag
Card inserted.

enumerator kUSDHC_CommandLineLevelFlag
Command line signal level.

enumerator kUSDHC_Data0LineLevelFlag
Data0 line signal level.

enumerator kUSDHC_Data1LineLevelFlag
Data1 line signal level.

enumerator kUSDHC_Data2LineLevelFlag
Data2 line signal level.

enumerator kUSDHC_Data3LineLevelFlag
Data3 line signal level.

enumerator kUSDHC_Data4LineLevelFlag
Data4 line signal level.

enumerator kUSDHC_Data5LineLevelFlag
Data5 line signal level.

enumerator kUSDHC_Data6LineLevelFlag
Data6 line signal level.

enumerator kUSDHC_Data7LineLevelFlag
Data7 line signal level.

Enum _usdhc_interrupt_status_flag. Interrupt status flag mask. .

Values:

enumerator kUSDHC_CommandCompleteFlag
Command complete.

enumerator kUSDHC_DataCompleteFlag
Data complete.

enumerator kUSDHC_BlockGapEventFlag
Block gap event.

enumerator kUSDHC_DmaCompleteFlag
DMA interrupt.

enumerator kUSDHC_BufferWriteReadyFlag
Buffer write ready.

enumerator kUSDHC_BufferReadReadyFlag
Buffer read ready.

enumerator kUSDHC_CardInsertionFlag
Card inserted.

enumerator kUSDHC_CardRemovalFlag
Card removed.

enumerator kUSDHC_CardInterruptFlag
Card interrupt.

enumerator kUSDHC_ReTuningEventFlag
Re-Tuning event, only for SD3.0 SDR104 mode.

enumerator kUSDHC_TuningPassFlag
SDR104 mode tuning pass flag.

enumerator kUSDHC_TuningErrorFlag
SDR104 tuning error flag.

enumerator kUSDHC_CommandTimeoutFlag
Command timeout error.

enumerator kUSDHC_CommandCrcErrorFlag
Command CRC error.

enumerator kUSDHC_CommandEndBitErrorFlag
Command end bit error.

enumerator kUSDHC_CommandIndexErrorFlag
Command index error.

enumerator kUSDHC_DataTimeoutFlag
Data timeout error.

enumerator kUSDHC_DataCrcErrorFlag
Data CRC error.

enumerator kUSDHC_DataEndBitErrorFlag
Data end bit error.

enumerator kUSDHC_AutoCommand12ErrorFlag
Auto CMD12 error.

enumerator kUSDHC_DmaErrorFlag
DMA error.

enumerator kUSDHC_CommandErrorFlag
Command error

enumerator kUSDHC_DataErrorFlag
Data error

enumerator kUSDHC_ErrorFlag
All error

enumerator kUSDHC_DataFlag
Data interrupts

enumerator kUSDHC_DataDMAFlag
Data interrupts

enumerator kUSDHC_CommandFlag
Command interrupts

enumerator kUSDHC_CardDetectFlag
Card detection interrupts

enumerator kUSDHC_SDR104TuningFlag
SDR104 tuning flag.

enumerator kUSDHC_AllInterruptFlags
All flags mask

Enum _usdhc_auto_command12_error_status_flag. Auto CMD12 error status flag mask. .

Values:

enumerator kUSDHC_AutoCommand12NotExecutedFlag
Not executed error.

enumerator kUSDHC_AutoCommand12TimeoutFlag
Timeout error.

enumerator kUSDHC_AutoCommand12EndBitErrorFlag
End bit error.

enumerator kUSDHC_AutoCommand12CrcErrorFlag
CRC error.

enumerator kUSDHC_AutoCommand12IndexErrorFlag
Index error.

enumerator kUSDHC_AutoCommand12NotIssuedFlag
Not issued error.

Enum _usdhc_standard_tuning. Standard tuning flag.

Values:

enumerator kUSDHC_ExecuteTuning
Used to start tuning procedure.

enumerator kUSDHC_TuningSampleClockSel
When **std_tuning_en** bit is set, this bit is used to select sampleing clock.

Enum _usdhc_adma_error_status_flag. ADMA error status flag mask. .

Values:

enumerator kUSDHC_AdmaLenghMismatchFlag
Length mismatch error.

enumerator kUSDHC_AdmaDescriptorErrorFlag
Descriptor error.

Enum _usdhc_adma_error_state. ADMA error state.

This state is the detail state when ADMA error has occurred.

Values:

enumerator kUSDHC_AdmaErrorStateStopDma
Stop DMA, previous location set in the ADMA system address is errored address.

enumerator kUSDHC_AdmaErrorStateFetchDescriptor
Fetch descriptor, current location set in the ADMA system address is errored address.

enumerator kUSDHC_AdmaErrorStateChangeAddress
Change address, no DMA error has occurred.

enumerator kUSDHC_AdmaErrorStateTransferData
Transfer data, previous location set in the ADMA system address is errored address.

enumerator kUSDHC_AdmaErrorStateInvalidLength
Invalid length in ADMA descriptor.

enumerator kUSDHC_AdmaErrorStateInvalidDescriptor
Invalid descriptor fetched by ADMA.

enumerator kUSDHC_AdmaErrorState
ADMA error state

Enum _usdhc_force_event. Force event bit position. .

Values:

enumerator kUSDHC_ForceEventAutoCommand12NotExecuted
Auto CMD12 not executed error.

enumerator kUSDHC_ForceEventAutoCommand12Timeout
Auto CMD12 timeout error.

enumerator kUSDHC_ForceEventAutoCommand12CrcError
Auto CMD12 CRC error.

enumerator kUSDHC_ForceEventEndBitError
Auto CMD12 end bit error.

enumerator kUSDHC_ForceEventAutoCommand12IndexError
Auto CMD12 index error.

enumerator kUSDHC_ForceEventAutoCommand12NotIssued
Auto CMD12 not issued error.

enumerator kUSDHC_ForceEventCommandTimeout
Command timeout error.

enumerator kUSDHC_ForceEventCommandCrcError
Command CRC error.

enumerator kUSDHC_ForceEventCommandEndBitError
Command end bit error.

enumerator kUSDHC_ForceEventCommandIndexError
Command index error.

enumerator kUSDHC_ForceEventDataTimeout
Data timeout error.

enumerator kUSDHC_ForceEventDataCrcError
Data CRC error.

enumerator kUSDHC_ForceEventDataEndBitError
Data end bit error.

enumerator kUSDHC_ForceEventAutoCommand12Error
Auto CMD12 error.

enumerator kUSDHC_ForceEventCardInt
Card interrupt.

enumerator kUSDHC_ForceEventDmaError
Dma error.

enumerator kUSDHC_ForceEventTuningError
Tuning error.

enumerator kUSDHC_ForceEventsAll
All force event flags mask.

enum _usdhc_transfer_direction
Data transfer direction.

Values:

enumerator kUSDHC_TransferDirectionReceive
USDHC transfer direction receive.

enumerator kUSDHC_TransferDirectionSend
USDHC transfer direction send.

enum _usdhc_data_bus_width
Data transfer width.

Values:

enumerator kUSDHC_DataBusWidth1Bit
1-bit mode

enumerator kUSDHC_DataBusWidth4Bit
4-bit mode

enumerator kUSDHC_DataBusWidth8Bit
8-bit mode

enum _usdhc_endian_mode
Endian mode.

Values:

enumerator kUSDHC_EndianModeBig
Big endian mode.

enumerator kUSDHC_EndianModeHalfWordBig
Half word big endian mode.

enumerator kUSDHC_EndianModeLittle
Little endian mode.

enum _usdhc_dma_mode
DMA mode.

Values:

enumerator kUSDHC_DmaModeSimple
External DMA.

enumerator kUSDHC_DmaModeAdma1
ADMA1 is selected.

enumerator kUSDHC_DmaModeAdma2
ADMA2 is selected.

enumerator kUSDHC_ExternalDMA
External DMA mode selected.

Enum _usdhc_sdio_control_flag. SDIO control flag mask. .

Values:

enumerator kUSDHC_StopAtBlockGapFlag
Stop at block gap.

enumerator kUSDHC_ReadWaitControlFlag
Read wait control.

enumerator kUSDHC_InterruptAtBlockGapFlag
Interrupt at block gap.

enumerator kUSDHC_ReadDoneNo8CLK
Read done without 8 clk for block gap.

enumerator kUSDHC_ExactBlockNumberReadFlag
Exact block number read.

enum _usdhc_boot_mode
MMC card boot mode.

Values:

enumerator kUSDHC_BootModeNormal
Normal boot

enumerator kUSDHC_BootModeAlternative
Alternative boot

enum _usdhc_card_command_type
The command type.

Values:

enumerator kCARD_CommandTypeNormal
Normal command

enumerator kCARD_CommandTypeSuspend
Suspend command

enumerator kCARD_CommandTypeResume
Resume command

enumerator kCARD_CommandTypeAbort
Abort command

enumerator kCARD_CommandTypeEmpty
Empty command

enum _usdhc_card_response_type
The command response type.

Defines the command response type from card to host controller.

Values:

enumerator kCARD_ResponseTypeNone
Response type: none

enumerator kCARD_ResponseTypeR1
Response type: R1

enumerator kCARD_ResponseTypeR1b
Response type: R1b

enumerator kCARD_ResponseTypeR2
Response type: R2

enumerator kCARD_ResponseTypeR3
Response type: R3

enumerator kCARD_ResponseTypeR4
Response type: R4

enumerator kCARD_ResponseTypeR5
Response type: R5

enumerator kCARD_ResponseTypeR5b
Response type: R5b

enumerator kCARD_ResponseTypeR6
Response type: R6

enumerator kCARD_ResponseTypeR7
Response type: R7

Enum _usdhc_adma1_descriptor_flag. The mask for the control/status field in ADMA1 descriptor.

Values:

enumerator kUSDHC_Adma1DescriptorValidFlag
Valid flag.

enumerator kUSDHC_Adma1DescriptorEndFlag
End flag.

enumerator kUSDHC_Adma1DescriptorInterrupFlag
Interrupt flag.

enumerator kUSDHC_Adma1DescriptorActivity1Flag
Activity 1 flag.

enumerator kUSDHC_Adma1DescriptorActivity2Flag
Activity 2 flag.

enumerator kUSDHC_Adma1DescriptorTypeNop
No operation.

enumerator kUSDHC_Adma1DescriptorTypeTransfer
Transfer data.

enumerator kUSDHC_Adma1DescriptorTypeLink
Link descriptor.

enumerator kUSDHC_Adma1DescriptorTypeSetLength
Set data length.

Enum _usdhc_adma2_descriptor_flag. ADMA1 descriptor control and status mask.

Values:

enumerator kUSDHC_Adma2DescriptorValidFlag
Valid flag.

enumerator kUSDHC_Adma2DescriptorEndFlag
End flag.

enumerator kUSDHC_Adma2DescriptorInterruptFlag
Interrupt flag.

enumerator kUSDHC_Adma2DescriptorActivity1Flag
Activity 1 mask.

enumerator kUSDHC_Adma2DescriptorActivity2Flag
Activity 2 mask.

enumerator kUSDHC_Adma2DescriptorTypeNop
No operation.

enumerator kUSDHC_Adma2DescriptorTypeReserved
Reserved.

enumerator kUSDHC_Adma2DescriptorTypeTransfer
Transfer type.

enumerator kUSDHC_Adma2DescriptorTypeLink
Link type.

Enum _usdhc_adma_flag. ADMA descriptor configuration flag. .

Values:

enumerator kUSDHC_AdmaDescriptorSingleFlag
Try to finish the transfer in a single ADMA descriptor. If transfer size is bigger than one ADMA descriptor's ability, new another descriptor for data transfer.

enumerator kUSDHC_AdmaDescriptorMultipleFlag
Create multiple ADMA descriptors within the ADMA table, this is used for mmc boot mode specifically, which need to modify the ADMA descriptor on the fly, so the flag should be used combining with stop at block gap feature.

enum _usdhc_burst_len

DMA transfer burst len config.

Values:

enumerator kUSDHC_EnBurstLenForINCR

Enable burst len for INCR.

enumerator kUSDHC_EnBurstLenForINCR4816

Enable burst len for INCR4/INCR8/INCR16.

enumerator kUSDHC_EnBurstLenForINCR4816WRAP

Enable burst len for INCR4/8/16 WRAP.

Enum _usdhc_transfer_data_type. Tansfer data type definition.

Values:

enumerator kUSDHC_TransferDataNormal

Transfer normal read/write data.

enumerator kUSDHC_TransferDataTuning

Transfer tuning data.

enumerator kUSDHC_TransferDataBoot

Transfer boot data.

enumerator kUSDHC_TransferDataBootcontinous

Transfer boot data continuously.

typedef enum *_usdhc_transfer_direction* usdhc_transfer_direction_t

Data transfer direction.

typedef enum *_usdhc_data_bus_width* usdhc_data_bus_width_t

Data transfer width.

typedef enum *_usdhc_endian_mode* usdhc_endian_mode_t

Endian mode.

typedef enum *_usdhc_dma_mode* usdhc_dma_mode_t

DMA mode.

typedef enum *_usdhc_boot_mode* usdhc_boot_mode_t

MMC card boot mode.

typedef enum *_usdhc_card_command_type* usdhc_card_command_type_t

The command type.

typedef enum *_usdhc_card_response_type* usdhc_card_response_type_t

The command response type.

Defines the command response type from card to host controller.

typedef enum *_usdhc_burst_len* usdhc_burst_len_t

DMA transfer burst len config.

typedef uint32_t usdhc_adma1_descriptor_t

Defines the ADMA1 descriptor structure.

typedef struct *_usdhc_adma2_descriptor* usdhc_adma2_descriptor_t

Defines the ADMA2 descriptor structure.

typedef struct *_usdhc_capability* usdhc_capability_t
: USDHC capability information.

  Defines a structure to save the capability information of USDHC.

typedef struct *_usdhc_boot_config* usdhc_boot_config_t
: Data structure to configure the MMC boot feature.

typedef struct *_usdhc_config* usdhc_config_t
: Data structure to initialize the USDHC.

typedef struct *_usdhc_command* usdhc_command_t
: Card command descriptor.

  Defines card command-related attribute.

typedef struct *_usdhc_adma_config* usdhc_adma_config_t
: ADMA configuration.

typedef struct *_usdhc_scatter_gather_data_list* usdhc_scatter_gather_data_list_t
: Card scatter gather data list.

  Allow application register uncontinuous data buffer for data transfer.

typedef struct *_usdhc_scatter_gather_data* usdhc_scatter_gather_data_t
: Card scatter gather data descriptor.

  Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

typedef struct *_usdhc_scatter_gather_transfer* usdhc_scatter_gather_transfer_t
: usdhc scatter gather transfer.

typedef struct *_usdhc_data* usdhc_data_t
: Card data descriptor.

  Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

typedef struct *_usdhc_transfer* usdhc_transfer_t
: Transfer state.

typedef struct *_usdhc_handle* usdhc_handle_t
: USDHC handle typedef.

typedef struct *_usdhc_transfer_callback* usdhc_transfer_callback_t
: USDHC callback functions.

typedef *status_t* (*usdhc_transfer_function_t)(USDHC_Type *base, *usdhc_transfer_t* *content)
: USDHC transfer function.

typedef struct *_usdhc_host* usdhc_host_t
: USDHC host descriptor.

USDHC_MAX_BLOCK_COUNT
: Maximum block count can be set one time.

FSL_USDHC_ENABLE_SCATTER_GATHER_TRANSFER
: USDHC scatter gather feature control macro.

USDHC_ADMA1_ADDRESS_ALIGN

The alignment size for ADDRESS filed in ADMA1's descriptor.

USDHC_ADMA1_LENGTH_ALIGN

The alignment size for LENGTH field in ADMA1's descriptor.

USDHC_ADMA2_ADDRESS_ALIGN

The alignment size for ADDRESS field in ADMA2's descriptor.

USDHC_ADMA2_LENGTH_ALIGN

The alignment size for LENGTH filed in ADMA2's descriptor.

USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT

The bit shift for ADDRESS filed in ADMA1's descriptor.

Address/page field	Reserved	Attribute					
31 12	11 6	05	04	03	02	01	00
address or data length	000000	Act2	Act1	0	Int	End	Valid

Act2	Act1	Comment	31-28	27-12
0	0	No op	Don't care	
0	1	Set data length	0000	Data Length
1	0	Transfer data	Data address	
1	1	Link descriptor	Descriptor address	

USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK

The bit mask for ADDRESS field in ADMA1's descriptor.

USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT

The bit shift for LENGTH filed in ADMA1's descriptor.

USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK

The mask for LENGTH field in ADMA1's descriptor.

USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY

The maximum value of LENGTH filed in ADMA1's descriptor. Since the max transfer size ADMA1 support is 65535 which is indivisible by 4096, so to make sure a large data load transfer (>64KB) continuously (require the data address be always align with 4096), software will set the maximum data length for ADMA1 to (64 - 4)KB.

USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT

The bit shift for LENGTH field in ADMA2's descriptor.

Address field	Length	Reserved	Attribute					
63 32	31 16	15 06	05	04	03	02	01	00
32-bit address	16-bit length	0000000000	Act2	Act1	0	Int	End	Valid

Act2	Act1	Comment	Operation
0	0	No op	Don't care
0	1	Reserved	Read this line and go to next one
1	0	Transfer data	Transfer data with address and length set in this descriptor line
1	1	Link descriptor	Link to another descriptor

USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK
The bit mask for LENGTH field in ADMA2's descriptor.

USDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_ENTRY
The maximum value of LENGTH field in ADMA2's descriptor.

struct _usdhc_adma2_descriptor
#include <fsl_usdhc.h> Defines the ADMA2 descriptor structure.

Public Members

uint32_t attribute
The control and status field.

uint32_t address
The address field.

struct _usdhc_capability
#include <fsl_usdhc.h> USDHC capability information.

Defines a structure to save the capability information of USDHC.

Public Members

uint32_t sdVersion
Support SD card/sdio version.

uint32_t mmcVersion
Support EMMC card version.

uint32_t maxBlockLength
Maximum block length united as byte.

uint32_t maxBlockCount
Maximum block count can be set one time.

uint32_t flags
Capability flags to indicate the support information(_usdhc_capability_flag).

struct _usdhc_boot_config
#include <fsl_usdhc.h> Data structure to configure the MMC boot feature.

Public Members

uint32_t ackTimeoutCount
Timeout value for the boot ACK. The available range is 0 ~ 15.

usdhc_boot_mode_t bootMode
Boot mode selection.

uint32_t blockCount
: Stop at block gap value of automatic mode. Available range is 0 ~ 65535.

size_t blockSize
: Block size.

bool enableBootAck
: Enable or disable boot ACK.

bool enableAutoStopAtBlockGap
: Enable or disable auto stop at block gap function in boot period.

struct _usdhc_config
: *#include <fsl_usdhc.h>* Data structure to initialize the USDHC.

Public Members

uint32_t dataTimeout
: Data timeout value.

usdhc_endian_mode_t endianMode
: Endian mode.

uint8_t readWatermarkLevel
: Watermark level for DMA read operation. Available range is 1 ~ 128.

uint8_t writeWatermarkLevel
: Watermark level for DMA write operation. Available range is 1 ~ 128.

uint8_t readBurstLen
: Read burst len.

uint8_t writeBurstLen
: Write burst len.

struct _usdhc_command
: *#include <fsl_usdhc.h>* Card command descriptor.

Defines card command-related attribute.

Public Members

uint32_t index
: Command index.

uint32_t argument
: Command argument.

usdhc_card_command_type_t type
: Command type.

usdhc_card_response_type_t responseType
: Command response type.

uint32_t response[4U]
: Response for this command.

uint32_t responseErrorFlags
: Response error flag, which need to check the command reponse.

uint32_t flags
: Cmd flags.

struct _usdhc_adma_config
: *#include <fsl_usdhc.h>* ADMA configuration.

Public Members

usdhc_dma_mode_t dmaMode
: DMA mode.

usdhc_burst_len_t burstLen
: Burst len config.

uint32_t *admaTable
: ADMA table address, can't be null if transfer way is ADMA1/ADMA2.

uint32_t admaTableWords
: ADMA table length united as words, can't be 0 if transfer way is ADMA1/ADMA2.

struct _usdhc_scatter_gather_data_list
: *#include <fsl_usdhc.h>* Card scatter gather data list.

Allow application register uncontinuous data buffer for data transfer.

struct _usdhc_scatter_gather_data
: *#include <fsl_usdhc.h>* Card scatter gather data descriptor.

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

Public Members

bool enableAutoCommand12
: Enable auto CMD12.

bool enableAutoCommand23
: Enable auto CMD23.

bool enableIgnoreError
: Enable to ignore error event to read/write all the data.

usdhc_transfer_direction_t dataDirection
: data direction

uint8_t dataType
: this is used to distinguish the normal/tuning/boot data.

size_t blockSize
: Block size.

usdhc_scatter_gather_data_list_t sgData
: scatter gather data

struct _usdhc_scatter_gather_transfer
: *#include <fsl_usdhc.h>* usdhc scatter gather transfer.

Public Members

usdhc_scatter_gather_data_t *data

Data to transfer.

usdhc_command_t *command

Command to send.

struct __usdhc_data

#include <fsl_usdhc.h> Card data descriptor.

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

Public Members

bool enableAutoCommand12

Enable auto CMD12.

bool enableAutoCommand23

Enable auto CMD23.

bool enableIgnoreError

Enable to ignore error event to read/write all the data.

uint8_t dataType

this is used to distinguish the normal/tuning/boot data.

size_t blockSize

Block size.

uint32_t blockCount

Block count.

uint32_t *rxData

Buffer to save data read.

const uint32_t *txData

Data buffer to write.

struct __usdhc_transfer

#include <fsl_usdhc.h> Transfer state.

Public Members

usdhc_data_t *data

Data to transfer.

usdhc_command_t *command

Command to send.

struct __usdhc_transfer_callback

#include <fsl_usdhc.h> USDHC callback functions.

Public Members

void (*CardInserted)(USDHC_Type *base, void *userData)
Card inserted occurs when DAT3/CD pin is for card detect

void (*CardRemoved)(USDHC_Type *base, void *userData)
Card removed occurs

void (*SdioInterrupt)(USDHC_Type *base, void *userData)
SDIO card interrupt occurs

void (*BlockGap)(USDHC_Type *base, void *userData)
stopped at block gap event

void (*TransferComplete)(USDHC_Type *base, *usdhc_handle_t* *handle, *status_t* status, void *userData)
Transfer complete callback.

void (*ReTuning)(USDHC_Type *base, void *userData)
Handle the re-tuning.

struct __usdhc__handle
#include <fsl_usdhc.h> USDHC handle.

Defines the structure to save the USDHC state information and callback function.

Note: All the fields except interruptFlags and transferredWords must be allocated by the user.

Public Members

usdhc_data_t *volatile data
Transfer parameter. Data to transfer.

usdhc_command_t *volatile command
Transfer parameter. Command to send.

volatile uint32_t transferredWords
Transfer status. Words transferred by DATAPORT way.

usdhc_transfer_callback_t callback
Callback function.

void *userData
Parameter for transfer complete callback.

struct __usdhc__host
#include <fsl_usdhc.h> USDHC host descriptor.

Public Members

USDHC_Type *base
USDHC peripheral base address.

uint32_t sourceClock__Hz
USDHC source clock frequency united in Hz.

usdhc_config_t config
USDHC configuration.

usdhc_capability_t capability
USDHC capability information.

usdhc_transfer_function_t transfer
USDHC transfer function.

2.55 WDOG: Watchdog Timer Driver

void WDOG_GetDefaultConfig(*wdog_config_t* *config)
Initializes the WDOG configuration structure.

This function initializes the WDOG configuration structure to default values. The default values are as follows.

```
wdogConfig->enableWdog = true;
wdogConfig->workMode.enableWait = true;
wdogConfig->workMode.enableStop = true;
wdogConfig->workMode.enableDebug = true;
wdogConfig->enableInterrupt = false;
wdogConfig->enablePowerdown = false;
wdogConfig->resetExtension = flase;
wdogConfig->timeoutValue = 0xFFU;
wdogConfig->interruptTimeValue = 0x04u;
```

See also:

wdog_config_t

Parameters

- config – Pointer to the WDOG configuration structure.

void WDOG_Init(WDOG_Type *base, const *wdog_config_t* *config)
Initializes the WDOG.

This function initializes the WDOG. When called, the WDOG runs according to the configuration.

This is an example.

```
wdog_config_t config;
WDOG_GetDefaultConfig(&config);
config.timeoutValue = 0xffU;
config->interruptTimeValue = 0x04u;
WDOG_Init(wdog_base,&config);
```

Parameters

- base – WDOG peripheral base address
- config – The configuration of WDOG

void WDOG_Deinit(WDOG_Type *base)
Shuts down the WDOG.

This function shuts down the WDOG. Watchdog Enable bit is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. This bit(WDE) can be set/reset only in debug mode(exception).

static inline void WDOG_Enable(WDOG_Type *base)

Enables the WDOG module.

This function writes a value into the WDOG_WCR register to enable the WDOG. This is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. only debug mode exception.

Parameters

- base – WDOG peripheral base address

static inline void WDOG_Disable(WDOG_Type *base)

Disables the WDOG module.

This function writes a value into the WDOG_WCR register to disable the WDOG. This is a write one once only bit. It is not possible to clear this bit by a software write,once the bit is set. only debug mode exception

Parameters

- base – WDOG peripheral base address

static inline void WDOG_TriggerSystemSoftwareReset(WDOG_Type *base)

Trigger the system software reset.

This function will write to the WCR[SRS] bit to trigger a software system reset. This bit will automatically resets to "1" after it has been asserted to "0". Note: Calling this API will reset the system right now, please using it with more attention.

Parameters

- base – WDOG peripheral base address

static inline void WDOG_TriggerSoftwareSignal(WDOG_Type *base)

Trigger an output assertion.

This function will write to the WCR[WDA] bit to trigger WDOG_B signal assertion. The WDOG_B signal can be routed to external pin of the chip, the output pin will turn to assertion along with WDOG_B signal. Note: The WDOG_B signal will remain assert until a power on reset occurred, so, please take more attention while calling it.

Parameters

- base – WDOG peripheral base address

static inline void WDOG_EnableInterrupts(WDOG_Type *base, uint16_t mask)

Enables the WDOG interrupt.

This bit is a write once only bit. Once the software does a write access to this bit, it will get locked and cannot be reprogrammed until the next system reset assertion

Parameters

- base – WDOG peripheral base address
- mask – The interrupts to enable The parameter can be combination of the following source if defined.
  - kWDOG_InterruptEnable

uint16_t WDOG_GetStatusFlags(WDOG_Type *base)

Gets the WDOG all reset status flags.

This function gets all reset status flags.

```
uint16_t status;
status = WDOG_GetStatusFlags (wdog_base);
```

See also:

_wdog_status_flags

- true: a related status flag has been set.
- false: a related status flag is not set.

Parameters

- base – WDOG peripheral base address

Returns

State of the status flag: asserted (true) or not-asserted (false).

void WDOG_ClearInterruptStatus(WDOG_Type *base, uint16_t mask)

Clears the WDOG flag.

This function clears the WDOG status flag.

This is an example for clearing the interrupt flag.

```
WDOG_ClearStatusFlags(wdog_base,KWDOG_InterruptFlag);
```

Parameters

- base – WDOG peripheral base address
- mask – The status flags to clear. The parameter could be any combination of the following values. kWDOG_TimeoutFlag

static inline void WDOG_SetTimeoutValue(WDOG_Type *base, uint16_t timeoutCount)

Sets the WDOG timeout value.

This function sets the timeout value. This function writes a value into WCR registers. The time-out value can be written at any point of time but it is loaded to the counter at the time when WDOG is enabled or after the service routine has been performed.

Parameters

- base – WDOG peripheral base address
- timeoutCount – WDOG timeout value; count of WDOG clock tick.

static inline void WDOG_SetInterrputTimeoutValue(WDOG_Type *base, uint16_t timeoutCount)

Sets the WDOG interrupt count timeout value.

This function sets the interrupt count timeout value. This function writes a value into WIC registers which are wirte-once. This field is write once only. Once the software does a write access to this field, it will get locked and cannot be reprogrammed until the next system reset assertion.

Parameters

- base – WDOG peripheral base address
- timeoutCount – WDOG timeout value; count of WDOG clock tick.

static inline void WDOG_DisablePowerDownEnable(WDOG_Type *base)

Disable the WDOG power down enable bit.

This function disable the WDOG power down enable(PDE). This function writes a value into WMCR registers which are wirte-once. This field is write once only. Once software sets this bit it cannot be reset until the next system reset.

Parameters

- base – WDOG peripheral base address

void WDOG_Refresh(WDOG_Type *base)

Refreshes the WDOG timer.

This function feeds the WDOG. This function should be called before the WDOG timer is in timeout. Otherwise, a reset is asserted.

Parameters

- base – WDOG peripheral base address

FSL_WDOG_DRIVER_VERSION

Defines WDOG driver version.

WDOG_REFRESH_KEY

enum _wdog_interrupt_enable

WDOG interrupt configuration structure, default settings all disabled.

This structure contains the settings for all of the WDOG interrupt configurations.

Values:

enumerator kWDOG_InterruptEnable

WDOG timeout generates an interrupt before reset

enum _wdog_status_flags

WDOG status flags.

This structure contains the WDOG status flags for use in the WDOG functions.

Values:

enumerator kWDOG_RunningFlag

Running flag, set when WDOG is enabled

enumerator kWDOG_PowerOnResetFlag

Power On flag, set when reset is the result of a powerOnReset

enumerator kWDOG_TimeoutResetFlag

Timeout flag, set when reset is the result of a timeout

enumerator kWDOG_SoftwareResetFlag

Software flag, set when reset is the result of a software

enumerator kWDOG_InterruptFlag

interrupt flag,whether interrupt has occurred or not

typedef struct *_wdog_work_mode* wdog_work_mode_t

Defines WDOG work mode.

typedef struct *_wdog_config* wdog_config_t

Describes WDOG configuration structure.

struct _wdog_work_mode

#include <fsl_wdog.h> Defines WDOG work mode.

Public Members

bool enableWait

If set to true, WDOG continues in wait mode

bool enableStop

If set to true, WDOG continues in stop mode

bool enableDebug
: If set to true, WDOG continues in debug mode

struct __wdog__config
: *#include <fsl_wdog.h>* Describes WDOG configuration structure.

Public Members

bool enableWdog
: Enables or disables WDOG

wdog_work_mode_t workMode
: Configures WDOG work mode in debug stop and wait mode

bool enableInterrupt
: Enables or disables WDOG interrupt

uint16_t timeoutValue
: Timeout value

uint16_t interruptTimeValue
: Interrupt count timeout value

bool softwareResetExtension
: software reset extension

bool enablePowerDown
: power down enable bit

bool enableTimeOutAssert
: Enable WDOG_B timeout assertion.

Chapter 3

Middleware

3.1 Motor Control

3.1.1 FreeMASTER

Communication Driver User Guide

Introduction

What is FreeMASTER? FreeMASTER is a PC-based application developed by NXP for NXP customers. It is a versatile tool usable as a real-time monitor, visualization tool, and a graphical control panel of embedded applications based on the NXP processing units.

This document describes the embedded-side software driver which implements an interface between the application and the host PC. The interface covers the following communication:

- **Serial** UART communication either over plain RS232 interface or more typically over a USB-to-Serial either external or built in a debugger probe.
- **USB** direct connection to target microcontroller
- **CAN bus**
- **TCP/IP network** wired or WiFi
- **Segger J-Link RTT**
- **JTAG** debug port communication
- ...and all of the above also using a **Zephyr** generic drivers.

The driver also supports so-called "packet-driven BDM" interface which enables a protocol-based communication over a debugging port. The BDM stands for Background Debugging Module and its physical implementation is different on each platform. Some platforms leverage a semi-standard JTAG interface, other platforms provide a custom implementation called BDM. Regardless of the name, this debugging interface enables non-intrusive access to the memory space while the target CPU is running. For basic memory read and write operations, there is no communication driver required on the target when communicating with the host PC. Use this driver to get more advanced FreeMASTER protocol features over the BDM interface. The driver must be configured for the packet-driven BDM mode, in which the host PC uses the debugging interface to write serial command frames directly to the target memory buffer. The same method is then used to read response frames from that memory buffer.

Similar to “packet-driven BDM”, the FreeMASTER also supports a communication over [J-Link RTT]((https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/) interface defined by SEGGER Microcontroller GmbH for ARM CortexM-based microcontrollers. This method also uses JTAG physical interface and enables high-speed real time communication to run over the same channel as used for application debugging.

Driver version 3 This document describes version 3 of the FreeMASTER Communication Driver. This version features the implementation of the new Serial Protocol, which significantly extends the features and security of its predecessor. The new protocol internal number is v4 and its specification is available in the documentation accompanying the driver code.

Driver V3 is deployed to modern 32-bit MCU platforms first, so the portfolio of supported platforms is smaller than for the previous V2 versions. It is recommended to keep using the V2 driver for legacy platforms, such as S08, S12, ColdFire, or Power Architecture. Reach out to FreeMASTER community or to the local NXP representative with requests for more information or to port the V3 driver to legacy MCU devices.

Thanks to a layered approach, the new driver simplifies the porting of the driver to new UART, CAN or networking communication interfaces significantly. Users are encouraged to port the driver to more NXP MCU platforms and contribute the code back to NXP for integration into future releases. Existing code and low-level driver layers may be used as an example when porting to new targets.

Note: Using the FreeMASTER tool and FreeMASTER Communication Driver is only allowed in systems based on NXP microcontroller or microprocessor unit. Use with non-NXP MCU platforms is **not permitted** by the license terms.

Target platforms The driver implementation uses the following abstraction mechanisms which simplify driver porting and supporting new communication modules:

- **General CPU Platform** (see source code in the src/platforms directory). The code in this layer is only specific to native data type sizes and CPU architectures (for example; alignment-aware memory copy routines). This driver version brings two generic implementations of 32-bit platforms supporting both little-endian and big-endian architectures. There are also implementations customized for the 56F800E family of digital signal controllers and S12Z MCUs. **Zephyr** is treated as a specific CPU platform as it brings unified user configuration (Kconfig) and generic hardware device drivers. With Zephyr, the transport layer and low-level communication layers described below are configured automatically using Kconfig and Device Tree technologies.
- **Transport Communication Layer** - The Serial, CAN, Networking, PD-BDM, and other methods of transport logic are implemented as a driver layer called FMSTR_TRANSPORT with a uniform API. A support of the Network transport also extends single-client modes of operation which are native for Serial, USB and CAN by a concept of multiple client sessions.
- **Low-level Communication Driver** - Each type of transport further defines a low-level API used to access the physical communication module. For example, the Serial transport defines a character-oriented API implemented by different serial communication modules like UART, LPUART, USART, and also USB-CDC. Similarly, the CAN transport defines a message-oriented API implemented by the FlexCAN or MCAN modules. Moreover, there are multiple different implementations for the same kind of communication peripherals. The difference between the implementation is in the way the low-level hardware registers are accessed. The *mcuxsdk* folder contains implementations which use MCUXpresso SDK drivers. These drivers should be used in applications based on the NXP MCUXpresso SDK. The “ampsdk” drivers target automotive-specific MCUs and their respective SDKs. The “dreg” implementations use a plain C-language access to hardware register addresses which makes it a universal and the most portable solution. In this case, users are encouraged to add more drivers for other communication modules or other respective SDKs and contribute the code back to NXP for integration.

The low-level drivers defined for the Networking transport enable datagram-oriented UDP and stream TCP communication. This implementation is demonstrated using the lwIP software stack but shall be portable to other TCP/IP stacks. It may sound surprisingly, but also the Segger J-Link RTT communication driver is linked to the Networking transport (RTT is stream oriented communication handled similarly to TCP).

Replacing existing drivers For all supported platforms, the driver described in this document replaces the V2 implementation and also older driver implementations that were available separately for individual platforms (PC Master SCI drivers).

Clocks, pins, and peripheral initialization The FreeMASTER communication driver is only responsible for runtime processing of the communication and must be integrated with an user application code to function properly. The user application code is responsible for general initialization of clock sources, pin multiplexers, and peripheral registers related to the communication speed. Such initialization should be done before calling the FMSTR_Init function.

It is recommended to develop the user application using one of the Software Development Kits (SDKs) available from third parties or directly from NXP, such as MCUXpresso SDK, MCUXpresso IDE, and related tools. This approach simplifies the general configuration process significantly.

MCUXpresso SDK The MCUXpresso SDK is a software package provided by NXP which contains the device initialization code, linker files, and software drivers with example applications for the NXP family of MCUs. The MCUXpresso Config Tools may be used to generate the clock-setup and pin-multiplexer setup code suitable for the selected processor.

The MCUXpresso SDK also contains this FreeMASTER communication driver as a "middleware" component which may be downloaded along with the example applications from https://mcuxpresso.nxp.com/en/welcome.

MCUXpresso SDK on GitHub The FreeMASTER communication driver is also released as one of the middleware components of the MCUXpresso SDK on the GitHub. This release enables direct integration of the FreeMASTER source code Git repository into a target applications including Zephyr applications.

Related links:

- The official FreeMASTER middleware repository.
- Online version of this document

FreeMASTER in Zephyr The FreeMASTER middleware repository can be used with MCUXpresso SDK as well as a Zephyr module. Zephyr-specific samples which include examples of Kconfig and Device Tree configurations for Serial, USB and Network communications are available in separate repository. West manifest in this sample repository fetches the full Zephyr package including the FreeMASTER middleware repository used as a Zephyr module.

Example applications

MCUX SDK Example applications There are several example applications available for each supported MCU platform.

- **fmstr_uart** demonstrates a plain serial transmission, typically connecting to a computer's physical or virtual COM port. The typical transmission speed is 115200 bps.

- **fmstr_can** demonstrates CAN bus communication. This requires a suitable CAN interface connected to the computer and interconnected with the target MCU using a properly terminated CAN bus. The typical transmission speed is 500 kbps. A FreeMASTER-over-CAN communication plug-in must be used.
- **fmstr_usb_cdc** uses an on-chip USB controller to implement a CDC communication class. It is connected directly to a computer's USB port and creates a virtual COM port device. The typical transmission speed is above 1 Mbps.
- **fmstr_net** demonstrates the Network communication over UDP or TCP protocol. Existing examples use lwIP stack to implement the communication, but in general, it shall be possible to use any other TCP/IP stack to achieve the same functionality.
- **fmstr_wifi** is the fmstr_net application modified to use a WiFi network interface instead of a wired Ethernet connection.
- **fmstr_rtt** demonstrates the communication over SEGGER J-Link RTT interface. Both fmstr_net and fmstr_rtt examples require the FreeMASTER TCP/UDP communication plug-in to be used on the PC host side.
- **fmstr_eonce** uses the real-time data unit on the JTAG EOnCE module of the 56F800E family to implement pseudo-serial communication over the JTAG port. The typical transmission speed is around 10 kbps. This communication requires FreeMASTER JTAG/EOnCE communication plug-in.
- **fmstr_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 spplications Zephyr sample applications demonstrate Kconfig and Device Tree configuration which configure the FreeMASTER middleware module for a selected communication option (Serial, CAN, Network or RTT).

Refer to *readme.md* files in each sample directory for description of configuration options required to implement FreeMASTER connectivity.

Description

This section shows how to add the FreeMASTER Communication Driver into application and how to configure the connection to the FreeMASTER visualization tool.

Features The FreeMASTER driver implements the FreeMASTER protocol V4 and provides the following features which may be accessed using the FreeMASTER visualization tool:

- Read/write access to any memory location on the target.
- Optional password protection of the read, read/write, and read/write/flash access levels.

- Atomic bit manipulation on the target memory (bit-wise write access).
- Optimal size-aligned access to memory which is also suitable to access the peripheral register space.
- Oscilloscope access—real-time access to target variables. The sample rate may be limited by the communication speed.
- Recorder— access to the fast transient recorder running on the board as a part of the FreeMASTER driver. The sample rate is only limited by the MCU CPU speed. The length of the data recorded depends on the amount of available memory.
- Multiple instances of Oscilloscopes and Recorders without the limitation of maximum number of variables.
- Application commands—high-level message delivery from the PC to the application.
- TSA tables—describing the data types, variables, files, or hyperlinks exported by the target application. The TSA newly supports also non-memory mapped resources like external EEPROM or SD Card files.
- Pipes—enabling the buffered stream-oriented data exchange for a general-purpose terminal-like communication, diagnostic data streaming, or other data exchange.

The FreeMASTER driver features:

- Full FreeMASTER protocol V4 implementation with a new V4 style of CRC used.
- Layered approach supporting Serial, CAN, Network, PD-BDM, and other transports.
- Layered low-level Serial transport driver architecture enabling to select UART, LPUART, USART, and other physical implementations of serial interfaces, including USB-CDC.
- Layered low-level CAN transport driver architecture enabling to select FlexCAN, msCAN, MCAN, and other physical implementations of the CAN interface.
- Layered low-level Networking transport enabling to select TCP, UDP or J-Link RTT communication.
- TSA support to write-protect memory regions or individual variables and to deny the access to the unsafe memory.
- The pipe callback handlers are invoked whenever new data is available for reading from the pipe.
- Two Serial Single-Wire modes of operation are enabled. The “external” mode has the RX and TX shorted on-board. The “true” single-wire mode interconnects internally when the MCU or UART modules support it.

The following sections briefly describe all FreeMASTER features implemented by the driver. See the PC-based FreeMASTER User Manual for more details on how to use the features to monitor, tune, or control an embedded application.

Board Detection The FreeMASTER protocol V4 defines the standard set of configuration values which the host PC tool reads to identify the target and to access other target resources properly. The configuration includes the following parameters:

- Version of the driver and the version of the protocol implemented.
- MTU as the Maximum size of the Transmission Unit (for example; communication buffer size).
- Application name, description, and version strings.
- Application build date and time as a string.
- Target processor byte ordering (little/big endian).
- Protection level that requires password authentication.

- Number of the Recorder and Oscilloscope instances.
- RAM Base Address for optimized memory access commands.

Memory Read This basic feature enables the host PC to read any data memory location by specifying the address and size of the required memory area. The device response frame must be shorter than the MTU to fit into the outgoing communication buffer. To read a device memory of any size, the host uses the information retrieved during the Board Detection and splits the large-block request to multiple partial requests.

The driver uses size-aligned operations to read the target memory (for example; uses proper read-word instruction when an address is aligned to 4 bytes).

Memory Write Similarly to the Memory Read operation, the Memory Write feature enables to write to any RAM memory location on the target device. A single write command frame must be shorter than the MTU to fit into the target communication buffer. Larger requests must be split into smaller ones.

The driver uses size-aligned operations to write to the target memory (for example; uses proper write-word instruction when an address is aligned to 4 bytes).

Masked Memory Write To implement the write access to a single bit or a group of bits of target variables, the Masked Memory Write feature is available in the FreeMASTER protocol and it is supported by the driver using the Read-Modify-Write approach.

Be careful when writing to bit fields of volatile variables that are also modified in an application interrupt. The interrupt may be serviced in the middle of a read-modify-write operation and it may cause data corruption.

Oscilloscope The protocol and driver enables any number of variables to be read at once with a single request from the host. This feature is called Oscilloscope and the FreeMASTER tool uses it to display a real-time graph of variable values.

The driver can be configured to support any number of Oscilloscope instances and enable simultaneously running graphs to be displayed on the host computer screen.

Recorder The protocol enables the host to select target variables whose values are then periodically recorded into a dedicated on-board memory buffer. After such data sampling stops (either on a host request or by evaluating a threshold-crossing condition), the data buffer is downloaded to the host and displayed as a graph. The data sampling rate is not limited by the speed of the communication line, so it enables displaying the variable transitions in a very high resolution.

The driver can be configured to support multiple Recorder instances and enable multiple recorder graphs to be displayed on the host screen. Having multiple recorders also enables setting the recording point differently for each instance. For example; one instance may be recording data in a general timer interrupt while another instance may record at a specific control algorithm time in the PWM interrupt.

TSA With the TSA feature, data types and variables can be described directly in the application source code. Such information is later provided to the FreeMASTER tool which may use it instead of reading symbol data from the application ELF executable file.

The information is encoded as so-called TSA tables which become direct part of the application code. The TSA tables contain descriptors of variables that shall be visible to the host tool. The descriptors can describe the memory areas by specifying the address and size of the memory

block or more conveniently using the C variable names directly. Different set of TSA descriptors can be used to encode information about the structure types, unions, enumerations, or arrays.

The driver also supports special types of TSA table entries to describe user resources like external EEPROM and SD Card files, memory-mapped files, virtual directories, web URL hyperlinks, and constant enumerations.

TSA Safety When the TSA is enabled in the application, the TSA Safety can be enabled and validate the memory accesses directly by the embedded-side driver. When the TSA Safety is turned on, any memory request received from the host is validated and accepted only if it belongs to a TSA-described object. The TSA entries can be declared as Read-Write or Read-Only so that the driver can actively deny the write access to the Read-Only objects.

Application commands The Application Commands are high-level messages that can be delivered from the PC Host to the embedded application for further processing. The embedded application can either poll the status, or be called back when a new Application Command arrives to be processed. After the embedded application acknowledges that the command is handled, the host receives the Result Code and reads the other return data from memory. Both the Application Commands and the Result Codes are specific to a given application and it is user's responsibility to define them. The FreeMASTER protocol and the FreeMASTER driver only implement the delivery channel and a set of API calls to enable the Application Command processing in general.

Pipes The Pipes enable buffered and stream-oriented data exchange between the PC Host and the target application. Any pipe can be written to and read from at both ends (either on the PC or the MCU). The data transmission is acknowledged using the special FreeMASTER protocol commands. It is guaranteed that the data bytes are delivered from the writer to the reader in a proper order and without losses.

Serial single-wire operation The MCU Serial Communication Driver natively supports normal dual-wire operation. Because the protocol is half-duplex only, the driver can also operate in two single-wire modes:

- "External" single-wire operation where the Receiver and Transmitter pins are shorted on the board. This mode is supported by default in the MCU driver because the Receiver and Transmitter units are enabled or disabled whenever needed. It is also easy to extend this operation for the RS485 communication.
- "True" single-wire mode which uses only a single pin and the direction switching is made by the UART module. This mode of operation must be enabled by defining the FMSTR_SERIAL_SINGLEWIRE configuration option.

Multi-session support With networking interface it is possible for multiple clients to access the target MCU simultaneously. Reading and writing of target memory is processed atomically so there is no risk of data corruption. The state-full resources such as Recorders or Oscilloscopes are locked to a client session upon first use and access is denied to other clients until lock is released..

Zephyr-specific

Dedicated communication task FreeMASTER communication may run isolated in a dedicated task. The task automates the FMSTR_Init and FMSTR_Poll calls together with periodic activities enabling the FreeMASTER UI to fetch information about tasks and CPU utilization. The task can be started automatically or manually, and it must be assigned a priority to be able to react on interrupts and other communication events. Refer to Zephyr FreeMASTER sample applications which all use this communication task.

Zephyr shell and logging over FreeMASTER pipe FreeMASTER implements a shell backend which may use FreeMASTER pipe as a I/O terminal and logging output. Refer to Zephyr FreeMASTER sample applications which all use this feature.

Automatic TSA tables TSA tables can be declared as "automatic" in Zephyr which make them automatically registered in the table list. This may be very useful when there are many TSA tables or when the tables are defined in different (often unrelated) libraries linked together. In this case user does not need to build a list of all tables manually.

Driver files The driver source files can be found in a top-level src folder, further divided into the sub-folders:

- ***src/platforms*** platform-specific folder—one folder exists for each supported processor platform (for example; 32-bit Little Endian platform). Each such folder contains a platform header file with data types and a code which implements the potentially platform-specific operations, such as aligned memory access.
- ***src/common*** folder—contains the common driver source files shared by the driver for all supported platforms. All the *.c* files must be added to the project, compiled, and linked together with the application.
  - *freemaster.h* - master driver header file, which declares the common data types, macros, and prototypes of the FreeMASTER driver API functions.
  - *freemaster_cfg.h.example* - this file can serve as an example of the FreeMASTER driver configuration file. Save this file into a project source code folder and rename it to *freemaster_cfg.h*. The FreeMASTER driver code includes this file to get the project-specific configuration options and to optimize the compilation of the driver.
  - *freemaster_defcfg.h* - defines the default values for each FreeMASTER configuration option if the option is not set in the *freemaster_cfg.h* file.
  - *freemaster_protocol.h* - defines the FreeMASTER protocol constants used internally by the driver.
  - *freemaster_protocol.c* - implements the FreeMASTER protocol decoder and handles the basic Get Configuration Value, Memory Read, and Memory Write commands.
  - *freemaster_rec.c* - handles the Recorder-specific commands and implements the Recorder sampling and triggering routines. When the Recorder is disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
  - *freemaster_scope.c* - handles the Oscilloscope-specific commands. If the Oscilloscope is disabled by the FreeMASTER driver configuration file, this file compiles as void.
  - *freemaster_pipes.c* - implements the Pipes functionality when the Pipes feature is enabled.
  - *freemaster_appcmd.c* - handles the communication commands used to deliver and execute the Application Commands within the context of the embedded application. When the Application Commands are disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.

  - *freemaster_tsa.c* - handles the commands specific to the TSA feature. This feature enables the FreeMASTER host tool to obtain the TSA memory descriptors declared in the embedded application. If the TSA is disabled by the FreeMASTER driver configuration file, this file compiles as void.
  - *freemaster_tsa.h* - contains the declaration of the macros used to define the TSA memory descriptors. This file is indirectly included into the user application code (via *freemaster.h*).
  - *freemaster_sha.c* - implements the SHA-1 hash code used in the password authentication algorithm.
  - *freemaster_private.h* - contains the declarations of functions and data types used internally in the driver. It also contains the C pre-processor statements to perform the compile-time verification of the user configuration provided in the *freemaster_cfg.h* file.
  - *freemaster_serial.c* - implements the serial protocol logic including the CRC, FIFO queuing, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a character-oriented API exported by the specific low-level driver.
  - *freemaster_serial.h* - defines the low-level character-oriented Serial API.
  - *freemaster_can.c* - implements the CAN protocol logic including the CAN message preparation, signalling using the first data byte in the CAN frame, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a message-oriented API exported by the specific low-level driver.
  - *freemaster_can.h* - defines the low-level message-oriented CAN API.
  - *freemaster_net.c* - implements the Network protocol transport logic including multiple session management code.
  - *freemaster_net.h* - definitions related to the Network transport.
  - *freemaster_pdbdm.c* - implements the packet-driven BDM communication buffer and other communication-related operations.
  - *freemaster_utils.c* - aligned memory copy routines, circular buffer management and other utility functions
  - *freemaster_utils.h* - definitions related to utility code.
- ***src/drivers/[sdk]/serial*** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster_serial_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the UART, LPUART, USART, and other kinds of Serial communication modules.
- ***src/drivers/[sdk]/can*** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the FlexCAN, msCAN, MCAN, and other kinds of CAN communication modules.
- ***src/drivers/[sdk]/network*** - contains low-level code adapting the FreeMASTER Network transport to an underlying TCP/IP or RTT stack.
  - *freemaster_net_lwip_tcp.c* and *_udp.c* - default networking implementation of TCP and UDP transports using lwIP stack.
  - *freemaster_net_segger_rtt.c* - implementation of network transport using Segger J-Link RTT interface

Driver configuration The driver is configured using a single header file (*freemaster_cfg.h*). Create this file and save it together with other project source files before compiling the driver code. All FreeMASTER driver source files include the *freemaster_cfg.h* file and use the macros defined here for the conditional and parameterized compilation. The C compiler must locate the configuration file when compiling the driver files. Typically, it can be achieved by putting this file into a folder where the other project-specific included files are stored.

As a starting point to create the configuration file, get the *freemaster_cfg.h.example* file, rename it to *freemaster_cfg.h*, and save it into the project area.

Note: It is NOT recommended to leave the *freemaster_cfg.h* file in the FreeMASTER driver source code folder. The configuration file must be placed at a project-specific location, so that it does not affect the other applications that use the same driver.

Configurable items This section describes the configuration options which can be defined in *freemaster_cfg.h*.

Interrupt modes

```
#define FMSTR_LONG_INTR   [0|1]
#define FMSTR_SHORT_INTR  [0|1]
#define FMSTR_POLL_DRIVEN [0|1]
```

Value Type boolean (0 or 1)

Description Exactly one of the three macros must be defined to non-zero. The others must be defined to zero or left undefined. The non-zero-defined constant selects the interrupt mode of the driver. See *Driver interrupt modes*.

- FMSTR_LONG_INTR — long interrupt mode
- FMSTR_SHORT_INTR — short interrupt mode
- FMSTR_POLL_DRIVEN — poll-driven mode

Note: Some options may not be supported by all communication interfaces. For example, the FMSTR_SHORT_INTR option is not supported by the USB_CDC interface.

Protocol transport

```
#define FMSTR_TRANSPORT [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER source code. Specify one of existing instances to make use of the protocol transport.

Description Use one of the pre-defined constants, as implemented by the FreeMASTER code. The current driver supports the following transports:

- **FMSTR_SERIAL** - serial communication protocol
- **FMSTR_CAN** - using CAN communication
- **FMSTR_PDBDM** - using packet-driven BDM communication
- **FMSTR_NET** - network communication using TCP or UDP protocol

Serial transport This section describes configuration parameters used when serial transport is used:

```
#define FMSTR_TRANSPORT FMSTR_SERIAL
```

FMSTR_SERIAL_DRV Select what low-level driver interface will be used when implementing the Serial communication.

```
#define FMSTR_SERIAL_DRV [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing serial driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/serial* implementation):

- **FMSTR_SERIAL_MCUX_UART** - UART driver
- **FMSTR_SERIAL_MCUX_LPUART** - LPUART driver
- **FMSTR_SERIAL_MCUX_USART** - USART driver
- **FMSTR_SERIAL_MCUX_MINIUSART** - miniUSART driver
- **FMSTR_SERIAL_MCUX_QSCI** - DSC QSCI driver
- **FMSTR_SERIAL_MCUX_USB** - USB/CDC class driver (also see code in the */support/mcuxsdk_usb* folder)
- **FMSTR_SERIAL_56F800E_EONCE** - DSC JTAG EOnCE driver

Other SDKs or BSPs may define custom low-level driver interface structure which may be used as FMSTR_SERIAL_DRV. For example:

- **FMSTR_SERIAL_DREG_UART** - demonstrates the low-level interface implemented without the MCUXpresso SDK and using direct access to peripheral registers.

FMSTR_SERIAL_BASE

```
#define FMSTR_SERIAL_BASE [address|symbol]
```

Value Type Optional address value (numeric or symbolic)

Description Specify the base address of the UART, LPUART, USART, or other serial peripheral module to be used for the communication. This value is not defined by default. User application should call `FMSTR_SetSerialBaseAddress()` to select the peripheral module.

FMSTR_COMM_BUFFER_SIZE

```
#define FMSTR_COMM_BUFFER_SIZE [number]
```

Value Type 0 or a value in range 32...255

Description Specify the size of the communication buffer to be allocated by the driver. Default value, which suits all driver features, is used when this option is defined as 0.

FMSTR_COMM_RQUEUE_SIZE

```
#define FMSTR_COMM_RQUEUE_SIZE [number]
```

Value Type Value in range 0...255

Description Specify the size of the FIFO receiver queue used to quickly receive and store characters in the FMSTR_SHORT_INTR interrupt mode.
The default value is 32 B.

FMSTR_SERIAL_SINGLEWIRE

```
#define FMSTR_SERIAL_SINGLEWIRE [0|1]
```

Value Type Boolean 0 or 1.

Description Set to non-zero to enable the "True" single-wire mode which uses a single MCU pin to communicate. The low-level driver enables the pin direction switching when the MCU peripheral supports it.

CAN Bus transport This section describes configuration parameters used when CAN transport is used:

```
#define FMSTR_TRANSPORT FMSTR_CAN
```

FMSTR_CAN_DRV Select what low-level driver interface will be used when implementing the CAN communication.

```
#define FMSTR_CAN_DRV [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing CAN driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/can implementation*):

- **FMSTR_CAN_MCUX_FLEXCAN** - FlexCAN driver
- **FMSTR_CAN_MCUX_MCAN** - MCAN driver
- **FMSTR_CAN_MCUX_MSCAN** - msCAN driver
- **FMSTR_CAN_MCUX_DSCFLEXCAN** - DSC FlexCAN driver
- **FMSTR_CAN_MCUX_DSCMSCAN** - DSC msCAN driver

Other SDKs or BSPs may define the custom low-level driver interface structure which may be used as FMSTR_CAN_DRV.

FMSTR_CAN_BASE

```
#define FMSTR_CAN_BASE [address|symbol]
```

Value Type Optional address value (numeric or symbolic)

Description Specify the base address of the FlexCAN, msCAN, or other CAN peripheral module to be used for the communication. This value is not defined by default. User application should call FMSTR_SetCanBaseAddress() to select the peripheral module.

FMSTR_CAN_CMDID

```
#define FMSTR_CAN_CMDID [number]
```

Value Type CAN identifier (11-bit or 29-bit number)

Description CAN message identifier used for FreeMASTER commands (direction from PC Host tool to target application). When declaring 29-bit identifier, combine the numeric value with FMSTR_CAN_EXTID bit. Default value is 0x7AA.

FMSTR_CAN_RSPID

```
#define FMSTR_CAN_RSPID [number]
```

Value Type CAN identifier (11-bit or 29-bit number)

Description CAN message identifier used for responding messages (direction from target application to PC Host tool). When declaring 29-bit identifier, combine the numeric value with FMSTR_CAN_EXTID bit. Note that both *CMDID* and *RSPID* values may be the same. Default value is 0x7AA.

FMSTR_FLEXCAN_TXMB

```
#define FMSTR_FLEXCAN_TXMB [number]
```

Value Type Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame transmission. Default value is 0.

FMSTR_FLEXCAN_RXMB

```
#define FMSTR_FLEXCAN_RXMB [number]
```

Value Type Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame reception. Note that the FreeMASTER driver may also operate with a common message buffer used by both TX and RX directions. Default value is 1.

Network transport This section describes configuration parameters used when Network transport is used:

```
#define FMSTR_TRANSPORT FMSTR_NET
```

FMSTR_NET_DRV Select network interface implementation.

```
#define FMSTR_NET_DRV [identifier]
```

Value Type Identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing NET driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/network implementation*):

- **FMSTR_NET_LWIP_TCP** - TCP communication using lwIP stack
- **FMSTR_NET_LWIP_UDP** - UDP communication using lwIP stack
- **FMSTR_NET_SEGGER_RTT** - Communication using SEGGER J-Link RTT interface

Other SDKs or BSPs may define the custom networking interface which may be used as FMSTR_CAN_DRV.

Add another row below:

FMSTR_NET_PORT

```
#define FMSTR_NET_PORT [number]
```

Value Type TCP or UDP port number (short integer)

Description Specifies the server port number used by TCP or UDP protocols.

FMSTR_NET_BLOCKING_TIMEOUT

```
#define FMSTR_NET_BLOCKING_TIMEOUT [number]
```

Value Type Timeout as number of milliseconds

Description This value specifies a timeout in milliseconds for which the network socket operations may block the execution inside *FMSTR_Poll*. This may be set high (e.g. 250) when a dedicated RTOS task is used to handle FreeMASTER protocol polling. Set to a lower value when the polling task is also responsible for other operations. Set to 0 to attempt to use non-blocking socket operations.

FMSTR_NET_AUTODISCOVERY

```
#define FMSTR_NET_AUTODISCOVERY [0|1]
```

Value Type Boolean 0 or 1.

Description This option enables the FreeMASTER driver to use a separate UDP socket to broadcast auto-discovery messages to network. This helps the FreeMASTER tool to discover the target device address, port and protocol options.

Debugging options

FMSTR_DISABLE

```
#define FMSTR_DISABLE [0|1]
```

Value Type boolean (0 or 1)

Description Define as non-zero to disable all FreeMASTER features, exclude the driver code from build, and compile all its API functions empty. This may be useful to remove FreeMASTER without modifying any application source code. Default value is 0 (false).

FMSTR_DEBUG_TX

```
#define FMSTR_DEBUG_TX [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to enable the driver to periodically transmit test frames out on the selected communication interface (SCI or CAN). With the debug transmission enabled, it is simpler to detect problems in the baudrate or other communication configuration settings.

The test frames are transmitted until the first valid command frame is received from the PC Host tool. The test frame is a valid error status frame, as defined by the protocol format. On the serial line, the test frame consists of three printable characters (+©W) which are easy to capture using the serial terminal tools.

This feature requires the FMSTR_Poll() function to be called periodically. Default value is 0 (false).

FMSTR_APPLICATION_STR

```
#define FMSTR_APPLICATION_STR
```

Value Type String.

Description Name of the application visible in FreeMASTER host application.

Memory access

FMSTR_USE_READMEM

```
#define FMSTR_USE_READMEM [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Memory Read command and enable FreeMASTER to have read access to memory and variables. The access can be further restricted by using a TSA feature.
Default value is 1 (true).

FMSTR_USE_WRITEMEM

```
#define FMSTR_USE_WRITEMEM [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Memory Write command.
The default value is 1 (true).

Oscilloscope options

FMSTR_USE_SCOPE

```
#define FMSTR_USE_SCOPE [number]
```

Value Type Integer number.

Description Number of Oscilloscope instances to be supported. Set to 0 to disable the Oscilloscope feature.
Default value is 0.

FMSTR_MAX_SCOPE_VARS

```
#define FMSTR_MAX_SCOPE_VARS [number]
```

Value Type Integer number larger than 2.

Description Number of variables to be supported by each Oscilloscope instance.
Default value is 8.

Recorder options

FMSTR_USE_RECORDER

```
#define FMSTR_USE_RECORDER [number]
```

Value Type Integer number.

Description Number of Recorder instances to be supported. Set to 0 to disable the Recorder feature.
Default value is 0.

FMSTR_REC_BUFF_SIZE

```
#define FMSTR_REC_BUFF_SIZE [number]
```

Value Type Integer number larger than 2.

Description Defines the size of the memory buffer used by the Recorder instance #0.
Default: not defined, user shall call 'FMSTR_RecorderCreate()" API function to specify this parameter in run time.

FMSTR_REC_TIMEBASE

```
#define FMSTR_REC_TIMEBASE [time specification]
```

Value Type Number (nanoseconds time).

Description Defines the base sampling rate in nanoseconds (sampling speed) Recorder instance #0.

Use one of the following macros:

- FMSTR_REC_BASE_SECONDS(x)
- FMSTR_REC_BASE_MILLISEC(x)
- FMSTR_REC_BASE_MICROSEC(x)
- FMSTR_REC_BASE_NANOSEC(x)

Default: not defined, user shall call 'FMSTR_RecorderCreate()" API function to specify this parameter in run time.

FMSTR_REC_FLOAT_TRIG

```
#define FMSTR_REC_FLOAT_TRIG [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the floating-point triggering. Be aware that floating-point triggering may grow the code size by linking the floating-point standard library.

Default value is 0 (false).

Application Commands options

FMSTR_USE_APPCMD

```
#define FMSTR_USE_APPCMD [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Application Commands feature.
Default value is 0 (false).

FMSTR_APPCMD_BUFF_SIZE

```
#define FMSTR_APPCMD_BUFF_SIZE [size]
```

Value Type Numeric buffer size in range 1..255

Description The size of the Application Command data buffer allocated by the driver. The buffer stores the (optional) parameters of the Application Command which waits to be processed.

FMSTR_MAX_APPCMD_CALLS

```
#define FMSTR_MAX_APPCMD_CALLS [number]
```

Value Type Number in range 0..255

Description The number of different Application Commands that can be assigned a callback handler function using FMSTR_RegisterAppCmdCall(). Default value is 0.

TSA options

FMSTR_USE_TSA

```
#define FMSTR_USE_TSA [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the FreeMASTER TSA feature to be used. With this option enabled, the TSA tables defined in the applications are made available to the FreeMASTER host tool.
Default value is 0 (false).

FMSTR_USE_TSA_SAFETY

```
#define FMSTR_USE_TSA_SAFETY [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the memory access validation in the FreeMASTER driver. With this option, the host tool is not able to access the memory which is not described by at least one TSA descriptor. Also a write access is denied for objects defined as read-only in TSA tables.
Default value is 0 (false).

FMSTR_USE_TSA_INROM

```
#define FMSTR_USE_TSA_INROM [0|1]
```

Value Type Boolean 0 or 1.

Description Declare all TSA descriptors as *const*, which enables the linker to put the data into the flash memory. The actual result depends on linker settings or the linker commands used in the project.
Default value is 0 (false).

FMSTR_USE_TSA_DYNAMIC

```
#define FMSTR_USE_TSA_DYNAMIC [0|1]
```

Value Type Boolean 0 or 1.

Description Enable runtime-defined TSA entries to be added to the TSA table by the FMSTR_SetUpTsaBuff() and FMSTR_TsaAddVar() functions.
Default value is 0 (false).

Pipes options

FMSTR_USE_PIPES

```
#define FMSTR_USE_PIPES [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the FreeMASTER Pipes feature to be used.
Default value is 0 (false).

FMSTR_MAX_PIPES_COUNT

```
#define FMSTR_MAX_PIPES_COUNT [number]
```

Value Type Number in range 1..63.

Description The number of simultaneous pipe connections to support.
The default value is 1.

Driver interrupt modes To implement the communication, the FreeMASTER driver handles the Serial or CAN module's receive and transmit requests. Use the *freemaster_cfg.h* configuration file to select whether the driver processes the communication automatically in the interrupt service routine handler or if it only polls the status of the module (typically during the application idle time).

This section describes each of the interrupt mode in more details.

Completely Interrupt-Driven operation Activated using:

```
#define FMSTR_LONG_INTR 1
```

In this mode, both the communication and the FreeMASTER protocol decoding is done in the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, or other interrupt service routine. Because the protocol execution may be a lengthy task (especially with the TSA-Safety enabled) it is recommended to use this mode only if the interrupt prioritization scheme is possible in the application and the FreeMASTER interrupt is assigned to a lower (the lowest) priority.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the FMSTR_SerialIsr or FMSTR_CanIsr functions from that handler.

Mixed Interrupt and Polling Modes Activated using:

```
#define FMSTR_SHORT_INTR 1
```

In this mode, the communication processing time is split between the interrupt routine and the main application loop or task. The raw communication is handled by the *FMSTR_SerialIsr, FMSTR_CanIsr*, or other interrupt service routine, while the protocol decoding and execution is handled by the *FMSTR_Poll* routine. Call *FMSTR_Poll* during the idle time in the application main loop.

The interrupt processing in this mode is relatively fast and deterministic. Upon a serial-receive event, the received character is only placed into a FIFO-like queue and it is not further processed. Upon a CAN receive event, the received frame is stored into a receive buffer. When transmitting, the characters are fetched from the prepared transmit buffer.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR_SerialIsr* or *FMSTR_CanIsr* functions from that handler.

When the serial interface is used as the serial communication interface, ensure that the *FMSTR_Poll* function is called at least once per *N* character time periods. *N* is the length of the FreeMASTER FIFO queue (*FMSTR_COMM_RQUEUE_SIZE*) and the character time is the time needed to transmit or receive a single byte over the SCI line.

Completely Poll-driven

```
#define FMSTR_POLL_DRIVEN 1
```

In this mode, both the communication and the FreeMASTER protocol decoding are done in the *FMSTR_Poll* routine. No interrupts are needed and the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, and similar handlers compile to an empty code.

When using this mode, ensure that the *FMSTR_Poll* function is called by the application at least once per the serial "character time" which is the time needed to transmit or receive a single character.

In the latter two modes (*FMSTR_SHORT_INTR* and *FMSTR_POLL_DRIVEN*), the protocol handling takes place in the FMSTR_Poll routine. An application interrupt can occur in the middle of the

Read Memory or Write Memory commands' execution and corrupt the variable being accessed by the FreeMASTER driver. In these two modes, some issues or glitches may occur when using FreeMASTER to visualize or monitor volatile variables modified in interrupt servicing code.

The same issue may appear even in the full interrupt mode (FMSTR_LONG_INTR), if volatile variables are modified in the interrupt code with a priority higher than the priority of the communication interrupt.

Data types Simple portability was one of the main requirements when writing the FreeMASTER driver. This is why the driver code uses the privately-declared data types and the vast majority of the platform-dependent code is separated in the platform-dependent source files. The data types used in the driver API are all defined in the platform-specific header file.

To prevent name conflicts with the symbols used in the application, all data types, macros, and functions have the FMSTR_ prefix. The only global variables used in the driver are the transport and low-level API structures exported from the driver-implementation layer to upper layers. Other than that, all private variables are declared as static and named using the fmstr_ prefix.

Communication interface initialization The FreeMASTER driver does not perform neither the initialization nor the configuration of the peripheral module that it uses to communicate. It is the application startup code responsibility to configure the communication module before the FreeMASTER driver is initialized by the FMSTR_Init call.

When the Serial communication module is used as the FreeMASTER communication interface, configure the UART receive and transmit pins, the serial communication baud rate, parity (no-parity), the character length (eight bits), and the number of stop bits (one) before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see *Driver interrupt modes*), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected serial peripheral module. Call the FMSTR_SerialIsr function from the application handler.

When a CAN module is used as the FreeMASTER communication interface, configure the CAN receive and transmit pins and the CAN module bit rate before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see *Driver interrupt modes*), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected CAN peripheral module. Call the FMSTR_CanIsr function from the application handler.

Note: It is not necessary to enable or unmask the serial nor the CAN interrupts before initializing the FreeMASTER driver. The driver enables or disables the interrupts and communication lines, as required during runtime.

FreeMASTER Recorder calls When using the FreeMASTER Recorder in the application (FMSTR_USE_RECORDER > 0), call the FMSTR_RecorderCreate function early after FMSTR_Init to set up each recorder instance to be used in the application. Then call the FMSTR_Recorder function periodically in the code where the data recording should occur. A typical place to call the Recorder routine is at the timer or PWM interrupts, but it can be anywhere else. The example applications provided together with the driver code call the FMSTR_Recorder in the main application loop.

In applications where FMSTR_Recorder is called periodically with a constant period, specify the period in the Recorder configuration structure before calling FMSTR_RecorderCreate. This setting enables the PC Host FreeMASTER tool to display the X-axis of the Recorder graph properly scaled for the time domain.

Driver usage Start using or evaluating FreeMASTER by opening some of the example applications available in the driver setup package.

Follow these steps to enable the basic FreeMASTER connectivity in the application:

- Make sure that all *.c files of the FreeMASTER driver from the *src/common/platforms/[your_platform]* folder are a part of the project. See *Driver files* for more details.
- Configure the FreeMASTER driver by creating or editing the *freemaster_cfg.h* file and by saving it into the application project directory. See *Driver configuration* for more details.
- Include the *freemaster.h* file into any application source file that makes the FreeMASTER API calls.
- Initialize the Serial or CAN modules. Set the baud rate, parity, and other parameters of the communication. Do not enable the communication interrupts in the interrupt mask registers.
- For the FMSTR_LONG_INTR and FMSTR_SHORT_INTR modes, install the application-specific interrupt routine and call the FMSTR_SerialIsr or FMSTR_CanIsr functions from this handler.
- Call the FMSTR_Init function early on in the application initialization code.
- Call the FMSTR_RecorderCreate functions for each Recorder instance to enable the Recorder feature.
- In the main application loop, call the FMSTR_Poll API function periodically when the application is idle.
- For the FMSTR_SHORT_INTR and FMSTR_LONG_INTR modes, enable the interrupts globally so that the interrupts can be handled by the CPU.

Communication troubleshooting The most common problem that causes communication issues is a wrong baud rate setting or a wrong pin multiplexer setting of the target MCU. When a communication between the PC Host running FreeMASTER and the target MCU cannot be established, try enabling the FMSTR_DEBUG_TX option in the *freemaster_cfg.h* file and call the FMSTR_Poll function periodically in the main application task loop.

With this feature enabled, the FreeMASTER driver periodically transmits a test frame through the Serial or CAN lines. Use a logic analyzer or an oscilloscope to monitor the signals at the communication pins of the CPU device to examine whether the bit rate and signal polarity are configured properly.

Driver API

This section describes the driver Application Programmers' Interface (API) needed to initialize and use the FreeMASTER serial communication driver.

Control API There are three key functions to initialize and use the driver.

FMSTR_Init

Prototype

```
FMSTR_BOOL FMSTR_Init(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_protocol.c*

Description This function initializes the internal variables of the FreeMASTER driver and enables the communication interface. This function does not change the configuration of the selected communication module. The hardware module must be initialized before the *FMSTR_Init* function is called.

A call to this function must occur before calling any other FreeMASTER driver API functions.

FMSTR_Poll

Prototype

```
void FMSTR_Poll(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_protocol.c*

Description In the poll-driven or short interrupt modes, this function handles the protocol decoding and execution (see *Driver interrupt modes*). In the poll-driven mode, this function also handles the communication interface with the PC. Typically, the *FMSTR_Poll* function is called during the "idle" time in the main application task loop.

To prevent the receive data overflow (loss) on a serial interface, make sure that the FMSTR_Poll function is called at least once per the time calculated as:

*N * Tchar*

where:

- *N* is equal to the length of the receive FIFO queue (configured by the FMSTR_COMM_RQUEUE_SIZE macro). *N* is 1 for the poll-driven mode.
- *Tchar* is the character time, which is the time needed to transmit or receive a single byte over the SCI line.

Note: In the long interrupt mode, this function typically compiles as an empty function and can still be called. It is worthwhile to call this function regardless of the interrupt mode used in the application. This approach enables a convenient switching between the different interrupt modes only by changing the configuration macros in the *freemaster_cfg.h* file.

FMSTR_SerialIsr / FMSTR_CanIsr

Prototype

```
void FMSTR_SerialIsr(void);
void FMSTR_CanIsr(void);
```

- Declaration: *freemaster.h*
- Implementation: *hw-specific low-level driver C file*

Description This function contains the interrupt-processing code of the FreeMASTER driver. In long or short interrupt modes (see *Driver interrupt modes*), this function must be called from the application interrupt service routine registered for the communication interrupt vector. On platforms where the communication module uses multiple interrupt vectors, the application should register a handler for all vectors and call this function at each interrupt.

Note: In a poll-driven mode, this function is compiled as an empty function and does not have to be used.

Recorder API

FMSTR_RecorderCreate

Prototype

```
FMSTR_BOOL FMSTR_RecorderCreate(FMSTR_INDEX recIndex, FMSTR_REC_BUFF* buffCfg);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function registers a recorder instance and enables it to be used by the PC Host tool. Call this function for all recorder instances from 0 to the maximum number defined by the FMSTR_USE_RECORDER configuration option (minus one). An exception to this requirement is the recorder of instance *0* which may be automatically configured by FMSTR_Init when the *freemaster_cfg.h* configuration file defines the *FMSTR_REC_BUFF_SIZE* and *FMSTR_REC_TIMEBASE* options.

For more information, see *Configurable items*.

FMSTR_Recorder

Prototype

```
void FMSTR_Recorder(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function takes a sample of the variables being recorded using the FreeMASTER Recorder instance *recIndex*. If the selected Recorder is not active when the *FMSTR_Recorder* function is being called, the function returns immediately. When the Recorder is active, the values of the variables being recorded are copied into the recorder buffer and the trigger conditions are evaluated.

If a trigger condition is satisfied, the Recorder enters the post-trigger mode, where it counts down the follow-up samples (number of *FMSTR_Recorder* function calls) and de-activates the Recorder when the required post-trigger samples are finished.

The *FMSTR_Recorder* function is typically called in the timer or PWM interrupt service routines. This function can also be called in the application main loop (for testing purposes).

FMSTR_RecorderTrigger

Prototype

```
void FMSTR_RecorderTrigger(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function forces the Recorder trigger condition to happen, which causes the Recorder to be automatically deactivated after the post-trigger samples are sampled. Use this function in the application code for programmatic control over the Recorder triggering. This can be useful when a more complex triggering conditions need to be used.

Fast Recorder API The Fast Recorder feature is not available in the FreeMASTER driver version 3. This feature was heavily dependent on the target platform and it was only available for the 56F8xxxx DSCs.

TSA Tables When the TSA is enabled in the FreeMASTER driver configuration file (by setting the FMSTR_USE_TSA macro to a non-zero value), it defines the so-called TSA tables in the application. This section describes the macros that must to be used to define the TSA tables.

There can be any number of TSA tables spread across the application source files. There must be always exactly one TSA Table List defined, which informs the FreeMASTER driver about the active TSA tables.

When there is at least one TSA table and one TSA Table List defined in the application, the TSA information automatically appears in the FreeMASTER symbols list. The symbols can then be used to create FreeMASTER variables for visualization or control.

TSA table definition The TSA table describes the static or global variables together with their address, size, type, and access-protection information. If the TSA-described variables are of a structure type, the TSA table may also describe this type and provide an access to the individual structure members of the variable.

The TSA table definition begins with the FMSTR_TSA_TABLE_BEGIN macro with a *table_id* identifying the table. The *table_id* shall be a valid C-langiage symbol.

```
FMSTR_TSA_TABLE_BEGIN(table_id)
```

After this opening macro, the TSA descriptors are placed using these macros:

```
/* Adding variable descriptors */
FMSTR_TSA_RW_VAR(name, type)  /* read/write variable entry */
FMSTR_TSA_RO_VAR(name, type)  /* read-only variable entry */

/* Description of complex data types */
FMSTR_TSA_STRUCT(struct_name) /* structure or union type entry */
FMSTR_TSA_MEMBER(struct_name, member_name, type)  /* structure member entry */

/* Memory blocks */
FMSTR_TSA_RW_MEM(name, type, address, size) /* read/write memory block */
FMSTR_TSA_RO_MEM(name, type, address, size) /* read-only memory block */
```

The table is closed using the FMSTR_TSA_TABLE_END macro:

```
FMSTR_TSA_TABLE_END()
```

TSA descriptor parameters The TSA descriptor macros accept these parameters:

- *name* — variable name. The variable must be defined before the TSA descriptor references it.
- *type* — variable or member type. Only one of the pre-defined type constants may be used (see below).
- *struct_name* — structure type name. The type must be defined (typedef) before the TSA descriptor references it.

- *member_name* — structure member name.

Note: The structure member descriptors (FMSTR_TSA_MEMBER) must immediately follow the parent structure descriptor (FMSTR_TSA_STRUCT) in the table.

Note: To write-protect the variables in the FreeMASTER driver (FMSTR_TSA_RO_VAR), enable the TSA-Safety feature in the configuration file.

TSA variable types The table lists *type* identifiers which can be used in TSA descriptors:

Constant	Description
FMSTR_TSA_UINT*n*	Unsigned integer type of size *n* bits (n=8,16,32,64)
FMSTR_TSA_SINT*n*	Signed integer type of size *n* bits (n=8,16,32,64)
FMSTR_TSA_FRAC*n*	Fractional number of size *n* bits (n=16,32,64).
FMSTR_TSA_FRAC_Q(*m*,*n*)	Signed fractional number in general Q form (m+n+1 total bits)
FMSTR_TSA_FRAC_UQ(*m*,*n*)	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)
FMSTR_TSA_USERTYPE(*name*)	Structure or union type declared with FMSTR_TSA_STRUCT record

TSA table list There shall be exactly one TSA Table List in the application. The list contains one entry for each TSA table defined anywhere in the application.

The TSA Table List begins with the FMSTR_TSA_TABLE_LIST_BEGIN macro and continues with the TSA table entries for each table.

```
FMSTR_TSA_TABLE_LIST_BEGIN()

FMSTR_TSA_TABLE(table_id)
FMSTR_TSA_TABLE(table_id2)
FMSTR_TSA_TABLE(table_id3)
...
```

The list is closed with the FMSTR_TSA_TABLE_LIST_END macro:

```
FMSTR_TSA_TABLE_LIST_END()
```

TSA Active Content entries FreeMASTER v2.0 and higher supports TSA Active Content, enabling the TSA tables to describe the memory-mapped files, virtual directories, and URL hyperlinks. FreeMASTER can access such objects similarly to accessing the files and folders on the local hard drive.

With this set of TSA entries, the FreeMASTER pages can be embedded directly into the target MCU flash and accessed by FreeMASTER directly over the communication line. The HTML-coded pages rendered inside the FreeMASTER window can access the TSA Active Content resources using a special URL referencing the *fmstr:* protocol.

This example provides an overview of the supported TSA Active Content entries:

```
FMSTR_TSA_TABLE_BEGIN(files_and_links)

/* Directory entry applies to all subsequent MEMFILE entries */
FMSTR_TSA_DIRECTORY("/text_files")     /* entering a new virtual directory */
```

(continues on next page)

(continued from previous page)

```
/* The readme.txt file will be accessible at the fmstr://text_files/readme.txt URL */
FMSTR_TSA_MEMFILE("readme.txt", readme_txt, sizeof(readme_txt)) /* memory-mapped file */

/* Files can also be specified with a full path so the DIRECTORY entry does not apply */
FMSTR_TSA_MEMFILE("/index.htm", index, sizeof(index))          /* memory-mapped file */
FMSTR_TSA_MEMFILE("/prj/demo.pmp", demo_pmp, sizeof(demo_pmp)) /* memory-mapped file */

/* Hyperlinks can point to a local MEMFILE object or to the Internet */
FMSTR_TSA_HREF("Board's Built-in Welcome Page", "/index.htm")
FMSTR_TSA_HREF("FreeMASTER Home Page", "http://www.nxp.com/freemaster")

/* Project file links simplify opening the projects from any URLs */
FMSTR_TSA_PROJECT("Demonstration Project (embedded)", "/prj/demo.pmp")
FMSTR_TSA_PROJECT("Full Project (online)", "http://mycompany.com/prj/demo.pmp")

FMSTR_TSA_TABLE_END()
```

TSA API

FMSTR_SetUpTsaBuff

Prototype

```
FMSTR_BOOL FMSTR_SetUpTsaBuff(FMSTR_ADDR buffAddr, FMSTR_SIZE buffSize);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_tsa.c*

Arguments

- *buffAddr* [in] - address of the memory buffer for the dynamic TSA table
- *buffSize* [in] - size of the memory buffer which determines the maximum number of TSA entries to be added in the runtime

Description This function must be used to assign the RAM memory buffer to the TSA subsystem when FMSTR_USE_TSA_DYNAMIC is enabled. The memory buffer is then used to store the TSA entries added dynamically to the runtime TSA table using the FMSTR_TsaAddVar function call. The runtime TSA table is processed by the FreeMASTER PC Host tool along with all static tables as soon as the communication port is open.

The size of the memory buffer determines the number of TSA entries that can be added dynamically. Depending on the MCU platform, one TSA entry takes either 8 or 16 bytes.

FMSTR_TsaAddVar

Prototype

```
FMSTR_BOOL FMSTR_TsaAddVar(FMSTR_TSATBL_STRPTR tsaName, FMSTR_TSATBL_STRPTR
↪tsaType,
    FMSTR_TSATBL_VOIDPTR varAddr, FMSTR_SIZE32 varSize,
    FMSTR_SIZE flags);
```

- Declaration: *freemaster.h*

- Implementation: *freemaster_tsa.c*

Arguments

- *tsaName* [in] - name of the object
- *tsaType* [in] - name of the object type
- *varAddr* [in] - address of the object
- *varSize* [in] - size of the object
- *flags* [in] - access flags; a combination of these values:
  - *FMSTR_TSA_INFO_RO_VAR* — read-only memory-mapped object (typically a variable)
  - *FMSTR_TSA_INFO_RW_VAR* — read/write memory-mapped object
  - *FMSTR_TSA_INFO_NON_VAR* — other entry, describing structure types, structure members, enumerations, and other types

Description This function can be called only when the dynamic TSA table is enabled by the FMSTR_USE_TSA_DYNAMIC configuration option and when the FMSTR_SetUpTsaBuff function call is made to assign the dynamic TSA table memory. This function adds an entry into the dynamic TSA table. It can be used to register a read-only or read/write memory object or describe an item of the user-defined type.

See *TSA table definition* for more details about the TSA table entries.

Application Commands API

FMSTR_GetAppCmd

Prototype

```
FMSTR_APPCMD_CODE FMSTR_GetAppCmd(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Description This function can be used to detect if there is an Application Command waiting to be processed by the application. If no command is pending, this function returns the FMSTR_APPCMDRESULT_NOCMD constant. Otherwise, this function returns the code of the Application Command that must be processed. Use the FMSTR_AppCmdAck call to acknowledge the Application Command after it is processed and to return the appropriate result code to the host.

The FMSTR_GetAppCmd function does not report the commands for which a callback handler function exists. If the FMSTR_GetAppCmd function is called when a callback-registered command is pending (and before it is actually processed by the callback function), this function returns FMSTR_APPCMDRESULT_NOCMD.

FMSTR_GetAppCmdData

Prototype

```
FMSTR_APPCMD_PDATA FMSTR_GetAppCmdData(FMSTR_SIZE* dataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *dataLen* [out] - pointer to the variable that receives the length of the data available in the buffer. It can be NULL when this information is not needed.

Description This function can be used to retrieve the Application Command data when the application determines that an Application Command is pending (see *FMSTR_GetAppCmd*).

There is just a single buffer to hold the Application Command data (the buffer length is FMSTR_APPCMD_BUFF_SIZE bytes). If the data are to be used in the application after the command is processed by the FMSTR_AppCmdAck call, copy the data out to a private buffer.

FMSTR_AppCmdAck

Prototype

```
void FMSTR_AppCmdAck(FMSTR_APPCMD_RESULT resultCode);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultCode* [in] - the result code which is to be returned to FreeMASTER

Description This function is used when the Application Command processing finishes in the application. The resultCode passed to this function is returned back to the host and the driver is re-initialized to expect the next Application Command.

After this function is called and before the next Application Command arrives, the return value of the FMSTR_GetAppCmd function is FMSTR_APPCMDRESULT_NOCMD.

FMSTR_AppCmdSetResponseData

Prototype

```
void FMSTR_AppCmdSetResponseData(FMSTR_ADDR 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
FMSTR_ADDR	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.
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.	
FMSTR_SIZE	Data type used to hold the memory block size.
It is required that this type is unsigned and at least 16 bits wide integer.	
FMSTR_BOOL	Data type used as a general boolean type.
This type is used only in zero/non-zero conditions in the driver code.	
FMSTR_APPCM	Data type used to hold the Application Command code.
Generally, this is an unsigned 8-bit value.	
FMSTR_APPCM	Data type used to create the Application Command data buffer.
Generally, this is an unsigned 8-bit value.	
FMSTR_APPCM	Data type used to hold the Application Command result code.
Generally, this is an unsigned 8-bit value.	

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.

FMSTR_TSA_TI	Data type used to hold a descriptor index in the TSA table or a table index in the list of TSA tables.
By default, this is defined as FMSTR_SIZE.	
FMSTR_TSA_TS	Data type used to hold a memory block size, as used in the TSA descriptors.
By default, this is defined as FMSTR_SIZE.	

Public Pipes types The table describes the data types used by the FreeMASTER Pipes API:

FMSTR_HPIPE	Pipe handle that identifies the open-pipe object.
Generally, this is a pointer to a void type.	
FMSTR_PIPE_P	Integer type required to hold at least 7 bits of data.
Generally, this is an unsigned 8-bit or 16-bit type.	
FMSTR_PIPE_SI	Integer type required to hold at least 16 bits of data.
This is used to store the data buffer sizes.	
FMSTR_PPIPEF	Pointer to the pipe handler function.
See *FMSTR_PipeOpen* for more details.	

Internal types The table describes the data types used internally by the FreeMASTER driver. The data types are declared in the platform-specific header file and they are not available in the application code.

FMSTR_U8	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.
FMSTR_U16	Unsigned 16-bit integer.
FMSTR_U32	Unsigned 32-bit integer.
FMSTR_S8	Signed 8-bit integer.
FMSTR_S16	Signed 16-bit integer.
FMSTR_S32	Signed 32-bit integer.
FMSTR_FLOAT	4-byte standard IEEE floating-point type.
FMSTR_FLAGS	Data type forming a union with a structure of flag bit-fields.
FMSTR_SIZE8	Data type holding a general size value, at least 8 bits wide.
FMSTR_INDEX	General for-loop index. Must be signed, at least 16 bits wide.
FMSTR_BCHR	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.
FMSTR_BPTR	A pointer to the communication buffer (an array of FMSTR_BCHR).

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
- FreeMASTER community area: community.nxp.com/community/freemaster
- FreeMASTER GitHub code repo: https://github.com/nxp-mcuxpresso/mcux-freemaster
- MCUXpresso SDK home: www.nxp.com/mcuxpresso
- MCUXpresso SDK builder: mcuxpresso.nxp.com/en

Documents

- *FreeMASTER Usage Serial Driver Implementation* (document AN4752)
- *Integrating FreeMASTER Time Debugging Tool With CodeWarrior For Microcontrollers v10.X Project* (document AN4771)
- *Flash Driver Library For MC56F847xx And MC56F827xx DSC Family* (document AN4860)

Revision history This Table summarizes the changes done to this document since the initial release.

Revision	Date	Description
1.0	03/2006	Limited initial release
2.0	09/2007	Updated for FreeMASTER version. New Freescale document template used.
2.1	12/2007	Added description of the new Fast Recorder feature and its API.
2.2	04/2010	Added support for MPC56xx platform, Added new API for use CAN interface.
2.3	04/2011	Added support for Kxx Kinetis platform and MQX operating system.
2.4	06/2011	Serial driver update, adds support for USB CDC interface.
2.5	08/2011	Added Packet Driven BDM interface.
2.7	12/2013	Added FLEXCAN32 interface, byte access and isr callback configuration option.
2.8	06/2014	Removed obsolete license text, see the software package content for up-to-date license.
2.9	03/2015	Update for driver version 1.8.2 and 1.9: FreeMASTER Pipes, TSA Active Content, LIN Transport Layer support, DEBUG-TX communication troubleshooting, Kinetis SDK support.
3.0	08/2016	Update for driver version 2.0: Added support for MPC56xx, MPC57xx, KEAxx and S32Kxx platforms. New NXP document template as well as new license agreement used. added MCAN interface. Folders structure at the installation destination was rearranged.
4.0	04/2019	Update for driver released as part of FreeMASTER v3.0 and MCUXpresso SDK 2.6. Updated to match new V4 serial communication protocol and new configuration options. This version of the document removes substantial portion of outdated information related to S08, S12, ColdFire, Power and other legacy platforms.
4.1	04/2020	Minor update for FreeMASTER driver included in MCUXpresso SDK 2.8.
4.2	09/2020	Added example applications description and information about the MCUXpresso Config Tools. Fixed the pipe-related API description.
4.3	10/2024	Added description of Network and Segger J-Link RTT interface configuration. Accompanying the MCUXpresso SDK version 24.12.00.
4.4	04/2025	Added Zephyr-specific information. Accompanying the MCUXpresso SDK version 25.06.00.

Chapter 4

RTOS

4.1 FreeRTOS

4.1.1 FreeRTOS kernel

Open source RTOS kernel for small devices.

FreeRTOS kernel for MCUXpresso SDK Readme

FreeRTOS kernel for MCUXpresso SDK ChangeLog

FreeRTOS kernel Readme

4.1.2 FreeRTOS drivers

This is set of NXP provided FreeRTOS reentrant bus drivers.

4.1.3 backoffalgorithm

Algorithm for calculating exponential backoff with jitter for network retry attempts.

Readme

4.1.4 corehttp

C language HTTP client library designed for embedded platforms.

4.1.5 corejson

JSON parser.

Readme

4.1.6 coremqtt

MQTT publish/subscribe messaging library.

4.1.7 coremqtt-agent

The coreMQTT Agent library is a high level API that adds thread safety to the coreMQTT library.

Readme

4.1.8 corepkcs11

PKCS #11 key management library.

Readme

4.1.9 freertos-plus-tcp

Open source RTOS FreeRTOS Plus TCP.

Readme