



MCUXpresso SDK Documentation

Release 25.12.00-pvw2



NXP
Nov 14, 2025



Table of contents

1	IMX95LPD5EVK-19	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	12
1.3.1	Getting Started with MCUXpresso SDK Repository	13
1.4	Release Notes	25
1.4.1	MCUXpresso SDK Release Notes	25
1.5	ChangeLog	29
1.5.1	MCUXpresso SDK Changelog	29
1.6	Driver API Reference Manual	127
1.7	Middleware Documentation	127
1.7.1	Multicore	127
1.7.2	FreeRTOS	127
1.7.3	lwIP	127
2	MIMX9596	129
2.1	CACHE: ARMV7-M7 CACHE Memory Controller	129
2.2	CAMERA MIX CSR: Camera Domain Block Control	132
2.3	Clock Driver	133
2.4	Csi2rx	133
2.5	Dpu	138
2.6	eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver	169
2.7	eDMA core Driver	200
2.8	eDMA soc Driver	207
2.9	EIM: error injection module	214
2.10	ERM: error recording module	214
2.11	FGPIO Driver	216
2.12	FlexCAN: Flex Controller Area Network Driver	216
2.13	FlexCAN Driver	216
2.14	FlexCAN eDMA Driver	263
2.15	FlexIO: FlexIO Driver	266
2.16	FlexIO Driver	266
2.17	FlexIO eDMA I2S Driver	283
2.18	FlexIO eDMA SPI Driver	287
2.19	FlexIO eDMA UART Driver	290
2.20	FlexIO I2C Master Driver	293
2.21	FlexIO I2S Driver	301
2.22	FlexIO SPI Driver	312
2.23	FlexIO UART Driver	325
2.24	FLEXSPI: Flexible Serial Peripheral Interface Driver	335
2.25	FLEXSPI eDMA Driver	352
2.26	I3C: I3C Driver	355
2.27	I3C Common Driver	357
2.28	I3C Master Driver	359
2.29	I3C Master DMA Driver	385

2.30	I3C Slave Driver	388
2.31	I3C Slave DMA Driver	401
2.32	INTM: Interrupt Monitor Driver	403
2.33	IOMUXC: IOMUX Controller	406
2.34	IRQSTEER: Interrupt Request Steering Driver	430
2.35	ISI: Image Sensing Interface	435
2.36	Common Driver	450
2.37	LDB: LVDS Display Bridge	462
2.38	Lin_lpuart_driver	464
2.39	LPI2C: Low Power Inter-Integrated Circuit Driver	472
2.40	LPI2C Master Driver	473
2.41	LPI2C Master DMA Driver	488
2.42	LPI2C Slave Driver	490
2.43	LPIT: Low-Power Interrupt Timer	500
2.44	LPSPi: Low Power Serial Peripheral Interface	507
2.45	LPSPi Peripheral driver	507
2.46	LPSPi eDMA Driver	529
2.47	LPTMR: Low-Power Timer	536
2.48	LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver	541
2.49	LPUART Driver	541
2.50	LPUART eDMA Driver	560
2.51	MCM: Miscellaneous Control Module	563
2.52	Mipi_dsi	567
2.53	MIPI_DSI: MIPI DSI Host Controller	584
2.54	MSGINTR: Message Unit	584
2.55	MSGINTR Driver	585
2.56	MU: Messaging Unit	585
2.57	NETC driver	597
2.58	Abbreviation in NETC driver	603
2.59	API layer	603
2.60	NETC Endpoint (EP) Driver	603
2.61	Endpoint (EP) Generic Configuration	603
2.62	Endpoint (EP) data path	608
2.63	Endpoint (EP) Interrupt Module	608
2.64	Endpoint (EP) Table Management Module	610
2.65	Endpoint (EP) PSI/VSI	610
2.66	Endpoint (EP) Ingress data path configuration	612
2.67	Endpoint (EP) Statistic Module	624
2.68	Endpoint (EP) Egress data path configuration	625
2.69	Endpoint (EP) Transmit/Receive	627
2.70	Hardware layer	632
2.71	Hardware Common Functions	640
2.72	Hardware ENETC	641
2.73	Hardware Port	647
2.74	Hardware Port MAC	659
2.75	Hardware Port Rx	666
2.76	Hardware Port Tx	666
2.77	Hardware Station Interface(SI)	668
2.78	Hardware Switch	685
2.79	Hardware Table Access Functions	691
2.80	NETC MDIO Driver	771
2.81	MDIO initialization module	771
2.82	MDIO PHY status module	772
2.83	MDIO write/read module	773
2.84	NETC Timer Driver	774
2.85	Timer adjustment module	774
2.86	Timer initialization module	775
2.87	Local time synchronization module	777

2.88	OTFAD: On The Fly AES-128 Decryption Driver	782
2.89	PDM: Microphone Interface	785
2.90	PDM Driver	785
2.91	PDM EDMA Driver	803
2.92	RGPIO: Rapid General-Purpose Input/Output Driver	807
2.93	RGPIO Driver	809
2.94	SAI: Serial Audio Interface	812
2.95	SAI Driver	812
2.96	SAI EDMA Driver	839
2.97	SAR_ADC: SAR_ADC Module	845
2.98	SEMA42: Hardware Semaphores Driver	873
2.99	SFA: Signal Frequency Analyser	876
2.100	TEMPSENSOR: Temperature Sensor Module	885
2.101	TPM: Timer PWM Module	894
2.102	TRDC: Trusted Resource Domain Controller	911
2.103	Trdc_core	931
2.104	Trdc_soc	945
2.105	TRGMUX: Trigger Mux Driver	948
2.106	TSTMR: Timestamp Timer Driver	949
2.107	CACHE: CACHE Memory Controller	950
3	Middleware	955
4	RTOS	957
4.1	FreeRTOS	957
4.1.1	FreeRTOS kernel	957
4.1.2	FreeRTOS drivers	957
4.1.3	backoffalgorithm	957
4.1.4	corehttp	957
4.1.5	corejson	957
4.1.6	coremqtt	958
4.1.7	corepkcs11	958
4.1.8	freertos-plus-tcp	958

This documentation contains information specific to the imx95lpd5evk19 board.

Chapter 1

IMX95LPD5EVK-19

1.1 Overview



MCU device and part on board is shown below:

- Device: MIMX9596
- PartNumber: MIMX9596AVZXN

1.2 Getting Started with MCUXpresso SDK Package

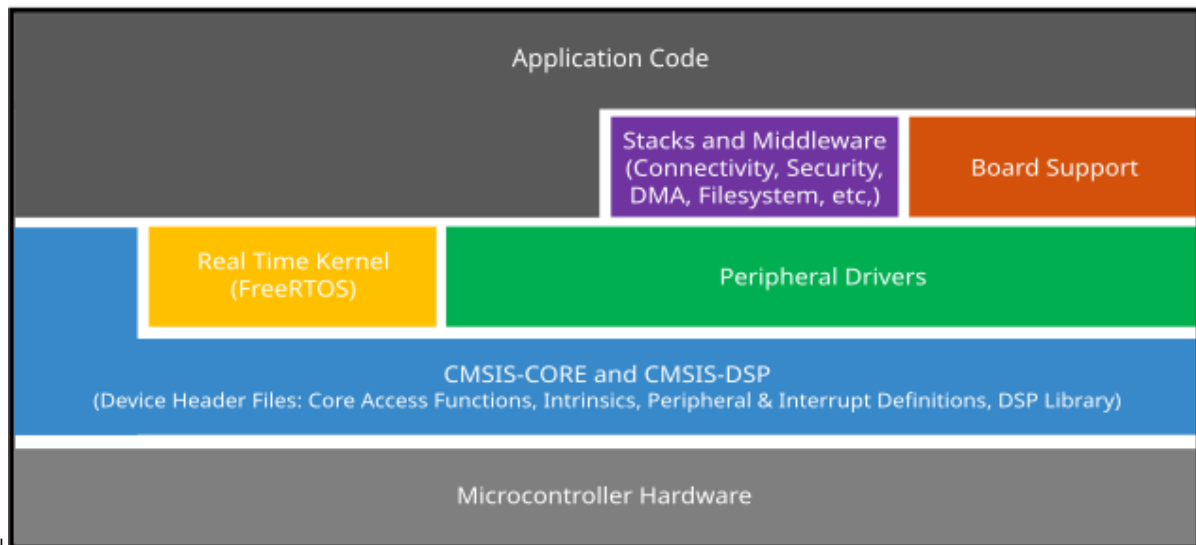
1.2.1 Getting Started with Package

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 IMX95LPD5EVK-19 (document MCUXSDKIMX95LPD5EVK19RN)*.

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



MCUXpresso SDK board support folders

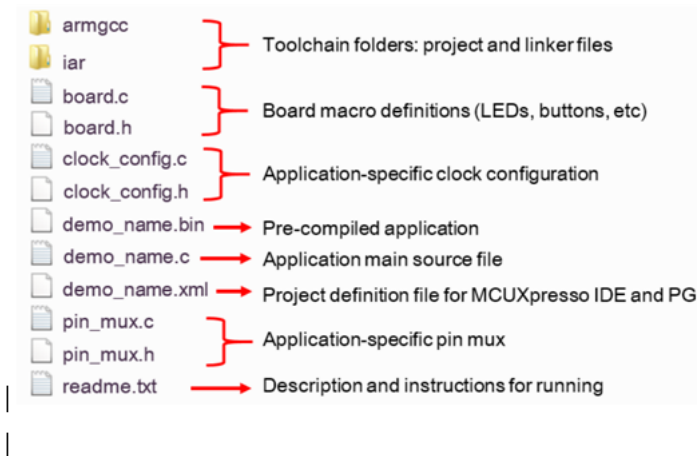
MCUXpresso SDK board support provides example applications for NXP development and evaluation boards for Arm Cortex-M cores. Board support packages are found inside of the top level boards folder, and each supported board has its own folder (MCUXpresso SDK package can support multiple boards). Within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

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

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

Each `<board_name>` folder in the boards directory contains a comprehensive set of examples that are relevant to that specific piece of hardware. Although we use the `hello_world_sm` 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_sm` application folder you see the following contents:



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

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

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

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

- devices/<device_name>: The device's CMSIS header file, MCUXpresso SDK feature file and a few other files
- devices/<device_name>/cmsis_drivers: All the CMSIS drivers for your specific MCU
- devices/<device_name>/drivers: All of the peripheral drivers for your specific MCU
- devices/<device_name>/<tool_name>: Toolchain-specific startup code, including vector table definitions
- devices/<device_name>/utilities: Items such as the debug console that are used by many of the example applications
- devices/<device_name>/project_template: 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 95 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 95 release supports building and debugging with the toolchains listed in Table 1.

The user can choose the appropriate one for development.

For supported toolchain versions, see MCUXpresso SDK Release Notes for for IMX95LPD5EVK-19, IMX95LP4XEVK-15, and IMX95VERDINEVK (*document: MCUXSDKIMX95SERIESRN*).

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

Com- piler/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 95, the patch for IAR and Segger J-Link must be installed. To get the patch, download it from [the NXP website](#) or contact your local field applications engineer (FAE) or the sales representative.

Parent topic: [Toolchain introduction](#)

Build a demo application using Arm GCC

This section describes the steps to configure the command-line Arm GCC tools to build, run, and debug demo applications. Additionally, this section lists the necessary driver libraries provided in the MCUXpresso SDK. The `hello_world_sm` demo application targeted for the IMX95 series hardware platform is used as an example, though these steps can be applied to any board, demo, or example application in the MCUXpresso SDK.

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

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

Install GCC Arm embedded toolchain Download and run the installer from the GNU Arm Embedded Toolchain Downloads page. The GNU Arm embedded toolchain contains the GCC compiler, libraries, and other tools required for bare-metal software development. The GCC toolchain should correspond to the latest supported version, as described in the MCUXpresso SDK Release Notes for IMX95 Series (document MCUXSDKIMX95SERIESRN).

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 toolchain installation path. For this example, the path is: \$ export ARMGCC_DIR=<path_to_GNUARM_GCC_installation_dir>.

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 example, the exact path is: <install_dir>/boards/imx95lpd5evk19/demo_apps/hello_world_sm/armgcc.
2. Run the build_debug.sh script at the command-line to perform the build. The output is shown as below:

```
$ ./build_debug.sh
-- TOOLCHAIN_DIR:
-- BUILD_TYPE: debug
-- TOOLCHAIN_DIR:
-- BUILD_TYPE: debug
-- The ASM compiler identification is GNU
-- Found assembler:
-- Configuring done
-- Generating done
-- Build files have been written to:
Scanning dependencies of target hello_world.elf
< -- skipping lines -- >
[100%] Linking C executable debug/hello_world.elf
[100%] Built target hello_world.elf
```

Note: build_debug/release.sh are ram target, build_ddr_debug/release.sh are ddr target (only imx95lpd5evk19 hello_world_sm support ddr target this release, the same as iar).

Parent topic:Linux OS host

Parent topic:[Build a demo application 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 toolchain Download and run the installer from the GNU Arm Embedded Toolchain Downloads page. The GNU Arm embedded toolchain contains the GCC compiler, libraries, and other tools required for bare-metal software development. The GCC toolchain should correspond to the latest supported version, as described in MCUXpresso SDK Release Notes for IMX95 Series(document MCUXSDKIMX95SERIESRN).

Parent topic:Set up toolchain

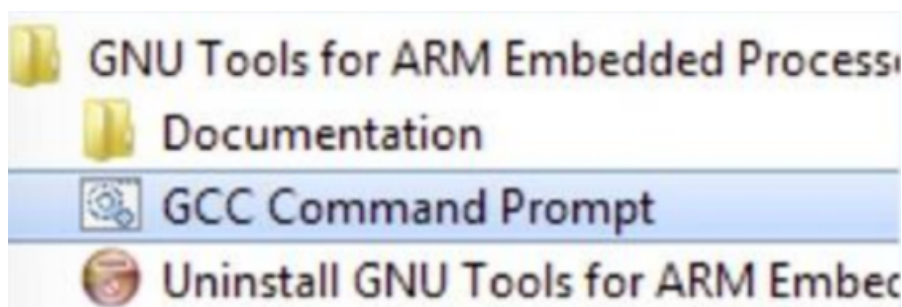
Add a new system environment variable for ARMGCC_DIR Create a new system environment variable and name it ARMGCC_DIR. The value of this variable should point to the Arm GCC embedded toolchain installation path. For this example, the path is: C:\Program Files (x86)\GNU Arm Embedded Toolchain\9 2020-q2-update. Reference the installation folder of the GNU Arm GCC embedded tools for the exact path.

Parent topic:Set up toolchain

Parent topic:Windows OS host

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

1. Open the GCC Arm embedded toolchain command window. To launch the window on the Windows operating system, select **Start-> Programs-> GNU Tools ARM Embedded <version>-> 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/imx95lpd5evk19/demo_apps/hello_world/armgcc.
3. Type build_debug.bat at the command-line or double-click the build_debug.bat file in Windows Explorer to perform the build. The output is as shown in Figure 2.

```
[100%] Linking C executable debug/hello_world_sm_cm7.elf
Memory region      Used Size  Region Size  %age Used
  m_interrupts:      1612 B        2 KB       78.71%
    m_text:          18060 B       254 KB        6.94%
m_m33_suspend_ram:    0 GB         8 KB         0.00%
m_a55_suspend_ram:    0 GB         8 KB         0.00%
    m_data:          4072 B       104 KB        3.82%
    m_ncache:         0 GB        64 KB         0.00%
    m_reserved:       0 GB        56 KB         0.00%
    m_rsc_tbl:        0 GB        16 KB         0.00%
m_rsc_tbl_in_dram:    0 GB        16 KB         0.00%
[100%] Built target hello_world_sm_cm7.elf
```

Parent topic:Windows OS host

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

Build a demo application with IAR

This section describes the steps to run the example applications provided in the MCUXpresso SDK. The demo application targeted for the i.MX 95 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 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.

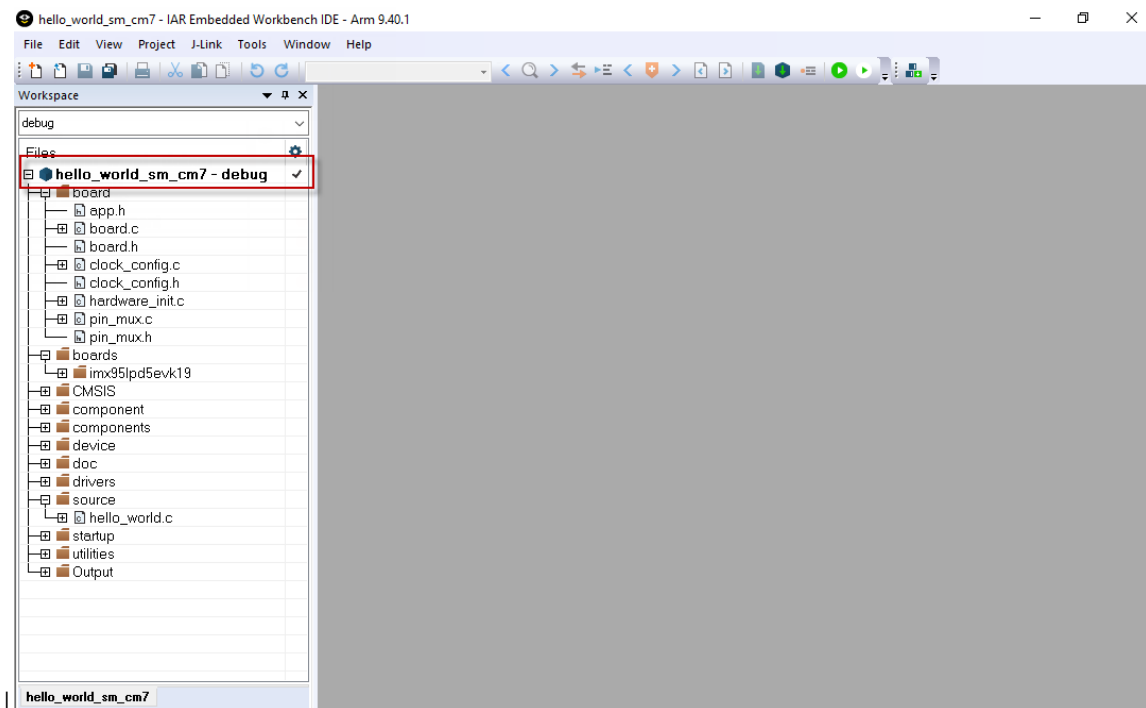
1. If not already done, open the desired demo application workspace. Most example application workspace files can be located using the following path:

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

Using the i.MX 95 LPD EVK board as an example, the workspace is located in:

```
<install_dir>/boards/imx95lpd5evk19/demo_apps/hello_world/cm7/hello_world_cm7.eww
```

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



4. The build completes without errors.

Parent topic: [Build a demo application with IAR](#)

Run a demo application

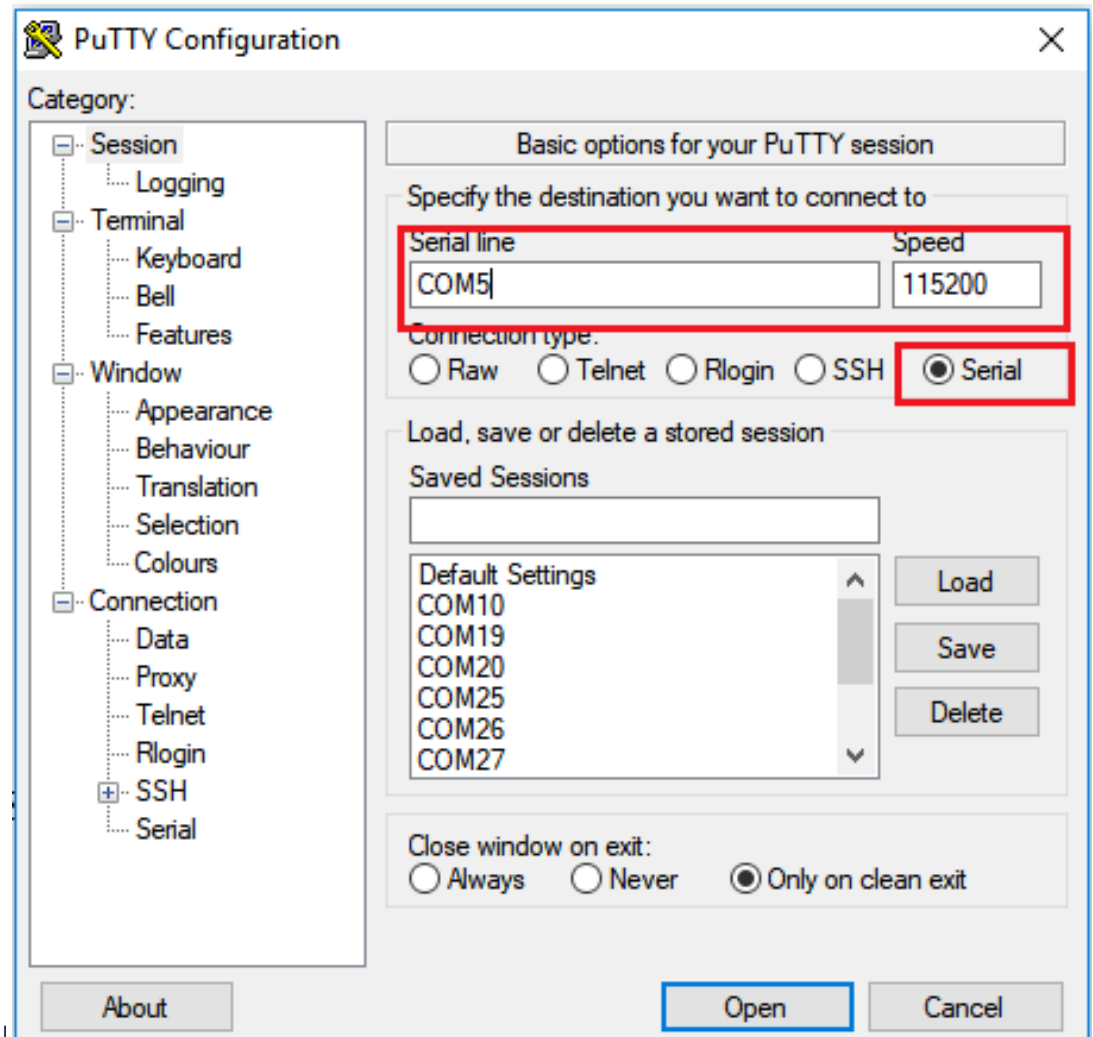
This section describes the steps to download the flash.bin to sd and emmc, run the example applications provided in the MCUXpresso SDK. The `hello_world_sm` demo application targeted for the i.MX 95 hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

Connect LPUART on board to console on PC To download and run the application, perform these steps:

1. Connect the development platform to your PC via USB cable between the DBG USB connector (J31) and the PC. It provides console output.

2. 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

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

Make a flash.bin

1. Get the boot images and the imx-mkimage source repository from corresponding Linux BSP release. The boot images required to be put into imx-mkimage/i.MX95 are:
 - oei-m33-tcm.bin
 - oei-m33-ddr.bin
 - m33_image.bin (m33_image-mx95alt.bin/m33_image-mx95evk.bin/m33_image-mx95netc. bin)
 - lpddr5_dmem_qb_v202311.bin (lpddr4x_dmem_qb_v202311.bin for IMX95LP4XEVK-15)
 - lpddr5_dmem_v202311.bin (lpddr4x_dmem_v202311.bin for IMX95LP4XEVK-15)
 - lpddr5_imem_qb_v202311.bin (lpddr4x_imem_qb_v202311.bin for IMX95LP4XEVK-15)

- lpddr5_imem_v202311.bin (lpddr4x_imem_v202311.bin for IMX95LP4XEVK-15)
- u-boot.bin
- u-boot-spl.bin
- bl31.bin
- tee.bin
- mx95a0-ahab-container.img

Note:

- **mx95evk** for `m33_image.bin` is used for `rpmsg str echo`, `rpmsg ping pong` and `power_mode_switch_rtos`.
- **mx95netc** for `m33_image.bin` is used for `netc_share_sm`.
- **mx95alt** for `m33_image.bin` is used for almost other examples.

2. Copy binary built by ARMGCC/IAR into `imx-mkimage/i.MX95`, and rename it to `m7_image.bin`.

3. make image for ram target.

1. Make `flash.bin` with `imx-mkimage`.

```
make SOC=iMX95 OEI=YES flash_all LPDDR_TYPE=lpddr5 (boot A core and M7)
```

or

```
make SOC=iMX95 OEI=YES flash_lpboot_sm_m7 LPDDR_TYPE=lpddr5 (does not  
boot A core, just boot M7)
```

2. Make image for ddr target:

```
make SOC=i.MX95 OEI=YES flash_m7_ddr LPDDR_TYPE=lpddr5
```

```
make SOC=iMX95 OEI=YES flash_lpboot_sm_m7 LPDDR_TYPE=lpddr5
```

Note: For IMX95LP4XEVK-15, `LPDDR_TYPE=lpddr4x`.

4. Burn `flash.bin` to MicroSD/eMMC at 32 K(0x8000) offset with `dd` or `HxD` or `UUU` and then plug the MicroSD card to the board.

For example:

- Burn `flash.bin` to Micro SD card with `dd`

```
dd if=flash.bin of=/dev/sdh bs=1k seek=32 && sync
```

- Burn `flash.bin` to SD/eMMC with `UUU`

1. Connect USB Type-C port to PC through the USB cable. It is used for downloading firmware of the board.

2. Switch to serial downloader mode; boot core is cortex-m33. `sd: uuu -b sd imx-boot-imx95-19x19-lpddr5-evk-sd.bin-flash_all new-flash.bin`

3. Burn `flash.bin` with `uuu`.

```
emmc: uuu -b emmc imx-boot-imx95-19x19-lpddr5-evk-sd.bin-flash_all new-flash.bin
```

Note:

- `imx-boot-imx95-19x19-lpddr5-evk-sd.bin-flash_all` (`imx-boot-imx95-15x15-lpddr4x-evk-sd.bin-flash_all` for IMX95LP4XEVK-15 and `imx-boot-imx95-19x19-verdin-sd.bin-flash_all` for `imx95verdinevk`). Get it from `linux bsp`.
- `new-flash.bin`. Generate it yourself.

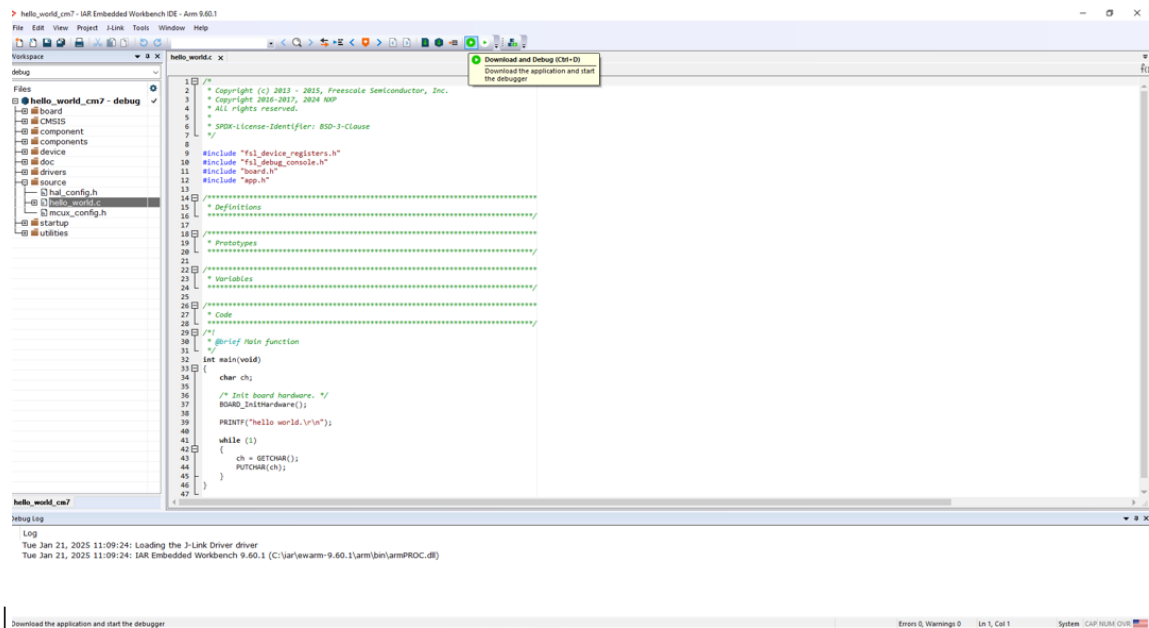
5. Change the boot mode to `SW7[1:4] = 1011` for sd boot, `SW7[1:4] = 1010` for emmc boot.

6. Power on the board .

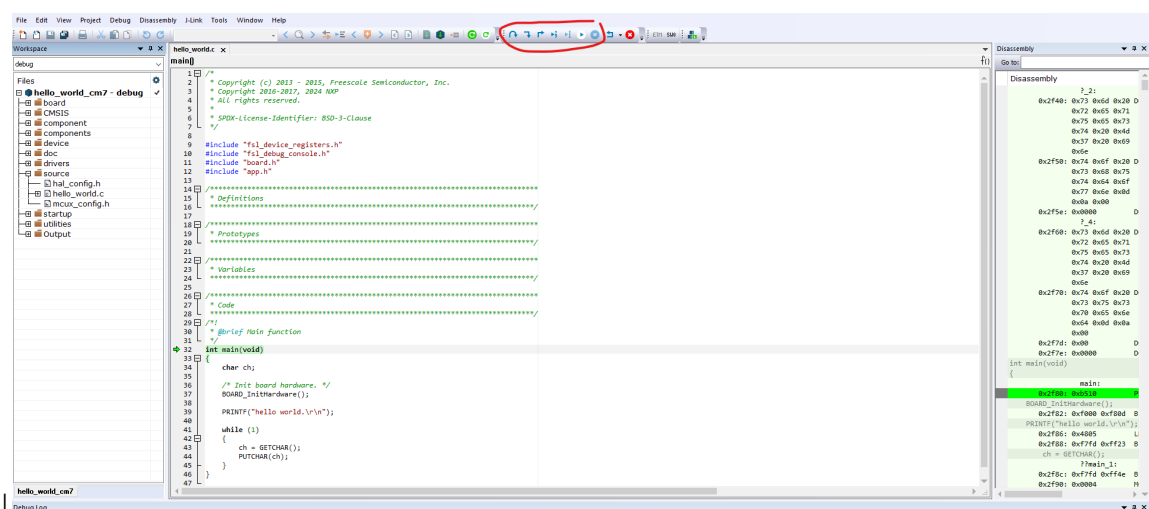
Parent topic: [Run a demo application](#)

Debug an demo application with IAR To debug an application with IAR, perform these steps:

1. download the patch for IAR and Segger J-Link from [the NXP website](#), install it as readme in the patch.
2. burn a flash.bin to board, see [Run a demo application](#), make sure the size of M7 binary in flash.bin not less than the binary you want to debug.
3. power on the board, make sure the M7 is booted by SM
4. Open hello world example with IAR, click **download and debug** to debug



5. click button such as **step over** to debug



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

1.3 Getting Started with MCUXpresso SDK GitHub

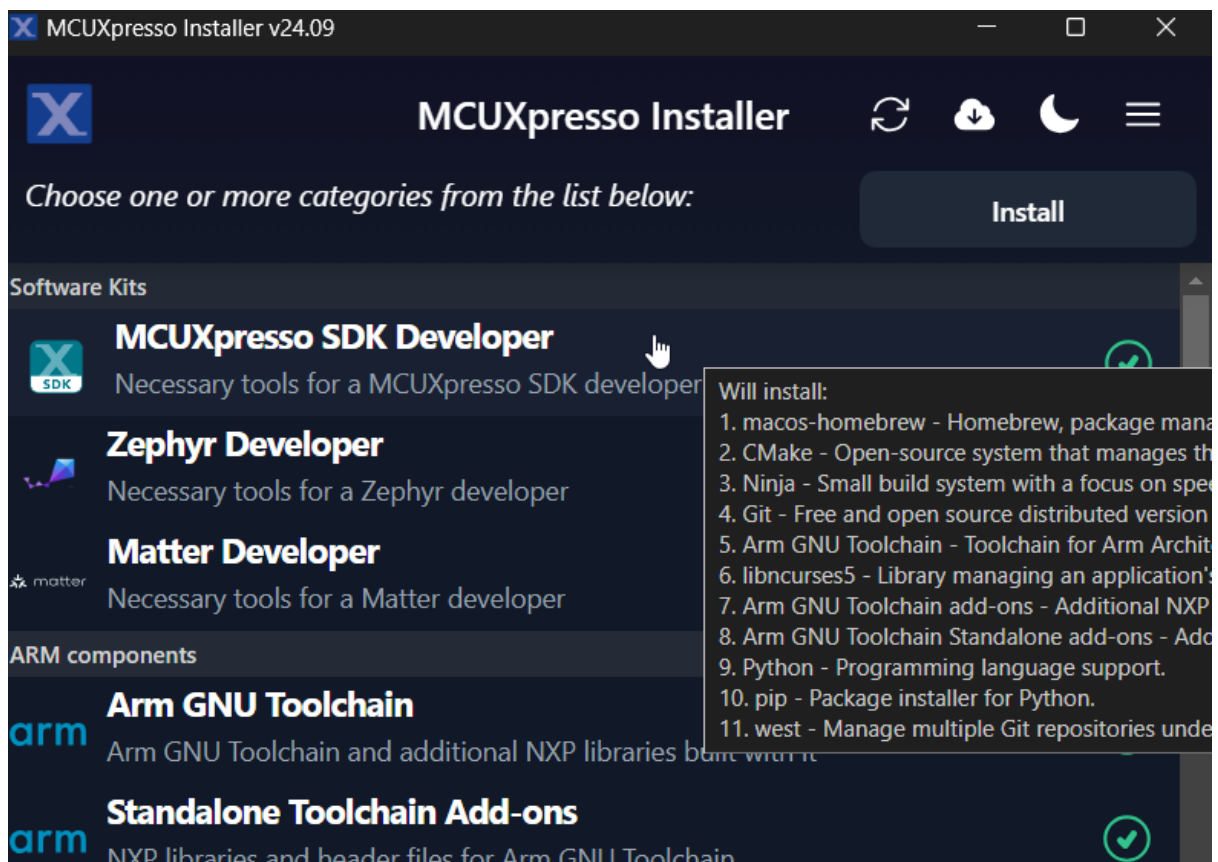
1.3.1 Getting Started with MCUXpresso SDK Repository

Installation

NOTE

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

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



Alternative: Manual Installation

Basic tools

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

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

Then configure your username and email using the commands:

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

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

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

West Please use the west version equal or greater than 1.2.0

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

Build And Configuration System

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

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

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

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

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

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

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

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

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

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

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

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

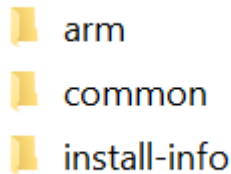
Toolchain MCUXpresso SDK supports all mainstream toolchains for embedded development. You can install your used or interested toolchains following the guides.

Toolchain	Download and Installation Guide	Note
Armgcc	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	ARM-CLANG_DIR	C:\ArmCompilerforEmbedded6.22 for Windows/opt/ArmCompilerforEmbedded6.21 for Linux	– toolchain mdk
Zephyr	ZEPHYR_DIR	c:\NXP\zephyr-sdk-<version> for windows/opt/zephyr-sdk-<version> for Linux	– toolchain zephyr
CodeWarrior	CW_DIR	C:\Freescall\CW MCU v11.2 for windowsCodeWarrior is not supported with Linux	– toolchain code-warrior
Xtensa	XCC_DIR	C:\xtensa\XtDevTools\install\tools\RI-2023.11-win32\XtensaTools for windows/opt/xtensa/XtDevTools/install/tools/RI-2023.11-Linux/XtensaTools for Linux	– toolchain xtensa
NXP S32Compiler RISC-V Zen-V	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 riscv-llvm

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



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

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

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

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

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

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

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

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

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

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

- Linux:
 1. Open the \$HOME/.bashrc file using a text editor, such as vim.
 2. Go to the end of the file.
 3. Add the line which appends the tool installation path to the PATH variable and export PATH at the end of the file. For example, export PATH="/Directory1:\$PATH".
 4. Save and exit.

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

Get MCUXpresso SDK Repo

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

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

# Update the west projects
cd mcuxpresso-sdk
west update

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

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

1. Navigate to the core directory:

```
cd mcuxsdk
```

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

```
python -m venv .venv

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

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

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

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

3. Install the required Python packages:

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

Explore Contents

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

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

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

All the projects record in the [Manifest repo](#) are checked out to the folder mcuxsdk/, the layout of mcuxsdk folder is shown as below:

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

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

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

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

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

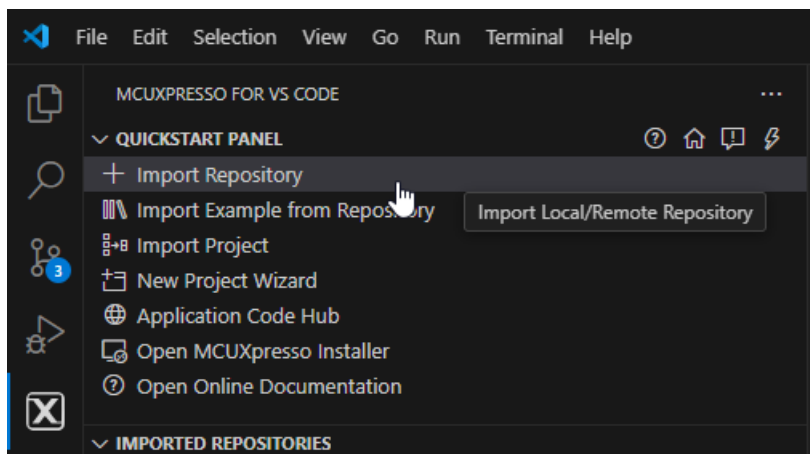
Run a demo using MCUXpresso for VS Code

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

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

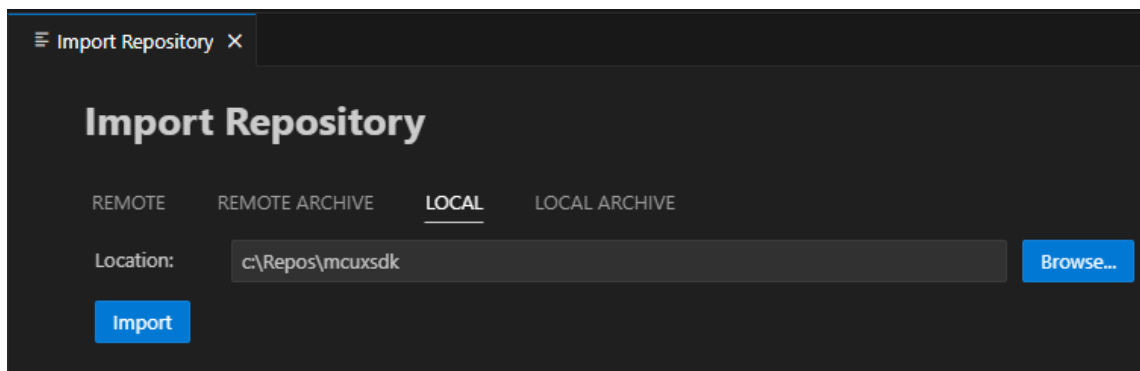
To build an example application:

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

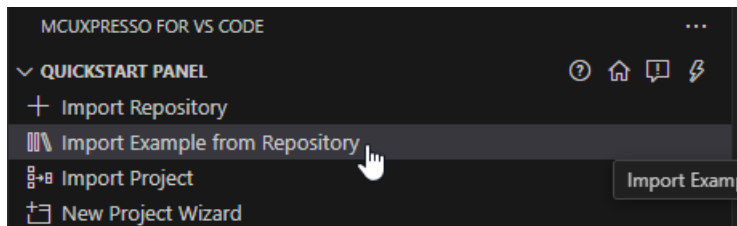


Note: You can import the SDK in several ways. Refer to [MCUXpresso for VS Code Wiki](#) for details.

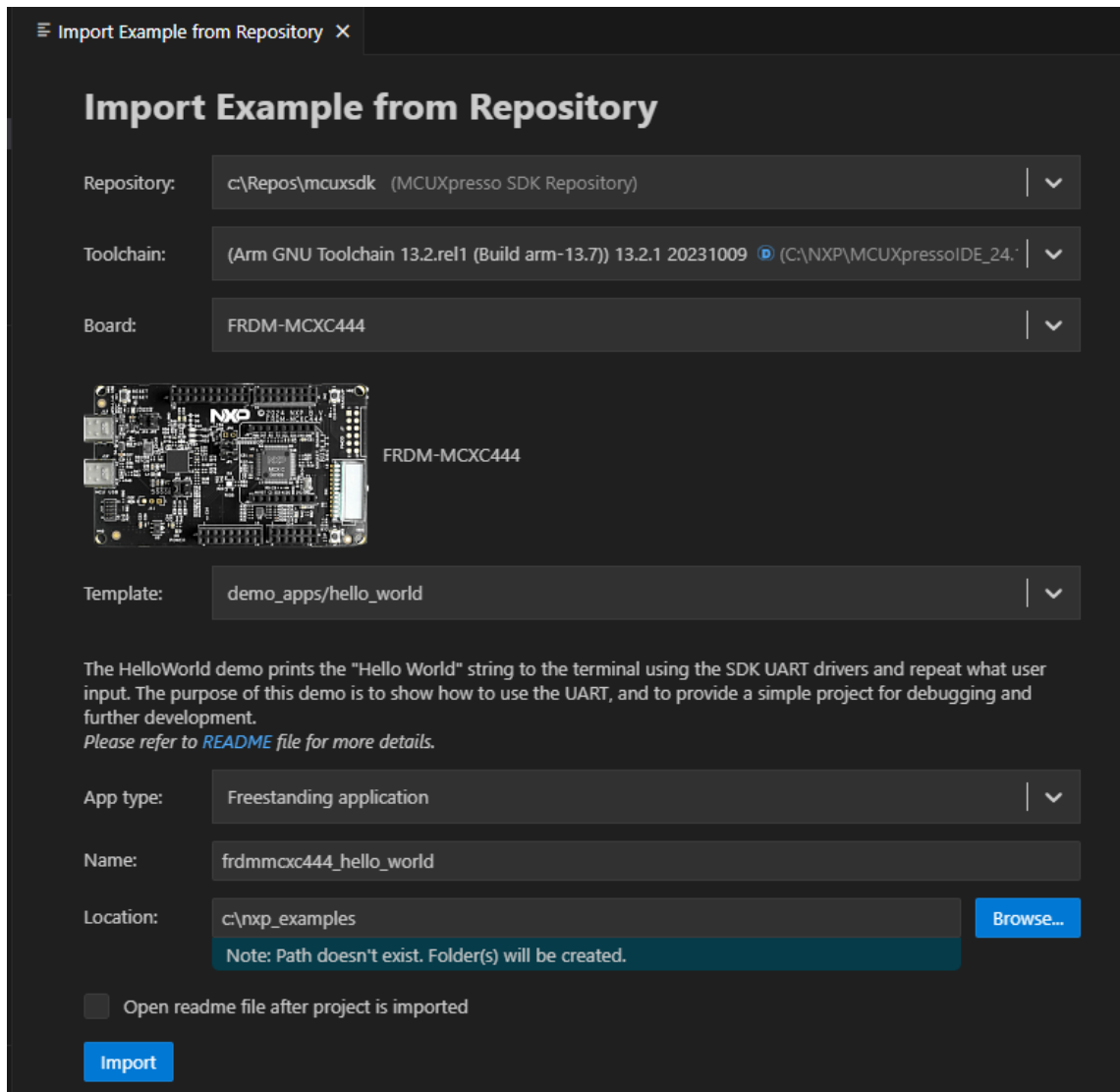
Select **Local** if you've already obtained the SDK as seen in [Get MCUXpresso SDK Repo](#). Select your location and click **Import**.



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

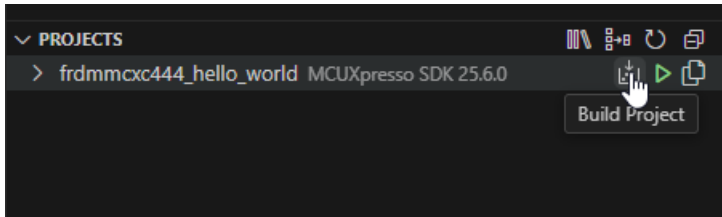


In the dropdown menu, select the MCUXpresso SDK, the Arm GNU Toolchain, your board, template, and application type. Click **Import**.

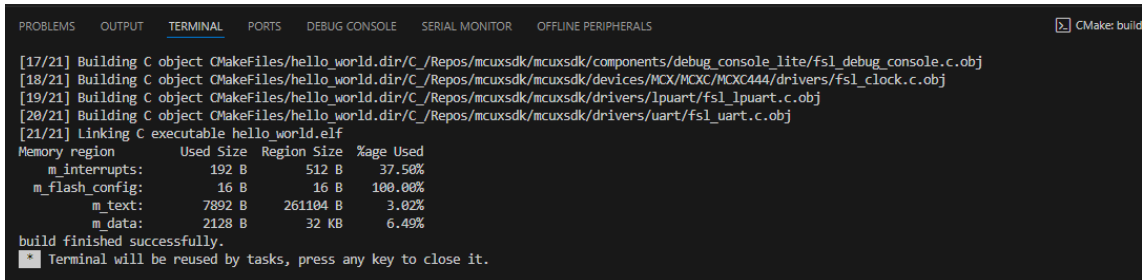


Note: The MCUXpresso SDK projects can be imported as **Repository applications** or **Free-standing applications**. The difference between the two is the import location. Projects imported as Repository examples will be located inside the MCUXpresso SDK, whereas Free-standing examples can be imported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

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

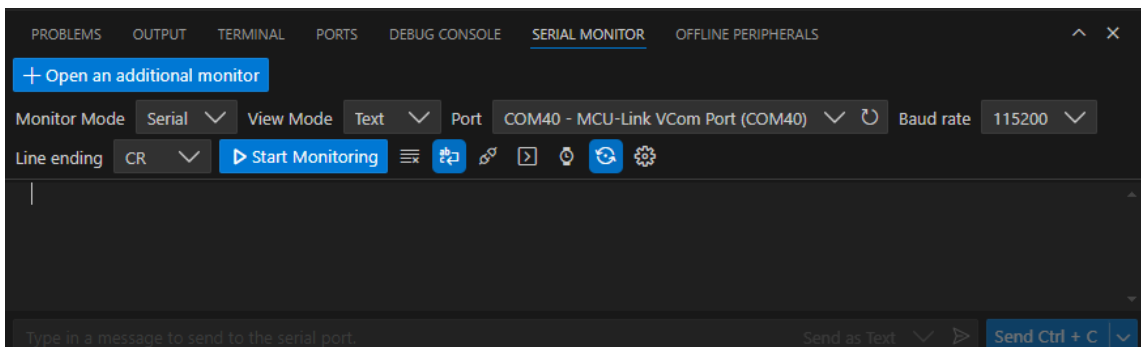


The integrated terminal will open at the bottom and will display the build output.

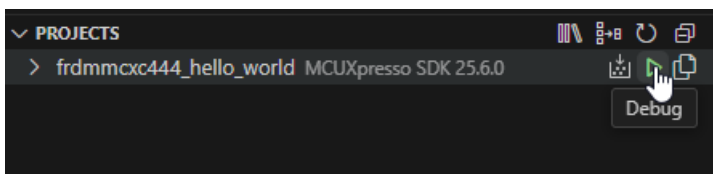


Run an example application **Note:** for full details on MCUXpresso for VS Code debug probe support, see [MCUXpresso for VS Code Wiki](#).

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



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



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

```

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

```

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

```

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

```

Running a demo using ARMGCC CLI/IAR/MDK

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

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

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

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

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

(continues on next page)

(continued from previous page)

```

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

```

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

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

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

Here are some typical usages for generating a SDK example:

```

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

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

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

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

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

```

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

```

west build -b evkbnmimxrt1170 examples/demo_apps/hello_world --toolchain iar -Dcore_id=cm7 --config_
↪ flexspi_nor_debug

```

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

```

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

```

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

```

west build -b evkbnmimxrt1170 --sysbuild ./examples/multicore_examples/hello_world/primary -Dcore_
↪ id=cm7 --config flexspi_nor_debug --toolchain=armgcc -p always

```

For more details, please refer to System build.

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

1. Run cmake configuration

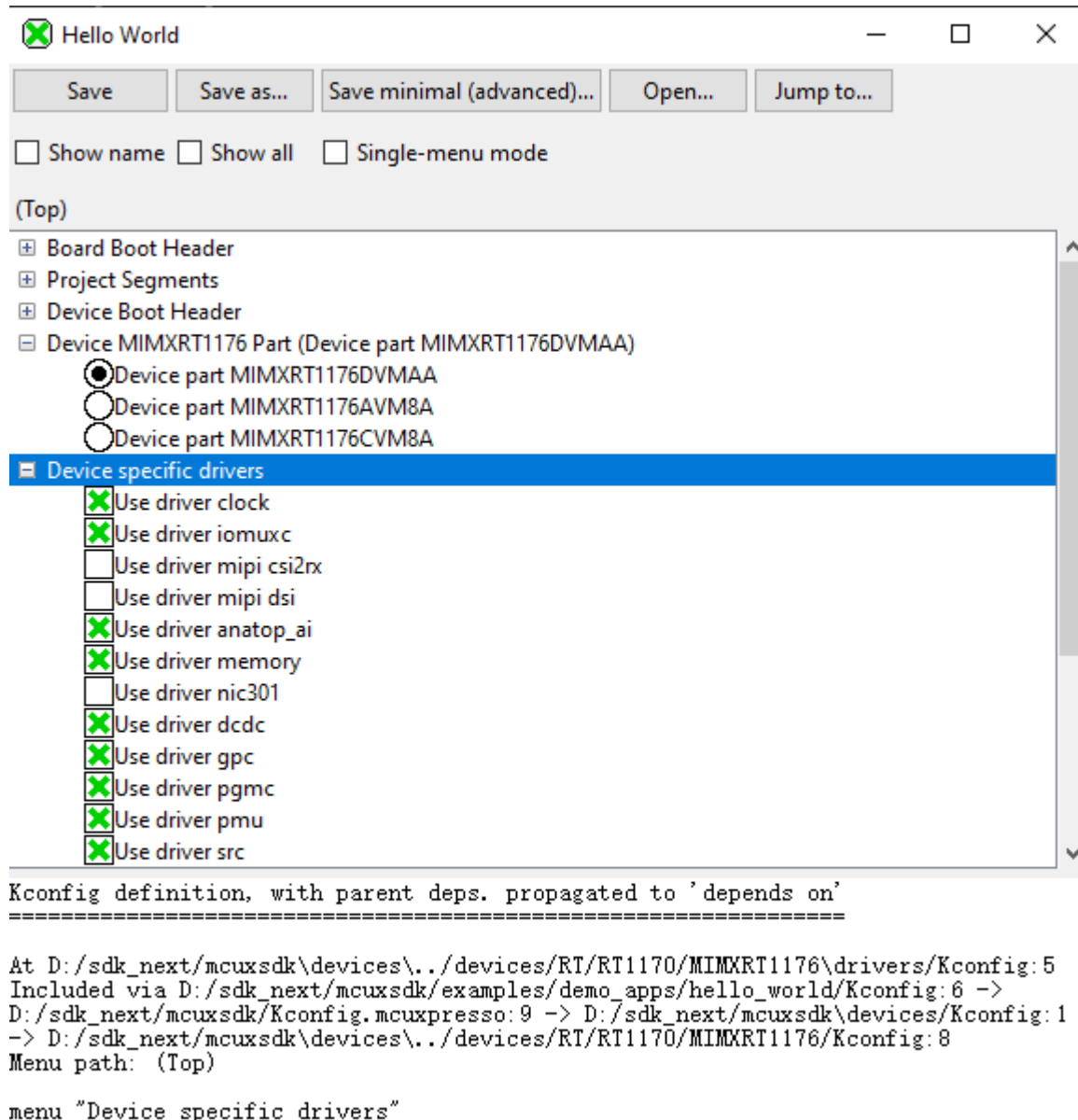
```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

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

2. Run guiconfig target

```
west build -t guiconfig
```

Then you will get the Kconfig GUI launched, like



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

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

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

Flash the hello_world example:

```
west flash -r linkserver
```

Debug Start a gdb interface by following command:

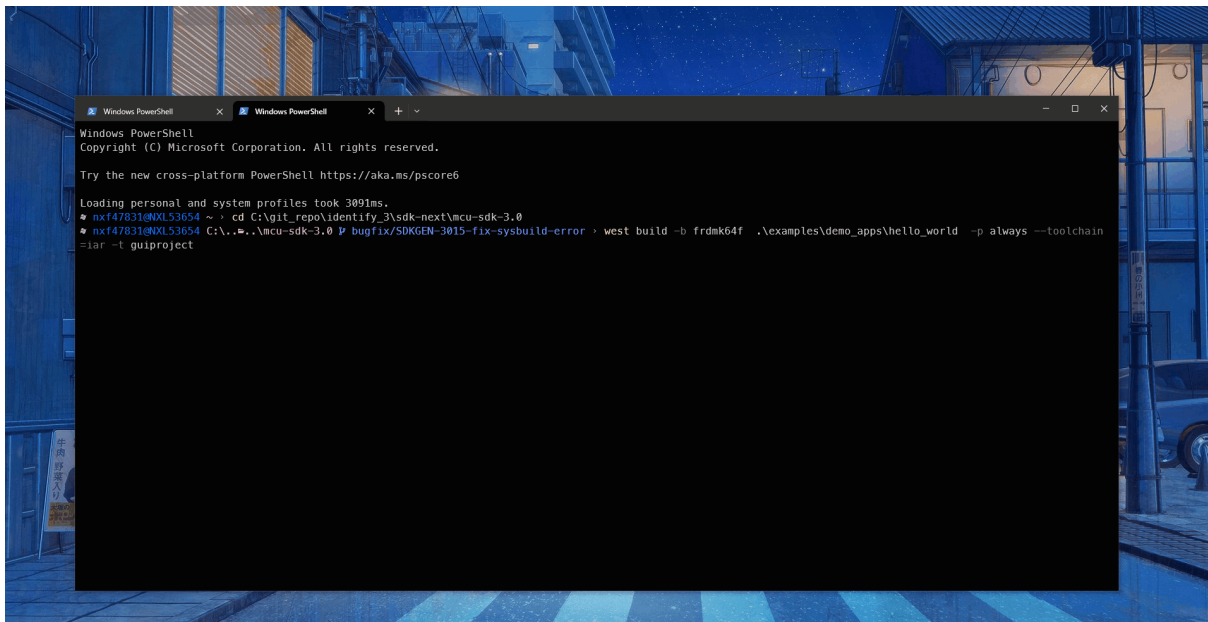
```
west debug -r linkserver
```

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

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

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

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



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

1.4 Release Notes

1.4.1 MCUXpresso SDK Release Notes

Overview

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

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

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

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

MCUXpresso SDK

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

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

Development tools

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

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

Supported development system

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

De-velop-ment boards	MCU devices			
IMX95L19	MIMX9596AVTXN, MIMX9596CVYXN, MIMX9596DVZXN, MIMX9596DVYXQ	MIMX9596AVYXN, MIMX9596CVZXN, MIMX9596XVTXN,	MIMX9596AVZXN , MIMX9596DVTXN, MIMX9596XVYXN,	MIMX9596CVTXN, MIMX9596DVYXN, MIMX9596XVZXN,

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).

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

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

TinyCBOR Concise Binary Object Representation (CBOR) Library

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

Multicore Multicore Software Development Kit

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

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

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

llhttp HTTP parser llhttp

Release contents

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

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

Known Issues

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

SEGGER J-Link debugger usage problem

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

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

1. Configure switch SW5 to M core boot; low-power boot. Ensure that there is no image on the boot source.
2. Power the board and start the debugger for use.
3. To restart the debugger, stop the debugger, power off the board, and repeat step #2.

1.5 ChangeLog

1.5.1 MCUXpresso SDK Changelog

Board Support Files

board

[25.06.00]

- Initial version

clock_config

[25.06.00]

- Initial version

pin_mux

[25.06.00]

- Initial version
-

CACHE 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.
-

CAMERA_CSR

[2.0.1]

- Update to adapter head file
 - There are some updates to the header file. Update it to adapt to this change.

[2.0.0]

- Initial version.
-
-

COMMON

[2.6.1]

- Improvements
 - Support Cortex M23.

[2.6.0]

- Bug Fixes
 - Fix CERT-C violations.

[2.5.0]

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

[2.4.3]

- Improvements
 - Enable 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.
-

DPU

[2.3.1]

- Bug Fixes
 - Fixed CERT-C violations.

[2.3.0]

- New feature.
 - Supported the SEERIS MDR7.

[2.2.0]

- New feature.
 - Supported the display stream 1.

[2.1.0]

- New feature.
 - Supported the dpu warp function.

[2.0.0]

- Initial version.
-

DWC MIPI CSI2RX

[2.0.0]

- Initial version.
-

EDMA

[2.10.7]

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

[2.10.6]

- Improvements
 - Add macro `FSL_FEATURE_EDMA_HAS_EDMA_TCD_CLOCK_ENABLE` to enable tcd clocks in `EDMA_Init` function.

[2.10.5]

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

[2.10.4]

- Improvements
 - Add new MP register macros to ensure compatibility with different devices.
 - Add macro `DMA_CHANNEL_ARRAY_STEPn` to adapt to complex addressing of edma tcd registers.

[2.10.3]

- Bug Fixes
 - Clear interrupt status flags in `EDMA_CreateHandle` to avoid triggering interrupt by mistake.

[2.10.2]

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

[2.10.1]

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

[2.10.0]

- Improvements
 - Modify the structures `edma_core_mp_t`, `edma_core_channel_t`, `edma_core_tcd_t` to adapt to edma5.
 - Add TCD register macro to facilitate confirmation of tcd type.
 - Modify the mask macro to a fixed value.
 - Add `EDMA_TCD_TYPE` macro to determine edma tcd type.
 - Add extension API to the following API to determine edma tcd type.
 - * `EDMA_ConfigChannelSoftwareTCD` -> `EDMA_ConfigChannelSoftwareTCDExt`
 - * `EDMA_TcdReset` -> `EDMA_TcdResetExt`
 - * `EDMA_TcdSetTransferConfig` -> `EDMA_TcdSetTransferConfigExt`

- * EDMA_TcdSetMinorOffsetConfig -> EDMA_TcdSetMinorOffsetConfigExt
- * EDMA_TcdSetChannelLink -> EDMA_TcdSetChannelLinkExt
- * EDMA_TcdSetBandWidth -> EDMA_TcdSetBandWidthExt
- * EDMA_TcdSetModulo -> EDMA_TcdSetModuloExt
- * EDMA_TcdEnableAutoStopRequest -> EDMA_TcdEnableAutoStopRequestExt
- * EDMA_TcdEnableInterrupts -> EDMA_TcdEnableInterruptsExt
- * EDMA_TcdDisableInterrupts -> EDMA_TcdDisableInterruptsExt
- * EDMA_TcdSetMajorOffsetConfig -> EDMA_TcdSetMajorOffsetConfigExt

[2.9.2]

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

[2.9.1]

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

[2.9.0]

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

[2.8.1]

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

[2.8.0]

- Improvements
 - Added feature FSL_FEATURE_EDMA_HAS_NO_CH_SBR_SEC to separate DMA without SEC bitfield.

[2.7.1]

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

[2.7.0]

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

[2.6.0]

- Improvements
 - Modify the type of parameter channelRequestSource from dma_request_source_t to int32_t in the EDMA_SetChannelMux.

[2.5.3]

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

[2.5.2]

- Improvements
 - Applied ERRATA 51327.

[2.5.1]

- Bug Fixes
 - Fixed the EDMA_ResetChannel function cannot reset channel DONE/ERROR status.

[2.5.0]

- Improvements
 - Added feature FSL_FEATURE_EDMA_HAS_NO_SBR_ATTR_BIT to separate DMA without ATTR bitfield.
 - Added api EDMA_GetChannelSystemBusInformation to gets the channel identification and attribute information on the system bus interface.
- Bug Fixes
 - Fixed the ESG bit not set in scatter gather mode issue.
 - Fixed the DREQ bit configuration missed in single transfer issue.
 - Cleared the interrupt status before invoke callback to avoid miss interrupt issue.
 - Removed disableRequestAfterMajorLoopComplete from edma_transfer_config_t structure as driver will handle it.
 - Fixed the channel mux configuration not compatible issue.
 - Fixed the out of bound access in function EDMA_DriverIRQHandler.

[2.4.4]

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

[2.4.3]

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

[2.4.2]

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

[2.4.1]

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

[2.4.0]

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

[2.3.2]

- Improvements
 - Fixed HIS ccm issue in function EDMA_PrepareTransferConfig.
 - Fixed violations of MISRA C-2012 rule 11.6, 10.7, 10.3, 18.1.
- Bug Fixes

- Added ACTIVE & BITER & CITER bitfields to determine the channel status to fixed the issue of the transfer request cannot submit by function EDMA_SubmitTransfer when channel is idle.

[2.3.1]

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

[2.3.0]

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

[2.2.0]

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

[2.1.9]

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

[2.1.8]

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

[2.1.7]

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

[2.1.6]

- Bug Fixes
 - Fixed KW3X MISRA Issue.
 - * Rule 14.4, 10.8, 10.4, 10.7, 10.1, 10.3, 13.5, and 13.2.
- Improvements
 - Cleared the IRQ handler unavailable for specific platform with macro `FSL_FEATURE_EDMA_MODULE_CHANNEL_IRQ_ENTRY_SHARED_OFFSET`.

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

- Improvements
 - Added documentation for eDMA data flow when scatter/gather is implemented for the `EDMA_HandleIRQ` API.
 - Updated and corrected some related comments in the `EDMA_HandleIRQ` API and `edma_handle_t` struct.

[2.1.0]

- Improvements
 - Changed the `EDMA_GetRemainingBytes` API into `EDMA_GetRemainingMajorLoopCount` due to eDMA IP limitation (see API comments/note for further details).

[2.0.5]

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

[2.0.4]

- Improvements
 - Added support for SoCs with multiple eDMA instances.
 - Added pubweak DriverIRQHandler for KL28T DMA1 and MCIMX7U5_M4.

[2.0.3]

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

[2.0.2]

- Bug Fixes
 - Fixed incorrect minorLoopBytes type definition in `_edma_transfer_config` struct, and defined `minorLoopBytes` as `uint32_t` instead of `uint16_t`.

[2.0.1]

- Bug Fixes
 - Fixed the eDMA callback issue (which did not check valid status) in `EDMA_HandleIRQ` API.

[2.0.0]

- Initial version.
-

FLEXCAN**[2.14.5]**

- Improvements
 - Make API `FLEXCAN_GetFDMailboxOffset` public.
 - Add API `FLEXCAN_SetMbID` and `FLEXCAN_SetFDMbID` to configure Message Buffer ID individually.
- Bug Fixes
 - Fixed violations of the CERT INT30-C INT31-C.
 - Fixed violations of the CERT ARR30-C.

[2.14.4]

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

[2.14.3]

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

[2.14.2]

- Improvements
 - Add Coverage Justification for uncovered code.
 - Adjust API FLEXCAN_TransferAbortReceive order.
 - Update FLEXCAN_Enable to enter Freeze Mode first when enter Disable mode on some platform.
 - Added while loop timeout for following API:
 - * FLEXCAN_EnterFreezeMode
 - * FLEXCAN_ExitFreezeMode
 - * FLEXCAN_Enable
 - * FLEXCAN_Reset
 - * FLEXCAN_TransferSendBlocking
 - * FLEXCAN_TransferReceiveBlocking
 - * FLEXCAN_TransferFDSendBlocking
 - * FLEXCAN_TransferFDReceiveBlocking
 - * FLEXCAN_TransferReceiveFifoBlocking
 - * FLEXCAN_TransferReceiveEnhancedFifoBlocking
- Bug Fixes
 - Remove remote frame feature in CANFD mode because there is no remote frame in the CANFD format.
 - Remove legacy Rx FIFO disabled branch in FLEXCAN_SubHandlerForLegacyRxFIFO and FLEXCAN_SubHandlerForDataTransferred.

[2.14.1]

- Bug Fixes
 - Fixed register IMASK2-4 IFLAG2-4 HR_TIME_STAMP_n access issue on FlexCAN instances with different number of MBs.
 - Fixed bit field MBDSR1-3 access issue on FlexCAN instances with different number of MBs.
- Improvements
 - Unified following API as same parameter and return value type:
 - * FLEXCAN_GetMbStatusFlags
 - * FLEXCAN_ClearMbStatusFlags
 - * FLEXCAN_EnableMbInterrupts
 - * FLEXCAN_DisableMbInterrupts
 - Add workaround for ERR050443 and ERR052403.
 - Update message buffer read process in API FLEXCAN_ReadRxMb and FLEXCAN_ReadFDRxMb to make critical section as short as possible.
 - Simplify API FLEXCAN_DriverDataIRQHandler implementation by remove parameter type.

[2.14.0]

- Improvements
 - Support external time tick feature.
 - Support high resolution timestamp feature.
 - Enter Freeze Mode first when enter Disable Mode on some platform.
 - Add feature macro for Pretended Networking because some FlexCAN instance do not have this feature.
 - Add feature macro for enhanced Rx FIFO because some FlexCAN instance do not have this feature.
 - Add new FlexCAN IRQ Handler FLEXCAN_DriverDataIRQHandler and FLEXCAN_DriverEventIRQHandler. These IRQ Handlers are used on soc which FlexCAN interrupts are grouped by specific function and assigned to different vector.
 - Update macro FLEXCAN_WAKE_UP_FLAG and FLEXCAN_PNWAKE_UP_FLAG to simplify code.
 - Replace macro FSL_FEATURE_FLEXCAN_HAS_NO_WAKMSK_SUPPORT with FSL_FEATURE_FLEXCAN_HAS_NO_SLFWAK_SUPPORT.
 - Replace macro FSL_FEATURE_FLEXCAN_HAS_NO_WAKSRC_SUPPORT with FSL_FEATURE_FLEXCAN_HAS_GLITCH_FILTER.
- Bug Fixes
 - Fixed wrong interrupt and status flag helper macro in enumeration flexcan_flags and API FLEXCAN_DisableInterrupts.
 - Fixed interrupt flag helper macro typo issue.
 - Remove flags which will be unassociated with interrupt in macro FLEXCAN_MEMORY_ERROR_INT_FLAG.
 - Remove flags which will be unassociated with interrupt in macro FLEXCAN_ERROR_AND_STATUS_INT_FLAG.

- Fixed array out-of-bounds access when read enhanced Rx FIFO.

[2.13.1]

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

[2.13.0]

- Improvements
 - Support payload endianness selection feature.

[2.12.0]

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

[2.11.8]

- Improvements
 - Synchronize flexcan driver update on s32z platform.

[2.11.7]

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

[2.11.6]

- Bug Fixes
 - Fixed ERRATA_9595 FLEXCAN_EnterFreezeMode() may result to bus fault on some platform.

[2.11.5]

- Bug Fixes
 - Fixed flexcan_memset() crash under high optimization compilation.

[2.11.4]

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

[2.11.3]

- Bug Fixes
 - Fixed FLEXCAN_TransferReceiveEnhancedFifoEDMA() compile error with DMA3.

[2.11.2]

- Bug Fixes
 - Fixed bug that timestamp in flexcan_handle_t not updated when RX overflow happens.

[2.11.1]

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

[2.11.0]

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

[2.10.1]

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

[2.10.0]

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

[2.9.2]

- Bug Fixes
 - Fixed the issue that FLEXCAN_CheckUnhandleInterruptEvents() can't detecting the exist enhanced RX FIFO interrupt status.
 - Fixed the issue that FLEXCAN_ReadPNWakeUpMB() does not return fail even no existing valid wake-up frame.
 - Fixed the issue that FLEXCAN_ReadEnhancedRxFifo() may clear bits other than the data available bit.
 - Fixed violations of the MISRA C-2012 rules 10.4, 10.8.
- Improvements

- Return `kStatus_FLEXCAN_RxFifoDisabled` instead of `kStatus_Fail` when read FIFO fail during IRQ handler.
- Remove unreachable code from timing calculates APIs.
- Update Enhanced Rx FIFO handler to make it deal with underflow/overflow status first.

[2.9.1]

- Bug Fixes
 - Fixed the issue that `FLEXCAN_TransferReceiveEnhancedFifoBlocking()` API clearing Fifo data available flag more than once.
 - Fixed the issue that entering `FLEXCAN_SubHandlerForEnhancedRxFifo()` even if Enhanced Rx fifo interrupts are not enabled.
 - Fixed the issue that `FLEXCAN_TransferReceiveEnhancedFifoEDMA()` update handle even if previous Rx FIFO receive not finished.
 - Fixed the issue that `FLEXCAN_SetEnhancedRxFifoConfig()` not configure the `ER_FCR[NFE]` bits to the correct value.
 - Fixed the issue that `FLEXCAN_ReceiveFifoEDMACallback()` can't differentiate between Rx fifo and enhanced rx fifo.
 - Fixed the issue that `FLEXCAN_TransferHandleIRQ()` can't report Legacy Rx FIFO warning status.

[2.9.0]

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

[2.8.7]

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

[2.8.6]

- Bug Fixes
- Fix Coverity overrun issues in `fsl_flexcan_edma` driver.

[2.8.5]

- Improvements
 - Make driver aarch64 compatible.

[2.8.4]

- Bug Fixes
 - Fixed FlexCan_Errata_6032 to disable all interrupts.

[2.8.3]

- Bug Fixes
 - Fixed an issue with the FLEXCAN_EnableInterrupts and FLEXCAN_DisableInterrupts interrupt enable bits in the CTRL1 register.

[2.8.2]

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

[2.8.1]

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

[2.8.0]

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

[2.7.1]

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

[2.7.0]

- Improvements
 - Update config configuration.
 - * Added enablePretendedNetworking member to support enable/disable Pretended Networking feature.
 - * Added enableTransceiverDelayMeasure member to support enable/disable Transceiver Delay Measurement Pretended feature.
 - * Added bitRate/bitRateFD member to work as baudRate/baudRateFD member union.
 - Rename all “baud” in code or comments to “bit” to align with the CAN spec.

- Added Pretended Networking mode related APIs.
 - * FLEXCAN_SetPNConfig
 - * FLEXCAN_GetPNMatchCount
 - * FLEXCAN_ReadPNWakeUpMB
- Added support for Enhanced Rx FIFO.
- Removed independent memory error interrupt/status APIs and put all interrupt/status control operation into FLEXCAN_EnableInterrupts/FLEXCAN_DisableInterrupts and FLEXCAN_GetStatusFlags/FLEXCAN_ClearStatusFlags APIs.
- Update improved timing APIs to make it calculate improved timing according to CiA doc recommended.
 - * FLEXCAN_CalculateImprovedTimingValues.
 - * FLEXCAN_FDCalculateImprovedTimingValues.
- Update FLEXCAN_SetBitRate/FLEXCAN_SetFDBitRate to added the use of enhanced timing registers.

[2.6.2]

- Improvements
 - Add CANFD frame data length enumeration.

[2.6.1]

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

[2.6.0]

- Improvements
 - Enable CANFD ISO mode in FLEXCAN_FDInit API.
 - Enable the transceiver delay compensation feature when enable FD operation and set bitrate switch.
 - Implementation memory error control in FLEXCAN_Init API.
 - Improve FLEXCAN_FDCalculateImprovedTimingValues API to get same value for FPRESDIV and PRES DIV.
 - Added memory error configuration for user.
 - * enableMemoryErrorControl
 - * enableNonCorrectableErrorEnterFreeze
 - Added memory error related APIs.
 - * FLEXCAN_GetMemoryErrorReportStatus
 - * FLEXCAN_GetMemoryErrorStatusFlags
 - * FLEXCAN_ClearMemoryErrorStatusFlags
 - * FLEXCAN_EnableMemoryErrorInterrupts
 - * FLEXCAN_DisableMemoryErrorInterrupts
- Bug Fixes

- Fixed the issue of sent duff CAN frame after call FLEXCAN_FDInit() API.

[2.5.2]

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

[2.5.1]

- Bug Fixes
 - Fixed the non-divisible case in improved timing APIs.
 - * FLEXCAN_CalculateImprovedTimingValues
 - * FLEXCAN_FDCalculateImprovedTimingValues

[2.5.0]

- Bug Fixes
 - MISRA C-2012 issue check.
 - * Fixed rules, containing: rule-10.1, rule-10.3, rule-10.4, rule-10.7, rule-10.8, rule-11.8, rule-12.2, rule-13.4, rule-14.4, rule-15.5, rule-15.6, rule-15.7, rule-16.4, rule-17.3, rule-5.8, rule-8.3, rule-8.5.
 - Fixed the issue that API FLEXCAN_SetFDRxMbConfig lacks inactive message buff.
 - Fixed the issue of Pa082 warning.
 - Fixed the issue of dead lock in the function of interruption handler.
 - Fixed the issue of Legacy Rx Fifo EDMA transfer data fail in evkmimxrt1060 and evk-mimxrt1064.
 - Fixed the issue of setting CANFD Bit Rate Switch.
 - Fixed the issue of operating unknown pointer risk.
 - * when used the pointer "handle->mbFrameBuf[mbIdx]" to update the timestamp in a short-live TX frame, the frame pointer became as unknown, the action of operating it would result in program stack destroyed.
 - Added assert to check current CAN clock source affected by other clock gates in current device.
 - * In some chips, CAN clock sources could be selected by CCM. But for some clock sources affected by other clock gates, if user insisted on using that clock source, they had to open these gates at the same time. However, they should take into consideration the power consumption issue at system level. In RT10xx chips, CAN clock source 2 was affected by the clock gate of lpuart1. ERRATA ID: (ERR050235 in CCM).

- Improvements
 - Implementation for new FLEXCAN with ECC feature able to exit Freeze mode.
 - Optimized the function of interruption handler.
 - Added two APIs for FLEXCAN EDMA driver.
 - * FLEXCAN_PrepareTransfConfiguration
 - * FLEXCAN_StartTransferDatafromRxFIFO
 - Added new API for FLEXCAN driver.
 - * FLEXCAN_GetTimeStamp
 - For TX non-blocking API, we wrote the frame into mailbox only, so no need to register TX frame address to the pointer, and the timestamp could be updated into the new global variable handle->timestamp[mbIdx], the FLEXCAN driver provided a new API for user to get it by handle and index number after TX DONE Success.
 - * FLEXCAN_EnterFreezeMode
 - * FLEXCAN_ExitFreezeMode
 - Added new configuration for user.
 - * disableSelfReception
 - * enableListenOnlyMode
 - Renamed the two clock source enum macros based on CLKSRC bit field value directly.
 - * The CLKSRC bit value had no property about Oscillator or Peripheral type in lots of devices, it acted as two different clock input source only, but the legacy enum macros name contained such property, that misled user to select incorrect CAN clock source.
 - Created two new enum macros for the FLEXCAN driver.
 - * kFLEXCAN_ClkSrc0
 - * kFLEXCAN_ClkSrc1
 - Deprecated two legacy enum macros for the FLEXCAN driver.
 - * kFLEXCAN_ClkSrcOsc
 - * kFLEXCAN_ClkSrcPeri
 - Changed the process flow for Remote request frame response..
 - * Created a new enum macro for the FLEXCAN driver.
 - kStatus_FLEXCAN_RxRemote
 - Changed the process flow for kFLEXCAN_StateRxRemote state in the interrupt handler.
 - * Should the TX frame not register to the pointer of frame handle, interrupt handler would not be able to read the remote response frame from the mail box to ram, so user should read the frame by manual from mail box after a complete remote frame transfer.

[2.4.0]

- Bug Fixes
 - MISRA C-2012 issue check.

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

[2.3.2]

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

[2.3.1]

- Bug Fixes
 - Added correct handle when kStatus_FLEXCAN_TxSwitchToRx is 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.
-

FLEXCAN_EDMA

[2.12.1]

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

[2.12.0]

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

[2.11.7]

- Refer FLEXCAN driver change log 2.7.0 to 2.11.7
-

FLEXIO**[2.3.0]**

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

- Improvements
 - Added doxygen index parameter comment in FLEXIO_SetClockMode.

[2.2.0]

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

[2.1.0]

- Improvements
 - Added API FLEXIO_SetClockMode to set flexio channel counter and source clock.

[2.0.4]

- Bug Fixes
 - Fixed MISRA 8.4 issues.

[2.0.3]

- Bug Fixes
 - Fixed MISRA 10.4 issues.

[2.0.2]

- Improvements
 - Split FLEXIO component which combines all flexio/flexio_uart/flexio_i2c/flexio_i2s drivers into several components: FlexIO component, flexio_uart component, flexio_i2c_master component, and flexio_i2s component.
- Bug Fixes
 - Fixed MISRA issues
 - * Fixed rules 10.1, 10.3, 10.4, 10.7, 11.6, 11.9, 14.4, 17.7.

[2.0.1]

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

FLEXIO_I2C

[2.6.2]

- Improvements
 - Added timeout for while loop in FLEXIO_I2C_MasterTransferBlocking().
- Bug Fixes
 - Fixed build issues related to I2C_RETRY_TIMES.

[2.6.1]

- Bug Fixes
 - Fixed coverity issues

[2.6.0]

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

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.0]

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

[2.2.0]

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

[2.1.7]

- Bug Fixes
 - Fixed the issue that FLEXIO_I2C_MasterTransferBlocking did not wait for STOP bit sent.
 - Fixed COVERITY issue of useless call in FLEXIO_I2C_MasterTransferRunStateMachine.
 - Fixed the issue that I2C master did not check whether bus was busy before transfer.

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

- Improvements
 - Changed the prototype of FLEXIO_I2C_MasterInit to return kStatus_Success if initialized successfully or to return kStatus_InvalidArgument if “(srcClock_Hz / masterConfig->baudRate_Bps) / 2 - 1” exceeds 0xFFU.

[2.1.2]

- Bug Fixes
 - Fixed the FLEXIO I2C issue where the master could not receive data from I2C slave in high baudrate.
 - Fixed the FLEXIO I2C issue where the master could not receive NAK when master sent non-existent addr.
 - Fixed the FLEXIO I2C issue where the master could not get transfer count successfully.
 - Fixed the FLEXIO I2C issue where the master could not receive data successfully when sending data first.
 - Fixed the Dozen mode configuration error in FLEXIO_I2C_MasterInit API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.
 - Fixed the issue that FLEXIO_I2C_MasterTransferBlocking API called FLEXIO_I2C_MasterTransferCreateHandle, which lead to the s_flexioHandle/s_flexioIsr/s_flexioType variable being written. Then, if calling FLEXIO_I2C_MasterTransferBlocking API multiple times, the s_flexioHandle/s_flexioIsr/s_flexioType variable would not be written any more due to it being out of range. This lead to the following situation: NonBlocking transfer APIs could not work due to the fail of register IRQ.

[2.1.1]

- Bug Fixes
 - Implemented the FLEXIO_I2C_MasterTransferBlocking API which is defined in header file but has no implementation in the C file.

[2.1.0]

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

FLEXIO_I2S**[2.2.2]**

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

[2.2.1]

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

[2.2.0]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

FLEXIO_I2S_EDMA

[2.1.9]

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

[2.1.8]

- Improvements
 - Applied EDMA ERRATA 51327.
-

FLEXIO_SPI

[2.4.3]

- Improvements
 - Make SPI_RETRY_TIMES configurable by CONFIG_SPI_RETRY_TIMES.

[2.4.2]

- Bug Fixes
 - Fixed FLEXIO_SPI_MasterTransferBlocking and FLEXIO_SPI_MasterTransferNonBlocking issue in CS continuous mode, the CS might not be continuous.

[2.4.1]

- Bug Fixes
 - Fixed coverity issues

[2.4.0]

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

[2.3.5]

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

[2.3.4]

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

[2.3.3]

- Bugfixes
 - Fixed cs-continuous mode.

[2.3.2]

- Improvements
 - Changed FLEXIO_SPI_DUMMYDATA to 0x00.

[2.3.1]

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

[2.3.0]

- New Features
 - Supported FLEXIO_SPI slave transfer with continuous master CS signal and CPHA=0.
 - Supported FLEXIO_SPI master transfer with continuous CS signal.
 - Support 32 bit transfer width.
- Bug Fixes

- Fixed wrong timer compare configuration for dma/edma transfer.
- Fixed wrong byte order of rx data if transfer width is 16 bit, since the we use shifter buffer bit swapped/byte swapped register to read in received data, so the high byte should be read from the high bits of the register when MSB.

[2.2.1]

- Bug Fixes
 - Fixed bug in FLEXIO_SPI_MasterTransferAbortEDMA that when aborting EDMA transfer EDMA_AbortTransfer should be used rather than EDMA_StopTransfer.

[2.2.0]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

- Bug Fixes
 - Fixed bug where FLEXIO SPI transfer data is in 16 bit per frame mode with eDMA.
 - Fixed bug when FLEXIO SPI works in eDMA and interrupt mode with 16-bit per frame and Lsbfirst.
 - Fixed the Dozen mode configuration error in FLEXIO_SPI_MasterInit/FLEXIO_SPI_SlaveInit API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.

- Improvements
 - Added `#ifndef/#endif` to allow users to change the default TX value at compile time.

[2.1.0]

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

FLEXIO_UART

[2.6.4]

- Improvements
 - Make `UART_RETRY_TIMES` configurable by `CONFIG_UART_RETRY_TIMES`.

[2.6.3]

- Bug Fixes
 - Fixed coverity issues

[2.6.2]

- Bug Fixes
 - Fixed coverity issues

[2.6.1]

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

[2.6.0]

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

[2.5.1]

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

[2.5.0]

- Improvements
 - Added API FLEXIO_UART_FlushShifters to flush UART fifo.

[2.4.0]

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

[2.3.0]

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

[2.2.0]

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

[2.1.6]

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

[2.1.5]

- Improvements
 - Triggered user callback after all the data in ringbuffer were received in FLEXIO_UART_TransferReceiveNonBlocking.

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

- Bug Fixes
 - Changed the API name FLEXIO_UART_StopRingBuffer to FLEXIO_UART_TransferStopRingBuffer to align with the definition in C file.

[2.1.0]

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

FLEXIO_UART_EDMA

[2.3.1]

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

[2.3.0]

- Refer FLEXIO_UART driver change log to 2.3.0
-

FLEXSPI

[2.8.1]

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

[2.8.0]

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

[2.7.0]

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

[2.6.4]

- Improvements
 - Added new macro “FSL_SDK_ENABLE_FLEXSPI_RESET_CONTROL” to support driver level reset control.

[2.6.3]

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

[2.6.2]

- Bug Fixes
 - Wait Bus IDLE before operation of FLEXSPI_SoftwareReset(), FLEXSPI_TransferBlocking() and FLEXSPI_TransferNonBlocking().

[2.6.1]

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

[2.6.0]

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

[2.5.1]

- Bugfixes
 - Fixed handling of W1C bits in the INTR register
 - Removed FIFO resets from FLEXSPI_CheckAndClearError
 - FLEXSPI_TransferBlocking is observing IPCMDDONE and then fetches the final status of the transfer
 - Fixed issue that FLEXSPI2_DriverIRQHandler not defined.

[2.5.0]

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

[2.4.0]

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

[2.3.5]

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

[2.3.4]

- Bug Fixes
 - Updated flexspi_config_t structure and FlexSPI_Init to support new feature FSL_FEATURE_FLEXSPI_HAS_NO_MCR0_CONBINATION.

[2.3.3]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

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

[2.1.0]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

- Bug Fixes
 - Fixed AHB receive buffer size configuration issue. The `FLEXSPI_AHBRXBUFCR0_BUFSZ` field should configure 64 bits size, and currently the AHB receive buffer size is in bytes which means 8-bit, so the correct configuration should be `config->ahbConfig.buffer[i].bufferSize / 8`.

[2.0.2]

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

[2.0.1]

- Bug Fixes
 - Fixed the flag clear issue and AHB read Command index configuration issue in FLEXSPI_SetFlashConfig.
 - Updated FLEXSPI_UpdateLUT function to update LUT table from any index instead of previous command index.
 - Added bus idle wait in FLEXSPI_SetFlashConfig and FLEXSPI_UpdateLUT to ensure bus is idle before any change to FlexSPI controller.
 - Updated interrupt API FLEXSPI_TransferNonBlocking and interrupt handle flow FLEXSPI_TransferHandleIRQ.
 - Updated eDMA API FLEXSPI_TransferEDMA.

[2.0.0]

- Initial version.
-

FLEXSPI EDMA Driver

[2.3.3]

- Bug Fixes
 - Fixed FLEXSPI_TransferEDMA bug that, the DMA channel not configured correctly when using kFLEXSPI_Read.

[2.3.2]

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

[2.0.2]

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

[2.0.0]

- Initial version.
-

I3C**[2.14.3]**

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

[2.14.2]

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

[2.14.1]

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

[2.14.0]

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

[2.13.1]

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

[2.13.0]

- New features
 - Added the hot-join support for I3C bus initialization API.
- Bug Fixes
 - Set read termination with START at the same time in case unknown issue.
 - Set MCTRL[TYPE] as 0 for DDR force exit.
- Improvements
 - Added the API to reset device count assigned by ENTDAAs.
 - Provided the method to set global macro I3C_MAX_DEVCNT to determine how many device addresses ENTDAAs can allocate at one time.
 - Initialized target management static array based on instance number for the case that multiple instances are used at the same time.

[2.12.0]

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

[2.11.1]

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

[2.11.0]

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

[2.10.8]

- Improvements
 - Support more instances.

[2.10.7]

- Improvements
 - Fixed the potential compile warning.

[2.10.6]

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

[2.10.5]

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

[2.10.4]

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

[2.10.3]

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

[2.10.2]

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

[2.10.1]

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

[2.10.0]

- New features
 - Added I3C extended IBI data support.

[2.9.0]

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

[2.8.2]

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

[2.8.1]

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

[2.8.0]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

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

[2.5.4]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

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

[2.5.0]

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

[2.4.0]

- Bug Fixes
 - Fixed kI3C_SlaveMatchedFlag interrupt is not properly handled in I3C_SlaveTransferHandleIRQ when it comes together with interrupt kI3C_SlaveBusStartFlag.
 - Fixed the inaccurate I2C baudrate calculation in I3C_MasterSetBaudRate.
 - Added new API I3C_MasterGetIBIRules to get registered IBI rules.
 - Added new variable isReadTerm in struct _i3c_master_handle for transfer state routine to check if MCTRL.RDTERM is configured for read transfer.
 - Changed to emit Auto IBI in transfer state routine for slave start flag assertion.
 - Fixed the slave maxWriteLength and maxReadLength does not be configured into SMAXLIMITS register issue.
 - Fixed incorrect state for IBI in I3C master interrupt transfer IRQ handle routine.
 - Added isHotJoin in i3c_slave_config_t to request hot-join event during slave init.

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.0]

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

[2.1.2]

- Bug Fixes
 - Reset default hkeep value to kI3C_MasterHighKeeperNone in I3C_MasterGetDefaultConfig

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.
-

I3C_EDMA

[2.2.12]

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

[2.2.11]

- Improvements
 - Fixed Coverity CERT-C violations.

[2.2.10]

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

[2.2.9]

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

[2.2.8]

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

[2.2.7]

- Bug Fixes
 - Fixed MISRA issue rule 11.6.

[2.2.6]

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

[2.2.5]

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

[2.2.4]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

- Improvements
 - Deleted legacy IBI data request code.

[2.1.0]

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

[2.0.1]

- Bug Fixes
 - Fixed MISRA issue rule 9.1.

[2.0.0]

- Initial version.
-
-

IRQSTEER

[2.1.5]

- Bug Fixes
 - Fixed the violation of MISRA C-2012 Rules and CERT-C.

[2.1.4]

- Bug Fixes
 - Fixed the bug that IRQSTEER_GetMasterNextInterrupt may check wrong register with platform which has odd number of CHn_MASK registers.

[2.1.3]

- Bug Fixes
 - Fixed the violation of MISRA C-2012 Rules.

[2.1.2]

- Improvements
 - Cleanup the index calculation.

[2.1.1]

- Bug Fixes
 - Fixed conditional compile with FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL false.

[2.1.0]

- New Features
 - Added new APIs IRQSTEER_InterruptIsEnabled, IRQSTEER_GetMasterIrqCount.
 - Added new configuration options FSL_IRQSTEER_USE_DRIVER_IRQ_HANDLER, and FSL_IRQSTEER_ENABLE_MASTER_INT.
 - Added new APIs IRQSTEER_GetMasterInterruptsStatus.

[2.0.2]

- Bug Fixes
 - Fixed the violation of MISRA C-2012 Rules: 10.1, 10.4, 10.7, 10.8, 12.2, 17.7, 20.7.
- New Features
 - Added control macro to enable/disable the CLOCK code in current driver.

[2.0.0]

- Initial version.
-

ISI

[2.2.5]

- Update to adapter head file
 - ISI interface have some channel, head file have been updated to support it, modified driver to compatible this changes.

[2.2.4]

- Bug Fixes
 - Fixed CERT INT31-C INT30-C violations.

[2.0.2]

- Bug Fixes
 - Fixed the violations of MISRA 2012 rules: 3.1, 5.6, 5.7, 10.1, 10.3, 10.4, 10.8, 12.2, 13.2, 14.4, 17.7.

[2.0.1]

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

[2.0.0]

- Initial version.
-

LDB (LVDS Display Bridge)

[2.2.0]

- New feature.
 - Supported data map standard setting in LDB_Init.
 - Fixed split mode bit setting logic.

[2.1.0]

- New feature.
 - Supported display input 1 in ldb driver.
 - Supported diIndex and dualpanelIndex in LDB_Init API.

[2.0.0]

- Initial version.
-

LPI2C

[2.6.2]

- Improvements
 - Added timeout for while loop in LPI2C_TransferStateMachineSendCommand().

[2.6.1]

- Bug Fixes
 - Fixed coverity issues.

[2.6.0]

- New Feature
 - Added common IRQ handler entry LPI2C_DriverIRQHandler.

[2.5.7]

- Improvements
 - Added support for separated IRQ handlers.

[2.5.6]

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

[2.5.5]

- Bug Fixes
 - Fixed LPI2C_SlaveInit() - allow to disable SDA/SCL glitch filter.

[2.5.4]

- Bug Fixes
 - Fixed LPI2C_MasterTransferBlocking() - the return value was sometime affected by call of LPI2C_MasterStop().

[2.5.3]

- Improvements
 - Added handler for LPI2C7 and LPI2C8.

[2.5.2]

- Bug Fixes
 - Fixed ERR051119 to ignore the nak flag when IGNACK=1 in LPI2C_MasterCheckAndClearError.

[2.5.1]

- Bug Fixes
 - Added bus stop incase of bus stall in LPI2C_MasterTransferBlocking.
- Improvements
 - Release peripheral from reset if necessary in init function.

[2.5.0]

- New Features
 - Added new function LPI2C_SlaveEnableAckStall to enable or disable ACKSTALL.

[2.4.1]

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

[2.4.0]

- Improvements
 - Split some functions, fixed CCM problem in file fsl_lpi2c.c.
- Bug Fixes
 - Fixed bug in LPI2C_MasterInit that the MCFGR2's value set in LPI2C_MasterSetBaudRate may be overwritten by mistake.

[2.3.2]

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

[2.3.1]

- Improvements
 - Updated LPI2C_GetCyclesForWidth to add the parameter of minimum cycle, because for master SDA/SCL filter, master bus idle/pin low timeout and slave SDA/SCL filter configuration, 0 means disabling the feature and cannot be used.
- Bug Fixes
 - Fixed bug in LPI2C_SlaveTransferHandleIRQ that when restart detect event happens the transfer structure should not be cleared.
 - Fixed bug in LPI2C_RunTransferStateMachine, that when only slave address is transferred or there is still data remaining in tx FIFO the last byte's nack cannot be ignored.
 - Fixed bug in slave filter doze enable, that when FILTDZ is set it means disable rather than enable.
 - Fixed bug in the usage of LPI2C_GetCyclesForWidth. First its return value cannot be used directly to configure the slave FILTSDA, FILTSCL, DATAVD or CLKHOLD, because the real cycle width for them should be FILTSDA+3, FILTSCL+3, FILTSCL+DATAVD+3 and CLKHOLD+3. Second when cycle period is not affected by the prescaler value, prescaler value should be passed as 0 rather than 1.

- Fixed wrong default setting for LPI2C slave. If enabling the slave tx SCL stall, then the default clock hold time should be set to 250ns according to I2C spec for 100kHz standard mode baudrate.
- Fixed bug that before pushing command to the tx FIFO the FIFO occupation should be checked first in case FIFO overflow.

[2.3.0]

- New Features
 - Supported reading more than 256 bytes of data in one transfer as master.
 - Added API LPI2C_GetInstance.
- Bug Fixes
 - Fixed bug in LPI2C_MasterTransferAbortEDMA, LPI2C_MasterTransferAbort and LPI2C_MasterTransferHandleIRQ that before sending stop signal whether master is active and whether stop signal has been sent should be checked, to make sure no FIFO error or bus error will be caused.
 - Fixed bug in LPI2C master EDMA transactional layer that the bus error cannot be caught and returned by user callback, by monitoring bus error events in interrupt handler.
 - Fixed bug in LPI2C_GetCyclesForWidth that the parameter used to calculate clock cycle should be $2^{\text{prescaler}}$ rather than prescaler.
 - Fixed bug in LPI2C_MasterInit that timeout value should be configured after baudrate, since the timeout calculation needs prescaler as parameter which is changed during baudrate configuration.
 - Fixed bug in LPI2C_MasterTransferHandleIRQ and LPI2C_RunTransferStateMachine that when master writes with no stop signal, need to first make sure no data remains in the tx FIFO before finishes the transfer.

[2.2.0]

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

[2.1.12]

- Bug Fixes
 - Fixed MISRA advisory 15.5 issues.

[2.1.11]

- Bug Fixes
 - Fixed the bug that, during master non-blocking transfer, after the last byte is sent/received, the kLPI2C_MasterNackDetectFlag is expected, so master should not check and clear kLPI2C_MasterNackDetectFlag when remainingBytes is zero, in case FIFO is emptied when stop command has not been sent yet.

- Fixed the bug that, during non-blocking transfer slave may nack master while master is busy filling tx FIFO, and NDF may not be handled properly.

[2.1.10]

- Bug Fixes
 - MISRA C-2012 issue fixed.
 - * Fixed rule 10.3, 14.4, 15.5.
 - Fixed unaligned access issue in LPI2C_RunTransferStateMachine.
 - Fixed uninitialized variable issue in LPI2C_MasterTransferHandleIRQ.
 - Used linked TCD to disable tx and enable rx in read operation to fix the issue that for platform sharing the same DMA request with tx and rx, during LPI2C read operation if interrupt with higher priority happened exactly after command was sent and before tx disabled, potentially both tx and rx could trigger dma and cause trouble.
 - Fixed MISRA issues.
 - * Fixed rules 10.1, 10.3, 10.4, 11.6, 11.9, 14.4, 17.7.
 - Fixed the waitTimes variable not re-assignment issue for each byte read.
- New Features
 - Added the IRQHandler for LPI2C5 and LPI2C6 instances.
- Improvements
 - Updated the LPI2C_WAIT_TIMEOUT macro to unified name I2C_RETRY_TIMES.

[2.1.9]

- Bug Fixes
 - Fixed Coverity issue of unchecked return value in I2C_RTOS_Transfer.
 - Fixed Coverity issue of operands did not affect the result in LPI2C_SlaveReceive and LPI2C_SlaveSend.
 - Removed STOP signal wait when NAK detected.
 - Cleared slave repeat start flag before transmission started in LPI2C_SlaveSend/LPI2C_SlaveReceive. The issue was that LPI2C_SlaveSend/LPI2C_SlaveReceive did not handle with the reserved repeat start flag. This caused the next slave to send a break, and the master was always in the receive data status, but could not receive data.

[2.1.8]

- Bug Fixes
 - Fixed the transfer issue with LPI2C_MasterTransferNonBlocking, kLPI2C_TransferNoStopFlag, with the wait transfer done through callback in a way of not doing a blocking transfer.
 - Fixed the issue that STOP signal did not appear in the bus when NAK event occurred.

[2.1.7]

- Bug Fixes
 - Cleared the stopflag before transmission started in LPI2C_SlaveSend/LPI2C_SlaveReceive. The issue was that LPI2C_SlaveSend/LPI2C_SlaveReceive did not handle with the reserved stop flag and caused the next slave to send a break, and the master always stayed in the receive data status but could not receive data.

[2.1.6]

- Bug Fixes
 - Fixed driver MISRA build error and C++ build error in LPI2C_MasterSend and LPI2C_SlaveSend.
 - Reset FIFO in LPI2C Master Transfer functions to avoid any byte still remaining in FIFO during last transfer.
 - Fixed the issue that LPI2C_MasterStop did not return the correct NAK status in the bus for second transfer to the non-existing slave address.

[2.1.5]

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

[2.1.4]

- Bug Fixes
 - Fixed the LPI2C_MasterTransferEDMA receive issue when LPI2C shared same request source with TX/RX DMA request. Previously, the API used scatter-gather method, which handled the command transfer first, then the linked TCD which was pre-set with the receive data transfer. The issue was that the TX DMA request and the RX DMA request were both enabled, so when the DMA finished the first command TCD transfer and handled the receive data TCD, the TX DMA request still happened due to empty TX FIFO. The result was that the RX DMA transfer would start without waiting on the expected RX DMA request.
 - Fixed the issue by enabling IntMajor interrupt for the command TCD and checking if there was a linked TCD to disable the TX DMA request in LPI2C_MasterEDMACallback API.

[2.1.3]

- Improvements
 - Added LPI2C_WATI_TIMEOUT macro to allow the user to specify the timeout times for waiting flags in functional API and blocking transfer API.
 - Added LPI2C_MasterTransferBlocking API.

[2.1.2]

- Bug Fixes
 - In LPI2C_SlaveTransferHandleIRQ, reset the slave status to idle when stop flag was detected.

[2.1.1]

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

[2.1.0]

- Improvements
 - API name changed:
 - * LPI2C_MasterTransferCreateHandle -> LPI2C_MasterCreateHandle.
 - * LPI2C_MasterTransferGetCount -> LPI2C_MasterGetTransferCount.
 - * LPI2C_MasterTransferAbort -> LPI2C_MasterAbortTransfer.
 - * LPI2C_MasterTransferHandleIRQ -> LPI2C_MasterHandleInterrupt.
 - * LPI2C_SlaveTransferCreateHandle -> LPI2C_SlaveCreateHandle.
 - * LPI2C_SlaveTransferGetCount -> LPI2C_SlaveGetTransferCount.
 - * LPI2C_SlaveTransferAbort -> LPI2C_SlaveAbortTransfer.
 - * LPI2C_SlaveTransferHandleIRQ -> LPI2C_SlaveHandleInterrupt.

[2.0.0]

- Initial version.
-

LPI2C_EDMA

[2.4.5]

- Improvements
 - Added condition to IRQ handler to check whether the interrupt is enabled - kLPI2C_MasterTxReadyFlag.

[2.4.4]

- Improvements
 - Added support for 2KB data transfer

[2.4.3]

- Improvements
 - Added support for separated IRQ handlers.

[2.4.2]

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

[2.4.1]

- Refer LPI2C driver change log 2.0.0 to 2.4.1
-

LPIT**[2.1.3]**

- Bug Fixes
 - Fixed doxygen generation warnings.

[2.1.2]

- Bug Fixes
 - Fix CERT INT31-C issues.

[2.1.1]

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

[2.1.0]

- Improvements
 - Add new function LPIT_SetTimerValue to set timeout period.

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

LPSPI

[2.7.3]

- Improvements
 - Added timeout for while loop in LPSPI_MasterTransferWriteAllTxData().
 - Make SPI_RETRY_TIMES configurable by CONFIG_SPI_RETRY_TIMES.

[2.7.2]

- Bug Fixes
 - Fixed coverity issues.

[2.7.1]

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

[2.7.0]

- New Feature
 - Added common IRQ handler entry LPSPI_DriverIRQHandler.

[2.6.10]

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

[2.6.9]

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

[2.6.8]

- Bug Fixes
 - Fixed build error when SPI_RETRY_TIMES is defined to non-zero value.

[2.6.7]

- Bug Fixes
 - Fixed the txData from void * to const void * in transmit API _lpspi_master_handle and _lpspi_slave_handle.

[2.6.6]

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

[2.6.5]

- Improvements
 - Introduced FSL_FEATURE_LPSPI_HAS_NO_PCSCFG and FSL_FEATURE_LPSPI_HAS_NO_MULTI_WIDTH for conditional compile.
 - Release peripheral from reset if necessary in init function.

[2.6.4]

- Bug Fixes
 - Added LPSPI6_DriverIRQHandler for LPSPI6 instance.

[2.6.3]

- Hot Fixes
 - Added macro switch in function LPSPI_Enable about ERRATA051472.

[2.6.2]

- Bug Fixes
 - Disabled lpspi before LPSPI_MasterSetBaudRate incase of LPSPI opened.

[2.6.1]

- Bug Fixes
 - Fixed return value while calling LPSPI_WaitTxFifoEmpty in function LPSPI_MasterTransferNonBlocking.

[2.6.0]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

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

[2.5.0]

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

[2.4.7]

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

[2.4.6]

- Improvements
 - Moved the configuration of registers for the 3-wire lpspi mode to the LPSPI_MasterInit and LPSPI_SlaveInit function.

[2.4.5]

- Improvements
 - Improved LPSPI_MasterTransferBlocking send performance when frame size is 1-byte.

[2.4.4]

- Bug Fixes
 - Fixed LPSPI_MasterGetDefaultConfig incorrect default inter-transfer delay calculation.

[2.4.3]

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

[2.4.2]

- Bug Fixes
 - Fixed bug that LPSPI_MasterTransferBlocking will modify the parameter txbuff and rxbuff pointer.

[2.4.1]

- Bug Fixes
 - Fixed bug that LPSPI_SlaveTransferNonBlocking can't detect RX error.

[2.4.0]

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

[2.3.1]

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

[2.3.0]

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

[2.2.1]

- Bug Fixes
 - Fixed bug in LPSPI_SetPCSContinuous when disabling PCS continuous mode.

[2.2.0]

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

[2.1.0]

- Improvements
 - Improved LPSPI_SlaveTransferHandleIRQ to fill up TX FIFO instead of write one data to TX register which improves the slave transmit performance.
 - Added new functional APIs LPSPI_SelectTransferPCS and LPSPI_SetPCSContinuous to support changing PCS selection and PCS continuous mode.
- Bug Fixes

- Fixed bug in non-blocking and EDMA transfer APIs that `kStatus_InvalidArgument` is returned if user configures 3-wire mode and full-duplex transfer at the same time, but transfer state is already set to `kLPSPI_Busy` by mistake causing following transfer can not start.
- Fixed bug when LPSPI slave using EDMA way to transfer, tx should be masked when tx data is null, otherwise in 3-wire mode which tx/rx use the same pin, the received data will be interfered.

[2.0.5]

- Improvements
 - Added timeout mechanism when waiting certain states in transfer driver.
- Bug Fixes
 - Fixed the bug that LPSPI can not transfer large data using EDMA.
 - Fixed MISRA 17.7 issues.
 - Fixed variable overflow issue introduced by MISRA fix.
 - Fixed issue that `rxFifoMaxBytes` should be calculated according to transfer width rather than FIFO width.
 - Fixed issue that completion flag was not cleared after transfer completed.

[2.0.4]

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

[2.0.3]

- Bug Fixes
 - Removed `LPSPI_Reset` from `LPSPI_MasterInit` and `LPSPI_SlaveInit`, because this API may glitch the slave select line. If needed, call this function manually.

[2.0.2]

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

[2.0.1]

- Bug Fixes
 - Fixed the bug that the clock source should be divided by the `PRESCALE` setting in `LPSPI_MasterSetDelayTimes` function.

- Fixed the bug that LPSPI_MasterTransferBlocking function would hang in some corner cases.
- Optimization
 - Added #ifndef/#endif to allow user to change the default TX value at compile time.

[2.0.0]

- Initial version.
-

LPSPI_EDMA

[2.4.9]

- Improvements
 - Removed unused code from LPSPI_SeparateEdmaReadData().

[2.4.8]

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

[2.4.7]

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

[2.4.6]

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

[2.4.5]

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

[2.4.4]

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

[2.4.3]

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

[2.4.2]

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

[2.4.1]

- Improvements
 - Add the TXMSK wait after TCR setting.

[2.4.0]

- Improvements
 - Separated LPSPI_MasterTransferEDMA functions to LP-SPI_MasterTransferPrepareEDMA and LPSPI_MasterTransferEDMALite to optimize the process of transfer.
-

LPTMR

[2.2.1]

- Bug Fixes
 - Fix CERT INT31-C issues.

[2.2.0]

- Improvements
 - Updated lptmr_prescaler_clock_select_t, only define the valid options.

[2.1.1]

- Improvements
 - Updated the characters from “PTMR” to “LPTMR” in “FSL_FEATURE_PTMR_HAS_NO_PRESCALER_CLOCK_SOURCE_1_SUPPORT” feature definition.

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

LPUART**[2.10.0]**

- New Feature
 - Added support to configure RTS watermark.

[2.9.4]

- Improvements
 - Merged duplicate code.

[2.9.3]

- Improvements
 - Added timeout for while loops in LPUART_Deinit().

[2.9.2]

- Bug Fixes
 - Fixed coverity issues.

[2.9.1]

- Bug Fixes
 - Fixed coverity issues.

[2.9.0]

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

[2.8.3]

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

[2.8.2]

- Bug Fix
 - Fixed the bug that LPUART_TransferEnable16Bit controled by wrong feature macro.

[2.8.1]

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

[2.8.0]

- Improvements
 - Added support of DATA register for 9bit or 10bit data transmit in write and read API. Such as: LPUART_WriteBlocking16bit, LPUART_ReadBlocking16bit, LPUART_TransferEnable16Bit, LPUART_WriteNonBlocking16bit, LPUART_ReadNonBlocking16bit.

[2.7.7]

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

[2.7.6]

- Bug Fixes
 - Fixed LPUART_EnableInterrupts and LPUART_DisableInterrupts bug that blocks if the LPUART address doesn't support exclusive access.

[2.7.5]

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

[2.7.4]

- Improvements
 - Added support for atomic register accessing in LPUART_EnableInterrupts and LPUART_DisableInterrupts.

[2.7.3]

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

[2.7.2]

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

[2.7.1]

- Improvements
 - Added support for LPUART_BASE_PTRS_NS in security mode in file fsl_lpuart.c.

[2.7.0]

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

[2.6.0]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

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

[2.5.0]

- Bug Fixes
 - Added missing interrupt enable masks kLPUART_Match1InterruptEnable and kLPUART_Match2InterruptEnable.
 - Fixed bug in LPUART_EnableInterrupts, LPUART_DisableInterrupts and LPUART_GetEnabledInterrupts that the BAUD[LBKDIE] bit field should be soc specific.

- Fixed bug in LPUART_TransferHandleIRQ that idle line interrupt should be disabled when rx data size is zero.
- Deleted unused status flags kLPUART_NoiseErrorInRxDataRegFlag and kLPUART_ParityErrorInRxDataRegFlag, since firstly their function are the same as kLPUART_NoiseErrorFlag and kLPUART_ParityErrorFlag, secondly to obtain them one data word must be read out thus interfering with the receiving process.
- Fixed bug in LPUART_GetStatusFlags that the STAT[LBKDIF], STAT[MA1F] and STAT[MA2F] should be soc specific.
- Fixed bug in LPUART_ClearStatusFlags that tx/rx FIFO is reset by mistake when clearing flags.
- Fixed bug in LPUART_TransferHandleIRQ that while clearing idle line flag the other bits should be masked in case other status bits be cleared by accident.
- Fixed bug of race condition during LPUART transfer using transactional APIs, by disabling and re-enabling the global interrupt before and after critical operations on interrupt enable register.
- Fixed DMA/eDMA transfer blocking issue by enabling tx idle interrupt after DMA/eDMA transmission finishes.
- New Features
 - Added APIs LPUART_GetRxFifoCount/LPUART_GetTxFifoCount to get rx/tx FIFO data count.
 - Added APIs LPUART_SetRxFifoWatermark/LPUART_SetTxFifoWatermark to set rx/tx FIFO water mark.

[2.4.1]

- Bug Fixes
 - Fixed MISRA advisory 17.7 issues.

[2.4.0]

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

[2.3.1]

- Bug Fixes
 - Fixed MISRA advisory 15.5 issues.

[2.3.0]

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

[2.2.8]

- Bug Fixes
 - Fixed issue for MISRA-2012 check.
 - * Fixed rule-10.3, rule-14.4, rule-15.5.
 - Eliminated Pa082 warnings by assigning volatile variables to local variables and using local variables instead.
 - Fixed MISRA issues.
 - * Fixed rules 10.1, 10.3, 10.4, 10.8, 14.4, 11.6, 17.7.
- Improvements
 - Added check for `kLPUART_TransmissionCompleteFlag` in `LPUART_WriteBlocking`, `LPUART_TransferHandleIRQ`, `LPUART_TransferSendDMACallback` and `LPUART_SendEDMACallback` to ensure all the data would be sent out to bus.
 - Rounded up the calculated `sbr` value in `LPUART_SetBaudRate` and `LPUART_Init` to achieve more accurate baudrate setting. Changed `osr` from `uint32_t` to `uint8_t` since `osr`'s biggest value is 31.
 - Modified `LPUART_ReadBlocking` so that if more than one receiver errors occur, all status flags will be cleared and the most severe error status will be returned.

[2.2.7]

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

[2.2.6]

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

[2.2.5]

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

[2.2.4]

- Improvements
 - Added hardware flow control function support.
 - Added idle-line-detecting feature in `LPUART_TransferNonBlocking` function. If an idle line is detected, a callback is triggered with status `kStatus_LPUART_IdleLineDetected` returned. This feature may be useful when the received Bytes is less than the expected received data size. Before triggering the callback, data in the FIFO (if has FIFO) is read out, and no interrupt will be disabled, except for that the receive data size reaches 0.

- Enabled the RX FIFO watermark function. With the idle-line-detecting feature enabled, users can set the watermark value to whatever you want (should be less than the RX FIFO size). Data is received and a callback will be triggered when data receive ends.

[2.2.3]

- Improvements
 - Changed parameter type in LPUART_RTOS_Init struct from rtos_lpuart_config to lpuart_rtos_config_t.
- Bug Fixes
 - Disabled LPUART receive interrupt instead of all NVICs when reading data from ring buffer. Otherwise when the ring buffer is used, receive nonblocking method will disable all NVICs to protect the ring buffer. This may has a negative effect on other IPs that are using the interrupt.

[2.2.2]

- Improvements
 - Added software reset feature support.
 - Added software reset API in LPUART_Init.

[2.2.1]

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

[2.2.0]

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

[2.1.1]

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

[2.1.0]

- Improvements
 - Update transactional APIs.
-

LPUART_EDMA

[2.4.0]

- Refer LPUART driver change log 2.1.0 to 2.4.0
-

MCM

[2.2.0]

- Improvements
 - Support platforms with less features.

[2.1.0]

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

[2.0.0]

- Initial version.
-
-

MIPI_DSI

[2.1.0]

- Bug Fixes
 - Fixed typo in member of `dsi_transfer_t` structure. The `sendDscCmd` and `dscCmd` shall be `sendDcsCmd` and `dcsCmd`.

[2.0.3]

- Improvements
 - Supported `DSI_ConfigDphy` api.
 - Supported `DSI_EnableVpgEnMode` to config the video mode pattern generator.

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

MSGINTR

[2.0.2]

- Improvements
 - Conditional compile IRQ handlers.

[2.0.1]

- Bug Fixes
 - Fixed MISRA issue rule 8.4, 11.9, 17.7.

[2.0.0]

- Initial version.
-

MU

[2.8.1]

- Bug Fixes
 - Avoid incorrect MU_BUSY_POLL_COUNT macro use.

[2.8.0]

- New Features
 - Added MU1_BUSY_POLL_COUNT parameter to prevent infinite polling loops in MU operations.
 - Added timeout mechanism to all polling loops in MU driver code.
- Improvements
 - Updated function signatures to return status codes for better error handling:
 - * Changed MU_ResetBothSides to return status_t instead of void
 - * Updated MU_SendMsg to return status_t for timeout indication
 - * Added new function MU_ReceiveMsgTimeout() to include timeout mechanism.
 - Enhanced documentation across all functions to clarify timeout behavior and return values.

[2.7.0]

- New Features
 - Added API MU_GetRxStatusFlags.

[2.6.0]

- New Features
 - Added API MU_GetInterruptsPending.

[2.5.1]

- Bug Fixes
 - Fixed the bug that MU_TriggerGeneralPurposeInterrupts and MU_TriggerInterrupts may trigger previous triggered general purpose interrupts again by mistake.

[2.5.0]

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

[2.4.0]

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

[2.3.3]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.1]

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

[2.2.0]

- New features
 - Added support for i.MX RT118x.
- Bug Fixes
 - Fixed general purpose interrupt bug.
- Other Changes
 - Change `_mu_interrupt_trigger` item value.

[2.1.2]

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

[2.1.1]

- Bug Fixes
 - Fixed MISRA C-2012 issues.

[2.1.0]

- Improvements
 - Added new enum `mu_msg_reg_index_t`.

[2.0.0]

- Initial version.
-

NETC

[2.10.3]

- New Features
 - Supported `rrt` member in `netc_tb_ipf_cfge_t`.
 - Moved timer reference clock definition to soc to support i.MX95/i.MX943.
- Bug Fixes
 - Got right size of TX timestamp response frame.

[2.10.2]

- Bug Fixes
 - Correct transmit start time bit width of transmit buffer descriptor format.

[2.10.1]

- New Features
 - Supported IP version class commands in VSI-PSI message driver.
 - Added VSI3 definition.

[2.10.0]

- Bug Fixes
 - Fixed structures used for time gate control list and stream gate request commands memset.
- New Features
 - Added public API to query table entry in Frame Modification Table.
- Improvements
 - Optimized NETC Transmit functions: cached configuration in driver handle instead of accessing directly device registers at each packet transmit, and removed calls to memset/memcpy in the transmit path.

[2.9.1]

- Improvements
 - NETC_TimerInit() will always ignore user config->atomicMode and set it to 1 internally. This guarantees that period updates, that change both the integer and fractional part are always done atomically.

[2.9.0]

- Bug Fixes
 - Fixed padding in netc_tb_sgi_rsp_data_t union structure for query operations on OEXEN and IRXEN parameters.
 - Fixed structure use for rate policer and stream gate request commands memset.
 - Updated ERRATA 052134 to 052206.
 - Fixed MII mode setting.
 - Fixed i.MX943 getting function instance.
- New Features
 - Added API to Reset IRX and OEX flags in stream gate instance entry.
 - Added API to configure the priority to traffic class map.
 - Added APIs to query table entry and get maximum entry number for Frame Modification Table.
 - Added APIs to configure Frame preemption.
 - Added APIs to configure PSRCR and PGCR registers to implement HSR feature.
 - Moved PHY WRAPPER init sequence (NETC_PHYInit) implementation to SoC. And Added i.MX943 support.
- Improvements
 - keep netc_tb_sgi_rsp_data_t local to the low level driver functions for SGI table entry query.

- Converted to use preinitVsi callback for VSI pre-init.
- Added note for ERRATA 052167 to remind that actual MAC Tx IPG is longer than configured when transmitting back-to-back packets in MII half duplex. When using MII protocol, using full-duplex mode is recommended instead of half-duplex. If using MII half-duplex mode, additional bandwidth loss should be expected and accounted for due to extended IPG.

[2.8.2]

- Bug Fixes
 - Fixed ingress port filter table frame attribute flags mask field issue.

[2.8.1]

- Bug Fixes
 - Fixed MAC/VLAN filter operations through VSI-PSI message.
 - Fixed NETC_PortConfigTxIpgPreamble compile.
- Improvements
 - Enabled standard VLAN EtherTypes for i.MX95 VSIs for VLAN support.
 - Added netc_timer_exttrig_index_t definition for i.MX95.
 - Updated default BPCR[STAMVD] value setting to align the register reset value.

[2.8.0]

- Bug Fixes
 - Fixed ERRATA 052024.
 - Fixed ERRATA 052129.
 - Fixed ERRATA 052134.
 - Fixed ERRATA 052031.
 - Fixed ERRATA 051994.
 - Fixed ERRATA 051936.
- New Features
 - Added interface to reset the mark frame red parameter.
 - Added support for FRER sequence generation reset.
 - Added NETC Switch Tag support.
 - Added the Tx offload feature support.
- Improvements
 - Simplify NETC_TimerGetFreeRunningTime. Hardware synchronizes reads from high/low registers for the free running time. No need to do it in software.

[2.7.2]

- Bug Fixes
 - Fixed MISRA issue rule 4.10, 10.1, 10.3, 10.4, 10.7, 10.8, 11.3, 16.1, 16.4, 17.7.

[2.7.1]

- Bug Fixes
 - Fixed Coverity issue with array out of bounds access.

[2.7.0]

- New Features
 - Added VSI-PSI messaging driver.
- Bug Fixes
 - Fixed the issue that EP/SWT_ReceiveFrame don't return error status when some errors occur.

[2.6.1]

- Bug Fixes
 - Updated the MAC loopback configuration as Reference Manual.

[2.6.0]

- New Features
 - Added API to get transmit max SDU for specified port Traffic Class.
 - Added API to query entry from the Rate Policing table.
 - Added API to retrieve maximum rate policer entries.
 - Added API to set switch port default VID separately.
 - Added API to set max frame size separately.
 - Added API to query FRER resource.
- Bug Fixes
 - Fixed the issue that stream gate query functions don't check return status.
 - Fixed the ISF table query function operation issue.
 - Fixed the wrong configuration of Tx max SDU check.
 - Fixed ERRATA 051524.
 - Fixed ERRATA 051649.
 - Fixed ERRATA 051707.
 - Fixed ERRATA 051710.
 - Fixed ERRATA 051711.
- Improvements
 - Factorized qbv basetime workaround code, and stop using synchronized time for the workaround code. Synchronized time functionality should be reserved for gPTP operation.

[2.5.1]

- Improvements
 - Conditional compile NETC_ETH_LINK_PM0_COMMAND_CONFIG_HD_FCEN register.

[2.5.0]

- New Features
 - Added PHY WRAPPER driver.
 - Added C45 support for internal MDIO.
 - Added 10G support.
- Bug Fixes
 - Fixed ERRATA 051130.
 - Fixed master bus and memory access.
- Improvements
 - Moved platform specific code to soc driver.
 - Split switch code.

[2.4.0]

- New Features
 - Added the interrupt control functions for port MAC module.
 - Added setting parameters including half-duplex back pressure, port timestamp capture point, RGMII Tx clock stop state during low power idle, ports default traffic class gating states and timer atomic writing setting.
 - Added NETC_TimerInitHandle() to initialize a timer handle without modifying hardware state. Required to be able to read timer from another CPU.
 - Added NETC_TimerGetFreeRunningTime() to be able to read free running timer.
 - Added support for ingress stream gate query.
- Improvements
 - Added necessary default settings in the GetDefaultConfig functions in case some features can't work after initialization.
 - Updated loopback function according to new bit field in CRR.
 - Deleted the useless error check for ERRATA051243.
 - Updated NETC_TimerGetCurrentTime() to avoid using synchronized time and be able to read the time from different threads/cpus without locking.
 - Deleted the useless priority check in NETC_PortConfigTcCBS().
- Bug Fixes
 - Fixed typo in NETC_PortConfig.

[2.3.2]

- Bug Fixes
 - Added workaround for ERRATA051587.

[2.3.1]

- Bug Fixes
 - Fixed MISRA issue rule 10.3, 10.4, 10.8, 11.6, 11.7.

[2.3.0]

- Bug Fixes
 - Added SWT_PortStop() API for ERRATA051398.
 - Fixed the build error by add feature macro for port FCS Error Action feature.
 - Removed duplicate code from NETC_PortEthMacGracefulStop() API.
 - Fixed MISRA issue rule 8.6, 10.4, 11.9, 14.4.

[2.2.2]

- Bug Fixes
 - Fixed the issue that NETC_PortSetSpeed() would overwrite the full PCR register.

[2.2.1]

- Improvements
 - Fixed cpp build warning.

[2.2.0]

- Bug Fixes
 - Fixed the issue that NETC_ConfigTGSAdminList() doesn't clear the previous command response data status filed.
 - Fixed the issue that EP_ReceiveFrameCopy(&handle, 0, NULL, 0, NULL) can't drop error frame.
 - Fixed the issue that SWT_GetTimestampRefResp can't get Switch Tx TS Resp with no MgmtRxBdRing.
 - Fixed the issue that RGMII Half Duplex mode misconfigured.
 - Fixed the issue that missing workaround for ERR050679, ERR051246 and ERR051254.
 - Fixed the issue that missing feature macro for ERR051130, ERR051202, ERR051260.
 - Fixed the issue that ep/swt_tx_opt struct use wrong vlan tag tpid value.
 - Fixed MISRA issue rule 5.8, 8.3, 8.12, 10.1, 10.3, 10.4, 10.6, 10.7, 10.8, 11.6, 11.8, 12.2, 14.4, 15.5, 15.6, 16.1, 16.3, 16.4, 17.7.
 - Fixed the issue that internal MDIO read function uses wrong register.
 - Fixed the issue that SWT_TxPortTGSEnable()/EP_TxPortTGSEnable() still uses the default timer after enabling the 1588 timer.
 - Remove the resetCount parameter from get port discard statistic APIs because the registers required by this function have been removed from hardware design.
 - Fixed the issue that SWT/EP_ReclaimTxDesc() can't call reclaim callback for each full frame.
 - Fixed the issue in NETC_TimerAdjustFreq().
- New Features
 - Added the support for 1588 One-Step timestamp when chip doesn't have ERR051255.
 - Added APIs to get dynamic table remaining available entry numbers.
 - Added APIs to get static table number of entries.

- Improvements
 - Return detail error status instead of `kStatus_Fail` in NTMP APIs.
 - Rename feature macros and move them into the feature file.
 - Optimize the implementation of the `NETC_TimerAddOffset()` function to avoid change the `TMR_CNT_L/H` registers, and add required procedure for call `NETC_TimerAddOffset()` API in the comments.
 - Update the `SWT_FMDUpdateTableEntry()/SWT_FMDQueryTableEntry()` APIs to make them use internal table buffer.
 - Update `SWT_TxPortTGSEnable()/EP_TxPortTGSEnable()` to make it can config the default administration gate control list gates' state.
 - Use `TMR_SRT_L/H` instead of `TMR_CUR_TIMER` when want to get current 1588 timer value.

[2.1.0]

- Bug Fixes
 - Fixed the issue that `EP_RxL2MFINit` doesn't set the multicast promiscuous correctly.
 - Fixed the timer add offset issue.
 - Fixed the issue that all ENETC/Switch PCIe functions must be enabled firstly before triggering EP/SWT NTMP access and MSIX messages.
 - Fixed RT1180 NETC errata 051202: Configure Tx MAC to wait until 32 bytes of data are built up in the transmit FIFO before beginning transmission on the link.
 - Added workaround for RT1180 NETC errata 051130: C Egress time gate scheduling can get corrupted when functional level reset is applied or when time gating is disabled.
 - Fixed bugs in statistic APIs.
- New Features
 - Added the support for EP VSI transmission and PSI-VSI message exchanging.
 - Added EP receive regular frame zero-copy support.
 - Integrated EMDIO support in NETC MDIO driver for accessing PHY when EP/Switch function isn't enabled.
 - Added Timer and Switch MSIX table configuration support.
 - Added update entry APIs for IPF/VF/FDB/L2MCF/IS/ISF/SGI/RP/FM/ET/ISEQG tables, and search entry APIs for FDB/L2MCF table.
 - Added Ingress buffer pool table config APIs.
 - Added MAC Tx padding and Rx min/max frame size configuration to support Tx/Rx frames smaller than 64 bytes.
 - Added API to do graceful Stop for ETH MAC.
- Improvements
 - Used Rx buffer address array provided by application instead of buffer start address with contiguous memory to make the Rx buffer setup more flexible.
 - Added ring and userData parameter in the Tx reclaim callback.
 - Updated NETC hardware layer folder name from 'hw' to 'netc_hw'.
 - Stored necessary EP and SWT configurations constant in handle structure instead of storing pointer which forces application to keep static configuration structure data.

- Updated NETC_MsixXxx to EP_MsixXxx to differentiate with corresponding SWT/Timer MSIX configuration APIs.
- Aligned TGSL/SGCL API with enet_qos high-level driver.
- Updated EP/Switch config structure to include all port related config.
- Updated Switch transfer API only send management frame (Direct enqueue and Switch Port Masquerading) and only receive Host Reason no-zero frames.
- Updated EP transfer API only send/receive regular frames.
- Updated Switch/EP handle to make them use independent cache maintain, alloc/free memory and reclaimCallback functions.

[2.0.0]

- Initial version.
-

PDM

[2.9.3]

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

[2.9.2]

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

[2.9.1]

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

[2.9.0]

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

[2.8.1]

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

[2.8.0]

- Improvements
- Added feature FSL_FEATURE_PDM_HAS_NO_HWVAD to remove the support of hardware voice activity detector.
- Added feature FSL_FEATURE_PDM_HAS_NO_FILTER_BUFFER to remove the support of FIR_RDY bitfield in STAT register.

[2.7.4]

- Bug Fixes
 - Fixed driver can not determine the specific float number of clock divider.
 - Fixed PDM_ValidateSrcClockRate calculates PDM channel in wrong method issue.

[2.7.3]

- Improvements
- Added feature FSL_FEATURE_PDM_HAS_NO_VADEF to remove the support of VADEF bitfield in VADO_STAT register.

[2.7.2]

- Improvements
- Added feature FSL_FEATURE_PDM_HAS_NO_MINIMUM_CLKDIV to decide whether the minimum clock frequency division is required.

[2.7.1]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 8.4, 10.3, 10.1, 10.4, 14.4

[2.7.0]

- Improvements
 - Added api PDM_EnableHwvadInterruptCallback to support handle hwvad IRQ in PDM driver.
 - Corrected the sample rate configuration for non high quality mode.
 - Added api PDM_SetChannelGain to support adjust the channel gain.

[2.6.0]

- Improvements
 - Added new features FSL_FEATURE_PDM_HAS_STATUS_LOW_FREQ/FSL_FEATURE_PDM_HAS_DC_OUT

[2.5.0]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 8.4, 16.5, 10.4, 10.3, 10.1, 11.9, 17.7, 10.6, 14.4, 11.8, 11.6.

[2.4.1]

- Bug Fixes
 - Fixed MDK 66-D warning in pdm driver.

[2.4.0]

- Improvements
 - Added api PDM_TransferSetChannelConfig/PDM_ReadFifo to support read different width data.
 - Added feature FSL_FEATURE_PDM_HAS_RANGE_CTRL and api PDM_ClearRangeStatus/PDM_GetRangeStatus for range register.
- Bug Fixes
 - Fixed violation of MISRA C-2012 Rule 14.4, 10.3, 10.4.

[2.3.0]

- Improvements
 - Enabled envelope/energy voice detect mode by adding apis PDM_SetHwvadInEnvelopeBasedMode/PDM_SetHwvadInEnergyBasedMode.
 - Added feature FSL_FEATURE_PDM_CHANNEL_NUM for different SOC.

[2.2.1]

- Bug Fixes
 - Fixed violation of MISRA C-2012 Rule 10.1, 10.3, 10.4, 10.6, 10.7, 11.3, 11.8, 14.4, 17.7, 18.4.
 - Added medium quality mode support in function PDM_SetSampleRateConfig.

[2.2.0]

- Improvements
 - Added api PDM_SetSampleRateConfig to improve user experience and marked api PDM_SetSampleRate as deprecated.

[2.1.1]

- Improvements
 - Used new SDMA API SDMA_SetDoneConfig instead of SDMA_EnableSwDone for PDM SDMA driver.

[2.1.0]

- Improvements
 - Added software buffer queue for transactional API.

[2.0.1]

- Improvements
 - Improved HWVAD feature.

[2.0.0]

- Initial version.
-

PDM_EDMA

[2.6.5]

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

[2.6.4]

- Improvements
 - Add handling for runtime change of number of linked transfers

[2.6.3]

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

[2.6.2]

- Improvements
 - Add macro `MCUX_SDK_PDM_EDMA_PDM_ENABLE_INTERNAL` to let the user decide whether to enable it when calling `PDM_TransferReceiveEDMA`.

[2.6.1]

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

[2.6.0]

- Improvements
 - Updated api `PDM_TransferReceiveEDMA` to support channel block interleave transfer.
 - Added new api `PDM_TransferSetMultiChannelInterleaveType` to support channel interleave type configurations.

[2.5.0]

- Refer PDM driver change log 2.1.0 to 2.5.0
-

RGPIO

[2.1.0]

- New feature:
 - Added API RGPIO_EnablePortInput()
 - Added API RGPIO_SetPinInterruptConfig()
 - Added API RGPIO_GetPinsInterruptFlags()
 - Added API RGPIO_ClearPinsInterruptFlags()

[2.0.3]

- Improvements:
 - Enhanced FGPIO_PinInit to enable clock internally.

[2.0.2]

- Bug fix
 - MISRA C-2012 issue fixed.
 - * Fix rules, containing: rule-10.3, rule-14.4, rule-15.5.

[2.0.1]

- API Interface Change:
 - Refined naming of API while keep all original APIs with marking them as deprecated. The original API will be removed in the next release. The main change is to update API with prefix of _PinXXX() and _PortXXX().

[2.0.0]

- Initial version.
-

SAI

[2.4.10]

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

[2.4.9]

- Added Errata ERR051421 workaround.

[2.4.8]

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

[2.4.7]

- Added conditional support for bit clock swap feature
- Added common IRQ handler entry SAI_DriverIRQHandler.

[2.4.6]

- Bug Fixes
 - Fixed the IAR build warning.

[2.4.5]

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

[2.4.4]

- Bug Fixes
 - Fixed enumeration sai_fifo_combine_t - add RX configuration.

[2.4.3]

- Bug Fixes
 - Fixed enumeration sai_fifo_combine_t value configuration issue.

[2.4.2]

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

[2.4.1]

- Bug Fixes
 - Fixed bitWidth incorrectly assigned issue.

[2.4.0]

- Improvements
 - Removed deprecated APIs.

[2.3.8]

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

[2.3.7]

- Improvements
 - Change feature “FSL_FEATURE_SAI_FIFO_COUNT” to “FSL_FEATURE_SAI_HAS_FIFO”.
 - Added feature “FSL_FEATURE_SAI_FIFO_COUNTn(x)” to align SAI fifo count function with IP in function

[2.3.6]

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

[2.3.5]

- Improvements
 - Make driver to be aarch64 compatible.

[2.3.4]

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

[2.3.3]

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

[2.3.2]

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

[2.3.1]

- Bug Fixes
 - Corrected the peripheral name in function SAI0_DriverIRQHandler.
 - Fixed violations of MISRA C-2012 rule 17.7.

[2.3.0]

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

[2.2.3]

- Bug Fixes
 - Corrected the peripheral name in function SAI0_DriverIRQHandler.

[2.2.2]

- Bug Fixes
 - Fixed the issue of MISRA 2004 rule 9.3.
 - Fixed sign-compare warning.
 - Fixed the PA082 build warning.
 - Fixed sign-compare warning.
 - Fixed violations of MISRA C-2012 rule 10.3,17.7,10.4,8.4,10.7,10.8,14.4,17.7,11.6,10.1,10.6,8.4,14.3,16.4,18.4
 - Allow to reset Rx or Tx FIFO pointers only when Rx or Tx is disabled.
- Improvements
 - Added 24bit raw audio data width support in sai sdma driver.
 - Disabled the interrupt/DMA request in the SAI_Init to avoid generates unexpected sai FIFO requests.

[2.2.1]

- Improvements
 - Added mclk post divider support in function SAI_SetMasterClockDivider.
 - Removed useless configuration code in SAI_RxSetSerialDataConfig.
- Bug Fixes
 - Fixed the SAI SDMA driver build issue caused by the wrong structure member name used in the function SAI_TransferRxSetConfigSDMA/SAI_TransferTxSetConfigSDMA.
 - Fixed BAD BIT SHIFT OPERATION issue caused by the FSL_FEATURE_SAI_CHANNEL_COUNTn.
 - Applied ERR05144: not set FCONT = 1 when TMR > 0, otherwise the TX may not work.

[2.2.0]

- Improvements
 - Added new APIs for parameters collection and simplified user interfaces:
 - * SAI_Init
 - * SAI_SetMasterClockConfig
 - * SAI_TxSetBitClockRate
 - * SAI_TxSetSerialDataConfig
 - * SAI_TxSetFrameSyncConfig
 - * SAI_TxSetFifoConfig
 - * SAI_TxSetBitclockConfig
 - * SAI_TxSetConfig
 - * SAI_TxSetTransferConfig
 - * SAI_RxSetBitClockRate
 - * SAI_RxSetSerialDataConfig
 - * SAI_RxSetFrameSyncConfig
 - * SAI_RxSetFifoConfig

- * SAI_RxSetBitclockConfig
- * SAI_RXSetConfig
- * SAI_RxSetTransferConfig
- * SAI_GetClassicI2SConfig
- * SAI_GetLeftJustifiedConfig
- * SAI_GetRightJustifiedConfig
- * SAI_GetTDMConfig

[2.1.9]

- Improvements
 - Improved SAI driver comment for clock polarity.
 - Added enumeration for SAI for sample inputs on different edges.
 - Changed FSL_FEATURE_SAI_CHANNEL_COUNT to FSL_FEATURE_SAI_CHANNEL_COUNTn(base) for the difference between the different SAI instances.
- Added new APIs:
 - SAI_TxSetBitClockDirection
 - SAI_RxSetBitClockDirection
 - SAI_RxSetFrameSyncDirection
 - SAI_TxSetFrameSyncDirection

[2.1.8]

- Improvements
 - Added feature macro test for the sync mode2 and mode 3.
 - Added feature macro test for masterClockHz in sai_transfer_format_t.

[2.1.7]

- Improvements
 - Added feature macro test for the mclkSource member in sai_config_t.
 - Changed “FSL_FEATURE_SAI5_SAI6_SHARE_IRQ” to “FSL_FEATURE_SAI_SAI5_SAI6_SHARE_IRQ”.
 - Added #ifndef #endif check for SAI_XFER_QUEUE_SIZE to allow redefinition.
- Bug Fixes
 - Fixed build error caused by feature macro test for mclkSource.

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

- Improvements
 - API name changes:
 - * SAI_GetSendRemainingBytes -> SAI_GetSentCount.
 - * SAI_GetReceiveRemainingBytes -> SAI_GetReceivedCount.
 - * All names of transactional APIs were added with “Transfer” prefix.
 - * All transactional APIs use base and handle as input parameter.
 - * Unified the parameter names.

- Bug Fixes
 - Fixed WLC bug while reading TCSR/RCSR registers.
 - Fixed MOE enable flow issue. Moved MOE enable after MICS settings in SAI_TxInit/SAI_RxInit.

[2.0.0]

- Initial version.
-

SAI_EDMA

[2.7.3]

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

[2.7.2]

- Improvements
 - Add macros `MCUX_SDK_SAI_EDMA_TX_ENABLE_INTERNAL` and `MCUX_SDK_SAI_EDMA_RX_ENABLE_INTERNAL` to let the user decide whether to enable SAI when calling `SAI_TransferSendEDMA/SAI_TransferReceiveEDMA`.

[2.7.1]

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

[2.7.0]

- Improvements
 - Updated api `SAI_TransferReceiveEDMA` to support voice channel block interleave transfer.
 - Updated api `SAI_TransferSendEDMA` to support voice channel block interleave transfer.
 - Added new api `SAI_TransferSetInterleaveType` to support channel interleave type configurations.

[2.6.0]

- Improvements
 - Removed deprecated APIs.

[2.5.1]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 20.7.

[2.5.0]

- Improvements
 - Added new api SAI_TransferSendLoopEDMA/SAI_TransferReceiveLoopEDMA to support loop transfer.
 - Added multi sai channel transfer support.

[2.4.0]

- Improvements
 - Added new api SAI_TransferGetValidTransferSlotsEDMA which can be used to get valid transfer slot count in the sai edma transfer queue.
 - Deprecated the api SAI_TransferRxSetFormatEDMA and SAI_TransferTxSetFormatEDMA.
- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.3,10.4.

[2.3.2]

- Refer SAI driver change log 2.1.0 to 2.3.2
-

SAR_ADC

[2.3.0]

- New Feature
 - Added new feature macro a for compatibility with ADCs on some platforms where some instances do not support group3.

[2.2.0]

- New Feature
 - Added new features to compatible with new platforms.

[2.1.1]

- Improvement
 - Change ADC sample rate phase duration default value from 0x08 to 0x14.

[2.1.0]

- New Feature
 - Added ADC_StopConvChain function to support stop scan in normal conversion scan operation mode.

[2.0.3]

- Bug Fixes
 - Fixed the array name usage error in function ADC_GetInstance.

[2.0.2]

- Bug Fixes
 - Fixed MISRA issues.

[2.0.1]

- Bug Fixes
 - Fixed the bug that when calling function `ADC_EnableWdgThresholdInt()` in function `ADC_SetAnalogWdgConfig()`, the parameter was passed incorrectly.

[2.0.0]

- Initial version.
-

SEMA42**[2.1.1]**

- Improvements
 - Updated `SEMA42_TryLock` function to avoid unsigned integer operations wrap issue.

[2.1.0]

- New Features
 - Added `SEMA42_BUSY_POLL_COUNT` parameter to prevent infinite polling loops in SEMA42 operations.
 - Added timeout mechanism to all polling loops in SEMA42 driver code.
- Improvements
 - Updated `SEMA42_Lock` function to return `status_t` instead of void for better error handling.
 - Enhanced documentation to clarify timeout behavior and return values.

[2.0.4]

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

[2.0.3]

- Improvements
 - Changed to implement `SEMA42_Lock` base on `SEMA42_TryLock`.

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

TPM

[2.4.1]

- Improvements
 - Add Coverage Justification for uncovered code.

[2.4.0]

- New Feature
 - Added while loop timeout for MOD CnV CnSC and SC register write sequence.
 - Change the return type from void to status_t for following API:
 - * TPM_DisableChannel
 - * TPM_EnableChannel
 - * TPM_SetupOutputCompare
 - * TPM_SetTimerPeriod
 - * TPM_StopTimer

[2.3.6]

- Bug Fixes
 - Fixed CERT INT30-C INT31-C issue for TPM_SetupDualEdgeCapture.

[2.3.5]

- New Feature
 - Added IRQ handler entry for TPM2.

[2.3.4]

- New Feature
 - Added common IRQ handler entry TPM_DriverIRQHandler.

[2.3.3]

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

[2.3.2]

- Bug Fixes
 - Fixed ERR008085 TPM writing the TPMx_MOD or TPMx_CnV registers more than once may fail when the timer is disabled.

[2.3.1]

- Bug Fixes
 - Fixed compilation error when macro FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL is 1.

[2.3.0]

- Improvements
 - Create callback feature for TPM match and timer overflow interrupts.

[2.2.4]

- Improvements
 - Add feature macros(FSL_FEATURE_TPM_HAS_GLOBAL_TIME_BASE_EN, FSL_FEATURE_TPM_HAS_GLOBAL_TIME_BASE_SYNC).

[2.2.3]

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

[2.2.2]

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

[2.2.1]

- Bug Fixes
 - Fixed CCM issue by splitting function from TPM_SetupPwm() function to reduce function complexity.
 - Fixed violations of MISRA C-2012 rule 17.7.

[2.2.0]

- Improvements
 - Added TPM_SetChannelPolarity to support select channel input/output polarity.
 - Added TPM_EnableChannelExtTrigger to support enable external trigger input to be used by channel.
 - Added TPM_CalculateCounterClkDiv to help calculates the counter clock prescaler.
 - Added TPM_GetChannelValue to support get TPM channel value.
 - Added new TPM configuration.

- * syncGlobalTimeBase
 - * extTriggerPolarity
 - * chnlPolarity
- Added new PWM signal configuration.
- * secPauseLevel
- Bug Fixes
 - Fixed TPM_SetupPwm can't configure 0% combined PWM issues.

[2.1.1]

- Improvements
 - Add feature macro for PWM pause level select feature.

[2.1.0]

- Improvements
 - Added TPM_EnableChannel and TPM_DisableChannel APIs.
 - Added new PWM signal configuration.
 - * pauseLevel - Support select output level when counter first enabled or paused.
 - * enableComplementary - Support enable/disable generate complementary PWM signal.
 - * deadTimeValue - Support deadtime insertion for each pair of channels in combined PWM mode.
- Bug Fixes
 - Fixed issues about channel MSnB:MSnA and ELSnB:ELSnA bit fields and CnV register change request acknowledgement. Writes to these bits are ignored when the interval between successive writes is less than the TPM clock period.

[2.0.8]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.1, 10.4, 10.7 and 14.4.

[2.0.7]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.4 and 17.7.

[2.0.6]

- Bug Fixes
 - Fixed Out-of-bounds issue.

[2.0.5]

- Bug Fixes
 - Fixed MISRA-2012 rules.
 - * Rule 10.6, 10.7

[2.0.4]

- Bug Fixes
 - Fixed ERR050050 in functions TPM_SetupPwm/TPM_UpdatePwmDutycycle. When TPM was configured in EPWM mode as PS = 0, the compare event was missed on the first reload/overflow after writing 1 to the CnV register.

[2.0.3]

- Bug Fixes
 - MISRA-2012 issue fixed.
 - * Fixed rules: rule-12.1, rule-17.7, rule-16.3, rule-14.4, rule-1.3, rule-10.4, rule-10.3, rule-10.7, rule-10.1, rule-10.6, and rule-18.1.

[2.0.2]

- Bug Fixes
 - Fixed issues in functions TPM_SetupPwm/TPM_UpdateChnEdgeLevelSelect/TPM_SetupInputCapture/TPM_SetupOutputCompare/TPM_SetupDualEdgeCapture, wait acknowledgement when the channel is disabled.

[2.0.1]

- Bug Fixes
 - Fixed TPM_UpdateChnEdgeLevelSelect ACK wait issue.
 - Fixed the issue that TPM_SetupDualEdgeCapture could not set FILTER register.
 - Fixed TPM_UpdateChnEdgeLevelSelect ACK wait issue.

[2.0.0]

- Initial version.
-

TRDC**[2.2.2]**

- Improvements
 - Update APIs to check whether the memory access configuration can be updated.
 - Update APIs to mask the MRC address since only high 18 bits are valid.

[2.2.1]

- Bug Fixes:
 - Fix MISRA violations.

[2.2.0]

- New Features:
 - Supported SoCs that do not have all TRDC modules.

[2.1.0]

- Bug Fixes:
 - Fix MISRA violations.
 - Fixed wrong operation of domain mask in TRDC_MbcNseClearAll and TRDC_MrcDomainNseClear.

[2.0.0]

- Initial version.
-

TSTMR

[2.1.0]

- New Features
 - Support configured clock frequency.
 - Add TSTMR_Init and TSTMR_init APIs.
- Improvements
 - Change TSTMR_DelayUs from static inline function to normal function.

[2.0.4]

- Bugfix
 - Fix MISRA C-2012 Rule 10.4 and 14.4 issues.

[2.0.3]

- Bugfix
 - Fix CERT INT30-C that Unsigned integer operation TSTMR_ReadTimeStamp(base) - startTime may wrap.

[2.0.2]

- Improvements
 - Support 24MHz clock source.
- Bugfix
 - Fix MISRA C-2012 Rule 10.4 issue.
 - Read of TSTMR HIGH must follow TSTMR LOW atomically: require masking interrupt around 2 LSB / MSB accesses.

[2.0.1]

- Bugfix
 - Restrict to read with 32-bit accesses only.
 - Restrict that TSTMR LOW read occurs first, followed by the TSTMR HIGH read.

[2.0.0]

- Initial version.
-

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.

[MIMX9596](#)

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 FreeRTOS

[FreeRTOS](#)

1.7.3 lwIP

lwip

Chapter 2

MIMX9596

2.1 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.

2.2 CAMERA MIX CSR: Camera Domain Block Control

uint32_t CAMERACSR_GetPixelDataIndex(*csi2rx_payload_t* datatype)

Get CAMERA CSR Pixel Data type index. This function get pinxe index by data type.

Parameters

- camera_csr – BLK_CTRL_CAMERAMIX module periperal address.

void CAMERACSR_PixelFormatting(BLK_CTRL_CAMERAMIX_Type *camera_csr,
camera_csr_pixel_formatting_config_t *config)

introduce function CAMERACSR_PixelFormatting. This function control the data process channel from CSI host to ISI pixel link module.

Parameters

- camera_csr – BLK_CTRL_CAMERAMIX module periperal address.
- config – pixel link module camera_csr formatting configuration structure.

FSL_Camera_Csr_DRIVER_VERSION

Camera Csr driver version.

typedef struct camera_csr_pixel_formatting_config camera_csr_pixel_formatting_config_t
CAMERA CSR configuration.

typedef struct pixel_link_transfer_pixel_data pixel_link_transfer_pixel_data_t

bool enablePixelDataRoute

whether enable pixel data route to a new channel.

bool enableNonPixelDataRoute

whether enable non-pixel data route to a new channel.

bool enableRAW32

whether enable RAW32 mode for specific channel which up to 4 pixels per clock cycle can be transported.

csi2rx_payload_t PixelDataType
transport pixel data type.

uint32_t NonPixelDataType
transport non-pixel data type.

uint8_t PixelDataNewVc
new virtual channel on which the pixel data are transported.

uint8_t NonPixelDataNewVc
new virtual channel on which the non pixel data are transported.

uint32_t mipiVc
virtual channel from csi host idi interface.

uint8_t csiinterface
csi ininterface number

csi2rx_payload_t datatype

uint8_t index

struct camera_csr_pixel_formatting_config
#include <fsl_camera_csr.h> CAMERA CSR configuration.

struct pixel_link_transfer_pixel_data
#include <fsl_camera_csr.h>

2.3 Clock Driver

2.4 Csi2rx

enum __csi2rx__data_lane

CSI2RX data lanes.

Values:

enumerator kCSI2RX_DataLane0

Data lane 0.

enumerator kCSI2RX_DataLane1

Data lane 1.

enumerator kCSI2RX_DataLane2

Data lane 2.

enumerator kCSI2RX_DataLane3

Data lane 3.

enum __csi2rx__payload

CSI2RX payload type.

Values:

enumerator kCSI2RX_DataTypeFS

Frame Start.

enumerator kCSI2RX_DataTypeFE

Frame End.

enumerator kCSI2RX_DataTypeLS

Line Start.

enumerator kCSI2RX_DataTypeLE

Line End.

enumerator kCSI2RX_DataTypeEOT

End of transmission.

enumerator kCSI2RX_DataTypeGeneric1

Data type generic short 1.

enumerator kCSI2RX_DataTypeGeneric2

Data type generic short 2.

enumerator kCSI2RX_DataTypeGeneric3

Data type generic short 3.

enumerator kCSI2RX_DataTypeGeneric4

Data type generic short 4.

enumerator kCSI2RX_DataTypeGeneric5

Data type generic short 5.

enumerator kCSI2RX_DataTypeGeneric6

Data type generic short 6.

enumerator kCSI2RX_DataTypeGeneric7

Data type generic short 7.

enumerator kCSI2RX_DataTypeGeneric8
Data type generic short 8.

enumerator kCSI2RX_DataTypeNULL
NULL.

enumerator kCSI2RX_DataTypeBlanking
Blanking.

enumerator kCSI2RX_DataTypeEmbedded
Embedded.

enumerator kCSI2RX_DataTypeYUV420_8Bit
YUV420 8 bit.

enumerator kCSI2RX_DataTypeYUV420_10Bit
YUV420 10 bit.

enumerator kCSI2RX_DataTypeYUV420_8BitLegacy
Legacy YUV420 8 bit.

enumerator kCSI2RX_DataTypeYUV420_8BitCS
YUV420 8 bit CS.

enumerator kCSI2RX_DataTypeYUV420_10BitCS
YUV420 10 bit CS.

enumerator kCSI2RX_DataTypeYUV422_8Bit
YUV422 8 bit.

enumerator kCSI2RX_DataTypeYUV422_10Bit
YUV422 10 bit.

enumerator kCSI2RX_DataTypeRGB444
RGB444.

enumerator kCSI2RX_DataTypeRGB555
RGB555.

enumerator kCSI2RX_DataTypeRGB565
RGB565.

enumerator kCSI2RX_DataTypeRGB666
RGB666.

enumerator kCSI2RX_DataTypeRGB888
RGB888.

enumerator kCSI2RX_DataTypeRAW28
RAW28.

enumerator kCSI2RX_DataTypeRAW24
RAW24.

enumerator kCSI2RX_DataTypeRAW6
RAW6.

enumerator kCSI2RX_DataTypeRAW7
RAW7.

enumerator kCSI2RX_DataTypeRAW8
RAW8.

enumerator kCSI2RX_DataTypeRAW10
RAW10.

enumerator kCSI2RX_DataTypeRAW12
RAW12.

enumerator kCSI2RX_DataTypeRAW14
RAW14.

enumerator kCSI2RX_DataTypeRAW16
RAW16.

enumerator kCSI2RX_DataTypeRAW20
RAW20.

enumerator kCSI2RX_DataTypeUserDefined1
User defined 8-bit data type 1.

enumerator kCSI2RX_DataTypeUserDefined2
User defined 8-bit data type 2.

enumerator kCSI2RX_DataTypeUserDefined3
User defined 8-bit data type 3.

enumerator kCSI2RX_DataTypeUserDefined4
User defined 8-bit data type 4.

enumerator kCSI2RX_DataTypeUserDefined5
User defined 8-bit data type 5.

enumerator kCSI2RX_DataTypeUserDefined6
User defined 8-bit data type 6.

enumerator kCSI2RX_DataTypeUserDefined7
User defined 8-bit data type 7.

enumerator kCSI2RX_DataTypeUserDefined8
User defined 8-bit data type 8.

enum _csi2rx_bit_error
MIPI CSI2RX bit errors.

Values:

enumerator kCSI2RX_BitErrorEccTwoBit
ECC two bit error has occurred.

enumerator kCSI2RX_BitErrorEccOneBit
ECC one bit error has occurred.

enum _csi2rx_ppi_error
MIPI CSI2RX PPI error types.

Values:

enumerator kCSI2RX_PpiErrorSotHs
CSI2RX DPHY PPI error ErrSotHS.

enumerator kCSI2RX_PpiErrorSotSyncHs
CSI2RX DPHY PPI error ErrSotSync_HS.

enumerator kCSI2RX_PpiErrorEsc
CSI2RX DPHY PPI error ErrEsc.

enumerator kCSI2RX_PpiErrorSyncEsc
CSI2RX DPHY PPI error ErrSyncEsc.

enumerator kCSI2RX_PpiErrorControl
CSI2RX DPHY PPI error ErrControl.

enum _csi2rx_interrupt
MIPI CSI2RX interrupt.

Values:

enumerator kCSI2RX_InterruptphyFatal

enumerator kCSI2RX_InterruptPktFatal

enumerator kCSI2RX_InterruptBndryFrameFatal

enumerator kCSI2RX_InterruptSeqFrameFatal

enumerator kCSI2RX_InterruptCrcFrameFatal

enumerator kCSI2RX_InterruptDataId

enumerator kCSI2RX_InterruptEccCorrected

enumerator kCSI2RX_InterruptStPhy

enumerator kCSI2RX_InterruptStline

typedef enum _csi2rx_payload csi2rx_payload_t
CSI2RX payload type.

typedef struct _pg_pattern_config pg_pattern_config_t
CSI2RX pattern injection configuration.

typedef struct _csi2rx_config csi2rx_config_t
CSI2RX configuration.

typedef enum _csi2rx_ppi_error csi2rx_ppi_error_t
MIPI CSI2RX PPI error types.

uint32_t CSI2RX_GetInstance(CAMERA_MIPI_CSI2_Type *base)
Get the CSI instance from peripheral base address.

Parameters

- base – CSI peripheral base address.

Returns

CSI instance.

void MIPI_CSI2RX_Startup(CAMERA_MIPI_CSI2_Type *base)
introduce function MIPI_CSI2RX_Startup. This function start up the CSI host controller.

Parameters

- base – CSI2RX peripheral address.

status_t MIPI_CSI2RX_InitInterface(CAMERA_MIPI_CSI2_Type *base,
CAMERA_DSI_OR_CSI_PHY_CSR_Type *phybase,
csi2rx_config_t *config)

introduce function MIPI_CSI2RX_InitInterface. This function deal with CSI and PHY initialization.

Parameters

- base – CSI2RX peripheral address.

- phybase – PHY module peripheral address.
- config – CSI2RX module configuration structure.

```
status_t MIPI_CSI2RX_Init(CAMERA_MIPI_CSI2_Type *base,
                        CAMERA_DSI_OR_CSI_PHY_CSR_Type *phybase, csi2rx_config_t
                        *config)
```

introduce function MIPI_CSI2RX_Init. The CSI host interface is basically configured and ready to receive sensor data after this function.

Parameters

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

```
void CSI2RX_Deinit(CAMERA_MIPI_CSI2_Type *base)
```

introduce function CSI2RX_Deinit. This function disables the CSI2 host and PHY module.

Parameters

- base – CSI2RX peripheral address.

```
FSL_CSI2RX_DRIVER_VERSION
CSI2RX driver version.
```

```
uint32_t pattern_vertical
Number of pattern vertical size.
```

```
uint32_t pattern_horizontal
Number of pattern horizontal size.
```

```
uint8_t pattern_data_type
Number of sent pattern data type.
```

```
uint32_t pattern_format
Number of pattern format.
```

```
uint32_t laneNum
Number of active lanes used for receiving data.
```

```
uint32_t vcnun
Number of used CSI host interface vittual channel.
```

```
csi2rx_payload_t datatype
Number of csi host channel received data type from sensor.
```

```
bool pg_enable
Whether enable CSI host pattern generator, CSI will halt camera received data if enable it.
Default disabled
```

```
pg_pattern_config_t pg_pattern
Number of active lanes used for receiving data.
```

```
uint32_t cfgclkfreqrange
Number of DPHY clock frequency.
```

```
uint32_t hsfreqrange
Number of DPHY frequency range, a lane operation range from 8Mbps to ?.
```

```
struct __pg_pattern_config
#include <fsl_dwc_mipi_csi2rx.h> CSI2RX pattern injection configuration.
```

```
struct __csi2rx_config
#include <fsl_dwc_mipi_csi2rx.h> CSI2RX configuration.
```

2.5 Dpu

void DPU_Init(DISPLAY_SEERIS_Type *base)

Initializes the DPU peripheral.

This function ungates the DPU clock.

Parameters

- *base* – DPU peripheral base address.

void DPU_Deinit(DISPLAY_SEERIS_Type *base)

Deinitializes the DPU peripheral.

This function gates the DPU clock.

Parameters

- *base* – DPU peripheral base address.

void DPU_PreparePathConfig(DISPLAY_SEERIS_Type *base)

Prepare the unit path configuration.

The DPU has a default path configuration. Before changing the configuration, this function could be used to break all the original path. This make sure one pixel engine unit is not used in multiple pipelines.

Parameters

- *base* – DPU peripheral base address.

void DPU_EnableInterrupts(DISPLAY_SEERIS_Type *base, uint8_t group, uint32_t mask)

brief Enable the selected DPU interrupts.

For example, to enable Store9 shadow load interrupt and Store9 frame complete interrupt, use like this:

```
code DPU_EnableInterrupts(DPU, 0, kDPU_Group0Store9ShadowLoadInterrupt |  
kDPU_Group0Store9FrameCompleteInterrupt);endcode
```

param *base* DPU peripheral base address. param *group* Interrupt group index. param *mask* The interrupts to enable, this is a logical OR of members in *ref_dpu_interrupt*. note Only the members in the same group could be OR'ed, at the same time, the parameter *p_group* should be passed in correctly.

status_t DPU_EnableShadowLoad(DISPLAY_SEERIS_Type *base, dpu_unit_t unit, bool enable)

Enable or disable the register shadowing for the DPU process units.

For example, to enable the shadowing of all RWS registers of the pipeline with endpoint Store9.

```
DPU_EnableShadowLoad(DPU, kDPU_PipelineStore9, true);
```

Parameters

- *base* – DPU peripheral base address.
- *unit* – The unit whose shadow load to enable or disable, see *dpu_unit_t*.
- *enable* – True to enable, false to disable.

Return values

- *kStatus_Success* – The shadow load is enabled or disabled successfully.
- *kStatus_InvalidArgument* – The unit does not support shadow load.

void DPU_SetUnitSrc(DISPLAY_SEERIS_Type *base, *dpu_unit_t* unit, uint32_t srcReg)

Set the DPU unit input source selection.

Sets the DPU unit input source, the input source is controlled by the register <unit>_dynamic in “Pixel Engin Top Level”. This function writes the register <unit>_dynamic directly, please check the reference manual for the register details. This function only changes the input source control bits in register.

Parameters

- base – DPU peripheral base address.
- unit – The DPU pipeline unit.
- srcReg – The value written to register <unit>_dynamic. Could be generated using DPU_MAKE_SRC_REG1, DPU_MAKE_SRC_REG2, and DPU_MAKE_SRC_REG3.

void DPU_InitPipeline(DISPLAY_SEERIS_Type *base, *dpu_unit_t* unit)

Initialize the pipeline.

Parameters

- base – DPU peripheral base address.
- unit – The DPU pipeline unit.

void DPU_DeinitPipeline(DISPLAY_SEERIS_Type *base, *dpu_unit_t* unit)

Deinitializes the pipeline.

Power down the pipeline and disable the shadow load feature.

Parameters

- base – DPU peripheral base address.
- unit – The DPU pipeline unit.

void DPU_TriggerPipelineShadowLoad(DISPLAY_SEERIS_Type *base, *dpu_unit_t* unit)

Trigger the pipeline shadow load.

This function triggers the pipeline reconfiguration.

Parameters

- base – DPU peripheral base address.
- unit – The DPU pipeline unit.

void DPU_DstBufferGetDefaultConfig(*dpu_dst_buffer_config_t* *config)

Get the default configuration for Store unit.

The default value is:

```
config->baseAddr = 0U;
config->strideBytes = 0x500U;
config->bitsPerPixel = 32U,
config->pixelFormat = kDPU_PixelFormatARGB8888;
config->bufferHeight = 0U;
config->bufferWidth = 0U;
```

Parameters

- config – Pointer to the configuration.

void DPU_InitStore(DISPLAY_SEERIS_Type *base, *dpu_unit_t* unit, uint32_t srcReg)

brief Initialize the Store unit.

The valid input source of the store unit could be:

- ref kDPU_UnitSrcNone
- ref kDPU_UnitSrcHScaler9
- ref kDPU_UnitSrcVScaler9
- ref kDPU_UnitSrcFilter9
- ref kDPU_UnitSrcBlitBlend9
- ref kDPU_UnitSrcFetchDecode9
- ref kDPU_UnitSrcFetchRot9

param base DPU peripheral base address. param unit DPU unit, see ref dpu_unit_t, must be Store unit here. param srcReg Input source select register value, pix-encfg_extdstX_dynamic see ref DPU_MAKE_SRC_REG1.

status_t DPU_SetStoreDstBufferConfig(DISPLAY_SEERIS_Type *base, *dpu_unit_t* unit, const *dpu_dst_buffer_config_t* *config)

Set the Store unit Destination buffer configuration.

Parameters

- base – DPU peripheral base address.
- unit – DPU unit, see dpu_unit_t, must be Store unit here.
- config – Pointer to the configuration.

Return values

- kStatus_Success – Initialization success.
- kStatus_InvalidArgument – Wrong argument.

void DPU_StartStore(DISPLAY_SEERIS_Type *base, *dpu_unit_t* unit)

Start the Store unit.

This function starts the Store unit to save the frame to output buffer. When the frame store completed, the interrupt flag kDPU_Group0Store9FrameCompleteInterrupt asserts.

This is an example shows how to use Store unit:

```
Initialize the Store unit, use FetchDecode9 output as its input.
DPU_InitStore(DPU, kDPU_Store9, DPU_MAKE_SRC_REG1(kDPU_UnitSrcFetchDecode9));

Configure the Store unit output buffer.
DPU_SetStoreDstBufferConfig(DPU, kDPU_Store9, &DstBufferConfig);

Configure FetchDecode9 unit, including source buffer setting and so on.
...

Initialize the Store9 pipeline
DPU_InitPipeline(DPU, kDPU_PipelineStore9);

DPU_ClearUserInterruptsPendingFlags(DPU, kDPU_Group0Store9ShadowLoadInterrupt);

Trigger the shadow load
DPU_TriggerPipelineShadowLoad(DPU, kDPU_PipelineStore9);

DPU_ClearUserInterruptsPendingFlags(DPU, kDPU_Group0Store9FrameCompleteInterrupt);

Start the Store9 to convert and output.
DPU_StartStore(DPU, kDPU_Store9);

Wait for Store 9 completed, this could also be monitored by interrupt.
while (!(kDPU_Group0Store9FrameCompleteInterrupt & DPU_GetUserInterruptsPendingFlags(DPU, kDPU_Group0Store9FrameCompleteInterrupt)))
{
    // ...
}
```

(continues on next page)

(continued from previous page)

```

    ↪0))
{
}

```

For better performance, it is allowed to set next operation while current is still in progress. Upper layer could set next operation immediately after shadow load finished.

Parameters

- base – DPU peripheral base address.
- unit – DPU unit, see `dpu_unit_t`, must be Store unit here.

`void DPU_InitBlitBlend(DISPLAY_SEERIS_Type *base, dpu_unit_t unit, uint32_t srcReg)`

brief Initialize the BlitBlend unit.

The valid input primary source could be:

- ref `kDPU_UnitSrcNone`
- ref `kDPU_UnitSrcHScaler9`
- ref `kDPU_UnitSrcVScaler9`
- ref `kDPU_UnitSrcFilter9`
- ref `kDPU_UnitSrcRop9`

The valid input secondary source could be:

- ref `kDPU_UnitSrcNone`
- ref `kDPU_UnitSrcFetchDecode9`
- ref `kDPU_UnitSrcFetchRot9`

param base DPU peripheral base address. param unit DPU unit, see ref `dpu_unit_t`, must be BlitBlend unit here. param srcReg Unit source selection, see ref `DPU_MAKE_SRC_REG2`.

`void DPU_EnableBlitBlend(DISPLAY_SEERIS_Type *base, dpu_unit_t unit, bool enable)`

brief Enable or disable the BlitBlend unit.

The BlitBlend unit could be runtime enabled or disabled, when disabled, the primary input is output directly.

param base DPU peripheral base address. param unit DPU unit, see ref `dpu_unit_t`, must be BlitBlend unit here. param enable Pass true to enable, false to disable.

`void DPU_LayerBlendGetDefaultConfig(dpu_layer_blend_config_t *config)`

Get default configuration structure for LayerBlend.

The default value is:

```

config->constAlpha = 0U;
config->secAlphaBlendMode = kDPU_BlendOne;
config->primAlphaBlendMode = kDPU_BlendZero;
config->secColorBlendMode = kDPU_BlendOne;
config->primColorBlendMode = kDPU_BlendZero;
config->enableAlphaMask = true;
config->alphaMaskMode = kDPU_AlphaMaskPrim;

```

Parameters

- config – Pointer to the configuration structure.

void DPU__InitLayerBlend(DISPLAY_SEERIS_Type *base, *dpu_unit_t* unit, uint32_t srcReg)

brief Initialize the LayerBlend.

The valid primary source:

- ref kDPU_UnitSrcNone
- ref kDPU_UnitSrcConstFrame0
- ref kDPU_UnitSrcConstFrame1
- ref kDPU_UnitSrcConstFrame4
- ref kDPU_UnitSrcConstFrame5

The valid secondary source:

- ref kDPU_UnitSrcNone
- ref kDPU_UnitSrcHScaler4
- ref kDPU_UnitSrcVScaler4
- ref kDPU_UnitSrcMatrix4
- ref kDPU_UnitSrcFetchRot9
- ref kDPU_UnitSrcFetchlayer0
- ref kDPU_UnitSrcFetchlayer1
- ref kDPU_UnitSrcFetchYuv0-3

param base DPU peripheral base address. param unit DPU unit, see ref *dpu_unit_t*, must be LayerBlend unit here. param srcReg Unit source selection, see ref DPU_MAKE_SRC_REG2.

void DPU__SetLayerBlendConfig(DISPLAY_SEERIS_Type *base, *dpu_unit_t* unit, const *dpu_layer_blend_config_t* *config)

Set the LayerBlend unit configuration.

Parameters

- base – DPU peripheral base address.
- unit – DPU unit, see *dpu_unit_t*, must be LayerBlend unit here.
- config – Pointer to the configuration structure.

void DPU__EnableLayerBlend(DISPLAY_SEERIS_Type *base, *dpu_unit_t* unit, bool enable)

Enable or disable the LayerBlend unit.

If enabled, the blend result is output, otherwise, the primary input is output.

Parameters

- base – DPU peripheral base address.
- unit – DPU unit, see *dpu_unit_t*, must be LayerBlend unit here.
- enable – Pass true to enable, false to disable.

void DPU__InitConstFrame(DISPLAY_SEERIS_Type *base, *dpu_unit_t* unit)

Initialize the ConstFrame unit.

Parameters

- base – DPU peripheral base address.
- unit – DPU unit, see *dpu_unit_t*, must be ConstFrame unit here.

void DPU_ConstFrameGetDefaultConfig(*dpu_const_frame_config_t* *config)

Get default configuration structure for ConstFrame unit.

The default value is:

```
config->frameHeight = 320U;
config->frameWidth = 480U;
config->constColor = DPU_MAKE_CONST_COLOR(0xFF, 0xFF, 0xFF, 0xFF);
```

Parameters

- config – Pointer to the configuration structure.

void DPU_SetConstFrameConfig(DISPLAY_SEERIS_Type *base, *dpu_unit_t* unit, const *dpu_const_frame_config_t* *config)

Set the ConstFrame unit configuration.

Parameters

- base – DPU peripheral base address.
- unit – DPU unit, see *dpu_unit_t*, must be ConstFrame unit here.
- config – Pointer to the configuration structure.

void DPU_InitRop(DISPLAY_SEERIS_Type *base, *dpu_unit_t* unit, uint32_t srcReg)

brief Initialize the ROp unit.

The primary input source of the unit could be:

- ref kDPU_UnitSrcNone
- ref kDPU_UnitSrcFetchDecode9
- ref kDPU_UnitSrcFetchRot9

The secondary input source of the unit could be:

- ref kDPU_UnitSrcNone
- ref kDPU_UnitSrcFetchEco9

The tert input source of the unit could be:

- ref kDPU_UnitSrcNone
- ref kDPU_UnitSrcFetchDecode9
- ref kDPU_UnitSrcFetchRot9

param base DPU peripheral base address. param unit DPU unit, see ref *dpu_unit_t*, must be Rop unit here. param srcReg Unit source selection, see ref DPU_MAKE_SRC_REG3.

void DPU_RopGetDefaultConfig(*dpu_rop_config_t* *config)

brief Get the default ROp unit configuration.

The default configuration is:

code config->controlFlags = 0U; config->alphaIndex = 0U; config->blueIndex = 0U; config->greenIndex = 0U; config->redIndex = 0U;endcode param config Pointer to the configuration structure.

void DPU_SetRopConfig(DISPLAY_SEERIS_Type *base, *dpu_unit_t* unit, const *dpu_rop_config_t* *config)

brief Set the ROp unit configuration.

param base DPU peripheral base address. param unit DPU unit, see ref *dpu_unit_t*, must be Rop unit here. param config Pointer to the configuration structure.

void DPU_EnableRop(DISPLAY_SEERIS_Type *base, *dpu_unit_t* unit, bool enable)

brief Enable or disable the ROp unit.

If disabled, only the primary input is output.

param base DPU peripheral base address. param unit DPU unit, see ref *dpu_unit_t*, must be Rop unit here. param enable Pass true to enable, false to disable.

void DPU_FetchUnitGetDefaultConfig(*dpu_fetch_unit_config_t* *config)

Get the default configuration for fetch unit.

The default value is:

```
config->srcReg = 0U;  
config->frameHeight = 320U;  
config->frameWidth = 480U;
```

Parameters

- config – Pointer to the configuration structure.

void DPU_SrcBufferGetDefaultConfig(*dpu_src_buffer_config_t* *config)

Get default configuration structure for fetch unit source buffer.

The default value is:

```
config->baseAddr = 0U;  
config->strideBytes = 0x500U;  
config->bitsPerPixel = 32U;  
config->pixelFormat = kDPU_PixelFormatARGB8888;  
config->bufferHeight = 0U;  
config->bufferWidth = 0U;  
config->constColor = DPU_MAKE_CONST_COLOR(0, 0, 0, 0);
```

Parameters

- config – Pointer to the configuration structure.

void DPU_InitFetchUnit(DISPLAY_SEERIS_Type *base, *dpu_unit_t* unit, const
dpu_fetch_unit_config_t *config)

brief Initialize the fetch unit.

This function initializes the fetch unit for the basic use, for other use case such as arbitrary warping, use the functions ref DPU_InitFetchUnitRot and ref DPU_InitWarpCoordinates.

The input source of fetch unit could be:

- ref kDPU_UnitSrcNone
- ref kDPU_UnitSrcFetchRot9
- ref kDPU_UnitSrcFetchEco2
- ref kDPU_UnitSrcFetchEco9
- ref kDPU_UnitSrcFetchEco0
- ref kDPU_UnitSrcFetchEco1
- ref kDPU_UnitSrcFetchYuv0

param base DPU peripheral base address. param unit DPU unit, see ref *dpu_unit_t*, must be fetch unit here. param config Pointer to the configuration structure.

status_t DPU_SetFetchUnitSrcBufferConfig(DISPLAY_SEERIS_Type *base, *dpu_unit_t* unit, uint8_t
sublayer, const *dpu_src_buffer_config_t* *config)

Set the fetch unit sublayer source buffer.

Parameters

- base – DPU peripheral base address.
- unit – DPU unit, see `dpu_unit_t`, must be fetch unit here.
- sublayer – Sublayer index, should be 0 to 7.
- config – Pointer to the configuration structure.

Return values

- `kStatus__Success` – Initialization success.
- `kStatus__InvalidArgument` – Wrong argument.

```
void DPU_SetFetchUnitOffset(DISPLAY_SEERIS_Type *base, dpu_unit_t unit, uint8_t sublayer,
                           uint16_t offsetX, uint16_t offsetY)
```

brief Set the fetch unit sublayer offset.

param base DPU peripheral base address. param unit DPU unit, see ref `dpu_unit_t`, must be fetch unit here. param sublayer Sublayer index, should be 0 to 7. param offsetX Horizontal offset. param offsetY Vertical offset.

```
void DPU_EnableFetchUnitSrcBuffer(DISPLAY_SEERIS_Type *base, dpu_unit_t unit, uint8_t
                                  sublayer, bool enable)
```

brief Enable or disable fetch unit sublayer source buffer.

param base DPU peripheral base address. param unit DPU unit, see ref `dpu_unit_t`, must be fetch unit here. param sublayer Sublayer index, should be 0 to 7. param enable True to enable, false to disable.

```
void DPU_SetFetchUnitClipColor(DISPLAY_SEERIS_Type *base, dpu_unit_t unit,
                               dpu_clip_color_mode_t clipColorMode, uint8_t sublayer)
```

brief Set the fetch unit clip color mode.

This function selects which color to take for pixels that do not lie inside the clip window of any layer.

param base DPU peripheral base address. param unit DPU unit, see ref `dpu_unit_t`, must be fetch unit here. param clipColorMode Select null color or use sublayer color. param sublayer Select which sublayer's color to use when p clipColorMode is ref `kDPU_ClipColorSublayer`.

```
void DPU_InitExtDst(DISPLAY_SEERIS_Type *base, dpu_unit_t unit, uint32_t srcReg)
```

brief Initialize the ExtDst unit.

param base DPU peripheral base address. param unit DPU unit, see ref `dpu_unit_t`, must be ExtDst unit here. param srcReg Input source selecte register value, `pix-encfg_extdstX_dynamic` see ref `DPU_MAKE_SRC_REG1`. The valid source:

- ref `kDPU_UnitSrcNone`
- ref `kDPU_UnitSrcLayerBlend1`
- ref `kDPU_UnitSrcLayerBlend2`
- ref `kDPU_UnitSrcLayerBlend3`
- ref `kDPU_UnitSrcLayerBlend4`
- ref `kDPU_UnitSrcLayerBlend5`
- ref `kDPU_UnitSrcLayerBlend6`

```
void DPU_SetStoreDstBufferAddr(DISPLAY_SEERIS_Type *base, dpu_unit_t unit, uint32_t
                               baseAddr)
```

brief Set the Store unit Destination buffer base address.

This function is run time used for better performance.

param base DPU peripheral base address. param unit DPU unit, see ref `dpu_unit_t`, must be Store unit here. param baseAddr Base address of the Destination buffer to set.

`void DPU_DisableInterrupts(DISPLAY_SEERIS_Type *base, uint8_t group, uint32_t mask)`

brief Disable the selected DPU interrupts.

For example, to disable Store9 shadow load interrupt and Store9 frame complete interrupt, use like this:

```
code DPU_DisableInterrupts(DPU, 0, kDPU_Group0Store9ShadowLoadInterrupt |  
kDPU_Group0Store9FrameCompleteInterrupt); endcode
```

param base DPU peripheral base address. param group Interrupt group index. param mask The interrupts to disable, this is a logical OR of members in ref `_dpu_interrupt`. note Only the members in the same group could be OR'ed, at the same time, the parameter p group should be passed in correctly.

`uint32_t DPU_GetInterruptsPendingFlags(DISPLAY_SEERIS_Type *base, uint8_t group)`

brief Get the DPU interrupts pending status.

The pending status are returned as mask.

param base DPU peripheral base address. param group Interrupt group index. return The interrupts pending status mask value, see ref `_dpu_interrupt`.

`void DPU_ClearInterruptsPendingFlags(DISPLAY_SEERIS_Type *base, uint8_t group, uint32_t mask)`

brief Clear the specified DPU interrupts pending status.

For example, to disable Store9 shadow load interrupt and Store9 frame complete interrupt pending status, use like this:

```
code DPU_ClearInterruptsPendingFlags(DPU, 0, kDPU_Group0Store9ShadowLoadInterrupt  
| kDPU_Group0Store9FrameCompleteInterrupt); endcode
```

param base DPU peripheral base address. param group Interrupt group index. param mask The interrupt pending flags to clear, this is a logical OR of members in ref `_dpu_interrupt`. note Only the members in the same group could be OR'ed, at the same time, the parameter p group should be passed in correctly.

`void DPU_DisplayTimingGetDefaultConfig(dpu_display_timing_config_t *config)`

brief Get default configuration structure for display mode.

The default value is: code `config->flags = kDPU_DisplayDeActiveHigh; config->width = 320U; config->hsw = 32U; config->hfp = 8U; config->hbp = 40U; config->height = 240U; config->vsw = 4U; config->vfp = 13U; config->vbp = 6U;` endcode

param config Pointer to the configuration structure.

`void DPU_InitDisplayTiming(DISPLAY_SEERIS_Type *base, uint8_t displayIndex, const dpu_display_timing_config_t *config)`

brief Initialize the display timing.

param base DPU peripheral base address. param displayIndex Index of the display. param config Pointer to the configuration structure.

`void DPU_DisplayGetDefaultConfig(dpu_display_config_t *config)`

brief Get default configuration structure for display frame mode.

The default value is: `code config->enablePrimAlpha = false; config->enableSecAlpha = false; config->displayMode = kDPU_DisplayTest; config->enablePrimAlphaInPanic = false; config->enableSecAlphaInPanic = false; config->displayModeInPanic = kDPU_DisplayTest; config->constRed = 0x3FFU; config->constGreen = 0x3FFU; config->constBlue = 0x3FFU; config->constAlpha = 1U; config->primAreaStartX = 1U; config->primAreaStartY = 1U; config->secAreaStartX = 1U; config->secAreaStartY = 1U; endcode`

param config Pointer to the configuration structure.

```
void DPU_SetDisplayConfig(DISPLAY_SEERIS_Type *base, uint8_t displayIndex, const
                        dpu_display_config_t *config)
```

brief Set the display mode.

param base DPU peripheral base address. param displayIndex Index of the display. param config Pointer to the configuration structure.

```
void DPU_StartDisplay(DISPLAY_SEERIS_Type *base, uint8_t displayIndex)
```

brief Start the display.

param base DPU peripheral base address. param displayIndex Index of the display.

```
void DPU_StopDisplay(DISPLAY_SEERIS_Type *base, uint8_t displayIndex)
```

brief Stop the display.

This function stops the display and wait the sequence complete.

param base DPU peripheral base address. param displayIndex Index of the display.

```
void DPU_TriggerDisplayShadowLoad(DISPLAY_SEERIS_Type *base, uint8_t displayIndex)
```

brief Trigger the display stream shadow load token.

Trigger the display stream shadow load token, then the shadow register will be loaded at the begining of next frame.

param base DPU peripheral base address. param displayIndex Display index.

```
void DPU_SetFetchUnitSrcBufferAddr(DISPLAY_SEERIS_Type *base, dpu_unit_t unit, uint8_t
                                   sublayer, uint32_t baseAddr)
```

brief Set the fetch unit sublayer source buffer base address.

param base DPU peripheral base address. param unit DPU unit, see ref `dpu_unit_t`, must be fetch unit here. param sublayer Sublayer index, should be 0 to 7. param baseAddr Source buffer base address.

```
void DPU_InitDomainBlend(DISPLAY_SEERIS_Type *base, dpu_unit_t unit)
```

brief Initialize the Domainblend.

param base DPU peripheral base address. param unit DPU unit, see ref `dpu_unit_t`, must be DomainBlend unit here.

```
void DPU_TriggerDisplayDbShadowLoad(DISPLAY_SEERIS_Type *base, dpu_unit_t unit)
```

brief Trigger the display stream domainblend shadow load token.

Trigger the display stream shadow load token, then the shadow register will be loaded at the begining of next frame.

param base DPU peripheral base address. param unit DPU unit, see ref `dpu_unit_t`, must be DomainBlend unit here.

```
void DPU_InitScaler(DISPLAY_SEERIS_Type *base, dpu_unit_t unit)
```

Initialize the VScaler or HScaler unit.

Parameters

- base – DPU peripheral base address.
- unit – DPU unit, see `dpu_unit_t`, must be HScaler or VScaler unit here.

`void DPU_ScalerGetDefaultConfig(dpu_scaler_config_t *config)`

Get default configuration structure for VScaler and HScaler.

The default value is:

```
config->srcReg = 0U;  
config->inputSize = 0U;  
config->outputSize = 0U;
```

Parameters

- `config` – Pointer to the configuration structure.

`void DPU_SetScalerConfig(DISPLAY_SEERIS_Type *base, dpu_unit_t unit, const dpu_scaler_config_t *config)`

Set the VScaler or HScaler units configuration.

The valid input source could be:

- `kDPU_UnitSrcNone`
- `kDPU_UnitSrcFetchYuv0-3`
- `kDPU_UnitSrcMatrix4`
- `kDPU_UnitSrcVScaler4`
- `kDPU_UnitSrcHScaler4`
- `kDPU_UnitSrcVScaler9`
- `kDPU_UnitSrcHScaler9`
- `kDPU_UnitSrcFilter9`
- `kDPU_UnitSrcMatrix9`

Parameters

- `base` – DPU peripheral base address.
- `unit` – DPU unit, see `dpu_unit_t`, must be HScaler or VScaler unit here.
- `config` – Pointer to the configuration structure.

`void DPU_FetcUnitGetDefaultWarpConfig(dpu_warp_config_t *config)`

brief Get the default warp configuration for FetchWarp unit.

The default value is: `code config->srcReg = 0U; config->frameHeight = 320U; config->frameWidth = 480U; config->warpBitsPerPixel = 0U; config->enableSymmetricOffset = false; config->coordMode = kDPU_WarpCoordinateModePNT; config->arbStartX = 0U; config->arbStartY = 0U; config->arbDeltaYY = 0U; config->arbDeltaYX = 0U; config->arbDeltaXY = 0U; config->arbDeltaXX = 0U;endcode`

param `config` Pointer to the configuration structure.

`status_t DPU_InitFetchUnitWarp(DISPLAY_SEERIS_Type *base, dpu_unit_t unit, const dpu_warp_config_t *config)`

brief Initialize the Warp function for FetchRot unit.

This function initializes the FetchWarp unit for the arbitrary warping.

The valid source of fetch warp unit could be:

- `ref kDPU_UnitSrcNone`
- `ref kDPU_UnitSrcFetchEco9`
- `ref kDPU_UnitSrcFetchDecode9`

param base DPU peripheral base address. param unit DPU unit, see ref dpu_unit_t, must be FetchWarp unit here. param config Pointer to the configuration structure. retval kStatus_Success Initialization success. retval kStatus_InvalidArgument Wrong argument.

void DPU_CorrdinatesGetDefaultConfig(*dpu_coordinates_config_t* *config)

brief Get the default configuration structure for arbitrary warping re-sampling coordinates.

The default value is: code config->bitsPerPixel = 0U; config->strideBytes = 0x500U; config->baseAddr = 0U; config->frameHeight = 320U; config->frameWidth = 480U; endcode

param config Pointer to the configuration structure.

status_t DPU_InitWarpCoordinates(DISPLAY_SEERIS_Type *base, *dpu_unit_t* unit, const *dpu_coordinates_config_t* *config)

brief Initialize the arbitrary warping coordinates.

This function initializes the FetchEco unit, so that it could be used as the arbitrary warping coordinates.

param base DPU peripheral base address. param unit DPU unit, see ref dpu_unit_t, must be FetchEco here. param config Pointer to the configuration structure. retval kStatus_Success Initialization success. retval kStatus_InvalidArgument Wrong argument.

FSL_DPU_DRIVER_VERSION

Driver version.

DPU_PALETTE_ENTRY_NUM

DPU palette entery number.

DPU_FETCH_UNIT_BURST_LENGTH

DPU fetch unit burst length, should be less than 16.

DPU_FETCH_UNIT_BURST_SIZE

DPU fetch unit burst size.

If prefetch is used, the frame buffer stride and base address should be aligned to the burst size.

DPU_USE_GENERATE_HEADER

DPU_MAKE_SRC_REG1(src)

Macro for one input source unit.

DPU_MAKE_SRC_REG2(primSrc, secSrc)

Macro for two input source unit.

DPU_MAKE_SRC_REG3(primSrc, secSrc, tertSrc)

Macro for three input source unit.

Values:

enumerator kDPU_Pipeline

enumerator kDPU_BlitBlend

enumerator kDPU_Rop

enumerator kDPU_FetchRot

enumerator KDPU_FetchYuv

enumerator kDPU_FetchDecode

enumerator kDPU_FetchEco

enumerator kDPU__FetchLayer
enumerator kDPU__HScaler
enumerator kDPU__VScaler
enumerator kDPU__ConstFrame
enumerator kDPU__ExtDst
enumerator kDPU__LayerBlend
enumerator kDPU__Store
enumerator kDPU__DomainBlend

Values:

enumerator kDPU__UnitAttrIsFetch
enumerator kDPU__UnitAttrHasSrc
enumerator kDPU__UnitAttrNoShadow
enumerator kDPU__UnitAttrSubLayer

enum _dpu_unit

DPU units.

Values:

enumerator kDPU__PipelineStore9
enumerator kDPU__FetchDecode9
enumerator kDPU__FetchEco9
enumerator kDPU__HScaler9
enumerator kDPU__VScaler9
enumerator kDPU__FetchRot9
enumerator kDPU__Rop9
enumerator kDPU__BlitBlend9
enumerator kDPU__Store9
enumerator kDPU__PipelineExtDst0
enumerator kDPU__PipelineExtDst1
enumerator kDPU__PipelineExtDst4
enumerator kDPU__PipelineExtDst5
enumerator kDPU__ConstFrame0
enumerator kDPU__ExtDst0
enumerator kDPU__ConstFrame4
enumerator kDPU__ExtDst4

enumerator kDPU__ConstFrame1
enumerator kDPU__ExtDst1
enumerator kDPU__ConstFrame5
enumerator kDPU__ExtDst5
enumerator kDPU__FetchEco0
enumerator kDPU__FetchEco1
enumerator kDPU__FetchLayer0
enumerator kDPU__FetchLayer1
enumerator kDPU__FetchYuv0
enumerator kDPU__FetchYuv1
enumerator kDPU__FetchYuv3
enumerator kDPU__Hscaler4
enumerator kDPU__Vscaler4
enumerator kDPU__LayerBlend1
enumerator kDPU__LayerBlend2
enumerator kDPU__LayerBlend3
enumerator kDPU__LayerBlend4
enumerator kDPU__LayerBlend5
enumerator kDPU__LayerBlend6
enumerator kDPU__FetchYuv2
enumerator kDPU__FetchEco2
enumerator kDPU__DomainBlend0
enumerator kDPU__DomainBlend1

enum _dpu_interrupt

DPU interrupt.

Values:

enumerator kDPU__Group0Store9ShadowLoadInterrupt
Store9 shadow load interrupt.
enumerator kDPU__Group0Store9FrameCompleteInterrupt
Store9 frame complete interrupt.
enumerator kDPU__Group0Store9SeqCompleteInterrupt
Store9 sequence complete interrupt.
enumerator kDPU__Group0ExtDst0ShadowLoadInterrupt
ExtDst0 shadow load interrupt.
enumerator kDPU__Group0ExtDst0FrameCompleteInterrupt
ExtDst0 frame complete interrupt.

enumerator kDPU_Group0ExtDst0SeqCompleteInterrupt
ExtDst0 sequence complete interrupt.

enumerator kDPU_Group0ExtDst4ShadowLoadInterrupt
ExtDst4 shadow load interrupt.

enumerator kDPU_Group0ExtDst4FrameCompleteInterrupt
ExtDst4 frame complete interrupt.

enumerator kDPU_Group0ExtDst4SeqCompleteInterrupt
ExtDst4 sequence complete interrupt.

enumerator kDPU_Group0ExtDst1ShadowLoadInterrupt
ExtDst1 shadow load interrupt.

enumerator kDPU_Group0ExtDst1FrameCompleteInterrupt
ExtDst1 frame complete interrupt.

enumerator kDPU_Group0ExtDst1SeqCompleteInterrupt
ExtDst1 sequence complete interrupt.

enumerator kDPU_Group0ExtDst5ShadowLoadInterrupt
ExtDst5 shadow load interrupt.

enumerator kDPU_Group0ExtDst5FrameCompleteInterrupt
ExtDst5 frame complete interrupt.

enumerator kDPU_Group0ExtDst5SeqCompleteInterrupt
ExtDst5 sequence complete interrupt.

enumerator kDPU_Group0DomainBlend0ShadowLoadInterrupt
DomainBlend0 shadow load interrupt.

enumerator kDPU_Group0DomainBlend0FrameCompleteInterrupt
DomainBlend0 frame complete interrupt.

enumerator kDPU_Group0DomainBlend0SeqCompleteInterrupt
DomainBlend0 sequence complete interrupt.

enumerator kDPU_Group0DiSengcfgShadowLoad0Interrupt
DiSengcfg shadow load0 interrupt

enumerator kDPU_Group0DiSengcfgFrameComplete0Interrupt
DiSengcfg frame complete0 interrupt.

enumerator kDPU_Group0DiSengcfgSeqComplete0Interrupt
DiSengcfg sequence complete0 interrupt.

enumerator kDPU_Group0FrameGen0Int0Interrupt
FrameGen 0 interrupt 0.

enumerator kDPU_Group0FrameGen0Int1Interrupt
FrameGen 0 interrupt 1.

enumerator kDPU_Group0FrameGen0Int2Interrupt
FrameGen 0 interrupt 2.

enumerator kDPU_Group0FrameGen0Int3Interrupt
FrameGen 0 interrupt 3.

enumerator kDPU_Group0Sig0ShadowLoadInterrupt
Sig0 shadow load interrupt.

enumerator kDPU_Group0Sig0ValidInterrupt
Sig0 measurement valid interrupt.

enumerator kDPU_Group0Sig0ErrorInterrupt
Sig0 error interrupt.

enumerator kDPU_Group0Sig0ClusterErrorInterrupt
Sig0 cluster error interrupt.

enumerator kDPU_Group0Sig0ClusterMatchInterrupt
Sig0 cluster match interrupt.

enumerator kDPU_Group0Sig2ShadowLoadInterrupt
Sig2 shadow load interrupt.

enumerator kDPU_Group0Sig2ValidInterrupt
Sig2 measurement valid interrupt.

enumerator kDPU_Group1Sig2ErrorInterrupt
Sig2 error interrupt.

enumerator kDPU_Group1Sig2ClusterErrorInterrupt
Sig2 cluster error interrupt.

enumerator kDPU_Group1Sig2ClusterMatchInterrupt
Sig2 cluster match interrupt.

enumerator kDPU_Group1IdHash0ShadowLoadInterrupt
IdHash0 shadow load interrupt.

enumerator kDPU_Group1IdHash0ValidInterrupt
IdHash0 measurement valid interrupt.

enumerator kDPU_Group1IdHash0WindowsErrorInterrupt
IdHash0 windows error interrupt.

enumerator kDPU_Group1DomainBlend1ShadowLoadInterrupt
DomainBlend1 shadow load interrupt.

enumerator kDPU_Group1DomainBlend1FrameCompleteInterrupt
DomainBlend1 frame complete interrupt.

enumerator kDPU_Group1DomainBlend1SeqCompleteInterrupt
DomainBlend1 sequence complete interrupt.

enumerator kDPU_Group1DiSengcfgShadowLoad1Interrupt
DiSengcfg shadow load1 interrupt

enumerator kDPU_Group1DiSengcfgFrameComplete1Interrupt
DiSengcfg frame complete1 interrupt.

enumerator kDPU_Group1DiSengcfgSeqComplete1Interrupt
DiSengcfg sequence complete1 interrupt.

enumerator kDPU_Group1FrameGen1Int0Interrupt
FrameGen 1 interrupt 0.

enumerator kDPU_Group1FrameGen1Int1Interrupt
FrameGen 1 interrupt 1.

enumerator kDPU_Group1FrameGen1Int2Interrupt
FrameGen 1 interrupt 2.

enumerator kDPU_Group1FrameGen1Int3Interrupt
FrameGen 1 interrupt 3.

enumerator kDPU_Group1Sig1ShadowLoadInterrupt
Sig1 shadow load interrupt.

enumerator kDPU_Group1Sig1ValidInterrupt
Sig1 measurement valid interrupt.

enumerator kDPU_Group1Sig1ErrorInterrupt
Sig1 error interrupt.

enumerator kDPU_Group1Sig1ClusterErrorInterrupt
Sig1 cluster error interrupt.

enumerator kDPU_Group1Sig1ClusterMatchInterrupt
Sig1 cluster match interrupt.

enumerator kDPU_Group1CmdSeqErrorInterrupt
CmdSeq Error interrupt.

enumerator kDPU_Group1ComCtrlSw0Interrupt
ComCtrlSw0 interrupt.

enumerator kDPU_Group1ComCtrlSw1Interrupt
ComCtrlSw1 interrupt.

enumerator kDPU_Group1ComCtrlSw2Interrupt
ComCtrlSw1 interrupt.

enumerator kDPU_Group1ComCtrlSw3Interrupt
ComCtrlSw1 interrupt.

enumerator kDPU_Group1FrameGen0PrimSyncOnInterrupt
FrameGen 0 primary sync on interrupt.

enumerator kDPU_Group1FrameGen0PrimSyncOffInterrupt
FrameGen 0 primary sync off interrupt.

enumerator kDPU_Group1FrameGen0OverFlow0OnInterrupt
FrameGen 0 over flow0 on interrupt.

enumerator kDPU_Group1FrameGen0OverFlow0OffInterrupt
FrameGen 0 over flow0 off interrupt.

enumerator kDPU_Group1FrameGen0UnderRun0OnInterrupt
FrameGen 0 under run0 on interrupt.

enumerator kDPU_Group1FrameGen0UnderRun0OffInterrupt
FrameGen 0 under run0 off interrupt.

enumerator kDPU_Group2FrameGen0Threshold0RiseInterrupt
FrameGen 0 Threshold0 rise interrupt.

enumerator kDPU_Group2FrameGen0Threshold0FailInterrupt
FrameGen 0 Threshold0 fail interrupt.

enumerator kDPU_Group2FrameGen0OverFlow1OnInterrupt
FrameGen 0 over flow1 on interrupt.

enumerator kDPU_Group2FrameGen0OverFlow1OffInterrupt
FrameGen 0 over flow1 off interrupt.

enumerator kDPU_Group2FrameGen0UnderRun1OnInterrupt

FrameGen 0 under run1 on interrupt.

enumerator kDPU_Group2FrameGen0UnderRun1OffInterrupt

FrameGen 0 under run1 off interrupt.

enumerator kDPU_Group2FrameGen0Threshold1RiseInterrupt

FrameGen 0 Threshold1 rise interrupt.

enumerator kDPU_Group2FrameGen0Threshold1FailInterrupt

FrameGen 0 Threshold1 fail interrupt.

enumerator kDPU_Group2FrameGen1PrimSyncOnInterrupt

FrameGen 1 primary sync on interrupt.

enumerator kDPU_Group2FrameGen1PrimSyncOffInterrupt

FrameGen 1 primary sync off interrupt.

enumerator kDPU_Group2FrameGen1OverFlow0OnInterrupt

FrameGen 1 over flow0 on interrupt.

enumerator kDPU_Group2FrameGen1OverFlow0OffInterrupt

FrameGen 1 over flow0 off interrupt.

enumerator kDPU_Group2FrameGen1UnderRun0OnInterrupt

FrameGen 1 under run0 on interrupt.

enumerator kDPU_Group2FrameGen1UnderRun0OffInterrupt

FrameGen 1 under run0 off interrupt.

enumerator kDPU_Group2FrameGen1Threshold0RiseInterrupt

FrameGen 1 Threshold0 rise interrupt.

enumerator kDPU_Group2FrameGen1Threshold0FailInterrupt

FrameGen 1 Threshold0 fail interrupt.

enumerator kDPU_Group2FrameGen1OverFlow1OnInterrupt

FrameGen 1 over flow1 on interrupt.

enumerator kDPU_Group2FrameGen1OverFlow1OffInterrupt

FrameGen 1 over flow1 off interrupt.

enumerator kDPU_Group2FrameGen1UnderRun1OnInterrupt

FrameGen 1 under run1 on interrupt.

enumerator kDPU_Group2FrameGen1UnderRun1OffInterrupt

FrameGen 1 under run1 off interrupt.

enumerator kDPU_Group2FrameGen1Threshold1RiseInterrupt

FrameGen 1 Threshold1 rise interrupt.

enumerator kDPU_Group2FrameGen1Threshold1FailInterrupt

FrameGen 1 Threshold1 fail interrupt.

enum _dpu_unit_source

Values:

enumerator kDPU_UnitSrcNone

Disable the input source.

enumerator kDPU_UnitSrcRop9

The input source is Rop 9.

enumerator KDPU__UnitSrcExtDst0

The input source is ExtDst 0.

enumerator KDPU__UnitSrcExtDst4

The input source is ExtDst 4.

enumerator kDPU__UnitSrcBlitBlend9

The input source is BlitBlend 9.

enumerator kDPU__UnitSrcFetchRot9

The input source is Rot 9.

enumerator kDPU__UnitSrcFetchDecode9

The input source is fetch decode 9.

enumerator kDPU__UnitSrcFetchEco9

input source is fetch eco 9.

enumerator kDPU__UnitSrcHScaler9

The input source is HScaler 9.

enumerator kDPU__UnitSrcVScaler9

The input source is VScaler 9.

enumerator kDPU__UnitSrcFilter9

The input source is Filter 9.

enumerator kDPU__UnitSrcConstFrame0

The input source is ConstFrame 0.

enumerator kDPU__UnitSrcConstFrame4

The input source is ConstFrame 4.

enumerator kDPU__UnitSrcConstFrame1

The input source is ConstFrame 1.

enumerator kDPU__UnitSrcConstFrame5

The input source is ConstFrame 5.

enumerator kDPU__UnitSrcLayerBlend1

The input source is LayerBlend 1.

enumerator kDPU__UnitSrcLayerBlend2

The input source is LayerBlend 2.

enumerator kDPU__UnitSrcLayerBlend3

The input source is LayerBlend 3.

enumerator kDPU__UnitSrcLayerBlend4

The input source is LayerBlend 4.

enumerator kDPU__UnitSrcLayerBlend5

The input source is LayerBlend 5.

enumerator kDPU__UnitSrcLayerBlend6

The input source is LayerBlend 6.

enumerator kDPU__UnitSrcFetchLayer0

The input source is FetchLayer 0.

enumerator kDPU__UnitSrcFetchLayer1

The input source is FetchLayer 1.

enumerator kDPU__UnitSrcFetchYUV3

The input source is Fetchyuv 3.

enumerator kDPU__UnitSrcFetchYUV0

The input source is Fetchyuv 0.

enumerator kDPU__UnitSrcFetchYUV1

The input source is Fetchyuv 1.

enumerator kDPU__UnitSrcFetchYUV2

The input source is Fetchyuv 2.

enumerator kDPU__UnitSrcMatrix4

The input source is Matrix 4.

enumerator kDPU__UnitSrcHScaler4

The input source is HScaler 4.

enumerator kDPU__UnitSrcVScaler4

The input source is VScaler 4.

enumerator kDPU__UnitSrcFetchEco0

The input source is FetchEco 0.

enumerator kDPU__UnitSrcFetchEco1

The input source is FetchEco 1.

enumerator kDPU__UnitSrcFetchEco2

The input source is FetchEco 2.

enum __dpu_layer_blend_shadow_token_mode

LayerBlend unit shadow token generate mode.

Values:

enumerator kDPU__LayerBlendShadowTokenPrim

Generate shadow load token when token received from primary input.

enumerator kDPU__LayerBlendShadowTokenSec

Generate shadow load token when token received from secondary input.

enumerator kDPU__LayerBlendShadowTokenBoth

Generate shadow load token when token received from any input.

enum __dpu_layer_blend_shadow_load_mode

LayerBlend unit shadow load mode.

Values:

enumerator kDPU__LayerBlendShadowLoadPrim

Load shadows when token received from primary input.

enumerator kDPU__LayerBlendShadowLoadSec

Load shadows when token received from secondary input.

enumerator kDPU__LayerBlendShadowLoadBoth

Load shadows when token received from any input.

enum __dpu_pixel_format

DPU pixel format.

To support more pixel format, enhance this enum and the array s_dpuColorComponentFormats.

Values:

enumerator kDPU__PixelFormatGray8
8-bit gray.

enumerator kDPU__PixelFormatRGB565
RGB565, 16-bit per pixel.

enumerator kDPU__PixelFormatARGB8888
ARGB8888, 32-bit per pixel.

enumerator kDPU__PixelFormatRGB888
RGB888, 24-bit per pixel.

enumerator kDPU__PixelFormatARGB1555
ARGB1555, 16-bit per pixel.

enum _dpu_warp_coordinate_mode
FetchWarp unit warp coordinate mode.

Values:

enumerator kDPU__WarpCoordinateModePNT
Sample points positions are read from coordinate layer.

enumerator kDPU__WarpCoordinateModeDPNT
Sample points start position and delta are read from coordinate layer.

enumerator kDPU__WarpCoordinateModeDDPNT
Sample points initial value and delta increase value are read from coordinate layer.

enum _dpu_clip_color_mode
Define the color to take for pixels that do not lie inside the clip window of any layer.

Values:

enumerator kDPU__ClipColorNull
Use null color.

enumerator kDPU__ClipColorSublayer
Use color of sublayer.

enum _dpu_alpha_mask_mode
LayerBlend unit AlphaMask mode.

Values:

enumerator kDPU__AlphaMaskPrim
Areas with primary input alpha > 128 mapped to alpha 255, the rest mapped to 0.

enumerator kDPU__AlphaMaskSec
Areas with secondary input alpha > 128 mapped to alpha 255, the rest mapped to 0.

enumerator kDPU__AlphaMaskPrimOrSec
Primary and secondary OR'ed together.

enumerator kDPU__AlphaMaskPrimAndSec
Primary and secondary AND'ed together.

enumerator kDPU__AlphaMaskPrimInv
Primary input alpha inverted.

enumerator kDPU__AlphaMaskSecInv
Secondary input alpha inverted.

enumerator kDPU__AlphaMaskPrimOrSecInv
Primary and inverted secondary OR'ed together.

enumerator kDPU__AlphaMaskPrimAndSecInv
Primary and inverted secondary AND'ed together.

enum __dpu_blend_mode
LayerBlend unit alpha blend mode.

Values:

enumerator kDPU__BlendZero
OUT = IN * 0.

enumerator kDPU__BlendOne
OUT = IN * 1.

enumerator kDPU__BlendPrimAlpha
OUT = IN * ALPHA_primary.

enumerator kDPU__BlendPrimAlphaInv
OUT = IN * (1 - ALPHA_primary).

enumerator kDPU__BlendSecAlpha
OUT = IN * ALPHA_secondary.

enumerator kDPU__BlendSecAlphaInv
OUT = IN * (1 - ALPHA_secondary).

enumerator kDPU__BlendConstAlpha
OUT = IN * ALPHA_const.

enumerator kDPU__BlendConstAlphaInv
OUT = IN * (1 - ALPHA_const).

enum __dpu_display_timing_flags
Display timing configuration flags.

Values:

enumerator kDPU__DisplayPixelActiveHigh
Pixel data active high.

enumerator kDPU__DisplayDataEnableActiveHigh
Set to make data enable high active.

enumerator kDPU__DisplayDataEnableActiveLow
Set to make data enable high low.

enumerator kDPU__DisplayHsyncActiveHigh
Set to make HSYNC high active.

enumerator kDPU__DisplayHsyncActiveLow
Set to make HSYNC low active.

enumerator kDPU__DisplayVsyncActiveHigh
Set to make VSYNC high active.

enumerator kDPU__DisplayVsyncActiveLow
Set to make VSYNC low active.

enum __dpu_display_mode
Display mode, safety stream is the primary input, content stream is the secondary input.

Values:

enumerator kDPU__DisplayBlackBackground

Black background is shown.

enumerator kDPU__DisplayConstBackground

Const color background is shown.

enumerator kDPU__DisplayOnlyPrim

Only primary input is shown.

enumerator kDPU__DisplayOnlySec

Only secondary input is shown.

enumerator kDPU__DisplayPrimOnTop

Both inputs overlaid with primary on top.

enumerator kDPU__DisplaySecOnTop

Both inputs overlaid with secondary on top.

enumerator kDPU__DisplayTest

White background with test pattern shown.

enum _dpu_rop_flags

Values:

enumerator kDPU__RopAddRed

Set to add the red component, otherwise raster with operation index.

enumerator kDPU__RopAddGreen

Set to add the green component, otherwise raster with operation index.

enumerator kDPU__RopAddBlue

Set to add the blue component, otherwise raster with operation index.

enumerator kDPU__RopAddAlpha

Set to add the alpha component, otherwise raster with operation index.

enumerator kDPU__RopTertDiv2

In add mode, set this to divide tertiary port input by 2.

enumerator kDPU__RopSecDiv2

In add mode, set this to divide secondary port input by 2.

enumerator kDPU__RopPrimDiv2

In add mode, set this to divide primary port input by 2.

typedef enum _dpu_unit dpu_unit_t

DPU units.

typedef enum _dpu_pixel_format dpu_pixel_format_t

DPU pixel format.

To support more pixel format, enhance this enum and the array `s_dpuColorComponentFormats`.

typedef struct _dpu_fetch_unit_config dpu_fetch_unit_config_t

Configuration structure for fetch units.

typedef struct _dpu_coordinates_config dpu_coordinates_config_t

Configuration structure for the arbitrary warping re-sampling coordinates.

The coordinate layer supports:

- 32 bpp: 2 x s12.4 (signed fix-point)

- 24 bpp: 2 x s8.
- 16 bpp: 2 x s4.4
- 8 bpp: 2 x s0.4
- 4 bpp: 2 x s(-2).4 (means total value size = 2 bits and lowest bit = 2^{-4})
- 2 bpp: 2 x s(-3).4
- 1 bpp: 1 x s(-3).4 (x and y alternating)

typedef enum *_dpu_warp_coordinate_mode* dpu_warp_coordinate_mode_t
FetchWarp unit warp coordinate mode.

typedef struct *_dpu_warp_config* dpu_warp_config_t
Warp configuration structure for FetchWarp unit.

typedef enum *_dpu_clip_color_mode* dpu_clip_color_mode_t
Define the color to take for pixels that do not lie inside the clip window of any layer.

typedef struct *_dpu_dst_buffer_config* dpu_dst_buffer_config_t
Store unit Destination buffer configuration structure.

Base address and stride alignment restrictions: 32 bpp: Base address and stride must be a multiple of 4 bytes. 16 bpp: Base address and stride must be a multiple of 2 bytes. others: any byte alignment allowed

typedef enum *_dpu_alpha_mask_mode* dpu_alpha_mask_mode_t
LayerBlend unit AlphaMask mode.

typedef enum *_dpu_blend_mode* dpu_blend_mode_t
LayerBlend unit alpha blend mode.

typedef struct *_dpu_layer_blend_config* dpu_layer_blend_config_t
LayerBlend unit configuration structure.

typedef struct *_dpu_const_frame_config* dpu_const_frame_config_t
ConstFrame unit configuration structure.

typedef struct *_dpu_display_timing_config* dpu_display_timing_config_t
Display timing configuration structure.

typedef enum *_dpu_display_mode* dpu_display_mode_t
Display mode, safety stream is the primary input, content stream is the secondary input.

typedef struct *_dpu_display_config* dpu_display_config_t
Display mode configuration structure.

typedef struct *_dpu_scaler_config* dpu_scaler_config_t
VScaler and HScaler configuration structure.

typedef struct *_dpu_rop_config* dpu_rop_config_t
Rop unit configuration structure.

typedef struct *_dpu_src_buffer_config* dpu_src_buffer_config_t
Fetch unit source buffer configuration structure.

Base address and stride alignment restrictions: 32 bpp: Base address and stride must be a multiple of 4 bytes. 16 bpp: Base address and stride must be a multiple of 2 bytes. others: any byte alignment allowed

Generally, the bitsPerPixel and pixelFormat specify the pixel format in frame buffer, they should match. But when the color palette is used, the bitsPerPixel specify the format in framebuffer, the pixelFormat specify the format in color palette entry.

DPU_MAKE_CONST_COLOR(red, green, blue, alpha)

Define the const value that write to <unit>_ConstantColor.

DPU_UNIT_TYPE_SHIFT

DPU_UNIT_TYPE_MASK

DPU_UNIT_ATTR_SHIFT

DPU_UNIT_ATTR_MASK

DPU_UNIT_OFFSET_SHIFT

DPU_UNIT_OFFSET_MASK

DPU_MAKE_UNIT_TYPE(type)

DPU_MAKE_UNIT_ATTR(attr)

DPU_MAKE_UNIT_OFFSET(offset)

DPU_GET_UNIT_TYPE(unit)

DPU_GET_UNIT_ATTR(unit)

DPU_GET_UNIT_OFFSET(unit)

DPU_MAKE_UNIT(type, attr, offset)

DPU_UNIT_OFFSET(unit)

DPU_COMCTRL_OFFSET

DPU_FETCH_DECODE9_OFFSET

DPU_FETCH_ROT9_OFFSET

DPU_FETCH_ECO9_OFFSET

DPU_ROP9_OFFSET

DPU_H_SCALER9_OFFSET

DPU_V_SCALER9_OFFSET

DPU_BLITBLEND9_OFFSET

DPU_STORE9_OFFSET

DPU_CONST_FRAME0_OFFSET

DPU_EXT_DST0_OFFSET

DPU_CONST_FRAME4_OFFSET

DPU_EXT_DST4_OFFSET

DPU_CONST_FRAME1_OFFSET

DPU_EXT_DST1_OFFSET

DPU_CONST_FRAME5_OFFSET

DPU_EXT_DST5_OFFSET

DPU_LAYER_BLEND1_OFFSET
DPU_LAYER_BLEND2_OFFSET
DPU_LAYER_BLEND3_OFFSET
DPU_LAYER_BLEND4_OFFSET
DPU_LAYER_BLEND5_OFFSET
DPU_DOMAIN_BLEND0_OFFSET
DPU_FRAME_GEN0_OFFSET
DPU_PIPELINE_EXTDST0_OFFSET
DPU_PIPELINE_EXTDST1_OFFSET
DPU_PIPELINE_EXTDST4_OFFSET
DPU_PIPELINE_EXTDST5_OFFSET
DPU_PIPELINE_STORE9_OFFSET
DPU_FETCH_ECO0_OFFSET
DPU_FETCH_ECO1_OFFSET
DPU_FETCH_ECO2_OFFSET
DPU_FETCH_LAYER0_OFFSET
DPU_FETCH_LAYER1_OFFSET
DPU_H_SCALER4_OFFSET
DPU_V_SCALER4_OFFSET
DPU_LAYER_BLEND6_OFFSET
DPU_FETCH_YUV0_OFFSET
DPU_FETCH_YUV1_OFFSET
DPU_FETCH_YUV2_OFFSET
DPU_FETCH_YUV3_OFFSET
DPU_DOMAIN_BLEND1_OFFSET
DPU_FRAME_GEN1_OFFSET
DPU_ID_HASH0_OFFSET
DPU_SIG0_OFFSET
DPU_SIG1_OFFSET
DPU_SIG2_OFFSET
DPU_DITHER0_CONFIG_OFFSET
DPU_DITHER1_CONFIG_OFFSET
DPU_ROP_CONTROL_Mode_MASK

DPU_ROP_CONTROL_RedMode_MASK
DPU_ROP_CONTROL_GreenMode_MASK
DPU_ROP_CONTROL_BlueMode_MASK
DPU_ROP_CONTROL_AlphaMode_MASK
DPU_ROP_CONTROL_TertDiv2_MASK
DPU_ROP_CONTROL_SecDiv2_MASK
DPU_ROP_CONTROL_PrimDiv2_MASK
DPU_DISENGCONF_POLARITYCTRL_PolEn_MASK
DPU_DISENGCONF_POLARITYCTRL_PolVs_MASK
DPU_DISENGCONF_POLARITYCTRL_PolHs_MASK
DPU_SIG_EVALUPPERLEFT_XEvalUpperLeft_SHIFT
DPU_SIG_EVALUPPERLEFT_YEvalUpperLeft_SHIFT
DPU_SIG_EVALLOWERRIGHT_XEvalLowerRight_SHIFT
DPU_SIG0_EVALLOWERRIGHT_YEvalLowerRight_SHIFT
DPU_SIG_SHADOWLOAD_ShdlReq_MASK
DOMAINMASK_ENABLE
SHDLREQSTICKY_ENABLE

struct _dpu_fetch_unit_config
 #include <fsl_dpu.h> Configuration structure for fetch units.

Public Members

uint32_t srcReg
 This value will be set to register pixengcfg_fetchX_dynamic to set the unit input source, see DPU_MAKE_SRC_REG1.

uint16_t frameHeight
 Frame height.

uint16_t frameWidth
 Frame width.

struct _dpu_coordinates_config
 #include <fsl_dpu.h> Configuration structure for the arbitrary warping re-sampling coordinates.

The coordinate layer supports:

- 32 bpp: 2 x s12.4 (signed fix-point)
- 24 bpp: 2 x s8.
- 16 bpp: 2 x s4.4
- 8 bpp: 2 x s0.4
- 4 bpp: 2 x s(-2).4 (means total value size = 2 bits and lowest bit = 2^{-4})

- 2 bpp: 2 x s(-3).4
- 1 bpp: 1 x s(-3).4 (x and y alternating)

Public Members

uint8_t bitsPerPixel

Number of bits per pixel in the source buffer. Must be 1, 2, 4, 8, 16, 32.

uint16_t strideBytes

Source buffer stride in bytes.

uint32_t baseAddr

Source buffer base address.

uint16_t frameHeight

Frame height.

uint16_t frameWidth

Frame width.

struct __dpu_warp_config

#include <fsl_dpu.h> Warp configuration structure for FetchWarp unit.

Public Members

uint32_t srcReg

This value will be set to register pixengcfg_fetchX_dynamic to set the unit input source, see DPU_MAKE_SRC_REG1.

uint16_t frameHeight

Frame height.

uint16_t frameWidth

Frame width.

uint8_t warpBitsPerPixel

Pixel bits of the coordinate layer.

bool enableSymmetricOffset

Enables symmetric range for negative and positive coordinate values by adding an offset of +0.03125 internally to all coordinate input values. Recommended for small coordinate formats in DD_PNT mode.

dpu_warp_coordinate_mode_t coordMode

Coordinate layer mode.

uint32_t arbStartX

X of start point position. Signed 16.5 fix-point. Used in D_PNT and DD_PNT.

uint32_t arbStartY

Y of start point position. Signed 16.5 fix-point. Used in D_PNT and DD_PNT.

uint8_t arbDeltaYY

Y of vector between start and first sample point. Signed 3.5 fix-point. Used in DD_PNT.

uint8_t arbDeltaYX

X of vector between start and first sample point. Signed 3.5 fix-point. Used in DD_PNT.

uint8_t arbDeltaXY

Y of vector between first and second sample point. Signed 3.5 fix-point. Used in DD_PNT.

uint8_t arbDeltaXX

X of vector between first and second sample point. Signed 3.5 fix-point. Used in DD_PNT.

struct __dpu_dst_buffer_config

#include <fsl_dpu.h> Store unit Destination buffer configuration structure.

Base address and stride alignment restrictions: 32 bpp: Base address and stride must be a multiple of 4 bytes. 16 bpp: Base address and stride must be a multiple of 2 bytes. others: any byte alignment allowed

Public Members

uint32_t baseAddr

Destination buffer base address, see alignment restrictions.

uint16_t strideBytes

Destination buffer stride in bytes, see alignment restrictions.

uint8_t bitsPerPixel

Bits per pixel.

dpu_pixel_format_t pixelFormat

Pixel format.

uint16_t bufferHeight

Buffer height.

uint16_t bufferWidth

Buffer width.

struct __dpu_layer_blend_config

#include <fsl_dpu.h> LayerBlend unit configuration structure.

Public Members

uint8_t constAlpha

The const alpha value used in blend.

dpu_blend_mode_t secAlphaBlendMode

Secondary (overlay) input alpha blending function.

dpu_blend_mode_t primAlphaBlendMode

Primary (background) input alpha blending function.

dpu_blend_mode_t secColorBlendMode

Secondary (overlay) input color blending function.

dpu_blend_mode_t primColorBlendMode

Primary (background) input color blending function.

uint32_t srcReg

This value will be set to `pixengcfg_layerblendX_dynamic` to set the unit input source, see `DPU_MAKE_SRC_REG2`.

bool enableAlphaMask

Enable AlphaMask feature.

dpu_alpha_mask_mode_t alphaMaskMode

AlphaMask mode, only valid when enableAlphaMask is true.

struct __dpu_const_frame_config

#include <fsl_dpu.h> ConstFrame unit configuration structure.

Public Members

uint16_t frameHeight

Frame height.

uint16_t frameWidth

Frame width.

uint32_t constColor

See DPU_MAKE_CONST_COLOR.

struct __dpu_display_timing_config

#include <fsl_dpu.h> Display timing configuration structure.

Public Members

uint16_t flags

OR'ed value of _dpu_display_timing_flags.

uint16_t width

Active width.

uint16_t hsw

HSYNC pulse width.

uint16_t hfp

Horizontal front porch.

uint16_t hbp

Horizontal back porch.

uint16_t height

Active height.

uint16_t vsw

VSYNC pulse width.

uint16_t vfp

Vrtical front porch.

uint16_t vbp

Vertical back porch.

struct __dpu_display_config

#include <fsl_dpu.h> Display mode configuration structure.

Public Members

bool enablePrimAlpha

Enable primary input alpha for screen composition.

bool enableSecAlpha

Enable secondary input alpha for screen composition.

dpu_display_mode_t displayMode

Display mode.

bool enablePrimAlphaInPanic

Enable primary input alpha for screen composition in panic mode.

bool enableSecAlphaInPanic

Enable secondary input alpha for screen composition in panic mode.

dpu_display_mode_t displayModeInPanic

Display mode in panic mode.

uint16_t constRed

Const red value, 10-bit.

uint16_t constGreen

Const green value, 10-bit.

uint16_t constBlue

Const green value, 10-bit.

uint8_t constAlpha

Const alpha value, 1-bit.

uint16_t primAreaStartX

Primary screen upper left corner, x component. 14-bit , start from 1.

uint16_t primAreaStartY

Primary screen upper left corner, y component. 14-bit, start from 1.

uint16_t secAreaStartX

Secondary screen upper left corner, x component. 14-bit, start from 1.

uint16_t secAreaStartY

Secondary screen upper left corner, y component. 14-bit, start from 1.

struct __dpu_scaler_config

#include <fsl_dpu.h> VScaler and HScaler configuration structure.

Public Members

uint32_t srcReg

This value will be set to register pixengcfg_slacer_dynamic to set the unit input source, see DPU_MAKE_SRC_REG1. When down-scaling horizontally, the path should be -> HScaler -> VScaler ->, When up-scaling horizontally, the path should be -> VScaler -> HScaler ->.

uint16_t inputSize

For HScaler, it is frame width, for VScaler, it is frame height.

uint16_t outputSize

For HScaler, it is frame width, for VScaler, it is frame height.

struct __dpu_rop_config

#include <fsl_dpu.h> Rop unit configuration structure.

Public Members

uint32_t controlFlags

Control flags, see `_dpu_rop_flags`.

uint8_t alphaIndex

Alpha operation index.

uint8_t blueIndex

Blue operation index.

uint8_t greenIndex

Green operation index.

uint8_t redIndex

Red operation index.

struct `_dpu_src_buffer_config`

#include <fsl_dpu.h> Fetch unit source buffer configuration structure.

Base address and stride alignment restrictions: 32 bpp: Base address and stride must be a multiple of 4 bytes. 16 bpp: Base address and stride must be a multiple of 2 bytes. others: any byte alignment allowed

Generally, the `bitsPerPixel` and `pixelFormat` specify the pixel format in frame buffer, they should match. But when the color palette is used, the `bitsPerPixel` specify the format in framebuffer, the `pixelFormat` specify the format in color palette entry.

Public Members

uint32_t baseAddr

Source buffer base address, see alignment restrictions.

uint16_t strideBytes

Source buffer stride in bytes, see alignment restrictions.

uint8_t bitsPerPixel

Bits per pixel in frame buffer.

dpu_pixel_format_t pixelFormat

Pixel format.

uint16_t bufferHeight

Buffer height.

uint16_t bufferWidth

Buffer width.

uint32_t constColor

Const color shown in the region out of frame buffer, see `DPU_MAKE_CONST_COLOR`.

2.6 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

void EDMA_Init(*EDMA_Type* *base, const *edma_config_t* *config)

Initializes the eDMA peripheral.

This function ungates the eDMA clock and configures the eDMA peripheral according to the configuration structure. All emda enabled request will be cleared in this function.

Note: This function enables the minor loop map feature.

Parameters

- base – eDMA peripheral base address.
- config – A pointer to the configuration structure, see “*edma_config_t*”.

void EDMA_Deinit(*EDMA_Type* *base)

Deinitializes the eDMA peripheral.

This function gates the eDMA clock.

Parameters

- base – eDMA peripheral base address.

void EDMA_InstallTCD(*EDMA_Type* *base, uint32_t channel, *edma_tcd_t* *tcd)

Push content of TCD structure into hardware TCD register.

Parameters

- base – EDMA peripheral base address.
- channel – EDMA channel number.
- tcd – Point to TCD structure.

void EDMA_GetDefaultConfig(*edma_config_t* *config)

Gets the eDMA default configuration structure.

This function sets the configuration structure to default values. The default configuration is set to the following values.

```
config.enableContinuousLinkMode = false;
config.enableHaltOnError = true;
config.enableRoundRobinArbitration = false;
config.enableDebugMode = false;
```

Parameters

- config – A pointer to the eDMA configuration structure.

void EDMA_InitChannel(*EDMA_Type* *base, uint32_t channel, *edma_channel_config_t* *channelConfig)

EDMA Channel initialization.

Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- channelConfig – pointer to user’s eDMA4 channel config structure, see *edma_channel_config_t* for detail.

static inline void EDMA_SetChannelMemoryAttribute(*EDMA_Type* *base, uint32_t channel, *edma_channel_memory_attribute_t* writeAttribute, *edma_channel_memory_attribute_t* readAttribute)

Set channel memory attribute.

Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- writeAttribute – Attributes associated with a write transaction.
- readAttribute – Attributes associated with a read transaction.

```
static inline void EDMA_SetChannelSignExtension(EDMA_Type *base, uint32_t channel, uint8_t position)
```

Set channel sign extension.

Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- position – A non-zero value specifying the sign extend bit position. If 0, sign extension is disabled.

```
static inline void EDMA_SetChannelSwapSize(EDMA_Type *base, uint32_t channel, edma_channel_swap_size_t swapSize)
```

Set channel swap size.

Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- swapSize – Swap occurs with respect to the specified transfer size. If 0, swap is disabled.

```
static inline void EDMA_SetChannelAccessType(EDMA_Type *base, uint32_t channel, edma_channel_access_type_t channelAccessType)
```

Set channel access type.

Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- channelAccessType – eDMA4's transactions type on the system bus when the channel is active.

```
static inline void EDMA_SetChannelMux(EDMA_Type *base, uint32_t channel, uint32_t channelRequestSource)
```

Set channel request source.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- channelRequestSource – eDMA hardware service request source for the channel. User need to use the dma_request_source_t type as the input parameter. Note that devices may use other enum type to express dma request source and User can find it in SOC header or fsl_edma_soc.h.

```
static inline uint32_t EDMA_GetChannelSystemBusInformation(EDMA_Type *base, uint32_t  
                                                         channel)
```

Gets the channel identification and attribute information on the system bus interface.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

Returns

The mask of the channel system bus information. Users need to use the `_edma_channel_sys_bus_info` type to decode the return variables.

```
static inline void EDMA_EnableChannelMasterIDReplication(EDMA_Type *base, uint32_t  
                                                         channel, bool enable)
```

Set channel master ID replication.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- enable – true is enable, false is disable.

```
static inline void EDMA_SetChannelProtectionLevel(EDMA_Type *base, uint32_t channel,  
                                                  edma_channel_protection_level_t level)
```

Set channel security level.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- level – security level.

```
void EDMA_ResetChannel(EDMA_Type *base, uint32_t channel)
```

Sets all TCD registers to default values.

This function sets TCD registers for this channel to default values.

Note: This function must not be called while the channel transfer is ongoing or it causes unpredictable results.

Note: This function enables the auto stop request feature.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

```
void EDMA_SetTransferConfig(EDMA_Type *base, uint32_t channel, const  
                           edma_transfer_config_t *config, edma_tcd_t *nextTcd)
```

Configures the eDMA transfer attribute.

This function configures the transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the TCD address. Example:

```
edma_transfer_t config;
edma_tcd_t tcd;
config.srcAddr = ..;
config.destAddr = ..;
...
EDMA_SetTransferConfig(DMA0, channel, &config, &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(EDMA_Type *base, uint32_t channel, const
                               edma_minor_offset_config_t *config)
```

Configures the eDMA minor offset feature.

The minor offset means that the signed-extended value is added to the source address or destination address after each minor loop.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- config – A pointer to the minor offset configuration structure.

```
void EDMA_SetChannelPreemptionConfig(EDMA_Type *base, uint32_t channel, const
                                     edma_channel_preemption_config_t *config)
```

Configures the eDMA channel preemption feature.

This function configures the channel preemption attribute and the priority of the channel.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number
- config – A pointer to the channel preemption configuration structure.

```
void EDMA_SetChannelLink(EDMA_Type *base, uint32_t channel, edma_channel_link_type_t
                        type, uint32_t linkedChannel)
```

Sets the channel link for the eDMA transfer.

This function configures either the minor link or the major link mode. The minor link means that the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note: Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- type – A channel link type, which can be one of the following:
 - kEDMA_LinkNone
 - kEDMA_MinorLink
 - kEDMA_MajorLink
- linkedChannel – The linked channel number.

```
void EDMA__SetBandWidth(EDMA_Type *base, uint32_t channel, edma_bandwidth_t  
                        bandWidth)
```

Sets the bandwidth for the eDMA transfer.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- bandWidth – A bandwidth setting, which can be one of the following:
 - kEDMABandwidthStallNone
 - kEDMABandwidthStall4Cycle
 - kEDMABandwidthStall8Cycle

```
void EDMA__SetModulo(EDMA_Type *base, uint32_t channel, edma_modulo_t srcModulo,  
                    edma_modulo_t destModulo)
```

Sets the source modulo and the destination modulo for the eDMA transfer.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- srcModulo – A source modulo value.
- destModulo – A destination modulo value.

```
static inline void EDMA__EnableAutoStopRequest(EDMA_Type *base, uint32_t channel, bool  
                                                enable)
```

Enables an auto stop request for the eDMA transfer.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

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

`void EDMA__EnableChannelInterrupts(EDMA_Type *base, uint32_t channel, uint32_t mask)`

Enables the interrupt source for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

`void EDMA__DisableChannelInterrupts(EDMA_Type *base, uint32_t channel, uint32_t mask)`

Disables the interrupt source for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of the interrupt source to be set. Use the defined `edma_interrupt_enable_t` type.

`void EDMA__SetMajorOffsetConfig(EDMA_Type *base, uint32_t channel, int32_t sourceOffset, int32_t destOffset)`

Configures the eDMA channel TCD major offset feature.

Adjustment value added to the source address at the completion of the major iteration count

Parameters

- base – eDMA peripheral base address.
- channel – edma channel number.
- sourceOffset – source address offset will be applied to source address after major loop done.
- destOffset – destination address offset will be applied to source address after major loop done.

`void EDMA__ConfigChannelSoftwareTCD(edma_tcd_t *tcd, const edma_transfer_config_t *transfer)`

Sets TCD fields according to the user's channel transfer configuration structure, `edma_transfer_config_t`.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA__ConfigChannelSoftwareTCDExt`

Application should be careful about the TCD pool buffer storage class,

- For the platform has cache, the software TCD should be put in non cache section
- The TCD pool buffer should have a consistent storage class.

Note: This function enables the auto stop request feature.

Parameters

- tcd – Pointer to the TCD structure.
- transfer – channel transfer configuration pointer.

void EDMA__TcdReset(*edma_tcd_t* *tcd)

Sets all fields to default values for the TCD structure.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA__TcdResetExt

This function sets all fields for this TCD structure to default value.

Note: This function enables the auto stop request feature.

Parameters

- tcd – Pointer to the TCD structure.

void EDMA__TcdSetTransferConfig(*edma_tcd_t* *tcd, const *edma_transfer_config_t* *config, *edma_tcd_t* *nextTcd)

Configures the eDMA TCD transfer attribute.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA__TcdSetTransferConfigExt

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The TCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

```
edma_transfer_t config = {  
...  
}  
edma_tcd_t tcd __aligned(32);  
edma_tcd_t nextTcd __aligned(32);  
EDMA__TcdSetTransferConfig(&tcd, &config, &nextTcd);
```

Note: TCD address should be 32 bytes aligned or it causes an eDMA error.

Note: If the nextTcd is not NULL, the scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the EDMA__TcdReset.

Parameters

- tcd – Pointer to the TCD structure.
- config – Pointer to eDMA transfer configuration structure.
- nextTcd – Pointer to the next TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

void EDMA__TcdSetMinorOffsetConfig(*edma_tcd_t* *tcd, const *edma_minor_offset_config_t* *config)

Configures the eDMA TCD minor offset feature.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA__TcdSetMinorOffsetConfigExt

A minor offset is a signed-extended value added to the source address or a destination address after each minor loop.

Parameters

- `tcd` – A point to the TCD structure.
- `config` – A pointer to the minor offset configuration structure.

```
void EDMA__TcdSetChannelLink(edma_tcd_t *tcd, edma_channel_link_type_t type, uint32_t  
linkedChannel)
```

Sets the channel link for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA__TcdSetChannelLinkExt`

This function configures either a minor link or a major link. The minor link means the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note: Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

- `tcd` – Point to the TCD structure.
- `type` – Channel link type, it can be one of:
 - `kEDMA_LinkNone`
 - `kEDMA_MinorLink`
 - `kEDMA_MajorLink`
- `linkedChannel` – The linked channel number.

```
static inline void EDMA__TcdSetBandWidth(edma_tcd_t *tcd, edma_bandwidth_t bandWidth)
```

Sets the bandwidth for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA__TcdSetBandWidthExt`

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- `tcd` – A pointer to the TCD structure.
- `bandWidth` – A bandwidth setting, which can be one of the following:
 - `kEDMABandwidthStallNone`
 - `kEDMABandwidthStall4Cycle`
 - `kEDMABandwidthStall8Cycle`

```
void EDMA__TcdSetModulo(edma_tcd_t *tcd, edma_modulo_t srcModulo, edma_modulo_t  
destModulo)
```

Sets the source modulo and the destination modulo for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA__TcdSetModuloExt`

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- `tcd` – A pointer to the TCD structure.
- `srcModulo` – A source modulo value.
- `destModulo` – A destination modulo value.

static inline void EDMA_TcdEnableAutoStopRequest(*edma_tcd_t* *tcd, bool enable)

Sets the auto stop request for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA_TcdEnableAutoStopRequestExt

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

- `tcd` – A pointer to the TCD structure.
- `enable` – The command to enable (true) or disable (false).

void EDMA_TcdEnableInterrupts(*edma_tcd_t* *tcd, uint32_t mask)

Enables the interrupt source for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA_TcdEnableInterruptsExt

Parameters

- `tcd` – Point to the TCD structure.
- `mask` – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

void EDMA_TcdDisableInterrupts(*edma_tcd_t* *tcd, uint32_t mask)

Disables the interrupt source for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA_TcdDisableInterruptsExt

Parameters

- `tcd` – Point to the TCD structure.
- `mask` – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

void EDMA_TcdSetMajorOffsetConfig(*edma_tcd_t* *tcd, int32_t sourceOffset, int32_t destOffset)

Configures the eDMA TCD major offset feature.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA_TcdSetMajorOffsetConfigExt

Adjustment value added to the source address at the completion of the major iteration count

Parameters

- `tcd` – A point to the TCD structure.
- `sourceOffset` – source address offset will be applied to source address after major loop done.
- `destOffset` – destination address offset will be applied to source address after major loop done.

void EDMA_ConfigChannelSoftwareTCDExt(*EDMA_Type* *base, *edma_tcd_t* *tcd, const *edma_transfer_config_t* *transfer)

Sets TCD fields according to the user's channel transfer configuration structure, `edma_transfer_config_t`.

Application should be careful about the TCD pool buffer storage class,

- For the platform has cache, the software TCD should be put in non cache section
- The TCD pool buffer should have a consistent storage class.

Note: This function enables the auto stop request feature.

Parameters

- `base` – eDMA peripheral base address.
- `tcd` – Pointer to the TCD structure.
- `transfer` – channel transfer configuration pointer.

`void EDMA_TcdResetExt(EDMA_Type *base, edma_tcd_t *tcd)`

Sets all fields to default values for the TCD structure.

This function sets all fields for this TCD structure to default value.

Note: This function enables the auto stop request feature.

Parameters

- `base` – eDMA peripheral base address.
- `tcd` – Pointer to the TCD structure.

`void EDMA_TcdSetTransferConfigExt(EDMA_Type *base, edma_tcd_t *tcd, const edma_transfer_config_t *config, edma_tcd_t *nextTcd)`

Configures the eDMA TCD transfer attribute.

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The TCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

```
edma_transfer_t config = {
...
}
edma_tcd_t tcd __aligned(32);
edma_tcd_t nextTcd __aligned(32);
EDMA_TcdSetTransferConfig(&tcd, &config, &nextTcd);
```

Note: TCD address should be 32 bytes aligned or it causes an eDMA error.

Note: If the `nextTcd` is not NULL, the scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the `EDMA_TcdReset`.

Parameters

- `base` – eDMA peripheral base address.
- `tcd` – Pointer to the TCD structure.
- `config` – Pointer to eDMA transfer configuration structure.
- `nextTcd` – Pointer to the next TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

```
void EDMA__TcdSetMinorOffsetConfigExt(EDMA_Type *base, edma_tcd_t *tcd, const  
                                     edma_minor_offset_config_t *config)
```

Configures the eDMA TCD minor offset feature.

A minor offset is a signed-extended value added to the source address or a destination address after each minor loop.

Parameters

- base – eDMA peripheral base address.
- tcd – A point to the TCD structure.
- config – A pointer to the minor offset configuration structure.

```
void EDMA__TcdSetChannelLinkExt(EDMA_Type *base, edma_tcd_t *tcd,  
                               edma_channel_link_type_t type, uint32_t linkedChannel)
```

Sets the channel link for the eDMA TCD.

This function configures either a minor link or a major link. The minor link means the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note: Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

- base – eDMA peripheral base address.
- tcd – Point to the TCD structure.
- type – Channel link type, it can be one of:
 - kEDMA_LinkNone
 - kEDMA_MinorLink
 - kEDMA_MajorLink
- linkedChannel – The linked channel number.

```
static inline void EDMA__TcdSetBandWidthExt(EDMA_Type *base, edma_tcd_t *tcd,  
                                             edma_bandwidth_t bandWidth)
```

Sets the bandwidth for the eDMA TCD.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- base – eDMA peripheral base address.
- tcd – A pointer to the TCD structure.
- bandWidth – A bandwidth setting, which can be one of the following:
 - kEDMABandwidthStallNone
 - kEDMABandwidthStall4Cycle
 - kEDMABandwidthStall8Cycle

```
void EDMA__TcdSetModuloExt(EDMA_Type *base, edma_tcd_t *tcd, edma_modulo_t srcModulo,  
                           edma_modulo_t destModulo)
```

Sets the source modulo and the destination modulo for the eDMA TCD.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- base – eDMA peripheral base address.
- tcd – A pointer to the TCD structure.
- srcModulo – A source modulo value.
- destModulo – A destination modulo value.

```
static inline void EDMA__TcdEnableAutoStopRequestExt(EDMA_Type *base, edma_tcd_t *tcd,  
                                                    bool enable)
```

Sets the auto stop request for the eDMA TCD.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

- base – eDMA peripheral base address.
- tcd – A pointer to the TCD structure.
- enable – The command to enable (true) or disable (false).

```
void EDMA__TcdEnableInterruptsExt(EDMA_Type *base, edma_tcd_t *tcd, uint32_t mask)
```

Enables the interrupt source for the eDMA TCD.

Parameters

- base – eDMA peripheral base address.
- tcd – Point to the TCD structure.
- mask – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

```
void EDMA__TcdDisableInterruptsExt(EDMA_Type *base, edma_tcd_t *tcd, uint32_t mask)
```

Disables the interrupt source for the eDMA TCD.

Parameters

- base – eDMA peripheral base address.
- tcd – Point to the TCD structure.
- mask – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

```
void EDMA__TcdSetMajorOffsetConfigExt(EDMA_Type *base, edma_tcd_t *tcd, int32_t  
                                       sourceOffset, int32_t destOffset)
```

Configures the eDMA TCD major offset feature.

Adjustment value added to the source address at the completion of the major iteration count

Parameters

- base – eDMA peripheral base address.
- tcd – A point to the TCD structure.
- sourceOffset – source address offset will be applied to source address after major loop done.

- `destOffset` – destination address offset will be applied to source address after major loop done.

static inline void `EDMA_EnableChannelRequest`(*EDMA_Type* *base, uint32_t channel)

Enables the eDMA hardware channel request.

This function enables the hardware channel request.

Parameters

- `base` – eDMA peripheral base address.
- `channel` – eDMA channel number.

static inline void `EDMA_DisableChannelRequest`(*EDMA_Type* *base, uint32_t channel)

Disables the eDMA hardware channel request.

This function disables the hardware channel request.

Parameters

- `base` – eDMA peripheral base address.
- `channel` – eDMA channel number.

static inline void `EDMA_TriggerChannelStart`(*EDMA_Type* *base, uint32_t channel)

Starts the eDMA transfer by using the software trigger.

This function starts a minor loop transfer.

Parameters

- `base` – eDMA peripheral base address.
- `channel` – eDMA channel number.

uint32_t `EDMA_GetRemainingMajorLoopCount`(*EDMA_Type* *base, uint32_t channel)

Gets the remaining major loop count from the eDMA current channel TCD.

This function checks the TCD (Task Control Descriptor) status for a specified eDMA channel and returns the number of major loop count that has not finished.

Note: 1. This function can only be used to get unfinished major loop count of transfer without the next TCD, or it might be inaccuracy.

- a. The unfinished/remaining transfer bytes cannot be obtained directly from registers while the channel is running. Because to calculate the remaining bytes, the initial NBYTES configured in `DMA_TCDn_NBYTES_MLNO` register is needed while the eDMA IP does not support getting it while a channel is active. In another word, the NBYTES value reading is always the actual (decrementing) NBYTES value the `dma_engine` is working with while a channel is running. Consequently, to get the remaining transfer bytes, a software-saved initial value of NBYTES (for example copied before enabling the channel) is needed. The formula to calculate it is shown below: $\text{RemainingBytes} = \text{RemainingMajorLoopCount} * \text{NBYTES}(\text{initially configured})$

Parameters

- `base` – eDMA peripheral base address.
- `channel` – eDMA channel number.

Returns

Major loop count which has not been transferred yet for the current TCD.

static inline uint32_t EDMA_GetErrorStatusFlags(*EDMA_Type* *base)

Gets the eDMA channel error status flags.

Parameters

- base – eDMA peripheral base address.

Returns

The mask of error status flags. Users need to use the `_edma_error_status_flags` type to decode the return variables.

uint32_t EDMA_GetChannelStatusFlags(*EDMA_Type* *base, uint32_t channel)

Gets the eDMA channel status flags.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

Returns

The mask of channel status flags. Users need to use the `_edma_channel_status_flags` type to decode the return variables.

void EDMA_ClearChannelStatusFlags(*EDMA_Type* *base, uint32_t channel, uint32_t mask)

Clears the eDMA channel status flags.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of channel status to be cleared. Users need to use the defined `_edma_channel_status_flags` type.

status_t EDMA_CreateHandle(*edma_handle_t* *handle, *EDMA_Type* *base, uint32_t channel)

Creates the eDMA handle.

This function is called if using the transactional API for eDMA. This function initializes the internal state of the eDMA handle.

Parameters

- handle – eDMA handle pointer. The eDMA handle stores callback function and parameters.
- base – eDMA peripheral base address.
- channel – eDMA channel number.

Return values

- `kStatus_Success` –
- `kStatus_InvalidArgument` –

void EDMA_InstallTCDMemory(*edma_handle_t* *handle, *edma_tcd_t* *tcdPool, uint32_t tcdSize)

Installs the TCDs memory pool into the eDMA handle.

This function is called after the `EDMA_CreateHandle` to use scatter/gather feature. This function shall only be used while users need to use scatter gather mode. Scatter gather mode enables EDMA to load a new transfer control block (tcd) in hardware, and automatically reconfigure that DMA channel for a new transfer. Users need to prepare tcd memory and also configure tcds using interface `EDMA_SubmitTransfer`.

Parameters

- handle – eDMA handle pointer.

- `tcdPool` – A memory pool to store TCDs. It must be 32 bytes aligned.
- `tcdSize` – The number of TCD slots.

`void EDMA_SetCallback(edma_handle_t *handle, edma_callback callback, void *userData)`

Installs a callback function for the eDMA transfer.

This callback is called in the eDMA IRQ handler. Use the callback to do something after the current major loop transfer completes. This function will be called every time one tcd finished transfer.

Parameters

- `handle` – eDMA handle pointer.
- `callback` – eDMA callback function pointer.
- `userData` – A parameter for the callback function.

`void EDMA_PrepareTransferConfig(edma_transfer_config_t *config, void *srcAddr, uint32_t srcWidth, int16_t srcOffset, void *destAddr, uint32_t destWidth, int16_t destOffset, uint32_t bytesEachRequest, uint32_t transferBytes)`

Prepares the eDMA transfer structure configurations.

This function prepares the transfer configuration structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE). User can check if 128 bytes support is available for specific instance by `FSL_FEATURE_EDMA_INSTANCE_SUPPORT_128_BYTES_TRANSFERn`.

Parameters

- `config` – The user configuration structure of type `edma_transfer_t`.
- `srcAddr` – eDMA transfer source address.
- `srcWidth` – eDMA transfer source address width(bytes).
- `srcOffset` – source address offset.
- `destAddr` – eDMA transfer destination address.
- `destWidth` – eDMA transfer destination address width(bytes).
- `destOffset` – destination address offset.
- `bytesEachRequest` – eDMA transfer bytes per channel request.
- `transferBytes` – eDMA transfer bytes to be transferred.

`void EDMA_PrepareTransfer(edma_transfer_config_t *config, void *srcAddr, uint32_t srcWidth, void *destAddr, uint32_t destWidth, uint32_t bytesEachRequest, uint32_t transferBytes, edma_transfer_type_t type)`

Prepares the eDMA transfer structure.

This function prepares the transfer configuration structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

Parameters

- `config` – The user configuration structure of type `edma_transfer_t`.
- `srcAddr` – eDMA transfer source address.
- `srcWidth` – eDMA transfer source address width(bytes).
- `destAddr` – eDMA transfer destination address.
- `destWidth` – eDMA transfer destination address width(bytes).
- `bytesEachRequest` – eDMA transfer bytes per channel request.
- `transferBytes` – eDMA transfer bytes to be transferred.
- `type` – eDMA transfer type.

```
void EDMA__PrepareTransferTCD(edma_handle_t *handle, edma_tcd_t *tcd, void *srcAddr,
                             uint32_t srcWidth, int16_t srcOffset, void *destAddr, uint32_t
                             destWidth, int16_t destOffset, uint32_t bytesEachRequest,
                             uint32_t transferBytes, edma_tcd_t *nextTcd)
```

Prepares the eDMA transfer content descriptor.

This function prepares the transfer content descriptor structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

Parameters

- `handle` – eDMA handle pointer.
- `tcd` – Pointer to eDMA transfer content descriptor structure.
- `srcAddr` – eDMA transfer source address.
- `srcWidth` – eDMA transfer source address width(bytes).
- `srcOffset` – source address offset.
- `destAddr` – eDMA transfer destination address.
- `destWidth` – eDMA transfer destination address width(bytes).
- `destOffset` – destination address offset.
- `bytesEachRequest` – eDMA transfer bytes per channel request.
- `transferBytes` – eDMA transfer bytes to be transferred.
- `nextTcd` – eDMA transfer linked TCD address.

```
status_t EDMA__SubmitTransferTCD(edma_handle_t *handle, edma_tcd_t *tcd)
```

Submits the eDMA transfer content descriptor.

This function submits the eDMA transfer request according to the transfer content descriptor. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function `EDMA__InstallTCDMemory` before.

Typical user case:

a. submit single transfer

```
edma_tcd_t tcd;
EDMA__PrepareTransferTCD(handle, tcd, ...)
EDMA__SubmitTransferTCD(handle, tcd)
EDMA__StartTransfer(handle)
```

b. submit static link transfer,

```
edma_tcd_t tcd[2];
EDMA_PrepareTransferTCD(handle, &tcd[0], ...)
EDMA_PrepareTransferTCD(handle, &tcd[1], ...)
EDMA_SubmitTransferTCD(handle, &tcd[0])
EDMA_StartTransfer(handle)
```

c. submit dynamic link transfer

```
edma_tcd_t tcdpool[2];
EDMA_InstallTCDMemory(&g_DMA_Handle, tcdpool, 2);
edma_tcd_t tcd;
EDMA_PrepareTransferTCD(handle, tcd, ...)
EDMA_SubmitTransferTCD(handle, tcd)
EDMA_PrepareTransferTCD(handle, tcd, ...)
EDMA_SubmitTransferTCD(handle, tcd)
EDMA_StartTransfer(handle)
```

d. submit loop transfer

```
edma_tcd_t tcd[2];
EDMA_PrepareTransferTCD(handle, &tcd[0], ..., &tcd[1])
EDMA_PrepareTransferTCD(handle, &tcd[1], ..., &tcd[0])
EDMA_SubmitTransferTCD(handle, &tcd[0])
EDMA_StartTransfer(handle)
```

Parameters

- handle – eDMA handle pointer.
- tcd – Pointer to eDMA transfer content descriptor structure.

Return values

- kStatus_EDMA_Success – It means submit transfer request succeed.
- kStatus_EDMA_QueueFull – It means TCD queue is full. Submit transfer request is not allowed.
- kStatus_EDMA_Busy – It means the given channel is busy, need to submit request later.

status_t EDMA_SubmitTransfer(*edma_handle_t* *handle, const *edma_transfer_config_t* *config)

Submits the eDMA transfer request.

This function submits the eDMA transfer request according to the transfer configuration structure. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function EDMA_InstallTCDMemory before.

Parameters

- handle – eDMA handle pointer.
- config – Pointer to eDMA transfer configuration structure.

Return values

- kStatus_EDMA_Success – It means submit transfer request succeed.
- kStatus_EDMA_QueueFull – It means TCD queue is full. Submit transfer request is not allowed.
- kStatus_EDMA_Busy – It means the given channel is busy, need to submit request later.


```
status_t EDMA_SubmitLoopTransfer(edma_handle_t *handle, edma_transfer_config_t *transfer,
                                uint32_t transferLoopCount)
```

Submits the eDMA scatter gather transfer configurations.

The function is target for submit loop transfer request, the ring transfer request means that the transfer request TAIL is link to HEAD, such as, A->B->C->D->A, or A->A

To use the ring transfer feature, the application should allocate several transfer object, such as

```
edma_channel_transfer_config_t transfer[2];
EDMA_TransferSubmitLoopTransfer(psHandle, &transfer, 2U);
```

Then eDMA driver will link transfer[0] and transfer[1] to each other

Note: Application should check the return value of this function to avoid transfer request submit failed

Parameters

- handle – eDMA handle pointer
- transfer – pointer to user's eDMA channel configure structure, see edma_channel_transfer_config_t for detail
- transferLoopCount – the count of the transfer ring, if loop count is 1, that means that the one will link to itself.

Return values

- kStatus_Success – It means submit transfer request succeed
- kStatus_EDMA_Busy – channel is in busy status
- kStatus_InvalidArgument – Invalid Argument

```
void EDMA_StartTransfer(edma_handle_t *handle)
```

eDMA starts transfer.

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

Parameters

- handle – eDMA handle pointer.

```
void EDMA_StopTransfer(edma_handle_t *handle)
```

eDMA stops transfer.

This function disables the channel request to pause the transfer. Users can call EDMA_StartTransfer() again to resume the transfer.

Parameters

- handle – eDMA handle pointer.

```
void EDMA_AbortTransfer(edma_handle_t *handle)
```

eDMA aborts transfer.

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

Parameters

- handle – DMA handle pointer.

static inline uint32_t EDMA_GetUnusedTCDNumber(*edma_handle_t* *handle)

Get unused TCD slot number.

This function gets current tcd index which is run. If the TCD pool pointer is NULL, it will return 0.

Parameters

- handle – DMA handle pointer.

Returns

The unused tcd slot number.

static inline uint32_t EDMA_GetNextTCDAddress(*edma_handle_t* *handle)

Get the next tcd address.

This function gets the next tcd address. If this is last TCD, return 0.

Parameters

- handle – DMA handle pointer.

Returns

The next TCD address.

void EDMA_HandleIRQ(*edma_handle_t* *handle)

eDMA IRQ handler for the current major loop transfer completion.

This function clears the channel major interrupt flag and calls the callback function if it is not NULL.

Note: For the case using TCD queue, when the major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Assertion of an optional interrupt request also occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor (if scatter/gather is enabled).

For instance, when the time interrupt of TCD[0] happens, the TCD[1] has already been loaded into the eDMA engine. As sga and sga_index are calculated based on the DLAST_SGA bitfield lies in the TCD_CSR register, the sga_index in this case should be 2 (DLAST_SGA of TCD[1] stores the address of TCD[2]). Thus, the “tcdUsed” updated should be (tcdUsed - 2U) which indicates the number of TCDs can be loaded in the memory pool (because TCD[0] and TCD[1] have been loaded into the eDMA engine at this point already).

For the last two continuous ISRs in a scatter/gather process, they both load the last TCD (The last ISR does not load a new TCD) from the memory pool to the eDMA engine when major loop completes. Therefore, ensure that the header and tcdUsed updated are identical for them. tcdUsed are both 0 in this case as no TCD to be loaded.

See the “eDMA basic data flow” in the eDMA Functional description section of the Reference Manual for further details.

Parameters

- handle – eDMA handle pointer.

FSL_EDMA_DRIVER_VERSION

eDMA driver version

Version 2.10.7.

_edma_transfer_status eDMA transfer status

Values:

enumerator kStatus_EDMA_QueueFull

TCD queue is full.

enumerator kStatus_EDMA_Busy

Channel is busy and can't handle the transfer request.

enum _edma_transfer_size

eDMA transfer configuration

Values:

enumerator kEDMA_TransferSize1Bytes

Source/Destination data transfer size is 1 byte every time

enumerator kEDMA_TransferSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kEDMA_TransferSize4Bytes

Source/Destination data transfer size is 4 bytes every time

enumerator kEDMA_TransferSize8Bytes

Source/Destination data transfer size is 8 bytes every time

enumerator kEDMA_TransferSize16Bytes

Source/Destination data transfer size is 16 bytes every time

enumerator kEDMA_TransferSize32Bytes

Source/Destination data transfer size is 32 bytes every time

enumerator kEDMA_TransferSize64Bytes

Source/Destination data transfer size is 64 bytes every time

enumerator kEDMA_TransferSize128Bytes

Source/Destination data transfer size is 128 bytes every time

enum _edma_modulo

eDMA modulo configuration

Values:

enumerator kEDMA_ModuloDisable

Disable modulo

enumerator kEDMA_Modulo2bytes

Circular buffer size is 2 bytes.

enumerator kEDMA_Modulo4bytes

Circular buffer size is 4 bytes.

enumerator kEDMA_Modulo8bytes

Circular buffer size is 8 bytes.

enumerator kEDMA_Modulo16bytes

Circular buffer size is 16 bytes.

enumerator kEDMA_Modulo32bytes

Circular buffer size is 32 bytes.

enumerator kEDMA_Modulo64bytes

Circular buffer size is 64 bytes.

enumerator kEDMA_Modulo128bytes

Circular buffer size is 128 bytes.

enumerator kEDMA_Modulo256bytes
Circular buffer size is 256 bytes.

enumerator kEDMA_Modulo512bytes
Circular buffer size is 512 bytes.

enumerator kEDMA_Modulo1Kbytes
Circular buffer size is 1 K bytes.

enumerator kEDMA_Modulo2Kbytes
Circular buffer size is 2 K bytes.

enumerator kEDMA_Modulo4Kbytes
Circular buffer size is 4 K bytes.

enumerator kEDMA_Modulo8Kbytes
Circular buffer size is 8 K bytes.

enumerator kEDMA_Modulo16Kbytes
Circular buffer size is 16 K bytes.

enumerator kEDMA_Modulo32Kbytes
Circular buffer size is 32 K bytes.

enumerator kEDMA_Modulo64Kbytes
Circular buffer size is 64 K bytes.

enumerator kEDMA_Modulo128Kbytes
Circular buffer size is 128 K bytes.

enumerator kEDMA_Modulo256Kbytes
Circular buffer size is 256 K bytes.

enumerator kEDMA_Modulo512Kbytes
Circular buffer size is 512 K bytes.

enumerator kEDMA_Modulo1Mbytes
Circular buffer size is 1 M bytes.

enumerator kEDMA_Modulo2Mbytes
Circular buffer size is 2 M bytes.

enumerator kEDMA_Modulo4Mbytes
Circular buffer size is 4 M bytes.

enumerator kEDMA_Modulo8Mbytes
Circular buffer size is 8 M bytes.

enumerator kEDMA_Modulo16Mbytes
Circular buffer size is 16 M bytes.

enumerator kEDMA_Modulo32Mbytes
Circular buffer size is 32 M bytes.

enumerator kEDMA_Modulo64Mbytes
Circular buffer size is 64 M bytes.

enumerator kEDMA_Modulo128Mbytes
Circular buffer size is 128 M bytes.

enumerator kEDMA_Modulo256Mbytes
Circular buffer size is 256 M bytes.

enumerator kEDMA_Modulo512Mbytes

Circular buffer size is 512 M bytes.

enumerator kEDMA_Modulo1Gbytes

Circular buffer size is 1 G bytes.

enumerator kEDMA_Modulo2Gbytes

Circular buffer size is 2 G bytes.

enum _edma_bandwidth

Bandwidth control.

Values:

enumerator kEDMA_BandwidthStallNone

No eDMA engine stalls.

enumerator kEDMA_BandwidthStall4Cycle

eDMA engine stalls for 4 cycles after each read/write.

enumerator kEDMA_BandwidthStall8Cycle

eDMA engine stalls for 8 cycles after each read/write.

enum _edma_channel_link_type

Channel link type.

Values:

enumerator kEDMA_LinkNone

No channel link

enumerator kEDMA_MinorLink

Channel link after each minor loop

enumerator kEDMA_MajorLink

Channel link while major loop count exhausted

_edma_channel_status_flags eDMA channel status flags.

Values:

enumerator kEDMA_DoneFlag

DONE flag, set while transfer finished, CITER value exhausted

enumerator kEDMA_ErrorFlag

eDMA error flag, an error occurred in a transfer

enumerator kEDMA_InterruptFlag

eDMA interrupt flag, set while an interrupt occurred of this channel

_edma_error_status_flags eDMA channel error status flags.

Values:

enumerator kEDMA_DestinationBusErrorFlag

Bus error on destination address

enumerator kEDMA_SourceBusErrorFlag

Bus error on the source address

enumerator kEDMA_ScatterGatherErrorFlag

Error on the Scatter/Gather address, not 32byte aligned.

enumerator kEDMA_NbytesErrorFlag
NBYTES/CITER configuration error

enumerator kEDMA_DestinationOffsetErrorFlag
Destination offset not aligned with destination size

enumerator kEDMA_DestinationAddressErrorFlag
Destination address not aligned with destination size

enumerator kEDMA_SourceOffsetErrorFlag
Source offset not aligned with source size

enumerator kEDMA_SourceAddressErrorFlag
Source address not aligned with source size

enumerator kEDMA_ErrorChannelFlag
Error channel number of the cancelled channel number

enumerator kEDMA_TransferCanceledFlag
Transfer cancelled

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

_edma_interrupt_enable eDMA interrupt source

Values:

enumerator kEDMA_ErrorInterruptEnable
Enable interrupt while channel error occurs.

enumerator kEDMA_MajorInterruptEnable
Enable interrupt while major count exhausted.

enumerator kEDMA_HalfInterruptEnable
Enable interrupt while major count to half value.

enum _edma_transfer_type

eDMA transfer type

Values:

enumerator kEDMA_MemoryToMemory
Transfer from memory to memory

enumerator kEDMA_PeripheralToMemory
Transfer from peripheral to memory

enumerator kEDMA_MemoryToPeripheral
Transfer from memory to peripheral

enumerator kEDMA_PeripheralToPeripheral
Transfer from Peripheral to peripheral

enum edma_channel_memory_attribute

eDMA channel memory attribute

Values:

enumerator kEDMA_ChannelNoWriteNoReadNoCacheNoBuffer
No write allocate, no read allocate, non-cacheable, non-bufferable.

enumerator kEDMA_ChannelNoWriteNoReadNoCacheBufferable

No write allocate, no read allocate, non-cacheable, bufferable.

enumerator kEDMA_ChannelNoWriteNoReadCacheableNoBuffer

No write allocate, no read allocate, cacheable, non-bufferable.

enumerator kEDMA_ChannelNoWriteNoReadCacheableBufferable

No write allocate, no read allocate, cacheable, bufferable.

enumerator kEDMA_ChannelNoWriteReadNoCacheNoBuffer

No write allocate, read allocate, non-cacheable, non-bufferable.

enumerator kEDMA_ChannelNoWriteReadNoCacheBufferable

No write allocate, read allocate, non-cacheable, bufferable.

enumerator kEDMA_ChannelNoWriteReadCacheableNoBuffer

No write allocate, read allocate, cacheable, non-bufferable.

enumerator kEDMA_ChannelNoWriteReadCacheableBufferable

No write allocate, read allocate, cacheable, bufferable.

enumerator kEDMA_ChannelWriteNoReadNoCacheNoBuffer

write allocate, no read allocate, non-cacheable, non-bufferable.

enumerator kEDMA_ChannelWriteNoReadNoCacheBufferable

write allocate, no read allocate, non-cacheable, bufferable.

enumerator kEDMA_ChannelWriteNoReadCacheableNoBuffer

write allocate, no read allocate, cacheable, non-bufferable.

enumerator kEDMA_ChannelWriteNoReadCacheableBufferable

write allocate, no read allocate, cacheable, bufferable.

enumerator kEDMA_ChannelWriteReadNoCacheNoBuffer

write allocate, read allocate, non-cacheable, non-bufferable.

enumerator kEDMA_ChannelWriteReadNoCacheBufferable

write allocate, read allocate, non-cacheable, bufferable.

enumerator kEDMA_ChannelWriteReadCacheableNoBuffer

write allocate, read allocate, cacheable, non-bufferable.

enumerator kEDMA_ChannelWriteReadCacheableBufferable

write allocate, read allocate, cacheable, bufferable.

enum _edma_channel_swap_size

eDMA4 channel swap size

Values:

enumerator kEDMA_ChannelSwapDisabled

Swap is disabled.

enumerator kEDMA_ChannelReadWith8bitSwap

Swap occurs with respect to the read 8bit.

enumerator kEDMA_ChannelReadWith16bitSwap

Swap occurs with respect to the read 16bit.

enumerator kEDMA_ChannelReadWith32bitSwap

Swap occurs with respect to the read 32bit.

enumerator kEDMA_ChannelWriteWith8bitSwap

Swap occurs with respect to the write 8bit.

enumerator kEDMA_ChannelWriteWith16bitSwap

Swap occurs with respect to the write 16bit.

enumerator kEDMA_ChannelWriteWith32bitSwap

Swap occurs with respect to the write 32bit.

eDMA channel system bus information, _edma_channel_sys_bus_info

Values:

enumerator kEDMA_PrivilegedAccessLevel

Privileged Access Level for DMA transfers. 0b - User protection level; 1b - Privileged protection level.

enumerator kEDMA_MasterId

DMA's master ID when channel is active and master ID replication is enabled.

enum _edma_channel_access_type

eDMA4 channel access type

Values:

enumerator kEDMA_ChannelDataAccess

Data access for eDMA4 transfers.

enumerator kEDMA_ChannelInstructionAccess

Instruction access for eDMA4 transfers.

enum _edma_channel_protection_level

eDMA4 channel protection level

Values:

enumerator kEDMA_ChannelProtectionLevelUser

user protection level for eDMA transfers.

enumerator kEDMA_ChannelProtectionLevelPrivileged

Privileged protection level eDMA transfers.

typedef enum _edma_transfer_size edma_transfer_size_t

eDMA transfer configuration

typedef enum _edma_modulo edma_modulo_t

eDMA modulo configuration

typedef enum _edma_bandwidth edma_bandwidth_t

Bandwidth control.

typedef enum _edma_channel_link_type edma_channel_link_type_t

Channel link type.

typedef enum _edma_transfer_type edma_transfer_type_t

eDMA transfer type

typedef struct _edma_channel_Preemption_config edma_channel_Preemption_config_t

eDMA channel priority configuration

typedef struct _edma_minor_offset_config edma_minor_offset_config_t

eDMA minor offset configuration


```
typedef enum edma_channel_memory_attribute edma_channel_memory_attribute_t
    eDMA channel memory attribute
```

```
typedef enum _edma_channel_swap_size edma_channel_swap_size_t
    eDMA4 channel swap size
```

```
typedef enum _edma_channel_access_type edma_channel_access_type_t
    eDMA4 channel access type
```

```
typedef enum _edma_channel_protection_level edma_channel_protection_level_t
    eDMA4 channel protection level
```

```
typedef struct _edma_channel_config edma_channel_config_t
    eDMA4 channel configuration
```

```
typedef edma_core_tcd_t edma_tcd_t
    eDMA TCD.
```

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

```
typedef struct _edma_transfer_config edma_transfer_config_t
    edma4 channel transfer configuration
```

The transfer configuration structure support full feature configuration of the transfer control descriptor.

1.To perform a simple transfer, below members should be initialized at least .srcAddr - source address .dstAddr - destination address .srcWidthOfEachTransfer - data width of source address .dstWidthOfEachTransfer - data width of destination address, normally it should be as same as srcWidthOfEachTransfer .bytesEachRequest - bytes to be transferred in each DMA request .totalBytes - total bytes to be transferred .srcOffsetOfEachTransfer - offset value in bytes unit to be applied to source address as each source read is completed .dstOffsetOfEachTransfer - offset value in bytes unit to be applied to destination address as each destination write is completed enablchannelRequest - channel request can be enabled together with transfer configure submission

2.The transfer configuration structure also support advance feature: Programmable source/destination address range(MODULO) Programmable minor loop offset Programmable major loop offset Programmable channel chain feature Programmable channel transfer control descriptor link feature

Note: User should pay attention to the transfer size alignment limitation

- a. the bytesEachRequest should align with the srcWidthOfEachTransfer and the dstWidthOfEachTransfer that is to say bytesEachRequest % srcWidthOfEachTransfer should be 0
 - b. the srcOffsetOfEachTransfer and dstOffsetOfEachTransfer must be aligne with transfer width
 - c. the totalBytes should align with the bytesEachRequest
 - d. the srcAddr should align with the srcWidthOfEachTransfer
 - e. the dstAddr should align with the dstWidthOfEachTransfer
 - f. the srcAddr should align with srcAddrModulo if modulo feature is enabled
 - g. the dstAddr should align with dstAddrModulo if modulo feature is enabled If anyone of above condition can not be satisfied, the edma4 interfaces will generate assert error.
-

```
typedef struct _edma_config edma_config_t
```

eDMA global configuration structure.

```
typedef void (*edma_callback)(struct _edma_handle *handle, void *userData, bool transferDone,
uint32_t tcds)
```

Define callback function for eDMA.

This callback function is called in the EDMA interrupt handle. In normal mode, run into callback function means the transfer users need is done. In scatter gather mode, run into callback function means a transfer control block (tcd) is finished. Not all transfer finished, users can get the finished tcd numbers using interface EDMA_GetUnusedTCDNumber.

Param handle

EDMA handle pointer, users shall not touch the values inside.

Param userData

The callback user parameter pointer. Users can use this parameter to involve things users need to change in EDMA callback function.

Param transferDone

If the current loaded transfer done. In normal mode it means if all transfer done. In scatter gather mode, this parameter shows is the current transfer block in EDMA register is done. As the load of core is different, it will be different if the new tcd loaded into EDMA registers while this callback called. If true, it always means new tcd still not loaded into registers, while false means new tcd already loaded into registers.

Param tcds

How many tcds are done from the last callback. This parameter only used in scatter gather mode. It tells user how many tcds are finished between the last callback and this.

```
typedef struct _edma_handle edma_handle_t
```

eDMA transfer handle structure

```
FSL_EDMA_DRIVER_EDMA4
```

eDMA driver name

```
EDMA_ALLOCATE_TCD(name, number)
```

Macro used for allocate edma TCD.

```
DMA_DCHPRI_INDEX(channel)
```

Compute the offset unit from DCHPRI3.

```
struct _edma_channel_Preemption_config
```

#include <fsl_edma.h> eDMA channel priority configuration

Public Members

```
bool enableChannelPreemption
```

If true: a channel can be suspended by other channel with higher priority

```
bool enablePreemptAbility
```

If true: a channel can suspend other channel with low priority

```
uint8_t channelPriority
```

Channel priority

```
struct _edma_minor_offset_config
```

#include <fsl_edma.h> eDMA minor offset configuration

Public Members

`bool enableSrcMinorOffset`
 Enable(true) or Disable(false) source minor loop offset.

`bool enableDestMinorOffset`
 Enable(true) or Disable(false) destination minor loop offset.

`uint32_t minorOffset`
 Offset for a minor loop mapping.

`struct __edma_channel_config`
`#include <fsl_edma.h>` eDMA4 channel configuration

Public Members

`edma_channel_preemption_config_t` channelPreemptionConfig
 channel preemption configuration

`edma_channel_memory_attribute_t` channelReadMemoryAttribute
 channel memory read attribute configuration

`edma_channel_memory_attribute_t` channelWriteMemoryAttribute
 channel memory write attribute configuration

`edma_channel_swap_size_t` channelSwapSize
 channel swap size configuration

`edma_channel_access_type_t` channelAccessType
 channel access type configuration

`uint8_t` channelDataSignExtensionBitPosition
 channel data sign extension bit position configuration

`uint32_t` channelRequestSource
 hardware service request source for the channel

`bool enableMasterIDReplication`
 enable master ID replication

`edma_channel_protection_level_t` protectionLevel
 protection level

`struct __edma_transfer_config`
`#include <fsl_edma.h>` edma4 channel transfer configuration

The transfer configuration structure support full feature configuration of the transfer control descriptor.

1.To perform a simple transfer, below members should be initialized at least .srcAddr - source address .dstAddr - destination address .srcWidthOfEachTransfer - data width of source address .dstWidthOfEachTransfer - data width of destination address, normally it should be as same as srcWidthOfEachTransfer .bytesEachRequest - bytes to be transferred in each DMA request .totalBytes - total bytes to be transferred .srcOffsetOfEachTransfer - offset value in bytes unit to be applied to source address as each source read is completed .dstOffsetOfEachTransfer - offset value in bytes unit to be applied to destination address as each destination write is completed enablchannelRequest - channel request can be enabled together with transfer configure submission

2.The transfer configuration structure also support advance feature: Programmable source/destination address range(MODULO) Programmable minor loop offset Programmable major loop offset Programmable channel chain feature Programmable channel transfer control descriptor link feature

Note: User should pay attention to the transfer size alignment limitation

- a. the bytesEachRequest should align with the srcWidthOfEachTransfer and the dstWidthOfEachTransfer that is to say bytesEachRequest % srcWidthOfEachTransfer should be 0
 - b. the srcOffsetOfEachTransfer and dstOffsetOfEachTransfer must be aligne with transfer width
 - c. the totalBytes should align with the bytesEachRequest
 - d. the srcAddr should align with the srcWidthOfEachTransfer
 - e. the dstAddr should align with the dstWidthOfEachTransfer
 - f. the srcAddr should align with srcAddrModulo if modulo feature is enabled
 - g. the dstAddr should align with dstAddrModulo if modulo feature is enabled If anyone of above condition can not be satisfied, the edma4 interfaces will generate assert error.
-

Public Members

uint32_t srcAddr

Source data address.

uint32_t destAddr

Destination data address.

edma_transfer_size_t srcTransferSize

Source data transfer size.

edma_transfer_size_t destTransferSize

Destination data transfer size.

int16_t srcOffset

Sign-extended offset value in byte unit applied to the current source address to form the next-state value as each source read is completed

int16_t destOffset

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

uint32_t minorLoopBytes

bytes in each minor loop or each request range: 1 - (2^{30} - 1) when minor loop mapping is enabled range: 1 - (2^{10} - 1) when minor loop mapping is enabled and source or dest minor loop offset is enabled range: 1 - (2^{32} - 1) when minor loop mapping is disabled

uint32_t majorLoopCounts

minor loop counts in each major loop, should be 1 at least for each transfer range: (0 - (2^{15} - 1)) when minor loop channel link is disabled range: (0 - (2^9 - 1)) when minor loop channel link is enabled total bytes in a transfer = minorLoopCountsEachMajorLoop * bytesEachMinorLoop

uint16_t enabledInterruptMask

channel interrupt to enable, can be OR'ed value of _edma_interrupt_enable

```

    edma_modulo_t srcAddrModulo
        source circular data queue range
    int32_t srcMajorLoopOffset
        source major loop offset
    edma_modulo_t dstAddrModulo
        destination circular data queue range
    int32_t dstMajorLoopOffset
        destination major loop offset
    bool enableSrcMinorLoopOffset
        enable source minor loop offset
    bool enableDstMinorLoopOffset
        enable dest minor loop offset
    int32_t minorLoopOffset
        burst offset, the offset will be applied after minor loop update
    bool enableChannelMajorLoopLink
        channel link when major loop complete
    uint32_t majorLoopLinkChannel
        major loop link channel number
    bool enableChannelMinorLoopLink
        channel link when minor loop complete
    uint32_t minorLoopLinkChannel
        minor loop link channel number
    edma_tcd_t *linkTCD
        pointer to the link transfer control descriptor
struct __edma_config
    #include <fsl_edma.h> eDMA global configuration structure.

```

Public Members

```

bool enableContinuousLinkMode
    Enable (true) continuous link mode. Upon minor loop completion, the channel activates again if that channel has a minor loop channel link enabled and the link channel is itself.

bool enableMasterIdReplication
    Enable (true) master ID replication. If Master ID replication is disabled, the privileged protection level (supervisor mode) for eDMA4 transfers is used.

bool enableGlobalChannelLink
    Enable(true) channel linking is available and controlled by each channel's link settings.

bool enableHaltOnError
    Enable (true) transfer halt on error. Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

bool enableDebugMode
    Enable(true) eDMA4 debug mode. When in debug mode, the eDMA4 stalls the start of a new channel. Executing channels are allowed to complete.

```

bool enableRoundRobinArbitration

Enable(true) channel linking is available and controlled by each channel's link settings.

edma_channel_config_t *channelConfig[1]

channel preemption configuration

struct *_edma_handle*

#include <fsl_edma.h> eDMA transfer handle structure

Public Members

edma_callback callback

Callback function for major count exhausted.

void *userData

Callback function parameter.

EDMA_ChannelType *channelBase

eDMA peripheral channel base address.

EDMA_Type *base

eDMA peripheral base address

EDMA_TCDType *tcdBase

eDMA peripheral tcd base address.

edma_tcd_t *tcdPool

Pointer to memory stored TCDs.

uint32_t channel

eDMA channel number.

volatile int8_t header

The first TCD index. Should point to the next TCD to be loaded into the eDMA engine.

volatile int8_t tail

The last TCD index. Should point to the next TCD to be stored into the memory pool.

volatile int8_t tcdUsed

The number of used TCD slots. Should reflect the number of TCDs can be used/loaded in the memory.

volatile int8_t tcdSize

The total number of TCD slots in the queue.

2.7 eDMA core Driver

enum *_edma_tcd_type*

eDMA tcd flag type

Values:

enumerator kEDMA_EDMA4Flag

Data access for eDMA4 transfers.

enumerator kEDMA_EDMA5Flag

Instruction access for eDMA4 transfers.

```
typedef struct _edma_core_mp edma_core_mp_t
    edma core channel struture definition
typedef struct _edma_core_channel edma_core_channel_t
    edma core channel struture definition
typedef enum _edma_tcd_type edma_tcd_type_t
    eDMA tcd flag type
typedef struct _edma5_core_tcd edma5_core_tcd_t
    edma5 core TCD struture definition
typedef struct _edma4_core_tcd edma4_core_tcd_t
    edma4 core TCD struture definition
typedef struct _edma_core_tcd edma_core_tcd_t
    edma core TCD struture definition
typedef edma_core_channel_t EDMA_ChannelType
    EDMA typedef.
typedef edma_core_tcd_t EDMA_TCDType
typedef void EDMA_Type

DMA_CORE_MP_CSR_EDBG_MASK
DMA_CORE_MP_CSR_ERCA_MASK
DMA_CORE_MP_CSR_HAE_MASK
DMA_CORE_MP_CSR_HALT_MASK
DMA_CORE_MP_CSR_GCLC_MASK
DMA_CORE_MP_CSR_GMRC_MASK
DMA_CORE_MP_CSR_EDBG(x)
DMA_CORE_MP_CSR_ERCA(x)
DMA_CORE_MP_CSR_HAE(x)
DMA_CORE_MP_CSR_HALT(x)
DMA_CORE_MP_CSR_GCLC(x)
DMA_CORE_MP_CSR_GMRC(x)
DMA_CSR_INTMAJOR_MASK
DMA_CSR_INTHALF_MASK
DMA_CSR_DREQ_MASK
DMA_CSR_ESG_MASK
DMA_CSR_BWC_MASK
DMA_CSR_BWC(x)
DMA_CSR_START_MASK
DMA_CITER_ELINKNO_CITER_MASK
```

DMA_BITER_ELINKNO_BITER_MASK
DMA_CITER_ELINKNO_CITER_SHIFT
DMA_CITER_ELINKYES_CITER_MASK
DMA_CITER_ELINKYES_CITER_SHIFT
DMA_ATTR_SMOD_MASK
DMA_ATTR_DMOD_MASK
DMA_CITER_ELINKNO_ELINK_MASK
DMA_CSR_MAJORELINK_MASK
DMA_BITER_ELINKYES_ELINK_MASK
DMA_CITER_ELINKYES_ELINK_MASK
DMA_CSR_MAJORLINKCH_MASK
DMA_BITER_ELINKYES_LINKCH_MASK
DMA_CITER_ELINKYES_LINKCH_MASK
DMA_NBYTES_MLOFFYES_MLOFF_MASK
DMA_NBYTES_MLOFFYES_DMLOE_MASK
DMA_NBYTES_MLOFFYES_SMLOE_MASK
DMA_NBYTES_MLOFFNO_NBYTES_MASK
DMA_ATTR_DMOD(x)
DMA_ATTR_SMOD(x)
DMA_BITER_ELINKYES_LINKCH(x)
DMA_CITER_ELINKYES_LINKCH(x)
DMA_NBYTES_MLOFFYES_MLOFF(x)
DMA_NBYTES_MLOFFYES_DMLOE(x)
DMA_NBYTES_MLOFFYES_SMLOE(x)
DMA_NBYTES_MLOFFNO_NBYTES(x)
DMA_NBYTES_MLOFFYES_NBYTES(x)
DMA_ATTR_DSIZE(x)
DMA_ATTR_SSIZE(x)
DMA_CSR_DREQ(x)
DMA_CSR_MAJORLINKCH(x)
DMA_CH_MATTR_WCACHE(x)
DMA_CH_MATTR_RCACHE(x)
DMA_CH_CSR_SIGNEXT_MASK

DMA_CH_CSR_SIGNEXT_SHIFT
DMA_CH_CSR_SWAP_MASK
DMA_CH_CSR_SWAP_SHIFT
DMA_CH_SBR_INSTR_MASK
DMA_CH_SBR_INSTR_SHIFT
DMA_CH_SBR_EMI_MASK
DMA_CH_SBR_EMI_SHIFT
DMA_CH_MUX_SOURCE(x)
DMA_ERR_DBE_FLAG
 DMA error flag.
DMA_ERR_SBE_FLAG
DMA_ERR_SGE_FLAG
DMA_ERR_NCE_FLAG
DMA_ERR_DOE_FLAG
DMA_ERR_DAE_FLAG
DMA_ERR_SOE_FLAG
DMA_ERR_SAE_FLAG
DMA_ERR_ERRCHAN_FLAG
DMA_ERR_ECX_FLAG
DMA_ERR_FLAG
DMA_CLEAR_DONE_STATUS(base, channel)
 get/clear DONE bit
DMA_GET_DONE_STATUS(base, channel)
DMA_ENABLE_ERROR_INT(base, channel)
 enable/disable error interrupt
DMA_DISABLE_ERROR_INT(base, channel)
DMA_CLEAR_ERROR_STATUS(base, channel)
 get/clear error status
DMA_GET_ERROR_STATUS(base, channel)
DMA_CLEAR_INT_STATUS(base, channel)
 get/clear INT status
DMA_GET_INT_STATUS(base, channel)
DMA_ENABLE_MAJOR_INT(base, channel)
 enable/disable MAJOR/HALF INT
DMA_ENABLE_HALF_INT(base, channel)
DMA_DISABLE_MAJOR_INT(base, channel)

DMA_DISABLE_HALF_INT(base, channel)

EDMA_TCD_ALIGN_SIZE

EDMA tcd align size.

EDMA_CORE_BASE(base)

EDMA base address convert macro.

EDMA_MP_BASE(base)

EDMA_CHANNEL_BASE(base, channel)

EDMA_TCD_BASE(base, channel)

EDMA_TCD_TYPE(x)

EDMA TCD type macro.

EDMA_TCD_SADDR(tcd, flag)

EDMA TCD address convert macro.

EDMA_TCD_SOFF(tcd, flag)

EDMA_TCD_ATTR(tcd, flag)

EDMA_TCD_NBYTES(tcd, flag)

EDMA_TCD_SLAST(tcd, flag)

EDMA_TCD_DADDR(tcd, flag)

EDMA_TCD_DOFF(tcd, flag)

EDMA_TCD_CITER(tcd, flag)

EDMA_TCD_DLAST_SGA(tcd, flag)

EDMA_TCD_CSR(tcd, flag)

EDMA_TCD_BITER(tcd, flag)

struct _edma_core_mp

#include <fsl_edma_core.h> edma core channel struture definition

Public Members

__IO uint32_t MP_CSR

Channel Control and Status, array offset: 0x10000, array step: 0x10000

__IO uint32_t MP_ES

Channel Error Status, array offset: 0x10004, array step: 0x10000

struct _edma_core_channel

#include <fsl_edma_core.h> edma core channel struture definition

Public Members

__IO uint32_t CH_CSR

Channel Control and Status, array offset: 0x10000, array step: 0x10000

__IO uint32_t CH_ES

Channel Error Status, array offset: 0x10004, array step: 0x10000

```

__IO uint32_t CH_INT
    Channel Interrupt Status, array offset: 0x10008, array step: 0x10000
__IO uint32_t CH_SBR
    Channel System Bus, array offset: 0x1000C, array step: 0x10000
__IO uint32_t CH_PRI
    Channel Priority, array offset: 0x10010, array step: 0x10000
struct __edma5_core_tcd
    #include <fsl_edma_core.h> edma5 core TCD struture definition

```

Public Members

```

__IO uint32_t SADDR
    SADDR register, used to save source address
__IO uint32_t SADDR_HIGH
    SADDR HIGH register, used to save source address
__IO uint16_t SOFF
    SOFF register, save offset bytes every transfer
__IO uint16_t ATTR
    ATTR register, source/destination transfer size and modulo
__IO uint32_t NBYTES
    Nbytes register, minor loop length in bytes
__IO uint32_t SLAST
    SLAST register
__IO uint32_t SLAST_SDA_HIGH
    SLAST SDA HIGH register
__IO uint32_t DADDR
    DADDR register, used for destination address
__IO uint32_t DADDR_HIGH
    DADDR HIGH register, used for destination address
__IO uint32_t DLAST_SGA
    DLASTSGA register, next tcd address used in scatter-gather mode
__IO uint32_t DLAST_SGA_HIGH
    DLASTSGA HIGH register, next tcd address used in scatter-gather mode
__IO uint16_t DOFF
    DOFF register, used for destination offset
__IO uint16_t CITER
    CITER register, current minor loop numbers, for unfinished minor loop.
__IO uint16_t CSR
    CSR register, for TCD control status
__IO uint16_t BITER
    BITER register, begin minor loop count.
uint8_t RESERVED[16]
    Aligned 64 bytes
struct __edma4_core_tcd
    #include <fsl_edma_core.h> edma4 core TCD struture definition

```

Public Members

__IO uint32_t SADDR
SADDR register, used to save source address

__IO uint16_t SOFF
SOFF register, save offset bytes every transfer

__IO uint16_t ATTR
ATTR register, source/destination transfer size and modulo

__IO uint32_t NBYTES
Nbytes register, minor loop length in bytes

__IO uint32_t SLAST
SLAST register

__IO uint32_t DADDR
DADDR register, used for destination address

__IO uint16_t DOFF
DOFF register, used for destination offset

__IO uint16_t CITER
CITER register, current minor loop numbers, for unfinished minor loop.

__IO uint32_t DLAST_SGA
DLASTSGA register, next tcd address used in scatter-gather mode

__IO uint16_t CSR
CSR register, for TCD control status

__IO uint16_t BITER
BITER register, begin minor loop count.

struct _edma_core_tcd
#include <fsl_edma_core.h> edma core TCD struture definition

union MP_REGS

Public Members

struct _edma_core_mp EDMA5_REG

struct EDMA5_REG

Public Members

__IO uint32_t MP_INT_LOW
Channel Control and Status, array offset: 0x10008, array step: 0x10000

__IO uint32_t MP_INT_HIGH
Channel Control and Status, array offset: 0x1000C, array step: 0x10000

__IO uint32_t MP_HRS_LOW
Channel Control and Status, array offset: 0x10010, array step: 0x10000

__IO uint32_t MP_HRS_HIGH
Channel Control and Status, array offset: 0x10014, array step: 0x10000

```

__IO uint32_t MP_STOPCH
    Channel Control and Status, array offset: 0x10020, array step: 0x10000
__IO uint32_t MP_SSR_LOW
    Channel Control and Status, array offset: 0x10030, array step: 0x10000
__IO uint32_t MP_SSR_HIGH
    Channel Control and Status, array offset: 0x10034, array step: 0x10000
__IO uint32_t CH_GRPRI [64]
    Channel Control and Status, array offset: 0x10100, array step: 0x10000
__IO uint32_t CH_MUX [64]
    Channel Control and Status, array offset: 0x10200, array step: 0x10000
__IO uint32_t CH_PROT [64]
    Channel Control and Status, array offset: 0x10400, array step: 0x10000
union CH_REGS

```

Public Members

```

struct _edma_core_channel EDMA5_REG
struct _edma_core_channel EDMA4_REG
struct EDMA5_REG

```

Public Members

```

__IO uint32_t CH_MATTR
    Memory Attributes Register, array offset: 0x10018, array step: 0x8000
struct EDMA4_REG

```

Public Members

```

__IO uint32_t CH_MUX
    Channel Multiplexor Configuration, array offset: 0x10014, array step: 0x10000
__IO uint16_t CH_MATTR
    Memory Attributes Register, array offset: 0x10018, array step: 0x8000
union TCD_REGS

```

Public Members

```

edma4_core_tcd_t edma4_tcd

```

2.8 eDMA soc Driver

```

enum _dma_request_source
    dma request source
    Values:

```

enumerator DmaRequestDisabled
 DSisabled

enumerator Dma3RequestMuxCAN1
 CAN1

enumerator Dma3RequestMuxLPTMR1Request
 LPTMR1 Request

enumerator Dma3RequestMuxELERequest
 ELE Request

enumerator Dma3RequestMuxTPM1OverflowRequest
 TPM1 Overflow Request

enumerator Dma3RequestMuxTPM2OverflowRequest
 TPM2 Overflow Request

enumerator Dma3RequestMuxPDMRequest
 PDM

enumerator Dma3RequestMuxADC1Request
 ADC1

enumerator Dma3RequestMuxGPIO1Request0
 GPIO1 channel 0

enumerator Dma3RequestMuxGPIO1Request1
 GPIO1 channel 1

enumerator Dma3RequestMuxI3C1ToBusRequest
 I3C1 To-bus Request

enumerator Dma3RequestMuxI3C1FromBusRequest
 I3C1 From-bus Request

enumerator Dma3RequestMuxLPI2C1Tx
 LPI2C1

enumerator Dma3RequestMuxLPI2C1Rx
 LPI2C1

enumerator Dma3RequestMuxLPI2C2Tx
 LPI2C2

enumerator Dma3RequestMuxLPI2C2Rx
 LPI2C2

enumerator Dma3RequestMuxLPSPI1Tx
 LPSPI1 Transmit

enumerator Dma3RequestMuxLPSPI1Rx
 LPSPI1 Receive

enumerator Dma3RequestMuxLPSPI2Tx
 LPSPI2 Transmit

enumerator Dma3RequestMuxLPSPI2Rx
 LPSPI2 Receive

enumerator Dma3RequestMuxLPUART1Tx
 LPUART1 Transmit

enumerator Dma3RequestMuxLPUART1Rx
LPUART1 Receive

enumerator Dma3RequestMuxLPUART2Tx
LPUART2 Transmit

enumerator Dma3RequestMuxLPUART2Rx
LPUART2 Receive

enumerator Dma3RequestMuxSai1Tx
SAI1 Transmit

enumerator Dma3RequestMuxSai1Rx
SAI1 Receive

enumerator Dma3RequestMuxTPM1Request0Request2
TPM1 request 0 and request 2

enumerator Dma3RequestMuxTPM1Request1Request3
TPM1 request 1 and request 3

enumerator Dma3RequestMuxTPM2Request0Request2
TPM2 request 0 and request 2

enumerator Dma3RequestMuxTPM2Request1Request3
TPM2 request 1 and request 3

enumerator Dma5RequestMuxCAN2
CAN2

enumerator Dma5RequestMuxGPIO2Request0
GPIO2 channel 0

enumerator Dma5RequestMuxGPIO2Request1
GPIO2 channel 1

enumerator Dma5RequestMuxGPIO3Request0
GPIO3 channel 0

enumerator Dma5RequestMuxGPIO3Request1
GPIO3 channel 1

enumerator Dma5RequestMuxI3C2ToBusRequest
I3C2 To-bus Request

enumerator Dma5RequestMuxI3C2FromBusRequest
I3C2 From-bus Request

enumerator Dma5RequestMuxLPI2C3Tx
LPI2C3

enumerator Dma5RequestMuxLPI2C3Rx
LPI2C3

enumerator Dma5RequestMuxLPI2C4Tx
LPI2C4

enumerator Dma5RequestMuxLPI2C4Rx
LPI2C2

enumerator Dma5RequestMuxLPSPI3Tx
LPSPI3 Transmit

enumerator Dma5RequestMuxLPSPI3Rx
LPSPI3 Receive

enumerator Dma5RequestMuxLPSPI4Tx
LPSPI4 Transmit

enumerator Dma5RequestMuxLPSPI4Rx
LPSPI4 Receive

enumerator Dma5RequestMuxLPTMR2Request
LPTMR2 Request

enumerator Dma5RequestMuxLPUART3Tx
LPUART3 Transmit

enumerator Dma5RequestMuxLPUART3Rx
LPUART3 Receive

enumerator Dma5RequestMuxLPUART4Tx
LPUART4 Transmit

enumerator Dma5RequestMuxLPUART4Rx
LPUART4 Receive

enumerator Dma5RequestMuxLPUART5Tx
LPUART5 Transmit

enumerator Dma5RequestMuxLPUART5Rx
LPUART5 Receive

enumerator Dma5RequestMuxLPUART6Tx
LPUART6 Transmit

enumerator Dma5RequestMuxLPUART6Rx
LPUART6 Receive

enumerator Dma5RequestMuxTPM3Request0Request2
TPM3 request 0 and request 2

enumerator Dma5RequestMuxTPM3Request1Request3
TPM3 request 1 and request 3

enumerator Dma5RequestMuxTPM3OverflowRequest
TPM3 Overflow request

enumerator Dma5RequestMuxTPM4Request0Request2
TPM4 request 0 and request 2

enumerator Dma5RequestMuxTPM4Request1Request3
TPM4 request 1 and request 3

enumerator Dma5RequestMuxTPM4OverflowRequest
TPM4 Overflow request

enumerator Dma5RequestMuxTPM5Request0Request2
TPM5 request 0 and request 2

enumerator Dma5RequestMuxTPM5Request1Request3
TPM5 request 1 and request 3

enumerator Dma5RequestMuxTPM5OverflowRequest
TPM5 Overflow request

enumerator Dma5RequestMuxTPM6Request0Request2
TPM6 request 0 and request 2

enumerator Dma5RequestMuxTPM6Request1Request3
TPM6 request 1 and request 3

enumerator Dma5RequestMuxTPM6OverflowRequest
TPM6 Overflow request

enumerator Dma5RequestMuxFlexIO1Request0
FlexIO1 Request0

enumerator Dma5RequestMuxFlexIO1Request1
FlexIO1 Request1

enumerator Dma5RequestMuxFlexIO1Request2
FlexIO1 Request2

enumerator Dma5RequestMuxFlexIO1Request3
FlexIO1 Request3

enumerator Dma5RequestMuxFlexIO1Request4
FlexIO1 Request4

enumerator Dma5RequestMuxFlexIO1Request5
FlexIO1 Request5

enumerator Dma5RequestMuxFlexIO1Request6
FlexIO1 Request6

enumerator Dma5RequestMuxFlexIO1Request7
FlexIO1 Request7

enumerator Dma5RequestMuxFlexIO2Request0
FlexIO2 Request0

enumerator Dma5RequestMuxFlexIO2Request1
FlexIO2 Request1

enumerator Dma5RequestMuxFlexIO2Request2
FlexIO2 Request2

enumerator Dma5RequestMuxFlexIO2Request3
FlexIO2 Request3

enumerator Dma5RequestMuxFlexIO2Request4
FlexIO2 Request4

enumerator Dma5RequestMuxFlexIO2Request5
FlexIO2 Request5

enumerator Dma5RequestMuxFlexIO2Request6
FlexIO2 Request6

enumerator Dma5RequestMuxFlexIO2Request7
FlexIO2 Request7

enumerator Dma5RequestMuxFlexSPI1Tx
FlexSPI1 Transmit

enumerator Dma5RequestMuxFlexSPI1Rx
FlexSPI1 Receive

enumerator Dma5RequestMuxGPIO5Request0
GPIO5 Request0

enumerator Dma5RequestMuxGPIO5Request1
GPIO5 Request1

enumerator Dma5RequestMuxCAN3
CAN3

enumerator Dma5RequestMuxSai2Tx
SAI2 Transmit

enumerator Dma5RequestMuxSai2Rx
SAI2 Receive

enumerator Dma5RequestMuxSai3Tx
SAI3 Transmit

enumerator Dma5RequestMuxSai3Rx
SAI3 Receive

enumerator Dma5RequestMuxGPIO4Request0
GPIO4 Request0

enumerator Dma5RequestMuxGPIO4Request1
GPIO4 Request1

enumerator Dma5RequestMuxeARCRequest0
eARC enhanced Audio Return Channel

enumerator Dma5RequestMuxeARCRequest1
eARC enhanced Audio Return Channel

enumerator Dma5RequestMuxSai4Tx
SAI4 Transmit

enumerator Dma5RequestMuxSai4Rx
SAI4 Receive

enumerator Dma5RequestMuxSai5Tx
SAI5 Transmit

enumerator Dma5RequestMuxSai5Rx
SAI5 Receive

enumerator Dma5RequestMuxLPI2C5Tx
LPI2C5

enumerator Dma5RequestMuxLPI2C5Rx
LPI2C5

enumerator Dma5RequestMuxLPI2C6Tx
LPI2C6

enumerator Dma5RequestMuxLPI2C6Rx
LPI2C6

enumerator Dma5RequestMuxLPI2C7Tx
LPI2C7

enumerator Dma5RequestMuxLPI2C7Rx
LPI2C7

```
enumerator Dma5RequestMuxLPI2C8Tx
    LPI2C8
enumerator Dma5RequestMuxLPI2C8Rx
    LPI2C8
enumerator Dma5RequestMuxLPSPi5Tx
    LPSPi5 Transmit
enumerator Dma5RequestMuxLPSPi5Rx
    LPSPi5 Receive
enumerator Dma5RequestMuxLPSPi6Tx
    LPSPi6 Transmit
enumerator Dma5RequestMuxLPSPi6Rx
    LPSPi6 Receive
enumerator Dma5RequestMuxLPSPi7Tx
    LPSPi7 Transmit
enumerator Dma5RequestMuxLPSPi7Rx
    LPSPi7 Receive
enumerator Dma5RequestMuxLPSPi8Tx
    LPSPi8 Transmit
enumerator Dma5RequestMuxLPSPi8Rx
    LPSPi8 Receive
enumerator Dma5RequestMuxLPUART7Tx
    LPUART7 Transmit
enumerator Dma5RequestMuxLPUART7Rx
    LPUART7 Receive
enumerator Dma5RequestMuxLPUART8Tx
    LPUART8 Transmit
enumerator Dma5RequestMuxLPUART8Rx
    LPUART8 Receive
enumerator Dma5RequestMuxCAN4
    CAN4
enumerator Dma5RequestMuxCAN5
    CAN5
typedef enum _dma_request_source dma__request__source_t
    dma request source
    Verify dma base and request source
FSL_EDMA_SOC_DRIVER_VERSION
    Driver version 2.0.0.
FSL_EDMA_SOC_IP_DMA3
    DMA IP version.
FSL_EDMA_SOC_IP_DMA5
EDMA_BASE_PTRS
    DMA base table.
```

EDMA_CHN_IRQS

EDMA_CHANNEL_HAS_REQUEST_SOURCE(base, source)

EDMA_CHANNEL_OFFSET

EDMA base address convert macro.

EDMA_CHANNEL_ARRAY_STEP(base)

2.9 EIM: error injection module

FSL_ERM_DRIVER_VERSION

Driver version.

void EIM_Init(EIM_Type *base)

EIM module initialization function.

Parameters

- base – EIM base address.

void EIM_Deinit(EIM_Type *base)

De-initializes the EIM.

2.10 ERM: error recording module

void ERM_Init(ERM_Type *base)

ERM module initialization function.

Parameters

- base – ERM base address.

void ERM_Deinit(ERM_Type *base)

De-initializes the ERM.

static inline void ERM_EnableInterrupts(ERM_Type *base, uint32_t channel, uint32_t mask)

ERM enable interrupts.

Parameters

- base – ERM peripheral base address.
- channel – memory channel.
- mask – single correction interrupt or non-correction interrupt enable to disable for one specific memory region. Refer to “_erm_interrupt_enable” enumeration.

static inline void ERM_DisableInterrupts(ERM_Type *base, uint32_t channel, uint32_t mask)

ERM module disable interrupts.

Parameters

- base – ERM base address.
- channel – memory channel.
- mask – single correction interrupt or non-correction interrupt enable to disable for one specific memory region. Refer to “_erm_interrupt_enable” enumeration.

```
static inline uint32_t ERM_GetInterruptStatus(ERM_Type *base, uint32_t channel)
```

Gets ERM interrupt flags.

Parameters

- base – ERM peripheral base address.

Returns

ERM event flags.

```
static inline void ERM_ClearInterruptStatus(ERM_Type *base, uint32_t channel, uint32_t mask)
```

ERM module clear interrupt status flag.

Parameters

- base – ERM base address.
- mask – event flag to clear. Refer to “_erm_interrupt_flag” enumeration.

```
uint32_t ERM_GetMemoryErrorAddr(ERM_Type *base, uint32_t channel)
```

ERM get memory error absolute address, which capturing the address of the last ECC event in Memory n.

Parameters

- base – ERM base address.
- channel – memory channel.

Return values

memory – error absolute address.

```
uint32_t ERM_GetSyndrome(ERM_Type *base, uint32_t channel)
```

ERM get syndrome, which identifies the pertinent bit position on a correctable, single-bit data inversion or a non-correctable, single-bit address inversion. The syndrome value does not provide any additional diagnostic information on non-correctable, multi-bit inversions.

Parameters

- base – ERM base address.
- channel – memory channel.

Return values

syndrome – value.

```
uint32_t ERM_GetErrorCount(ERM_Type *base, uint32_t channel)
```

ERM get error count, which records the count value of the number of correctable ECC error events for Memory n. Non-correctable errors are considered a serious fault, so the ERM does not provide any mechanism to count non-correctable errors. Only correctable errors are counted.

Parameters

- base – ERM base address.
- channel – memory channel.

Return values

error – count.

```
void ERM_ResetErrorCount(ERM_Type *base, uint32_t channel)
```

ERM reset error count.

Parameters

- base – ERM base address.
- channel – memory channel.

FSL_ERM_DRIVER_VERSION

Driver version.

ERM interrupt configuration structure, default settings all disabled, `_erm_interrupt_enable`.

This structure contains the settings for all of the ERM interrupt configurations.

Values:

enumerator `kERM_SingleCorrectionIntEnable`

Single Correction Interrupt Notification enable.

enumerator `kERM_NonCorrectableIntEnable`

Non-Correction Interrupt Notification enable.

enumerator `kERM_AllInterruptsEnable`

All Interrupts enable

ERM interrupt status, `_erm_interrupt_flag`.

This provides constants for the ERM event status for use in the ERM functions.

Values:

enumerator `kERM_SingleBitCorrectionIntFlag`

Single-Bit Correction Event.

enumerator `kERM_NonCorrectableErrorIntFlag`

Non-Correctable Error Event.

enumerator `kERM_AllIntsFlag`

All Events.

2.11 FGPIO Driver

2.12 FlexCAN: Flex Controller Area Network Driver

2.13 FlexCAN Driver

`bool FLEXCAN_IsInstanceHasFDMode(CAN_Type *base)`

Determine whether the FlexCAN instance support CAN FD mode at run time.

Note: Use this API only if different soc parts share the SOC part name macro define. Otherwise, a different SOC part name can be used to determine at compile time whether the FlexCAN instance supports CAN FD mode or not. If need use this API to determine if CAN FD mode is supported, the `FLEXCAN_Init` function needs to be executed first, and then call this API and use the return to value determines whether to supports CAN FD mode, if return true, continue calling `FLEXCAN_FDInit` to enable CAN FD mode.

Parameters

- `base` – FlexCAN peripheral base address.

Returns

return TRUE if instance support CAN FD mode, FALSE if instance only support classic CAN (2.0) mode.

uint32_t FLEXCAN_GetFDMailboxOffset(CAN_Type *base, uint8_t mbIdx)

Get Mailbox offset number by dword.

This function gets the offset number of the specified mailbox. Mailbox is not consecutive between memory regions when payload is not 8 bytes so need to calculate the specified mailbox address. For example, in the first memory region, MB[0].CS address is 0x4002_4080. For 32 bytes payload frame, the second mailbox is $((1/12)*512 + 1\%12*40)/4 = 10$, meaning 10 dword after the 0x4002_4080, which is actually the address of mailbox MB[1].CS.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – Mailbox index.

Returns

Mailbox address offset in word.

status_t FLEXCAN_EnterFreezeMode(CAN_Type *base)

Enter FlexCAN Freeze Mode.

This function makes the FlexCAN work under Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

Returns

kStatus_Success Enter Freeze Mode successful kStatus_Timeout Timeout when wait for Freeze Mode Acknowledge

status_t FLEXCAN_ExitFreezeMode(CAN_Type *base)

Exit FlexCAN Freeze Mode.

This function makes the FlexCAN leave Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

Returns

kStatus_Success Enter Freeze Mode successful kStatus_Timeout Timeout when wait for Freeze Mode Acknowledge

uint32_t FLEXCAN_GetInstance(CAN_Type *base)

Get the FlexCAN instance from peripheral base address.

Parameters

- base – FlexCAN peripheral base address.

Returns

FlexCAN instance.

bool FLEXCAN_CalculateImprovedTimingValues(CAN_Type *base, uint32_t bitRate, uint32_t sourceClock_Hz, flexcan_timing_config_t *pTimingConfig)

Calculates the improved timing values by specific bit Rates for classical CAN.

This function use to calculates the Classical CAN timing values according to the given bit rate. The Calculated timing values will be set in CTRL1/CBT/ENCBT register. The calculation is based on the recommendation of the CiA 301 v4.2.0 and previous version document.

Parameters

- base – FlexCAN peripheral base address.
- bitRate – The classical CAN speed in bps defined by user, should be less than or equal to 1Mbps.
- sourceClock_Hz – The Source clock frequency in Hz.
- pTimingConfig – Pointer to the FlexCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration.

void FLEXCAN_Init(CAN_Type *base, const *flexcan_config_t* *pConfig, uint32_t sourceClock_Hz)
Initializes a FlexCAN instance.

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the *flexcan_config_t* parameters and how to call the FLEXCAN_Init function by passing in these parameters.

```
flexcan_config_t flexcanConfig;  
flexcanConfig.clkSrc      = kFLEXCAN_ClkSrc0;  
flexcanConfig.bitRate     = 1000000U;  
flexcanConfig.maxMbNum    = 16;  
flexcanConfig.enableLoopBack = false;  
flexcanConfig.enableSelfWakeup = false;  
flexcanConfig.enableIndividMask = false;  
flexcanConfig.enableDoze    = false;  
flexcanConfig.disableSelfReception = false;  
flexcanConfig.enableListenOnlyMode = false;  
flexcanConfig.timingConfig = timingConfig;  
FLEXCAN_Init(CAN0, &flexcanConfig, 4000000UL);
```

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the user-defined configuration structure.
- sourceClock_Hz – FlexCAN Protocol Engine clock source frequency in Hz.

bool FLEXCAN_FDCalculateImprovedTimingValues(CAN_Type *base, uint32_t bitRate, uint32_t bitRateFD, uint32_t sourceClock_Hz, *flexcan_timing_config_t* *pTimingConfig)

Calculates the improved timing values by specific bit rates for CANFD.

This function use to calculates the CANFD timing values according to the given nominal phase bit rate and data phase bit rate. The Calculated timing values will be set in CBT/ENCBT and FDCBT/EDCBT registers. The calculation is based on the recommendation of the CiA 1301 v1.0.0 document.

Parameters

- base – FlexCAN peripheral base address.
- bitRate – The CANFD bus control speed in bps defined by user.
- bitRateFD – The CAN FD data phase speed in bps defined by user. Equal to bitRate means disable bit rate switching.
- sourceClock_Hz – The Source clock frequency in Hz.
- pTimingConfig – Pointer to the FlexCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration


```
void FLEXCAN_FDInit(CAN_Type *base, const flexcan_config_t *pConfig, uint32_t
                    sourceClock_Hz, flexcan_mb_size_t dataSize, bool brs)
```

Initializes a FlexCAN instance.

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the flexcan_config_t parameters and how to call the FLEXCAN_FDInit function by passing in these parameters.

```
flexcan_config_t flexcanConfig;
flexcanConfig.clkSrc      = kFLEXCAN_ClkSrc0;
flexcanConfig.bitRate     = 1000000U;
flexcanConfig.bitRateFD   = 2000000U;
flexcanConfig.maxMbNum    = 16;
flexcanConfig.enableLoopBack = false;
flexcanConfig.enableSelfWakeup = false;
flexcanConfig.enableIndividMask = false;
flexcanConfig.disableSelfReception = false;
flexcanConfig.enableListenOnlyMode = false;
flexcanConfig.enableDoze = false;
flexcanConfig.timingConfig = timingConfig;
FLEXCAN_FDInit(CAN0, &flexcanConfig, 8000000UL, kFLEXCAN_16BperMB, true);
```

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the user-defined configuration structure.
- sourceClock_Hz – FlexCAN Protocol Engine clock source frequency in Hz.
- dataSize – FlexCAN Message Buffer payload size. The actual transmitted or received CAN FD frame data size needs to be less than or equal to this value.
- brs – True if bit rate switch is enabled in FD mode.

```
void FLEXCAN_Deinit(CAN_Type *base)
```

De-initializes a FlexCAN instance.

This function disables the FlexCAN module clock and sets all register values to the reset value.

Parameters

- base – FlexCAN peripheral base address.

```
void FLEXCAN_GetDefaultConfig(flexcan_config_t *pConfig)
```

Gets the default configuration structure.

This function initializes the FlexCAN configuration structure to default values. The default values are as follows. flexcanConfig->clkSrc = kFLEXCAN_ClkSrc0; flexcanConfig->bitRate = 1000000U; flexcanConfig->bitRateFD = 2000000U; flexcanConfig->maxMbNum = 16; flexcanConfig->enableLoopBack = false; flexcanConfig->enableSelfWakeup = false; flexcanConfig->enableIndividMask = false; flexcanConfig->disableSelfReception = false; flexcanConfig->enableListenOnlyMode = false; flexcanConfig->enableDoze = false; flexcanConfig->enablePretendededeNetworking = false; flexcanConfig->enableMemoryErrorControl = true; flexcanConfig->enableNonCorrectableErrorEnterFreeze = true; flexcanConfig->enableTransceiverDelayMeasure = true; flexcanConfig->enableRemoteRequestFrameStored = true; flexcanConfig->payloadEndianness = kFLEXCAN_bigEndian; flexcanConfig.timingConfig = timingConfig;

Parameters

- pConfig – Pointer to the FlexCAN configuration structure.

```
void FLEXCAN_SetTimingConfig(CAN_Type *base, const flexcan_timing_config_t *pConfig)
```

Sets the FlexCAN classical CAN protocol timing characteristic.

This function gives user settings to classical CAN or CAN FD nominal phase timing characteristic. The function is for an experienced user. For less experienced users, call the FLEXCAN_SetBitRate() instead.

Note: Calling FLEXCAN_SetTimingConfig() overrides the bit rate set in FLEXCAN_Init() or FLEXCAN_SetBitRate().

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the timing configuration structure.

```
status_t FLEXCAN_SetBitRate(CAN_Type *base, uint32_t sourceClock_Hz, uint32_t bitRate_Bps)
```

Set bit rate of FlexCAN classical CAN frame or CAN FD frame nominal phase.

This function set the bit rate of classical CAN frame or CAN FD frame nominal phase base on FLEXCAN_CalculateImprovedTimingValues() API calculated timing values.

Note: Calling FLEXCAN_SetBitRate() overrides the bit rate set in FLEXCAN_Init().

Parameters

- base – FlexCAN peripheral base address.
- sourceClock_Hz – Source Clock in Hz.
- bitRate_Bps – Bit rate in Bps.

Returns

kStatus_Success - Set CAN baud rate (only Nominal phase) successfully.

```
void FLEXCAN_SetFDTimingConfig(CAN_Type *base, const flexcan_timing_config_t *pConfig)
```

Sets the FlexCAN CANFD data phase timing characteristic.

This function gives user settings to CANFD data phase timing characteristic. The function is for an experienced user. For less experienced users, call the FLEXCAN_SetFDBitRate() to set both Nominal/Data bit Rate instead.

Note: Calling FLEXCAN_SetFDTimingConfig() overrides the data phase bit rate set in FLEXCAN_FDInit()/FLEXCAN_SetFDBitRate().

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the timing configuration structure.

```
status_t FLEXCAN_SetFDBitRate(CAN_Type *base, uint32_t sourceClock_Hz, uint32_t  
bitRateN_Bps, uint32_t bitRateD_Bps)
```

Set bit rate of FlexCAN FD frame.

This function set the baud rate of FLEXCAN FD base on FLEXCAN_FDCalculateImprovedTimingValues() API calculated timing values.

Parameters

- base – FlexCAN peripheral base address.

- sourceClock_Hz – Source Clock in Hz.
- bitRateN_Bps – Nominal bit Rate in Bps.
- bitRateD_Bps – Data bit Rate in Bps.

Returns

kStatus_Success - Set CAN FD bit rate (include Nominal and Data phase) successfully.

void FLEXCAN_SetRxMbGlobalMask(CAN_Type *base, uint32_t mask)

Sets the FlexCAN receive message buffer global mask.

This function sets the global mask for the FlexCAN message buffer in a matching process. The configuration is only effective when the Rx individual mask is disabled in the FLEXCAN_Init().

Parameters

- base – FlexCAN peripheral base address.
- mask – Rx Message Buffer Global Mask value.

void FLEXCAN_SetRxFifoGlobalMask(CAN_Type *base, uint32_t mask)

Sets the FlexCAN receive FIFO global mask.

This function sets the global mask for FlexCAN FIFO in a matching process.

Parameters

- base – FlexCAN peripheral base address.
- mask – Rx Fifo Global Mask value.

void FLEXCAN_SetRxIndividualMask(CAN_Type *base, uint8_t maskIdx, uint32_t mask)

Sets the FlexCAN receive individual mask.

This function sets the individual mask for the FlexCAN matching process. The configuration is only effective when the Rx individual mask is enabled in the FLEXCAN_Init(). If the Rx FIFO is disabled, the individual mask is applied to the corresponding Message Buffer. If the Rx FIFO is enabled, the individual mask for Rx FIFO occupied Message Buffer is applied to the Rx Filter with the same index. Note that only the first 32 individual masks can be used as the Rx FIFO filter mask.

Parameters

- base – FlexCAN peripheral base address.
- maskIdx – The Index of individual Mask.
- mask – Rx Individual Mask value.

void FLEXCAN_SetTxMbConfig(CAN_Type *base, uint8_t mbIdx, bool enable)

Configures a FlexCAN transmit message buffer.

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- enable – Enable/disable Tx Message Buffer.
 - true: Enable Tx Message Buffer.
 - false: Disable Tx Message Buffer.

```
void FLEXCAN_SetRxMbConfig(CAN_Type *base, uint8_t mbIdx, const flexcan_rx_mb_config_t *pRxMbConfig, bool enable)
```

Configures a FlexCAN Receive Message Buffer.

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer. User should invoke this API when CTRL2[RRS]=1. When CTRL2[RRS]=1, frame's ID is compared to the IDs of the receive mailboxes with the CODE field configured as kFLEXCAN_RxMbEmpty, kFLEXCAN_RxMbFull or kFLEXCAN_RxMbOverrun. Message buffer will store the remote frame in the same fashion of a data frame. No automatic remote response frame will be generated. User need to setup another message buffer to respond remote request.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pRxMbConfig – Pointer to the FlexCAN Message Buffer configuration structure.
- enable – Enable/disable Rx Message Buffer.
 - true: Enable Rx Message Buffer.
 - false: Disable Rx Message Buffer.

```
static inline void FLEXCAN_SetMbID(CAN_Type *base, uint8_t mbIdx, uint32_t id)
```

Configures a FlexCAN Message Buffer identifier.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- id – CAN Message Buffer Identifier, should use FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

```
void FLEXCAN_SetFDTxMbConfig(CAN_Type *base, uint8_t mbIdx, bool enable)
```

Configures a FlexCAN transmit message buffer.

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- enable – Enable/disable Tx Message Buffer.
 - true: Enable Tx Message Buffer.
 - false: Disable Tx Message Buffer.

```
void FLEXCAN_SetFDRxMbConfig(CAN_Type *base, uint8_t mbIdx, const flexcan_rx_mb_config_t *pRxMbConfig, bool enable)
```

Configures a FlexCAN Receive Message Buffer.

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.

- pRxMbConfig – Pointer to the FlexCAN Message Buffer configuration structure.
- enable – Enable/disable Rx Message Buffer.
 - true: Enable Rx Message Buffer.
 - false: Disable Rx Message Buffer.

static inline void FLEXCAN_SetFDMbID(CAN_Type *base, uint8_t mbIdx, uint32_t id)
Configures a FlexCAN Message Buffer identifier.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- id – CAN Message Buffer Identifier, should use FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

void FLEXCAN_SetRemoteResponseMbConfig(CAN_Type *base, uint8_t mbIdx, const
flexcan_frame_t *pFrame)

Configures a FlexCAN Remote Response Message Buffer.

User should invoke this API when CTRL2[RRS]=0. When CTRL2[RRS]=0, frame's ID is compared to the IDs of the receive mailboxes with the CODE field configured as kFLEXCAN_RxMbRanswer. If there is a matching ID, then this mailbox content will be transmitted as response. The received remote request frame is not stored in receive buffer. It is only used to trigger a transmission of a frame in response.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pFrame – Pointer to CAN message frame structure for response.

void FLEXCAN_SetRxFifoConfig(CAN_Type *base, const flexcan_rx_fifo_config_t *pRxFifoConfig,
bool enable)

Configures the FlexCAN Legacy Rx FIFO.

This function configures the FlexCAN Rx FIFO with given configuration.

Note: Legacy Rx FIFO only can receive classic CAN message.

Parameters

- base – FlexCAN peripheral base address.
- pRxFifoConfig – Pointer to the FlexCAN Legacy Rx FIFO configuration structure. Can be NULL when enable parameter is false.
- enable – Enable/disable Legacy Rx FIFO.
 - true: Enable Legacy Rx FIFO.
 - false: Disable Legacy Rx FIFO.

void FLEXCAN_SetEnhancedRxFifoConfig(CAN_Type *base, const
flexcan_enhanced_rx_fifo_config_t *pConfig, bool
enable)

Configures the FlexCAN Enhanced Rx FIFO.

This function configures the Enhanced Rx FIFO with given configuration.

Note: Enhanced Rx FIFO support receive classic CAN or CAN FD messages, Legacy Rx FIFO and Enhanced Rx FIFO cannot be enabled at the same time.

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the FlexCAN Enhanced Rx FIFO configuration structure. Can be NULL when enable parameter is false.
- enable – Enable/disable Enhanced Rx FIFO.
 - true: Enable Enhanced Rx FIFO.
 - false: Disable Enhanced Rx FIFO.

void FLEXCAN_SetPNConfig(CAN_Type *base, const *flexcan_pn_config_t* *pConfig)

Configures the FlexCAN Pretended Networking mode.

This function configures the FlexCAN Pretended Networking mode with given configuration.

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the FlexCAN Rx FIFO configuration structure.

static inline uint64_t FLEXCAN_GetStatusFlags(CAN_Type *base)

Gets the FlexCAN module interrupt flags.

This function gets all FlexCAN status flags. The flags are returned as the logical OR value of the enumerators *_flexcan_flags*. To check the specific status, compare the return value with enumerators in *_flexcan_flags*.

Parameters

- base – FlexCAN peripheral base address.

Returns

FlexCAN status flags which are ORed by the enumerators in the *_flexcan_flags*.

static inline void FLEXCAN_ClearStatusFlags(CAN_Type *base, uint64_t mask)

Clears status flags with the provided mask.

This function clears the FlexCAN status flags with a provided mask. An automatically cleared flag can't be cleared by this function.

Parameters

- base – FlexCAN peripheral base address.
- mask – The status flags to be cleared, it is logical OR value of *_flexcan_flags*.

static inline void FLEXCAN_GetBusErrCount(CAN_Type *base, uint8_t *txErrBuf, uint8_t *rxErrBuf)

Gets the FlexCAN Bus Error Counter value.

This function gets the FlexCAN Bus Error Counter value for both Tx and Rx direction. These values may be needed in the upper layer error handling.

Parameters

- base – FlexCAN peripheral base address.
- txErrBuf – Buffer to store Tx Error Counter value.
- rxErrBuf – Buffer to store Rx Error Counter value.

```
static inline uint64_t FLEXCAN_GetMbStatusFlags(CAN_Type *base, uint64_t mask)
```

Gets the FlexCAN low 64 Message Buffer interrupt flags.

This function gets the interrupt flags of a given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

Returns

The status of given Message Buffers.

```
static inline uint64_t FLEXCAN_GetHigh64MbStatusFlags(CAN_Type *base, uint64_t mask)
```

Gets the FlexCAN High 64 Message Buffer interrupt flags.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

Returns

The status of given Message Buffers.

```
static inline void FLEXCAN_ClearMbStatusFlags(CAN_Type *base, uint64_t mask)
```

Clears the FlexCAN low 64 Message Buffer interrupt flags.

This function clears the interrupt flags of a given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

```
static inline void FLEXCAN_ClearHigh64MbStatusFlags(CAN_Type *base, uint64_t mask)
```

Clears the FlexCAN High 64 Message Buffer interrupt flags.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

```
void FLEXCAN_GetMemoryErrorReportStatus(CAN_Type *base,  
                                         flexcan_memory_error_report_status_t  
                                         *errorStatus)
```

Gets the FlexCAN Memory Error Report registers status.

This function gets the FlexCAN Memory Error Report registers status.

Parameters

- base – FlexCAN peripheral base address.
- errorStatus – Pointer to FlexCAN Memory Error Report registers status structure.

```
static inline uint8_t FLEXCAN_GetPNMatchCount(CAN_Type *base)
```

Gets the FlexCAN Number of Matches when in Pretended Networking.

This function gets the number of times a given message has matched the predefined filtering criteria for ID and/or PL before a wakeup event.

Parameters

- base – FlexCAN peripheral base address.

Returns

The number of received wake up messages.

```
static inline uint32_t FLEXCAN_GetEnhancedFifoDataCount(CAN_Type *base)
```

Gets the number of FlexCAN Enhanced Rx FIFO available frames.

This function gets the number of CAN messages stored in the Enhanced Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.

Returns

The number of available CAN messages stored in the Enhanced Rx FIFO.

```
static inline void FLEXCAN_EnableInterrupts(CAN_Type *base, uint64_t mask)
```

Enables FlexCAN interrupts according to the provided mask.

This function enables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see `_flexcan_interrupt_enable`.

Parameters

- base – FlexCAN peripheral base address.
- mask – The interrupts to enable. Logical OR of `_flexcan_interrupt_enable`.

```
static inline void FLEXCAN_DisableInterrupts(CAN_Type *base, uint64_t mask)
```

Disables FlexCAN interrupts according to the provided mask.

This function disables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see `_flexcan_interrupt_enable`.

Parameters

- base – FlexCAN peripheral base address.
- mask – The interrupts to disable. Logical OR of `_flexcan_interrupt_enable`.

```
static inline void FLEXCAN_EnableMbInterrupts(CAN_Type *base, uint64_t mask)
```

Enables FlexCAN low 64 Message Buffer interrupts.

This function enables the interrupts of given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

```
static inline void FLEXCAN_EnableHigh64MbInterrupts(CAN_Type *base, uint64_t mask)
```

Enables FlexCAN high 64 Message Buffer interrupts.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

```
static inline void FLEXCAN_DisableMbInterrupts(CAN_Type *base, uint64_t mask)
```

Disables FlexCAN low 64 Message Buffer interrupts.

This function disables the interrupts of given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.

- mask – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_DisableHigh64MbInterrupts(CAN_Type *base, uint64_t mask)

Disables FlexCAN high 64 Message Buffer interrupts.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

void FLEXCAN_EnableRxFifoDMA(CAN_Type *base, bool enable)

Enables or disables the FlexCAN Rx FIFO DMA request.

This function enables or disables the DMA feature of FlexCAN build-in Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.
- enable – true to enable, false to disable.

static inline uintptr_t FLEXCAN_GetRxFifoHeadAddr(CAN_Type *base)

Gets the Rx FIFO Head address.

This function returns the FlexCAN Rx FIFO Head address, which is mainly used for the DMA/eDMA use case.

Parameters

- base – FlexCAN peripheral base address.

Returns

FlexCAN Rx FIFO Head address.

static inline *status_t* FLEXCAN_Enable(CAN_Type *base, bool enable)

Enables or disables the FlexCAN module operation.

This function enables or disables the FlexCAN module.

Parameters

- base – FlexCAN base pointer.
- enable – true to enable, false to disable.

Returns

kStatus_Success Enable FlexCAN module successful
kStatus_Timeout Timeout when wait for Low-Power Mode Acknowledge

status_t FLEXCAN_WriteTxMb(CAN_Type *base, uint8_t mbIdx, const *flexcan_frame_t* *pTxFrame)

Writes a FlexCAN Message to the Transmit Message Buffer.

This function writes a CAN Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN Message transmit. After that the function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN Message Buffer index.
- pTxFrame – Pointer to CAN message frame to be sent.

Return values

- kStatus_Success -- Write Tx Message Buffer Successfully.

- `kStatus_Fail` – - Tx Message Buffer is currently in use.

`status_t FLEXCAN_ReadRxMb(CAN_Type *base, uint8_t mbIdx, flexcan_frame_t *pRxFrame)`

Reads a FlexCAN Message from Receive Message Buffer.

This function reads a CAN message from a specified Receive Message Buffer. The function fills a receive CAN message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

- `base` – FlexCAN peripheral base address.
- `mbIdx` – The FlexCAN Message Buffer index.
- `pRxFrame` – Pointer to CAN message frame structure for reception.

Return values

- `kStatus_Success` – - Rx Message Buffer is full and has been read successfully.
- `kStatus_FLEXCAN_RxOverflow` – - Rx Message Buffer is already overflowed and has been read successfully.
- `kStatus_Fail` – - Rx Message Buffer is empty.

`status_t FLEXCAN_WriteFDTxMb(CAN_Type *base, uint8_t mbIdx, const flexcan_fd_frame_t *pTxFrame)`

Writes a FlexCAN FD Message to the Transmit Message Buffer.

This function writes a CAN FD Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN FD Message transmit. After that the function returns immediately.

Parameters

- `base` – FlexCAN peripheral base address.
- `mbIdx` – The FlexCAN FD Message Buffer index.
- `pTxFrame` – Pointer to CAN FD message frame to be sent.

Return values

- `kStatus_Success` – - Write Tx Message Buffer Successfully.
- `kStatus_Fail` – - Tx Message Buffer is currently in use.

`status_t FLEXCAN_ReadFDRxMb(CAN_Type *base, uint8_t mbIdx, flexcan_fd_frame_t *pRxFrame)`

Reads a FlexCAN FD Message from Receive Message Buffer.

This function reads a CAN FD message from a specified Receive Message Buffer. The function fills a receive CAN FD message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

- `base` – FlexCAN peripheral base address.
- `mbIdx` – The FlexCAN FD Message Buffer index.
- `pRxFrame` – Pointer to CAN FD message frame structure for reception.

Return values

- `kStatus_Success` – - Rx Message Buffer is full and has been read successfully.
- `kStatus_FLEXCAN_RxOverflow` – - Rx Message Buffer is already overflowed and has been read successfully.
- `kStatus_Fail` – - Rx Message Buffer is empty.

status_t FLEXCAN_ReadRxFifo(CAN_Type *base, *flexcan_frame_t* *pRxFrame)

Reads a FlexCAN Message from Legacy Rx FIFO.

This function reads a CAN message from the FlexCAN Legacy Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – Read Message from Rx FIFO successfully.
- kStatus_Fail – Rx FIFO is not enabled.

status_t FLEXCAN_ReadEnhancedRxFifo(CAN_Type *base, *flexcan_fd_frame_t* *pRxFrame)

Reads a FlexCAN Message from Enhanced Rx FIFO.

This function reads a CAN or CAN FD message from the FlexCAN Enhanced Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – Read Message from Rx FIFO successfully.
- kStatus_Fail – Rx FIFO is not enabled.

status_t FLEXCAN_ReadPNWakeUpMB(CAN_Type *base, uint8_t mbIdx, *flexcan_frame_t* *pRxFrame)

Reads a FlexCAN Message from Wake Up MB.

This function reads a CAN message from the FlexCAN Wake up Message Buffers. There are four Wake up Message Buffers (WMBs) used to store incoming messages in Pretended Networking mode. The WMB index indicates the arrival order. The last message is stored in WMB3.

Parameters

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN message frame structure for reception.
- mbIdx – The FlexCAN Wake up Message Buffer index. Range in 0x0 ~ 0x3.

Return values

- kStatus_Success – Read Message from Wake up Message Buffer successfully.
- kStatus_Fail – Wake up Message Buffer has no valid content.

status_t FLEXCAN_TransferFDSendBlocking(CAN_Type *base, uint8_t mbIdx, *flexcan_fd_frame_t* *pTxFrame)

Performs a polling send transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN FD Message Buffer index.

- pTxFrame – Pointer to CAN FD message frame to be sent.

Return values

- kStatus__Success – - Write Tx Message Buffer Successfully.
- kStatus__Fail – - Tx Message Buffer is currently in use.
- kStatus__Timeout – - Failed to send frames within specific time.

status_t FLEXCAN__TransferFDReceiveBlocking(CAN_Type *base, uint8_t mbIdx,
flexcan_fd_frame_t *pRxFrame)

Performs a polling receive transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN FD Message Buffer index.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus__Success – - Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus__Fail – - Rx Message Buffer is empty.
- kStatus__Timeout – - Failed to receive frames within specific time.

status_t FLEXCAN__TransferFDSendNonBlocking(CAN_Type *base, flexcan_handle_t *handle,
flexcan_mb_transfer_t *pMbXfer)

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN FD Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus__Success – Start Tx Message Buffer sending process successfully.
- kStatus__Fail – Write Tx Message Buffer failed.
- kStatus_FLEXCAN_TxBusy – Tx Message Buffer is in use.

status_t FLEXCAN__TransferFDReceiveNonBlocking(CAN_Type *base, flexcan_handle_t *handle,
flexcan_mb_transfer_t *pMbXfer)

Receives a message using IRQ.

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.

- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN FD Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus_Success – - Start Rx Message Buffer receiving process successfully.
- kStatus_FLEXCAN_RxBusy – - Rx Message Buffer is in use.

```
void FLEXCAN_TransferFDAbortSend(CAN_Type *base, flexcan_handle_t *handle, uint8_t mbIdx)
```

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN FD Message Buffer index.

```
void FLEXCAN_TransferFDAbortReceive(CAN_Type *base, flexcan_handle_t *handle, uint8_t mbIdx)
```

Aborts the interrupt driven message receive process.

This function aborts the interrupt driven message receive process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN FD Message Buffer index.

```
status_t FLEXCAN_TransferSendBlocking(CAN_Type *base, uint8_t mbIdx, flexcan_frame_t *pTxFrame)
```

Performs a polling send transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN Message Buffer index.
- pTxFrame – Pointer to CAN message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.
- kStatus_Timeout – - Failed to send frames within specific time.

```
status_t FLEXCAN_TransferReceiveBlocking(CAN_Type *base, uint8_t mbIdx, flexcan_frame_t *pRxFrame)
```

Performs a polling receive transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN Message Buffer index.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – Rx Message Buffer is empty.
- kStatus_Timeout – Failed to receive frames within specific time.

status_t FLEXCAN_TransferReceiveFifoBlocking(CAN_Type *base, *flexcan_frame_t* *pRxFrame)
Performs a polling receive transaction from Legacy Rx FIFO on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – Read Message from Rx FIFO successfully.
- kStatus_Fail – Rx FIFO is not enabled.
- kStatus_Timeout – Failed to receive frames within specific time.

status_t FLEXCAN_TransferReceiveEnhancedFifoBlocking(CAN_Type *base, *flexcan_fd_frame_t* *pRxFrame)

Performs a polling receive transaction from Enhanced Rx FIFO on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – Read Message from Rx FIFO successfully.
- kStatus_Fail – Rx FIFO is not enabled.
- kStatus_Timeout – Failed to receive frames within specific time.

void FLEXCAN_TransferCreateHandle(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_transfer_callback_t* callback, void *userData)

Initializes the FlexCAN handle.

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

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- callback – The callback function.
- userData – The parameter of the callback function.

status_t FLEXCAN__TransferSendNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the *flexcan_mb_transfer_t*.

Return values

- kStatus_Success – Start Tx Message Buffer sending process successfully.
- kStatus_Fail – Write Tx Message Buffer failed.
- kStatus_FLEXCAN__TxBusy – Tx Message Buffer is in use.

status_t FLEXCAN__TransferReceiveNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Receives a message using IRQ.

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the *flexcan_mb_transfer_t*.

Return values

- kStatus_Success – Start Rx Message Buffer receiving process successfully.
- kStatus_FLEXCAN__RxBusy – Rx Message Buffer is in use.

status_t FLEXCAN__TransferReceiveFifoNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_fifo_transfer_t* *pFifoXfer)

Receives a message from Rx FIFO using IRQ.

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pFifoXfer – FlexCAN Rx FIFO transfer structure. See the *flexcan_fifo_transfer_t*.

Return values

- `kStatus__Success` -- Start Rx FIFO receiving process successfully.
- `kStatus__FLEXCAN__RxFifoBusy` -- Rx FIFO is currently in use.

`status_t` FLEXCAN__TransferGetReceiveFifoCount(`CAN_Type` *base, `flexcan_handle_t` *handle, `size_t` *count)

Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- `kStatus__InvalidArgument` – count is Invalid.
- `kStatus__Success` – Successfully return the count.

`status_t` FLEXCAN__TransferReceiveEnhancedFifoNonBlocking(`CAN_Type` *base, `flexcan_handle_t` *handle, `flexcan_fifo_transfer_t` *pFifoXfer)

Receives a message from Enhanced Rx FIFO using IRQ.

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pFifoXfer – FlexCAN Rx FIFO transfer structure. See the ref `flexcan_fifo_transfer_t`.

Return values

- `kStatus__Success` -- Start Rx FIFO receiving process successfully.
- `kStatus__FLEXCAN__RxFifoBusy` -- Rx FIFO is currently in use.

`static inline status_t` FLEXCAN__TransferGetReceiveEnhancedFifoCount(`CAN_Type` *base, `flexcan_handle_t` *handle, `size_t` *count)

Gets the Enhanced Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- `kStatus__InvalidArgument` – count is Invalid.
- `kStatus__Success` – Successfully return the count.

`uint32_t` FLEXCAN__GetTimeStamp(`flexcan_handle_t` *handle, `uint8_t` mbIdx)

Gets the detail index of Mailbox's Timestamp by handle.

Then function can only be used when calling non-blocking Data transfer (TX/RX) API, After TX/RX data transfer done (User can get the status by handler's callback

function), we can get the detail index of Mailbox's timestamp by handle, Detail non-blocking data transfer API (TX/RX) contain. -FLEXCAN_TransferSendNonBlocking -FLEXCAN_TransferFDSendNonBlocking -FLEXCAN_TransferReceiveNonBlocking -FLEXCAN_TransferFDReceiveNonBlocking -FLEXCAN_TransferReceiveFifoNonBlocking

Parameters

- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

Return values

the – index of mailbox 's timestamp stored in the handle.

void FLEXCAN_TransferAbortSend(CAN_Type *base, *flexcan_handle_t* *handle, uint8_t mbIdx)

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

void FLEXCAN_TransferAbortReceive(CAN_Type *base, *flexcan_handle_t* *handle, uint8_t mbIdx)

Aborts the interrupt driven message receive process.

This function aborts the interrupt driven message receive process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

void FLEXCAN_TransferAbortReceiveFifo(CAN_Type *base, *flexcan_handle_t* *handle)

Aborts the interrupt driven message receive from Rx FIFO process.

This function aborts the interrupt driven message receive from Rx FIFO process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_TransferAbortReceiveEnhancedFifo(CAN_Type *base, *flexcan_handle_t* *handle)

Aborts the interrupt driven message receive from Enhanced Rx FIFO process.

This function aborts the interrupt driven message receive from Enhanced Rx FIFO process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_TransferHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN IRQ handle function.

This function handles the FlexCAN Error, the Message Buffer, and the Rx FIFO IRQ request.

Parameters

- base – FlexCAN peripheral base address.

- handle – FlexCAN handle pointer.

void FLEXCAN_MbHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle, uint32_t startMbIdx, uint32_t endMbIdx)

FlexCAN Message Buffer IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- startMbIdx – First Message Buffer to handle.
- endMbIdx – Last Message Buffer to handle.

void FLEXCAN_EhancedRxFifoHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN Enhanced Rx FIFO IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_BusoffErrorHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN Bus Off, Error and Warning IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_PNWakeUpHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN Pretended Networking Wake-up IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_MemoryErrorHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN Memory Error IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

FSL_FLEXCAN_DRIVER_VERSION

FlexCAN driver version.

FlexCAN transfer status.

Values:

enumerator kStatus_FLEXCAN_TxBusy

Tx Message Buffer is Busy.

enumerator kStatus_FLEXCAN_TxIdle

Tx Message Buffer is Idle.

enumerator kStatus_FLEXCAN_TxSwitchToRx

Remote Message is send out and Message buffer changed to Receive one.

enumerator kStatus_FLEXCAN_RxBusy
Rx Message Buffer is Busy.

enumerator kStatus_FLEXCAN_RxIdle
Rx Message Buffer is Idle.

enumerator kStatus_FLEXCAN_RxOverflow
Rx Message Buffer is Overflowed.

enumerator kStatus_FLEXCAN_RxFifoBusy
Rx Message FIFO is Busy.

enumerator kStatus_FLEXCAN_RxFifoIdle
Rx Message FIFO is Idle.

enumerator kStatus_FLEXCAN_RxFifoOverflow
Rx Message FIFO is overflowed.

enumerator kStatus_FLEXCAN_RxFifoWarning
Rx Message FIFO is almost overflowed.

enumerator kStatus_FLEXCAN_RxFifoDisabled
Rx Message FIFO is disabled during reading.

enumerator kStatus_FLEXCAN_ErrorStatus
FlexCAN Module Error and Status.

enumerator kStatus_FLEXCAN_WakeUp
FlexCAN is waken up from STOP mode.

enumerator kStatus_FLEXCAN_UnHandled
UnHandled Interrupt asserted.

enumerator kStatus_FLEXCAN_RxRemote
Rx Remote Message Received in Mail box.

enumerator kStatus_FLEXCAN_RxFifoUnderflow
Enhanced Rx Message FIFO is underflow.

enumerator kStatus_FLEXCAN_MemoryError
FlexCAN Memory Error.

enum _flexcan_frame_format
FlexCAN frame format.

Values:

enumerator kFLEXCAN_FrameFormatStandard
Standard frame format attribute.

enumerator kFLEXCAN_FrameFormatExtend
Extend frame format attribute.

enum _flexcan_frame_type
FlexCAN frame type.

Values:

enumerator kFLEXCAN_FrameTypeData
Data frame type attribute.

enumerator kFLEXCAN_FrameTypeRemote
Remote frame type attribute.

enum _flexcan_clock_source
FlexCAN clock source.

Deprecated:

Do not use the kFLEXCAN_ClkSrcOs. It has been superceded kFLEXCAN_ClkSrc0

Do not use the kFLEXCAN_ClkSrcPeri. It has been superceded kFLEXCAN_ClkSrc1

Values:

enumerator kFLEXCAN_ClkSrcOsc

FlexCAN Protocol Engine clock from Oscillator.

enumerator kFLEXCAN_ClkSrcPeri

FlexCAN Protocol Engine clock from Peripheral Clock.

enumerator kFLEXCAN_ClkSrc0

FlexCAN Protocol Engine clock selected by user as SRC == 0.

enumerator kFLEXCAN_ClkSrc1

FlexCAN Protocol Engine clock selected by user as SRC == 1.

enum _flexcan_wake_up_source
FlexCAN wake up source.

Values:

enumerator kFLEXCAN_WakeupSrcUnfiltered

FlexCAN uses unfiltered Rx input to detect edge.

enumerator kFLEXCAN_WakeupSrcFiltered

FlexCAN uses filtered Rx input to detect edge.

enum _flexcan_rx_fifo_filter_type
FlexCAN Rx Fifo Filter type.

Values:

enumerator kFLEXCAN_RxFifoFilterTypeA

One full ID (standard and extended) per ID Filter element.

enumerator kFLEXCAN_RxFifoFilterTypeB

Two full standard IDs or two partial 14-bit ID slices per ID Filter Table element.

enumerator kFLEXCAN_RxFifoFilterTypeC

Four partial 8-bit Standard or extended ID slices per ID Filter Table element.

enumerator kFLEXCAN_RxFifoFilterTypeD

All frames rejected.

enum _flexcan_mb_size
FlexCAN Message Buffer Payload size.

Values:

enumerator kFLEXCAN_8BperMB

Selects 8 bytes per Message Buffer.

enumerator kFLEXCAN_16BperMB

Selects 16 bytes per Message Buffer.

enumerator kFLEXCAN_32BperMB

Selects 32 bytes per Message Buffer.

enumerator kFLEXCAN_64BperMB

Selects 64 bytes per Message Buffer.

enum _flexcan_fd_frame_length

FlexCAN CAN FD frame supporting data length (available DLC values).

For Tx, when the Data size corresponding to DLC value stored in the MB selected for transmission is larger than the MB Payload size, FlexCAN adds the necessary number of bytes with constant 0xCC pattern to complete the expected DLC. For Rx, when the Data size corresponding to DLC value received from the CAN bus is larger than the MB Payload size, the high order bytes that do not fit the Payload size will lose.

Values:

enumerator kFLEXCAN_0BperFrame

Frame contains 0 valid data bytes.

enumerator kFLEXCAN_1BperFrame

Frame contains 1 valid data bytes.

enumerator kFLEXCAN_2BperFrame

Frame contains 2 valid data bytes.

enumerator kFLEXCAN_3BperFrame

Frame contains 3 valid data bytes.

enumerator kFLEXCAN_4BperFrame

Frame contains 4 valid data bytes.

enumerator kFLEXCAN_5BperFrame

Frame contains 5 valid data bytes.

enumerator kFLEXCAN_6BperFrame

Frame contains 6 valid data bytes.

enumerator kFLEXCAN_7BperFrame

Frame contains 7 valid data bytes.

enumerator kFLEXCAN_8BperFrame

Frame contains 8 valid data bytes.

enumerator kFLEXCAN_12BperFrame

Frame contains 12 valid data bytes.

enumerator kFLEXCAN_16BperFrame

Frame contains 16 valid data bytes.

enumerator kFLEXCAN_20BperFrame

Frame contains 20 valid data bytes.

enumerator kFLEXCAN_24BperFrame

Frame contains 24 valid data bytes.

enumerator kFLEXCAN_32BperFrame

Frame contains 32 valid data bytes.

enumerator kFLEXCAN_48BperFrame

Frame contains 48 valid data bytes.

enumerator kFLEXCAN_64BperFrame

Frame contains 64 valid data bytes.

enum _flexcan_efifo_dma_per_read_length

FlexCAN Enhanced Rx Fifo DMA transfer per read length enumerations.

Values:

enumerator kFLEXCAN_1WordPerRead

Transfer 1 32-bit words (CS).

enumerator kFLEXCAN_2WordPerRead

Transfer 2 32-bit words (CS + ID).

enumerator kFLEXCAN_3WordPerRead

Transfer 3 32-bit words (CS + ID + 1~4 bytes data).

enumerator kFLEXCAN_4WordPerRead

Transfer 4 32-bit words (CS + ID + 5~8 bytes data).

enumerator kFLEXCAN_5WordPerRead

Transfer 5 32-bit words (CS + ID + 9~12 bytes data).

enumerator kFLEXCAN_6WordPerRead

Transfer 6 32-bit words (CS + ID + 13~16 bytes data).

enumerator kFLEXCAN_7WordPerRead

Transfer 7 32-bit words (CS + ID + 17~20 bytes data).

enumerator kFLEXCAN_8WordPerRead

Transfer 8 32-bit words (CS + ID + 21~24 bytes data).

enumerator kFLEXCAN_9WordPerRead

Transfer 9 32-bit words (CS + ID + 25~28 bytes data).

enumerator kFLEXCAN_10WordPerRead

Transfer 10 32-bit words (CS + ID + 29~32 bytes data).

enumerator kFLEXCAN_11WordPerRead

Transfer 11 32-bit words (CS + ID + 33~36 bytes data).

enumerator kFLEXCAN_12WordPerRead

Transfer 12 32-bit words (CS + ID + 37~40 bytes data).

enumerator kFLEXCAN_13WordPerRead

Transfer 13 32-bit words (CS + ID + 41~44 bytes data).

enumerator kFLEXCAN_14WordPerRead

Transfer 14 32-bit words (CS + ID + 45~48 bytes data).

enumerator kFLEXCAN_15WordPerRead

Transfer 15 32-bit words (CS + ID + 49~52 bytes data).

enumerator kFLEXCAN_16WordPerRead

Transfer 16 32-bit words (CS + ID + 53~56 bytes data).

enumerator kFLEXCAN_17WordPerRead

Transfer 17 32-bit words (CS + ID + 57~60 bytes data).

enumerator kFLEXCAN_18WordPerRead

Transfer 18 32-bit words (CS + ID + 61~64 bytes data).

enumerator kFLEXCAN_19WordPerRead

Transfer 19 32-bit words (CS + ID + 64 bytes data + ID HIT).

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.

enumerator kFLEXCAN_FDErrorInterruptEnable

CAN FD Error interrupt, use bit 31.

enumerator kFLEXCAN_PNMatchWakeUpInterruptEnable

PN Match Wake Up interrupt, use high word bit 17.

enumerator kFLEXCAN_PNTimeoutWakeUpInterruptEnable

PN Timeout Wake Up interrupt, use high word bit 16. Enhanced Rx FIFO Underflow interrupt, use high word bit 31.

enumerator kFLEXCAN_ERxFifoUnderflowInterruptEnable

Enhanced Rx FIFO Overflow interrupt, use high word bit 30.

enumerator kFLEXCAN_ERxFifoOverflowInterruptEnable

Enhanced Rx FIFO Watermark interrupt, use high word bit 29.

enumerator kFLEXCAN_ERxFifoWatermarkInterruptEnable

Enhanced Rx FIFO Data Available interrupt, use high word bit 28.

enumerator kFLEXCAN_ERxFifoDataAvlInterruptEnable

enumerator kFLEXCAN_HostAccessNCErrInterruptEnable

Host Access With Non-Correctable Errors interrupt, use high word bit 0.

enumerator kFLEXCAN_FlexCanAccessNCErrInterruptEnable

FlexCAN Access With Non-Correctable Errors interrupt, use high word bit 2.

enumerator kFLEXCAN_HostOrFlexCanCErrInterruptEnable

Host or FlexCAN Access With Correctable Errors interrupt, use high word bit 3.

enum _flexcan_flags

FlexCAN status flags.

This provides constants for the FlexCAN status flags for use in the FlexCAN functions.

Note: The CPU read action clears the bits corresponding to the FLEXCAN_ErrorFlag macro, therefore user need to read status flags and distinguish which error is occur using _flexcan_error_flags enumerations.

Values:

enumerator kFLEXCAN_ErrorOverrunFlag

Error Overrun Status.

enumerator kFLEXCAN_FDErrIntFlag

CAN FD Error Interrupt Flag.

enumerator kFLEXCAN_BusoffDoneIntFlag

Bus Off process completed Interrupt Flag.

enumerator kFLEXCAN_SynchFlag

CAN Synchronization Status.

enumerator kFLEXCAN_TxWarningIntFlag

Tx Warning Interrupt Flag.

enumerator kFLEXCAN_RxWarningIntFlag

Rx Warning Interrupt Flag.

enumerator kFLEXCAN_IdleFlag

FlexCAN In IDLE Status.

enumerator kFLEXCAN_FaultConfinementFlag

FlexCAN Fault Confinement State.

enumerator kFLEXCAN_TransmittingFlag

FlexCAN In Transmission Status.

enumerator kFLEXCAN_ReceivingFlag

FlexCAN In Reception Status.

enumerator kFLEXCAN_BusOffIntFlag

Bus Off Interrupt Flag.

enumerator kFLEXCAN_ErrorIntFlag

CAN Error Interrupt Flag.

enumerator kFLEXCAN_WakeUpIntFlag

Self Wake-Up Interrupt Flag.

enumerator kFLEXCAN_ErrorFlag

enumerator kFLEXCAN_PNMatchIntFlag

PN Matching Event Interrupt Flag.

enumerator kFLEXCAN_PNTimeoutIntFlag

PN Timeout Event Interrupt Flag.

enumerator kFLEXCAN_ERxFifoUnderflowIntFlag

Enhanced Rx FIFO underflow Interrupt Flag.

enumerator kFLEXCAN_ERxFifoOverflowIntFlag

Enhanced Rx FIFO overflow Interrupt Flag.

enumerator kFLEXCAN_ERxFifoWatermarkIntFlag

Enhanced Rx FIFO watermark Interrupt Flag.

enumerator kFLEXCAN_ERxFifoDataAvlIntFlag

Enhanced Rx FIFO data available Interrupt Flag.

enumerator kFLEXCAN_ERxFifoEmptyFlag

Enhanced Rx FIFO empty status.

enumerator kFLEXCAN_ERxFifoFullFlag

Enhanced Rx FIFO full status.

enumerator kFLEXCAN_HostAccessNonCorrectableErrorIntFlag

Host Access With Non-Correctable Error Interrupt Flag.

enumerator kFLEXCAN_FlexCanAccessNonCorrectableErrorIntFlag

FlexCAN Access With Non-Correctable Error Interrupt Flag.

enumerator kFLEXCAN_CorrectableErrorIntFlag

Correctable Error Interrupt Flag.

enumerator kFLEXCAN_HostAccessNonCorrectableErrorOverrunFlag

Host Access With Non-Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN_FlexCanAccessNonCorrectableErrorOverrunFlag

FlexCAN Access With Non-Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN_CorrectableErrorOverrunFlag

Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN_AllMemoryErrorIntFlag

All Memory Error Interrupt Flags.

enumerator kFLEXCAN_AllMemoryErrorFlag

All Memory Error Flags.

enum _flexcan_error_flags

FlexCAN error status flags.

The FlexCAN Error Status enumerations is used to report current error of the FlexCAN bus. This enumerations should be used with KFLEXCAN_ErrorFlag in _flexcan_flags enumerations to determine which error is generated.

Values:

enumerator kFLEXCAN_FDStuffingError

Stuffing Error.

enumerator kFLEXCAN_FDFormError

Form Error.

enumerator kFLEXCAN_FDCrcError

Cyclic Redundancy Check Error.

enumerator kFLEXCAN_FDBit0Error

Unable to send dominant bit.

enumerator kFLEXCAN_FDBit1Error

Unable to send recessive bit.

enumerator kFLEXCAN_TxErrorWarningFlag

Tx Error Warning Status.

enumerator kFLEXCAN_RxErrorWarningFlag

Rx Error Warning Status.

enumerator kFLEXCAN_StuffingError

Stuffing Error.

enumerator kFLEXCAN_FormError

Form Error.

enumerator kFLEXCAN_CrcError

Cyclic Redundancy Check Error.

enumerator kFLEXCAN_AckError

Received no ACK on transmission.

enumerator kFLEXCAN_Bit0Error

Unable to send dominant bit.

enumerator kFLEXCAN_Bit1Error

Unable to send recessive bit.

FlexCAN Legacy Rx FIFO status flags.

The FlexCAN Legacy Rx FIFO Status enumerations are used to determine the status of the Rx FIFO. Because Rx FIFO occupy the MB0 ~ MB7 (Rx Fifo filter also occupies more Message Buffer space), Rx FIFO status flags are mapped to the corresponding Message Buffer status flags.

Values:

enumerator kFLEXCAN_RxFifoOverflowFlag

Rx FIFO overflow flag.

enumerator kFLEXCAN_RxFifoWarningFlag

Rx FIFO almost full flag.

enumerator kFLEXCAN_RxFifoFrameAvlFlag

Frames available in Rx FIFO flag.

enum _flexcan_memory_error_type

FlexCAN Memory Error Type.

Values:

enumerator kFLEXCAN_CorrectableError

The memory error is correctable which means on bit error.

enumerator kFLEXCAN_NonCorrectableError

The memory error is non-correctable which means two bit errors.

enum _flexcan_memory_access_type

FlexCAN Memory Access Type.

Values:

enumerator kFLEXCAN_MoveOutFlexCanAccess

The memory error was detected during move-out FlexCAN access.

enumerator kFLEXCAN_MoveInAccess

The memory error was detected during move-in FlexCAN access.

enumerator kFLEXCAN_TxArbitrationAccess

The memory error was detected during Tx Arbitration FlexCAN access.

enumerator kFLEXCAN_RxMatchingAccess

The memory error was detected during Rx Matching FlexCAN access.

enumerator kFLEXCAN_MoveOutHostAccess

The memory error was detected during Rx Matching Host (CPU) access.

enum _flexcan_byte_error_syndrome

FlexCAN Memory Error Byte Syndrome.

Values:

enumerator kFLEXCAN_NoError

No bit error in this byte.

enumerator kFLEXCAN_ParityBits0Error

Parity bit 0 error in this byte.

enumerator kFLEXCAN_ParityBits1Error

Parity bit 1 error in this byte.

enumerator kFLEXCAN_ParityBits2Error

Parity bit 2 error in this byte.

enumerator kFLEXCAN_ParityBits3Error

Parity bit 3 error in this byte.

enumerator kFLEXCAN_ParityBits4Error

Parity bit 4 error in this byte.

enumerator kFLEXCAN_DataBits0Error

Data bit 0 error in this byte.

enumerator kFLEXCAN_DataBits1Error

Data bit 1 error in this byte.

enumerator kFLEXCAN_DataBits2Error

Data bit 2 error in this byte.

enumerator kFLEXCAN_DataBits3Error

Data bit 3 error in this byte.

enumerator kFLEXCAN_DataBits4Error

Data bit 4 error in this byte.

enumerator kFLEXCAN_DataBits5Error

Data bit 5 error in this byte.

enumerator kFLEXCAN_DataBits6Error

Data bit 6 error in this byte.

enumerator kFLEXCAN_DataBits7Error

Data bit 7 error in this byte.

enumerator kFLEXCAN_AllZeroError

All-zeros non-correctable error in this byte.

enumerator kFLEXCAN_AllOneError

All-ones non-correctable error in this byte.

enumerator kFLEXCAN_NonCorrectableErrors

Non-correctable error in this byte.

enum _flexcan_pn_match_source

FlexCAN Pretended Networking match source selection.

Values:

enumerator kFLEXCAN_PNMatSrcID

Message match with ID filtering.

enumerator kFLEXCAN_PNMatSrcIDAndData

Message match with ID filtering and payload filtering.

enum _flexcan_pn_match_mode

FlexCAN Pretended Networking mode match type.

Values:

enumerator kFLEXCAN_PNMatModeEqual

Match upon ID/Payload contents against an exact target value.

enumerator kFLEXCAN_PNMatModeGreater

Match upon an ID/Payload value greater than or equal to a specified target value.

enumerator kFLEXCAN_PNMatModeSmaller

Match upon an ID/Payload value smaller than or equal to a specified target value.

enumerator kFLEXCAN_PNMatModeRange

Match upon an ID/Payload value inside a range, greater than or equal to a specified lower limit, and smaller than or equal to a specified upper limit

typedef enum _flexcan_frame_format flexcan_frame_format_t

FlexCAN frame format.

typedef enum _flexcan_frame_type flexcan_frame_type_t

FlexCAN frame type.

typedef enum _flexcan_clock_source flexcan_clock_source_t

FlexCAN clock source.

Deprecated:

Do not use the kFLEXCAN_ClkSrcOs. It has been superceded kFLEXCAN_ClkSrc0

Do not use the kFLEXCAN_ClkSrcPeri. It has been superceded kFLEXCAN_ClkSrc1

typedef enum _flexcan_wake_up_source flexcan_wake_up_source_t

FlexCAN wake up source.

typedef enum _flexcan_rx_fifo_filter_type flexcan_rx_fifo_filter_type_t

FlexCAN Rx Fifo Filter type.

typedef enum *_flexcan_mb_size* flexcan_mb_size_t

FlexCAN Message Buffer Payload size.

typedef enum *_flexcan_efifo_dma_per_read_length* flexcan_efifo_dma_per_read_length_t

FlexCAN Enhanced Rx Fifo DMA transfer per read length enumerations.

typedef enum *_flexcan_rx_fifo_priority* flexcan_rx_fifo_priority_t

FlexCAN Enhanced/Legacy Rx FIFO priority.

The matching process starts from the Rx MB(or Enhanced/Legacy Rx FIFO) with higher priority. If no MB(or Enhanced/Legacy Rx FIFO filter) is satisfied, the matching process goes on with the Enhanced/Legacy Rx FIFO(or Rx MB) with lower priority.

typedef enum *_flexcan_memory_error_type* flexcan_memory_error_type_t

FlexCAN Memory Error Type.

typedef enum *_flexcan_memory_access_type* flexcan_memory_access_type_t

FlexCAN Memory Access Type.

typedef enum *_flexcan_byte_error_syndrome* flexcan_byte_error_syndrome_t

FlexCAN Memory Error Byte Syndrome.

typedef struct *_flexcan_memory_error_report_status* flexcan_memory_error_report_status_t

FlexCAN memory error register status structure.

This structure contains the memory access properties that caused a memory error access. It is used as the parameter of FLEXCAN_GetMemoryErrorReportStatus() function. And user can use FLEXCAN_GetMemoryErrorReportStatus to get the status of the last memory error access.

typedef struct *_flexcan_frame* flexcan_frame_t

FlexCAN message frame structure.

typedef struct *_flexcan_fd_frame* flexcan_fd_frame_t

CAN FD message frame structure.

The CAN FD message supporting up to sixty four bytes can be used for a data frame, depending on the length selected for the message buffers. The length should be a enumeration member, see *_flexcan_fd_frame_length*.

typedef struct *_flexcan_timing_config* flexcan_timing_config_t

FlexCAN protocol timing characteristic configuration structure.

typedef struct *_flexcan_config* flexcan_config_t

FlexCAN module configuration structure.

Deprecated:

Do not use the baudRate. It has been superceded bitRate

Do not use the baudRateFD. It has been superceded bitRateFD

typedef struct *_flexcan_rx_mb_config* flexcan_rx_mb_config_t

FlexCAN Receive Message Buffer configuration structure.

This structure is used as the parameter of FLEXCAN_SetRxMbConfig() function. The FLEXCAN_SetRxMbConfig() function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

typedef enum *_flexcan_pn_match_source* flexcan_pn_match_source_t

FlexCAN Pretended Networking match source selection.

`typedef enum _flexcan_pn_match_mode flexcan_pn_match_mode_t`

FlexCAN Pretended Networking mode match type.

`typedef struct _flexcan_pn_config flexcan_pn_config_t`

FlexCAN Pretended Networking configuration structure.

This structure is used as the parameter of `FLEXCAN_SetPNConfig()` function. The `FLEXCAN_SetPNConfig()` function is used to configure FlexCAN Networking work mode.

`typedef struct _flexcan_rx_fifo_config flexcan_rx_fifo_config_t`

FlexCAN Legacy Rx FIFO configuration structure.

`typedef struct _flexcan_enhanced_rx_fifo_std_id_filter flexcan_enhanced_rx_fifo_std_id_filter_t`

FlexCAN Enhanced Rx FIFO Standard ID filter element structure.

`typedef struct _flexcan_enhanced_rx_fifo_ext_id_filter flexcan_enhanced_rx_fifo_ext_id_filter_t`

FlexCAN Enhanced Rx FIFO Extended ID filter element structure.

`typedef struct _flexcan_enhanced_rx_fifo_config flexcan_enhanced_rx_fifo_config_t`

FlexCAN Enhanced Rx FIFO configuration structure.

`typedef struct _flexcan_mb_transfer flexcan_mb_transfer_t`

FlexCAN Message Buffer transfer.

`typedef struct _flexcan_fifo_transfer flexcan_fifo_transfer_t`

FlexCAN Rx FIFO transfer.

`typedef struct _flexcan_handle flexcan_handle_t`

FlexCAN handle structure definition.

`typedef void (*flexcan_transfer_callback_t)(CAN_Type *base, flexcan_handle_t *handle, status_t status, uint64_t result, void *userData)`

`FLEXCAN_WAIT_TIMEOUT`

`FLEXCAN_POLLING_TIMEOUT`

Max loops to wait for polling transfer.

`FLEXCAN_MODULE_TIMEOUT`

Max loops to wait for FlexCAN register access complete.

`DLC_LENGTH_DECODE(dlc)`

FlexCAN frame length helper macro.

`FLEXCAN_ID_STD(id)`

FlexCAN Frame ID helper macro.

Standard Frame ID helper macro.

`FLEXCAN_ID_EXT(id)`

Extend Frame ID helper macro.

`FLEXCAN_RX_MB_STD_MASK(id, rtr, ide)`

FlexCAN Rx Message Buffer Mask helper macro.

Standard Rx Message Buffer Mask helper macro.

`FLEXCAN_RX_MB_EXT_MASK(id, rtr, ide)`

Extend Rx Message Buffer Mask helper macro.

`FLEXCAN_RX_FIFO_STD_MASK_TYPE_A(id, rtr, ide)`

FlexCAN Legacy Rx FIFO Mask helper macro.

Standard Rx FIFO Mask helper macro Type A helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(id, rtr, ide)

Standard Rx FIFO Mask helper macro Type B upper part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(id, rtr, ide)

Standard Rx FIFO Mask helper macro Type B lower part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(id)

Standard Rx FIFO Mask helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(id)

Standard Rx FIFO Mask helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(id)

Standard Rx FIFO Mask helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(id)

Standard Rx FIFO Mask helper macro Type C lower part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(id, rtr, ide)

Extend Rx FIFO Mask helper macro Type A helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(id, rtr, ide)

Extend Rx FIFO Mask helper macro Type B upper part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(id, rtr, ide)

Extend Rx FIFO Mask helper macro Type B lower part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(id)

Extend Rx FIFO Mask helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(id)

Extend Rx FIFO Mask helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(id)

Extend Rx FIFO Mask helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id)

Extend Rx FIFO Mask helper macro Type C lower part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(id, rtr, ide)

FlexCAN Rx FIFO Filter helper macro.

Standard Rx FIFO Filter helper macro Type A helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH(id, rtr, ide)

Standard Rx FIFO Filter helper macro Type B upper part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW(id, rtr, ide)

Standard Rx FIFO Filter helper macro Type B lower part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH(id)

Standard Rx FIFO Filter helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH(id)

Standard Rx FIFO Filter helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW(id)

Standard Rx FIFO Filter helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW(id)

Standard Rx FIFO Filter helper macro Type C lower part helper macro.

`FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A(id, rtr, ide)`

Extend Rx FIFO Filter helper macro Type A helper macro.

`FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH(id, rtr, ide)`

Extend Rx FIFO Filter helper macro Type B upper part helper macro.

`FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW(id, rtr, ide)`

Extend Rx FIFO Filter helper macro Type B lower part helper macro.

`FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH(id)`

Extend Rx FIFO Filter helper macro Type C upper part helper macro.

`FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH(id)`

Extend Rx FIFO Filter helper macro Type C mid-upper part helper macro.

`FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW(id)`

Extend Rx FIFO Filter helper macro Type C mid-lower part helper macro.

`FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW(id)`

Extend Rx FIFO Filter helper macro Type C lower part helper macro.

`ENHANCED_RX_FIFO_FSC(x)`

FlexCAN Enhanced Rx FIFO Filter and Mask helper macro.

`RTR_STD_HIGH(x)`

`RTR_STD_LOW(x)`

`RTR_EXT(x)`

`ID_STD_LOW(id)`

`ID_STD_HIGH(id)`

`ID_EXT(id)`

`FLEXCAN_ENHANCED_RX_FIFO_STD_MASK_AND_FILTER(id, rtr, id_mask, rtr_mask)`

Standard ID filter element with filter + mask scheme.

`FLEXCAN_ENHANCED_RX_FIFO_STD_FILTER_WITH_RANGE(id_upper, rtr, id_lower,
rtr_mask)`

Standard ID filter element with filter range.

`FLEXCAN_ENHANCED_RX_FIFO_STD_TWO_FILTERS(id1, rtr1, id2, rtr2)`

Standard ID filter element with two filters without masks.

`FLEXCAN_ENHANCED_RX_FIFO_EXT_MASK_AND_FILTER_LOW(id, rtr)`

Extended ID filter element with filter + mask scheme low word.

`FLEXCAN_ENHANCED_RX_FIFO_EXT_MASK_AND_FILTER_HIGH(id_mask, rtr_mask)`

Extended ID filter element with filter + mask scheme high word.

`FLEXCAN_ENHANCED_RX_FIFO_EXT_FILTER_WITH_RANGE_LOW(id_upper, rtr)`

Extended ID filter element with range scheme low word.

`FLEXCAN_ENHANCED_RX_FIFO_EXT_FILTER_WITH_RANGE_HIGH(id_lower, rtr_mask)`

Extended ID filter element with range scheme high word.

`FLEXCAN_ENHANCED_RX_FIFO_EXT_TWO_FILTERS_LOW(id2, rtr2)`

Extended ID filter element with two filters without masks low word.

`FLEXCAN_ENHANCED_RX_FIFO_EXT_TWO_FILTERS_HIGH(id1, rtr1)`

Extended ID filter element with two filters without masks high word.

`FLEXCAN_PN_STD_MASK(id, rtr)`

FlexCAN Pretended Networking ID Mask helper macro.

Standard Rx Message Buffer Mask helper macro.

`FLEXCAN_PN_EXT_MASK(id, rtr)`

Extend Rx Message Buffer Mask helper macro.

`FLEXCAN_PN_INT_MASK(x)`

FlexCAN interrupt/status flag helper macro.

`FLEXCAN_PN_INT_UNMASK(x)`

`FLEXCAN_PN_STATUS_MASK(x)`

`FLEXCAN_PN_STATUS_UNMASK(x)`

`FLEXCAN_EFIFO_INT_MASK(x)`

`FLEXCAN_EFIFO_INT_UNMASK(x)`

`FLEXCAN_EFIFO_STATUS_MASK(x)`

`FLEXCAN_EFIFO_STATUS_UNMASK(x)`

`FLEXCAN_MECR_INT_MASK(x)`

`FLEXCAN_MECR_INT_UNMASK(x)`

`FLEXCAN_MECR_STATUS_MASK(x)`

`FLEXCAN_MECR_STATUS_UNMASK(x)`

`FLEXCAN_ERROR_AND_STATUS_INT_FLAG`

`FLEXCAN_PNWAKE_UP_FLAG`

`FLEXCAN_WAKE_UP_FLAG`

`FLEXCAN_MEMORY_ERROR_INT_FLAG`

`FLEXCAN_ENHANCED_RX_FIFO_INT_FLAG`

FlexCAN Enhanced Rx FIFO base address helper macro.

`E_RX_FIFO(base)`

`FLEXCAN_CALLBACK(x)`

FlexCAN transfer callback function.

The FlexCAN transfer callback returns a value from the underlying layer. If the status equals to `kStatus_FLEXCAN_ErrorStatus`, the result parameter is the Content of FlexCAN status register which can be used to get the working status(or error status) of FlexCAN module. If the status equals to other FlexCAN Message Buffer transfer status, the result is the index of Message Buffer that generate transfer event. If the status equals to other FlexCAN Message Buffer transfer status, the result is meaningless and should be Ignored.

`struct flexcan_memory_error_report_status`

`#include <fsl_flexcan.h>` FlexCAN memory error register status structure.

This structure contains the memory access properties that caused a memory error access. It is used as the parameter of `FLEXCAN_GetMemoryErrorReportStatus()` function. And user can use `FLEXCAN_GetMemoryErrorReportStatus` to get the status of the last memory error access.

Public Members

flexcan_memory_error_type_t errorType

The type of memory error that giving rise to the report.

flexcan_memory_access_type_t accessType

The type of memory access that giving rise to the memory error.

uint16_t accessAddress

The address where memory error detected.

uint32_t errorData

The raw data word read from memory with error.

struct _flexcan_frame

#include <fsl_flexcan.h> FlexCAN message frame structure.

struct _flexcan_fd_frame

#include <fsl_flexcan.h> CAN FD message frame structure.

The CAN FD message supporting up to sixty four bytes can be used for a data frame, depending on the length selected for the message buffers. The length should be a enumeration member, see *_flexcan_fd_frame_length*.

Public Members

uint32_t idhit

Note: ID HIT offset is changed dynamically according to data length code (DLC), when DLC is 15, they will be located below. Using FLEXCAN_FixEnhancedRxFifoFrameIdHit API is recommended to ensure this idhit value is correct. CAN Enhanced Rx FIFO filter hit id (This value is only used in Enhanced Rx FIFO receive mode).

struct _flexcan_timing_config

#include <fsl_flexcan.h> FlexCAN protocol timing characteristic configuration structure.

Public Members

uint32_t preDivider

Classic CAN or CAN FD nominal phase bit rate prescaler.

uint32_t rJumpwidth

Classic CAN or CAN FD nominal phase Re-sync Jump Width.

uint32_t phaseSeg1

Classic CAN or CAN FD nominal phase Segment 1.

uint32_t phaseSeg2

Classic CAN or CAN FD nominal phase Segment 2.

uint32_t propSeg

Classic CAN or CAN FD nominal phase Propagation Segment.

uint32_t fpreDivider

CAN FD data phase bit rate prescaler.

uint32_t frJumpwidth

CAN FD data phase Re-sync Jump Width.

uint32_t fphaseSeg1

CAN FD data phase Phase Segment 1.

uint32_t fphaseSeg2

CAN FD data phase Phase Segment 2.

uint32_t fpropSeg

CAN FD data phase Propagation Segment.

struct _flexcan_config

#include <fsl_flexcan.h> FlexCAN module configuration structure.

Deprecated:

Do not use the baudRate. It has been superceded bitRate

Do not use the baudRateFD. It has been superceded bitRateFD

Public Members

flexcan_clock_source_t clkSrc

Clock source for FlexCAN Protocol Engine.

flexcan_wake_up_source_t wakeupSrc

Wake up source selection.

uint8_t maxMbNum

The maximum number of Message Buffers used by user.

bool enableLoopBack

Enable or Disable Loop Back Self Test Mode.

bool enableTimerSync

Enable or Disable Timer Synchronization.

bool 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 enableDoze

Enable or Disable Doze Mode.

bool enablePretendedeNetworking

Enable or Disable the Pretended Networking mode.

bool enableMemoryErrorControl

Enable or Disable the memory errors detection and correction mechanism.

bool enableNonCorrectableErrorEnterFreeze

Enable or Disable Non-Correctable Errors In FlexCAN Access Put Device In Freeze Mode.

`bool enableTransceiverDelayMeasure`

Enable or Disable the transceiver delay measurement, when it is enabled, then the secondary sample point position is determined by the sum of the transceiver delay measurement plus the enhanced TDC offset.

`bool enableRemoteRequestFrameStored`

true: Store Remote Request Frame in the same fashion of data frame. false: Generate an automatic Remote Response Frame.

`struct _flexcan_rx_mb_config`

#include <fsl_flexcan.h> FlexCAN Receive Message Buffer configuration structure.

This structure is used as the parameter of FLEXCAN_SetRxMbConfig() function. The FLEXCAN_SetRxMbConfig() function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

Public Members

`uint32_t id`

CAN Message Buffer Frame Identifier, should be set using FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

flexcan_frame_format_t format

CAN Frame Identifier format(Standard of Extend).

flexcan_frame_type_t type

CAN Frame Type(Data or Remote for classical CAN only).

`struct _flexcan_pn_config`

#include <fsl_flexcan.h> FlexCAN Pretended Networking configuration structure.

This structure is used as the parameter of FLEXCAN_SetPNConfig() function. The FLEXCAN_SetPNConfig() function is used to configure FlexCAN Networking work mode.

Public Members

`bool enableTimeout`

Enable or Disable timeout event trigger wakeup.

`uint16_t timeoutValue`

The timeout value that generates a wakeup event, the counter timer is incremented based on 64 times the CAN Bit Time unit.

`bool enableMatch`

Enable or Disable match event trigger wakeup.

flexcan_pn_match_source_t matchSrc

Selects the match source (ID and/or data match) to trigger wakeup.

`uint8_t matchNum`

The number of times a given message must match the predefined ID and/or data before generating a wakeup event, range in 0x1 ~ 0xFF.

flexcan_pn_match_mode_t idMatchMode

The ID match type.

flexcan_pn_match_mode_t dataMatchMode

The data match type.

uint32_t idLower

The ID target values 1 which used either for ID match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in ID match “range detection”.

uint32_t idUpper

The ID target values 2 which used only as the upper limit value in ID match “range detection” or used to store the ID mask in “equal to”.

uint8_t lengthLower

The lower limit for length of data bytes which used only in data match “range detection”. Range in 0x0 ~ 0x8.

uint8_t lengthUpper

The upper limit for length of data bytes which used only in data match “range detection”. Range in 0x0 ~ 0x8.

struct _flexcan_rx_fifo_config

#include <fsl_flexcan.h> FlexCAN Legacy Rx FIFO configuration structure.

Public Members

uint32_t *idFilterTable

Pointer to the FlexCAN Legacy Rx FIFO identifier filter table.

uint8_t idFilterNum

The FlexCAN Legacy Rx FIFO Filter elements quantity.

flexcan_rx_fifo_filter_type_t idFilterType

The FlexCAN Legacy Rx FIFO Filter type.

flexcan_rx_fifo_priority_t priority

The FlexCAN Legacy Rx FIFO receive priority.

struct _flexcan_enhanced_rx_fifo_std_id_filter

#include <fsl_flexcan.h> FlexCAN Enhanced Rx FIFO Standard ID filter element structure.

Public Members

uint32_t filterType

FlexCAN internal Free-Running Counter Time Stamp.

uint32_t rtr1

CAN FD frame data length code (DLC), range see *_flexcan_fd_frame_length*, When the length <= 8, it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use *DLC_LENGTH_DECODE(length)* macro to get the number of valid data bytes.

uint32_t std1

CAN Frame Type(DATA or REMOTE).

uint32_t rtr2

CAN Frame Identifier(STD or EXT format).

uint32_t std2

Substitute Remote request.

struct _flexcan_enhanced_rx_fifo_ext_id_filter

#include <fsl_flexcan.h> FlexCAN Enhanced Rx FIFO Extended ID filter element structure.

Public Members

uint32_t filterType

FlexCAN internal Free-Running Counter Time Stamp.

uint32_t rtr1

CAN FD frame data length code (DLC), range see `_flexcan_fd_frame_length`, When the length ≤ 8 , it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use `DLC_LENGTH_DECODE(length)` macro to get the number of valid data bytes.

uint32_t std1

CAN Frame Type(DATA or REMOTE).

uint32_t rtr2

CAN Frame Identifier(STD or EXT format).

uint32_t std2

Substitute Remote request.

struct `_flexcan_enhanced_rx_fifo_config`

`#include <fsl_flexcan.h>` FlexCAN Enhanced Rx FIFO configuration structure.

Public Members

uint32_t *idFilterTable

Pointer to the FlexCAN Enhanced Rx FIFO identifier filter table, each table member occupies 32 bit word, table size should be equal to `idFilterNum`. There are two types of Enhanced Rx FIFO filter elements that can be stored in table : extended-ID filter element (1 word, occupie 1 table members) and standard-ID filter element (2 words, occupies 2 table members), the extended-ID filter element needs to be placed in front of the table.

uint8_t idFilterPairNum

`idFilterPairNum` is the Enhanced Rx FIFO identifier filter element pair numbers, each pair of filter elements occupies 2 words and can consist of one extended ID filter element or two standard ID filter elements.

uint8_t extendIdFilterNum

The number of extended ID filter element items in the FlexCAN enhanced Rx FIFO identifier filter table, each extended-ID filter element occupies 2 words, `extendIdFilterNum` need less than or equal to `idFilterPairNum`.

uint8_t fifoWatermark

$(\text{fifoWatermark} + 1)$ is the minimum number of CAN messages stored in the Enhanced RX FIFO which can trigger FIFO watermark interrupt or a DMA request.

flexcan_efifo_dma_per_read_length_t dmaPerReadLength

Define the length of each read of the Enhanced RX FIFO element by the DAM, see `_flexcan_fd_frame_length`.

flexcan_rx_fifo_priority_t priority

The FlexCAN Enhanced Rx FIFO receive priority.

struct `_flexcan_mb_transfer`

`#include <fsl_flexcan.h>` FlexCAN Message Buffer transfer.

Public Members

flexcan_frame_t *frame

The buffer of CAN Message to be transfer.

uint8_t mbIdx

The index of Message buffer used to transfer Message.

struct *_flexcan_fifo_transfer*

#include <fsl_flexcan.h> FlexCAN Rx FIFO transfer.

Public Members

flexcan_fd_frame_t *framefd

The buffer of CAN Message to be received from Enhanced Rx FIFO.

flexcan_frame_t *frame

The buffer of CAN Message to be received from Legacy Rx FIFO.

size_t frameNum

Number of CAN Message need to be received from Legacy or Enhanced Rx FIFO.

struct *_flexcan_handle*

#include <fsl_flexcan.h> FlexCAN handle structure.

Public Members

flexcan_transfer_callback_t callback

Callback function.

void *userData

FlexCAN callback function parameter.

flexcan_frame_t *volatile mbFrameBuf[CAN_WORD1_COUNT]

The buffer for received CAN data from Message Buffers.

flexcan_fd_frame_t *volatile mbFDFrameBuf[CAN_WORD1_COUNT]

The buffer for received CAN FD data from Message Buffers.

flexcan_frame_t *volatile rxFifoFrameBuf

The buffer for received CAN data from Legacy Rx FIFO.

flexcan_fd_frame_t *volatile rxFifoFDFrameBuf

The buffer for received CAN FD data from Enhanced Rx FIFO.

size_t rxFifoFrameNum

The number of CAN messages remaining to be received from Legacy or Enhanced Rx FIFO.

size_t rxFifoTransferTotalNum

Total CAN Message number need to be received from Legacy or Enhanced Rx FIFO.

volatile uint8_t mbState[CAN_WORD1_COUNT]

Message Buffer transfer state.

volatile uint8_t rxFifoState

Rx FIFO transfer state.

volatile uint32_t timestamp[CAN_WORD1_COUNT]

Mailbox transfer timestamp.

struct byteStatus

Public Members

bool byteIsRead

The byte n (0~3) was read or not. The type of error and which bit in byte (n) is affected by the error.

struct ____unnamed20____

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 ____unnamed22____

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 ____unnamed24____

Public Members

struct __flexcan_frame

struct __flexcan_frame

struct ____unnamed26____

Public Members

uint32_t dataWord0

CAN Frame payload word0.

uint32_t dataWord1

CAN Frame payload word1.

struct ____unnamed28____

Public Members

uint8_t dataByte3
CAN Frame payload byte3.

uint8_t dataByte2
CAN Frame payload byte2.

uint8_t dataByte1
CAN Frame payload byte1.

uint8_t dataByte0
CAN Frame payload byte0.

uint8_t dataByte7
CAN Frame payload byte7.

uint8_t dataByte6
CAN Frame payload byte6.

uint8_t dataByte5
CAN Frame payload byte5.

uint8_t dataByte4
CAN Frame payload byte4.

struct ____unnamed30____

Public Members

uint32_t timestamp
FlexCAN internal Free-Running Counter Time Stamp.

uint32_t length
CAN FD frame data length code (DLC), range see `_flexcan_fd_frame_length`, When the length <= 8, it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use `DLC_LENGTH_DECODE(length)` macro to get the number of valid data bytes.

uint32_t type
CAN Frame Type(DATA only).

uint32_t format
CAN Frame Identifier(STD or EXT format).

uint32_t srr
Substitute Remote request.

uint32_t esi
Error State Indicator.

uint32_t brs
Bit Rate Switch.

uint32_t edl
Extended Data Length.

struct ____unnamed32____

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 __unnamed34__

Public Members

struct __flexcan_fd_frame

struct __flexcan_fd_frame

struct __unnamed36__

Public Members

uint32_t dataWord[16]

CAN FD Frame payload, 16 double word maximum.

struct __unnamed38__

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 __unnamed40__

Public Members

struct __flexcan_config

struct __flexcan_config

struct ____unnamed42____

Public Members

uint32_t baudRate

FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.

uint32_t baudRateFD

FlexCAN FD bit rate in bps, for CANFD data phase.

struct ____unnamed44____

Public Members

uint32_t bitRate

FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.

uint32_t bitRateFD

FlexCAN FD bit rate in bps, for CANFD data phase.

union ____unnamed46____

Public Members

struct __flexcan_pn_config

< The data target values 1 which used either for data match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in data match “range detection”.

struct __flexcan_pn_config

struct ____unnamed50____

< The data target values 1 which used either for data match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in data match “range detection”.

Public Members

uint32_t lowerWord0

CAN Frame payload word0.

uint32_t lowerWord1

CAN Frame payload word1.

struct ____unnamed52____

Public Members

uint8_t lowerByte3
CAN Frame payload byte3.

uint8_t lowerByte2
CAN Frame payload byte2.

uint8_t lowerByte1
CAN Frame payload byte1.

uint8_t lowerByte0
CAN Frame payload byte0.

uint8_t lowerByte7
CAN Frame payload byte7.

uint8_t lowerByte6
CAN Frame payload byte6.

uint8_t lowerByte5
CAN Frame payload byte5.

uint8_t lowerByte4
CAN Frame payload byte4.

union __unnamed48__

Public Members

struct __flexcan_pn_config
< The data target values 2 which used only as the upper limit value in data match “range detection” or used to store the data mask in “equal to”.

struct __flexcan_pn_config

struct __unnamed54__

< The data target values 2 which used only as the upper limit value in data match “range detection” or used to store the data mask in “equal to”.

Public Members

uint32_t upperWord0
CAN Frame payload word0.

uint32_t upperWord1
CAN Frame payload word1.

struct __unnamed56__

Public Members

uint8_t upperByte3
CAN Frame payload byte3.

uint8_t upperByte2
CAN Frame payload byte2.

uint8_t upperByte1
CAN Frame payload byte1.

uint8_t upperByte0
CAN Frame payload byte0.

uint8_t upperByte7
CAN Frame payload byte7.

uint8_t upperByte6
CAN Frame payload byte6.

uint8_t upperByte5
CAN Frame payload byte5.

uint8_t upperByte4
CAN Frame payload byte4.

2.14 FlexCAN eDMA Driver

void FLEXCAN_TransferCreateHandleEDMA(CAN_Type *base, flexcan_edma_handle_t *handle, flexcan_edma_transfer_callback_t callback, void *userData, edma_handle_t *rxFifoEdmaHandle)

Initializes the FlexCAN handle, which is used in transactional functions.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- callback – The callback function.
- userData – The parameter of the callback function.
- rxFifoEdmaHandle – User-requested DMA handle for Rx FIFO DMA transfer.

void FLEXCAN_PrepareTransfConfiguration(CAN_Type *base, flexcan_fifo_transfer_t *pFifoXfer, edma_transfer_config_t *pEdmaConfig)

Prepares the eDMA transfer configuration for FLEXCAN Legacy RX FIFO.

This function prepares the eDMA transfer configuration structure according to FLEXCAN Legacy RX FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t.
- pEdmaConfig – The user configuration structure of type edma_transfer_t.

status_t FLEXCAN_StartTransferDatafromRxFIFO(CAN_Type *base, flexcan_edma_handle_t *handle, edma_transfer_config_t *pEdmaConfig)

Start Transfer Data from the FLEXCAN Legacy Rx FIFO using eDMA.

This function to Update edma transfer configuration and Start eDMA transfer

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- pEdmaConfig – The user configuration structure of type edma_transfer_t.

Return values

- kStatus__Success – if succeed, others failed.
- kStatus_FLEXCAN_RxFifoBusy – Previous transfer ongoing.

status_t FLEXCAN_TransferReceiveFifoEDMA(CAN_Type *base, flexcan_edma_handle_t *handle, flexcan_fifo_transfer_t *pFifoXfer)

Receives the CAN Message from the Legacy Rx FIFO using eDMA.

This function receives the CAN Message using eDMA. This is a non-blocking function, which returns right away. After the CAN Message is received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t.

Return values

- kStatus__Success – if succeed, others failed.
- kStatus_FLEXCAN_RxFifoBusy – Previous transfer ongoing.

status_t FLEXCAN_TransferGetReceiveFifoCountEMDA(CAN_Type *base, flexcan_edma_handle_t *handle, size_t *count)

Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus__Success – Successfully return the count.

void FLEXCAN_TransferAbortReceiveFifoEDMA(CAN_Type *base, flexcan_edma_handle_t *handle)

Aborts the receive Legacy/Enhanced Rx FIFO process which used eDMA.

This function aborts the receive Legacy/Enhanced Rx FIFO process which used eDMA.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.

status_t FLEXCAN_TransferReceiveEnhancedFifoEDMA(CAN_Type *base, flexcan_edma_handle_t *handle, flexcan_fifo_transfer_t *pFifoXfer)

Receives the CAN FD Message from the Enhanced Rx FIFO using eDMA.

This function receives the CAN FD Message using eDMA. This is a non-blocking function, which returns right away. After the CAN Message is received, the receive callback function is called.

Parameters

- `base` – FlexCAN peripheral base address.
- `handle` – Pointer to `flexcan_edma_handle_t` structure.
- `pFifoXfer` – FlexCAN Rx FIFO EDMA transfer structure, see `flexcan_fifo_transfer_t`.

Return values

- `kStatus_Success` – if succeed, others failed.
- `kStatus_FLEXCAN_RxFifoBusy` – Previous transfer ongoing.

```
static inline status_t FLEXCAN_TransferGetReceiveEnhancedFifoCountEMDA(CAN_Type *base,
                                                                    flex-
                                                                    can_edma_handle_t
                                                                    *handle, size_t
                                                                    *count)
```

Gets the Enhanced Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- `base` – FlexCAN peripheral base address.
- `handle` – FlexCAN handle pointer.
- `count` – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- `kStatus_InvalidArgument` – count is Invalid.
- `kStatus_Success` – Successfully return the count.

FSL_FLEXCAN_EDMA_DRIVER_VERSION

FlexCAN EDMA driver version.

```
typedef struct flexcan_edma_handle flexcan_edma_handle_t
```

```
typedef void (*flexcan_edma_transfer_callback_t)(CAN_Type *base, flexcan_edma_handle_t
*handle, status_t status, void *userData)
```

FlexCAN transfer callback function.

```
struct flexcan_edma_handle
```

```
#include <fsl_flexcan_edma.h> FlexCAN eDMA handle.
```

Public Members

`flexcan_edma_transfer_callback_t` callback
Callback function.

`void *userData`

FlexCAN callback function parameter.

`edma_handle_t *rxFifoEdmaHandle`

The EDMA handler for Rx FIFO.

volatile uint8_t rxFifoState

Rx FIFO transfer state.

size_t frameNum

The number of messages that need to be received.

flexcan_fd_frame_t *framefd

Point to the buffer of CAN Message to be received from Enhanced Rx FIFO.

2.15 FlexIO: FlexIO Driver

2.16 FlexIO Driver

void FLEXIO_GetDefaultConfig(*flexio_config_t* *userConfig)

Gets the default configuration to configure the FlexIO module. The configuration can be used directly to call the FLEXIO_Configure().

Example:

```
flexio_config_t config;
FLEXIO_GetDefaultConfig(&config);
```

Parameters

- userConfig – pointer to flexio_config_t structure

void FLEXIO_Init(FLEXIO_Type *base, const *flexio_config_t* *userConfig)

Configures the FlexIO with a FlexIO configuration. The configuration structure can be filled by the user or be set with default values by FLEXIO_GetDefaultConfig().

Example

```
flexio_config_t config = {
    .enableFlexio = true,
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false
};
FLEXIO_Configure(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- userConfig – pointer to flexio_config_t structure

void FLEXIO_Deinit(FLEXIO_Type *base)

Gates the FlexIO clock. Call this API to stop the FlexIO clock.

Note: After calling this API, call the FLEXIO_Init to use the FlexIO module.

Parameters

- base – FlexIO peripheral base address


```
uint32_t FLEXIO_GetInstance(FLEXIO_Type *base)
```

Get instance number for FLEXIO module.

Parameters

- base – FLEXIO peripheral base address.

```
void FLEXIO_Reset(FLEXIO_Type *base)
```

Resets the FlexIO module.

Parameters

- base – FlexIO peripheral base address

```
static inline void FLEXIO_Enable(FLEXIO_Type *base, bool enable)
```

Enables the FlexIO module operation.

Parameters

- base – FlexIO peripheral base address
- enable – true to enable, false to disable.

```
static inline uint32_t FLEXIO_ReadPinInput(FLEXIO_Type *base)
```

Reads the input data on each of the FlexIO pins.

Parameters

- base – FlexIO peripheral base address

Returns

FlexIO pin input data

```
static inline uint8_t FLEXIO_GetShifterState(FLEXIO_Type *base)
```

Gets the current state pointer for state mode use.

Parameters

- base – FlexIO peripheral base address

Returns

current State pointer

```
void FLEXIO_SetShifterConfig(FLEXIO_Type *base, uint8_t index, const flexio_shifter_config_t *shifterConfig)
```

Configures the shifter with the shifter configuration. The configuration structure covers both the SHIFTCTL and SHIFTCFG registers. To configure the shifter to the proper mode, select which timer controls the shifter to shift, whether to generate start bit/stop bit, and the polarity of start bit and stop bit.

Example

```
flexio_shifter_config_t config = {
    .timerSelect = 0,
    .timerPolarity = kFLEXIO_ShifterTimerPolarityOnPositive,
    .pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
    .pinPolarity = kFLEXIO_PinActiveLow,
    .shifterMode = kFLEXIO_ShifterModeTransmit,
    .inputSource = kFLEXIO_ShifterInputFromPin,
    .shifterStop = kFLEXIO_ShifterStopBitHigh,
    .shifterStart = kFLEXIO_ShifterStartBitLow
};
FLEXIO_SetShifterConfig(base, &config);
```

Parameters

- base – FlexIO peripheral base address

- index – Shifter index
- shifterConfig – Pointer to flexio_shifter_config_t structure

```
void FLEXIO_SetTimerConfig(FLEXIO_Type *base, uint8_t index, const flexio_timer_config_t *timerConfig)
```

Configures the timer with the timer configuration. The configuration structure covers both the TIMCTL and TIMCFG registers. To configure the timer to the proper mode, select trigger source for timer and the timer pin output and the timing for timer.

Example

```
flexio_timer_config_t config = {  
    .triggerSelect = FLEXIO_TIMER_TRIGGER_SEL_SHIFThSTAT(0),  
    .triggerPolarity = kFLEXIO_TimerTriggerPolarityActiveLow,  
    .triggerSource = kFLEXIO_TimerTriggerSourceInternal,  
    .pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,  
    .pinSelect = 0,  
    .pinPolarity = kFLEXIO_PinActiveHigh,  
    .timerMode = kFLEXIO_TimerModeDual8BitBaudBit,  
    .timerOutput = kFLEXIO_TimerOutputZeroNotAffectedByReset,  
    .timerDecrement = kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput,  
    .timerReset = kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput,  
    .timerDisable = kFLEXIO_TimerDisableOnTimerCompare,  
    .timerEnable = kFLEXIO_TimerEnableOnTriggerHigh,  
    .timerStop = kFLEXIO_TimerStopBitEnableOnTimerDisable,  
    .timerStart = kFLEXIO_TimerStartBitEnabled  
};  
FLEXIO_SetTimerConfig(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- index – Timer index
- timerConfig – Pointer to the flexio_timer_config_t structure

```
static inline void FLEXIO_SetClockMode(FLEXIO_Type *base, uint8_t index,  
                                       flexio_timer_decrement_source_t clocksource)
```

This function set the value of the prescaler on flexio channels.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.
- index – Timer index
- clocksource – Set clock value

```
static inline void FLEXIO_EnableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
```

Enables the shifter status interrupt. The interrupt generates when the corresponding SSF is set.

Note: For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « shifter index)

static inline void FLEXIO_DisableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
Disables the shifter status interrupt. The interrupt won't generate when the corresponding SSF is set.

Note: For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$

static inline void FLEXIO_EnableShifterErrorInterrupts(FLEXIO_Type *base, uint32_t mask)
Enables the shifter error interrupt. The interrupt generates when the corresponding SEF is set.

Note: For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$

static inline void FLEXIO_DisableShifterErrorInterrupts(FLEXIO_Type *base, uint32_t mask)
Disables the shifter error interrupt. The interrupt won't generate when the corresponding SEF is set.

Note: For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$

static inline void FLEXIO_EnableTimerStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
Enables the timer status interrupt. The interrupt generates when the corresponding SSF is set.

Note: For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by $(1 \ll \text{timer index})$

static inline void FLEXIO_DisableTimerStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
Disables the timer status interrupt. The interrupt won't generate when the corresponding SSF is set.

Note: For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by $(1 \ll \text{timer index})$

static inline uint32_t FLEXIO_GetShifterStatusFlags(FLEXIO_Type *base)

Gets the shifter status flags.

Parameters

- base – FlexIO peripheral base address

Returns

Shifter status flags

static inline void FLEXIO_ClearShifterStatusFlags(FLEXIO_Type *base, uint32_t mask)

Clears the shifter status flags.

Note: For clearing multiple shifter status flags, for example, two shifter status flags, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$

static inline uint32_t FLEXIO_GetShifterErrorFlags(FLEXIO_Type *base)

Gets the shifter error flags.

Parameters

- base – FlexIO peripheral base address

Returns

Shifter error flags

static inline void FLEXIO_ClearShifterErrorFlags(FLEXIO_Type *base, uint32_t mask)

Clears the shifter error flags.

Note: For clearing multiple shifter error flags, for example, two shifter error flags, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$

```
static inline uint32_t FLEXIO_GetTimerStatusFlags(FLEXIO_Type *base)
```

Gets the timer status flags.

Parameters

- base – FlexIO peripheral base address

Returns

Timer status flags

```
static inline void FLEXIO_ClearTimerStatusFlags(FLEXIO_Type *base, uint32_t mask)
```

Clears the timer status flags.

Note: For clearing multiple timer status flags, for example, two timer status flags, can calculate the mask by using $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by $(1 \ll \text{timer index})$

```
static inline void FLEXIO_EnableShifterStatusDMA(FLEXIO_Type *base, uint32_t mask, bool enable)
```

Enables/disables the shifter status DMA. The DMA request generates when the corresponding SSF is set.

Note: For multiple shifter status DMA enables, for example, calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$
- enable – True to enable, false to disable.

```
uint32_t FLEXIO_GetShifterBufferAddress(FLEXIO_Type *base, flexio_shifter_buffer_type_t type, uint8_t index)
```

Gets the shifter buffer address for the DMA transfer usage.

Parameters

- base – FlexIO peripheral base address
- type – Shifter type of flexio_shifter_buffer_type_t
- index – Shifter index

Returns

Corresponding shifter buffer index

```
status_t FLEXIO_RegisterHandleIRQ(void *base, void *handle, flexio_isr_t isr)
```

Registers the handle and the interrupt handler for the FlexIO-simulated peripheral.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.
- handle – Pointer to the handler for FlexIO simulated peripheral.
- isr – FlexIO simulated peripheral interrupt handler.

Return values

- `kStatus_Success` – Successfully create the handle.
- `kStatus_OutOfRange` – The FlexIO type/handle/ISR table out of range.

`status_t` FLEXIO_UnregisterHandleIRQ(void *base)

Unregisters the handle and the interrupt handler for the FlexIO-simulated peripheral.

Parameters

- `base` – Pointer to the FlexIO simulated peripheral type.

Return values

- `kStatus_Success` – Successfully create the handle.
- `kStatus_OutOfRange` – The FlexIO type/handle/ISR table out of range.

static inline void FLEXIO_ClearPortOutput(FLEXIO_Type *base, uint32_t mask)

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

Parameters

- `base` – FlexIO peripheral base address
- `mask` – FLEXIO pin number mask

static inline void FLEXIO_SetPortOutput(FLEXIO_Type *base, uint32_t mask)

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

Parameters

- `base` – FlexIO peripheral base address
- `mask` – FLEXIO pin number mask

static inline void FLEXIO_TogglePortOutput(FLEXIO_Type *base, uint32_t mask)

Reverses the current output logic of the multiple FLEXIO pins.

Parameters

- `base` – FlexIO peripheral base address
- `mask` – FLEXIO pin number mask

static inline void FLEXIO_PinWrite(FLEXIO_Type *base, uint32_t pin, uint8_t output)

Sets the output level of the FLEXIO pins to the logic 1 or 0.

Parameters

- `base` – FlexIO peripheral base address
- `pin` – FLEXIO pin number.
- `output` – FLEXIO pin output logic level.
 - 0: corresponding pin output low-logic level.
 - 1: corresponding pin output high-logic level.

static inline void FLEXIO_EnablePinOutput(FLEXIO_Type *base, uint32_t pin)

Enables the FLEXIO output pin function.

Parameters

- `base` – FlexIO peripheral base address
- `pin` – FLEXIO pin number.

```
static inline uint32_t FLEXIO_PinRead(FLEXIO_Type *base, uint32_t pin)
```

Reads the current input value of the FLEXIO pin.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

Return values

FLEXIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

```
static inline uint32_t FLEXIO_GetPinStatus(FLEXIO_Type *base, uint32_t pin)
```

Gets the FLEXIO input pin status.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

Return values

FLEXIO – port input status

- 0: corresponding pin input capture no status.
- 1: corresponding pin input capture rising or falling edge.

```
static inline void FLEXIO_SetPinLevel(FLEXIO_Type *base, uint8_t pin, bool level)
```

Sets the FLEXIO output pin level.

Parameters

- base – FlexIO peripheral base address
- pin – FlexIO pin number.
- level – FlexIO output pin level to set, can be either 0 or 1.

```
static inline bool FLEXIO_GetPinOverride(const FLEXIO_Type *const base, uint8_t pin)
```

Gets the enabled status of a FLEXIO output pin.

Parameters

- base – FlexIO peripheral base address
- pin – FlexIO pin number.

Return values

FlexIO – port enabled status

- 0: corresponding output pin is in disabled state.
- 1: corresponding output pin is in enabled state.

```
static inline void FLEXIO_ConfigPinOverride(FLEXIO_Type *base, uint8_t pin, bool enabled)
```

Enables or disables a FLEXIO output pin.

Parameters

- base – FlexIO peripheral base address
- pin – Flexio pin number.
- enabled – Enable or disable the FlexIO pin.

static inline void FLEXIO_ClearPortStatus(FLEXIO_Type *base, uint32_t mask)

Clears the multiple FLEXIO input pins status.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

FSL_FLEXIO_DRIVER_VERSION

FlexIO driver version.

enum __flexio_timer_trigger_polarity

Define time of timer trigger polarity.

Values:

enumerator kFLEXIO_TimerTriggerPolarityActiveHigh
Active high.

enumerator kFLEXIO_TimerTriggerPolarityActiveLow
Active low.

enum __flexio_timer_trigger_source

Define type of timer trigger source.

Values:

enumerator kFLEXIO_TimerTriggerSourceExternal
External trigger selected.

enumerator kFLEXIO_TimerTriggerSourceInternal
Internal trigger selected.

enum __flexio_pin_config

Define type of timer/shifter pin configuration.

Values:

enumerator kFLEXIO_PinConfigOutputDisabled
Pin output disabled.

enumerator kFLEXIO_PinConfigOpenDrainOrBidirection
Pin open drain or bidirectional output enable.

enumerator kFLEXIO_PinConfigBidirectionOutputData
Pin bidirectional output data.

enumerator kFLEXIO_PinConfigOutput
Pin output.

enum __flexio_pin_polarity

Definition of pin polarity.

Values:

enumerator kFLEXIO_PinActiveHigh
Active high.

enumerator kFLEXIO_PinActiveLow
Active low.

enum __flexio_timer_mode

Define type of timer work mode.

Values:

enumerator kFLEXIO__TimerModeDisabled
Timer Disabled.

enumerator kFLEXIO__TimerModeDual8BitBaudBit
Dual 8-bit counters baud/bit mode.

enumerator kFLEXIO__TimerModeDual8BitPWM
Dual 8-bit counters PWM mode.

enumerator kFLEXIO__TimerModeSingle16Bit
Single 16-bit counter mode.

enumerator kFLEXIO__TimerModeDual8BitPWMLow
Dual 8-bit counters PWM Low mode.

enum __flexio_timer_output
Define type of timer initial output or timer reset condition.
Values:

enumerator kFLEXIO__TimerOutputOneNotAffectedByReset
Logic one when enabled and is not affected by timer reset.

enumerator kFLEXIO__TimerOutputZeroNotAffectedByReset
Logic zero when enabled and is not affected by timer reset.

enumerator kFLEXIO__TimerOutputOneAffectedByReset
Logic one when enabled and on timer reset.

enumerator kFLEXIO__TimerOutputZeroAffectedByReset
Logic zero when enabled and on timer reset.

enum __flexio_timer_decrement_source
Define type of timer decrement.
Values:

enumerator kFLEXIO__TimerDecSrcOnFlexIOClockShiftTimerOutput
Decrement counter on FlexIO clock, Shift clock equals Timer output.

enumerator kFLEXIO__TimerDecSrcOnTriggerInputShiftTimerOutput
Decrement counter on Trigger input (both edges), Shift clock equals Timer output.

enumerator kFLEXIO__TimerDecSrcOnPinInputShiftPinInput
Decrement counter on Pin input (both edges), Shift clock equals Pin input.

enumerator kFLEXIO__TimerDecSrcOnTriggerInputShiftTriggerInput
Decrement counter on Trigger input (both edges), Shift clock equals Trigger input.

enum __flexio_timer_reset_condition
Define type of timer reset condition.
Values:

enumerator kFLEXIO__TimerResetNever
Timer never reset.

enumerator kFLEXIO__TimerResetOnTimerPinEqualToTimerOutput
Timer reset on Timer Pin equal to Timer Output.

enumerator kFLEXIO__TimerResetOnTimerTriggerEqualToTimerOutput
Timer reset on Timer Trigger equal to Timer Output.

enumerator kFLEXIO__TimerResetOnTimerPinRisingEdge

Timer reset on Timer Pin rising edge.

enumerator kFLEXIO__TimerResetOnTimerTriggerRisingEdge

Timer reset on Trigger rising edge.

enumerator kFLEXIO__TimerResetOnTimerTriggerBothEdge

Timer reset on Trigger rising or falling edge.

enum _flexio_timer_disable_condition

Define type of timer disable condition.

Values:

enumerator kFLEXIO__TimerDisableNever

Timer never disabled.

enumerator kFLEXIO__TimerDisableOnPreTimerDisable

Timer disabled on Timer N-1 disable.

enumerator kFLEXIO__TimerDisableOnTimerCompare

Timer disabled on Timer compare.

enumerator kFLEXIO__TimerDisableOnTimerCompareTriggerLow

Timer disabled on Timer compare and Trigger Low.

enumerator kFLEXIO__TimerDisableOnPinBothEdge

Timer disabled on Pin rising or falling edge.

enumerator kFLEXIO__TimerDisableOnPinBothEdgeTriggerHigh

Timer disabled on Pin rising or falling edge provided Trigger is high.

enumerator kFLEXIO__TimerDisableOnTriggerFallingEdge

Timer disabled on Trigger falling edge.

enum _flexio_timer_enable_condition

Define type of timer enable condition.

Values:

enumerator kFLEXIO__TimerEnabledAlways

Timer always enabled.

enumerator kFLEXIO__TimerEnableOnPrevTimerEnable

Timer enabled on Timer N-1 enable.

enumerator kFLEXIO__TimerEnableOnTriggerHigh

Timer enabled on Trigger high.

enumerator kFLEXIO__TimerEnableOnTriggerHighPinHigh

Timer enabled on Trigger high and Pin high.

enumerator kFLEXIO__TimerEnableOnPinRisingEdge

Timer enabled on Pin rising edge.

enumerator kFLEXIO__TimerEnableOnPinRisingEdgeTriggerHigh

Timer enabled on Pin rising edge and Trigger high.

enumerator kFLEXIO__TimerEnableOnTriggerRisingEdge

Timer enabled on Trigger rising edge.

enumerator kFLEXIO__TimerEnableOnTriggerBothEdge

Timer enabled on Trigger rising or falling edge.

enum _flexio_timer_stop_bit_condition

Define type of timer stop bit generate condition.

Values:

enumerator kFLEXIO_TimerStopBitDisabled

Stop bit disabled.

enumerator kFLEXIO_TimerStopBitEnableOnTimerCompare

Stop bit is enabled on timer compare.

enumerator kFLEXIO_TimerStopBitEnableOnTimerDisable

Stop bit is enabled on timer disable.

enumerator kFLEXIO_TimerStopBitEnableOnTimerCompareDisable

Stop bit is enabled on timer compare and timer disable.

enum _flexio_timer_start_bit_condition

Define type of timer start bit generate condition.

Values:

enumerator kFLEXIO_TimerStartBitDisabled

Start bit disabled.

enumerator kFLEXIO_TimerStartBitEnabled

Start bit enabled.

enum _flexio_timer_output_state

FlexIO as PWM channel output state.

Values:

enumerator kFLEXIO_PwmLow

The output state of PWM channel is low

enumerator kFLEXIO_PwmHigh

The output state of PWM channel is high

enum _flexio_shifter_timer_polarity

Define type of timer polarity for shifter control.

Values:

enumerator kFLEXIO_ShifterTimerPolarityOnPositive

Shift on positive edge of shift clock.

enumerator kFLEXIO_ShifterTimerPolarityOnNegative

Shift on negative edge of shift clock.

enum _flexio_shifter_mode

Define type of shifter working mode.

Values:

enumerator kFLEXIO_ShifterDisabled

Shifter is disabled.

enumerator kFLEXIO_ShifterModeReceive

Receive mode.

enumerator kFLEXIO_ShifterModeTransmit

Transmit mode.

enumerator kFLEXIO__ShifterModeMatchStore

Match store mode.

enumerator kFLEXIO__ShifterModeMatchContinuous

Match continuous mode.

enumerator kFLEXIO__ShifterModeState

SHIFTBUF contents are used for storing programmable state attributes.

enumerator kFLEXIO__ShifterModeLogic

SHIFTBUF contents are used for implementing programmable logic look up table.

enum _flexio_shifter_input_source

Define type of shifter input source.

Values:

enumerator kFLEXIO__ShifterInputFromPin

Shifter input from pin.

enumerator kFLEXIO__ShifterInputFromNextShifterOutput

Shifter input from Shifter N+1.

enum _flexio_shifter_stop_bit

Define of STOP bit configuration.

Values:

enumerator kFLEXIO__ShifterStopBitDisable

Disable shifter stop bit.

enumerator kFLEXIO__ShifterStopBitLow

Set shifter stop bit to logic low level.

enumerator kFLEXIO__ShifterStopBitHigh

Set shifter stop bit to logic high level.

enum _flexio_shifter_start_bit

Define type of START bit configuration.

Values:

enumerator kFLEXIO__ShifterStartBitDisabledLoadDataOnEnable

Disable shifter start bit, transmitter loads data on enable.

enumerator kFLEXIO__ShifterStartBitDisabledLoadDataOnShift

Disable shifter start bit, transmitter loads data on first shift.

enumerator kFLEXIO__ShifterStartBitLow

Set shifter start bit to logic low level.

enumerator kFLEXIO__ShifterStartBitHigh

Set shifter start bit to logic high level.

enum _flexio_shifter_buffer_type

Define FlexIO shifter buffer type.

Values:

enumerator kFLEXIO__ShifterBuffer

Shifter Buffer N Register.

enumerator kFLEXIO__ShifterBufferBitSwapped

Shifter Buffer N Bit Byte Swapped Register.

enumerator kFLEXIO__ShifterBufferByteSwapped
Shifter Buffer N Byte Swapped Register.

enumerator kFLEXIO__ShifterBufferBitByteSwapped
Shifter Buffer N Bit Swapped Register.

enumerator kFLEXIO__ShifterBufferNibbleByteSwapped
Shifter Buffer N Nibble Byte Swapped Register.

enumerator kFLEXIO__ShifterBufferHalfWordSwapped
Shifter Buffer N Half Word Swapped Register.

enumerator kFLEXIO__ShifterBufferNibbleSwapped
Shifter Buffer N Nibble Swapped Register.

enum _flexio_gpio_direction
FLEXIO gpio direction definition.
Values:

enumerator kFLEXIO__DigitalInput
Set current pin as digital input

enumerator kFLEXIO__DigitalOutput
Set current pin as digital output

enum _flexio_pin_input_config
FLEXIO gpio input config.
Values:

enumerator kFLEXIO__InputInterruptDisabled
Interrupt request is disabled.

enumerator kFLEXIO__InputInterruptEnable
Interrupt request is enable.

enumerator kFLEXIO__FlagRisingEdgeEnable
Input pin flag on rising edge.

enumerator kFLEXIO__FlagFallingEdgeEnable
Input pin flag on falling edge.

typedef enum _flexio_timer_trigger_polarity flexio_timer_trigger_polarity_t
Define time of timer trigger polarity.

typedef enum _flexio_timer_trigger_source flexio_timer_trigger_source_t
Define type of timer trigger source.

typedef enum _flexio_pin_config flexio_pin_config_t
Define type of timer/shifter pin configuration.

typedef enum _flexio_pin_polarity flexio_pin_polarity_t
Definition of pin polarity.

typedef enum _flexio_timer_mode flexio_timer_mode_t
Define type of timer work mode.

typedef enum _flexio_timer_output flexio_timer_output_t
Define type of timer initial output or timer reset condition.

typedef enum _flexio_timer_decrement_source flexio_timer_decrement_source_t
Define type of timer decrement.

typedef enum *_flexio_timer_reset_condition* flexio_timer_reset_condition_t

Define type of timer reset condition.

typedef enum *_flexio_timer_disable_condition* flexio_timer_disable_condition_t

Define type of timer disable condition.

typedef enum *_flexio_timer_enable_condition* flexio_timer_enable_condition_t

Define type of timer enable condition.

typedef enum *_flexio_timer_stop_bit_condition* flexio_timer_stop_bit_condition_t

Define type of timer stop bit generate condition.

typedef enum *_flexio_timer_start_bit_condition* flexio_timer_start_bit_condition_t

Define type of timer start bit generate condition.

typedef enum *_flexio_timer_output_state* flexio_timer_output_state_t

FlexIO as PWM channel output state.

typedef enum *_flexio_shifter_timer_polarity* flexio_shifter_timer_polarity_t

Define type of timer polarity for shifter control.

typedef enum *_flexio_shifter_mode* flexio_shifter_mode_t

Define type of shifter working mode.

typedef enum *_flexio_shifter_input_source* flexio_shifter_input_source_t

Define type of shifter input source.

typedef enum *_flexio_shifter_stop_bit* flexio_shifter_stop_bit_t

Define of STOP bit configuration.

typedef enum *_flexio_shifter_start_bit* flexio_shifter_start_bit_t

Define type of START bit configuration.

typedef enum *_flexio_shifter_buffer_type* flexio_shifter_buffer_type_t

Define FlexIO shifter buffer type.

typedef struct *_flexio_config* flexio_config_t

Define FlexIO user configuration structure.

typedef struct *_flexio_timer_config* flexio_timer_config_t

Define FlexIO timer configuration structure.

typedef struct *_flexio_shifter_config* flexio_shifter_config_t

Define FlexIO shifter configuration structure.

typedef enum *_flexio_gpio_direction* flexio_gpio_direction_t

FLEXIO gpio direction definition.

typedef enum *_flexio_pin_input_config* flexio_pin_input_config_t

FLEXIO gpio input config.

typedef struct *_flexio_gpio_config* flexio_gpio_config_t

The FLEXIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, use inputConfig param. If configured as an output pin, use outputLogic.

typedef void (*flexio_isr_t)(void *base, void *handle)

typedef for FlexIO simulated driver interrupt handler.

FLEXIO_Type *const s_flexioBases[]

Pointers to flexio bases for each instance.

```
const clock_ip_name_t s_flexioClocks[]
```

Pointers to flexio clocks for each instance.

```
void FLEXIO_SetPinConfig(FLEXIO_Type *base, uint32_t pin, flexio_gpio_config_t *config)
```

Configure a FLEXIO pin used by the board.

To Config the FLEXIO PIN, define a pin configuration, as either input or output, in the user file. Then, call the FLEXIO_SetPinConfig() function.

This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
flexio_gpio_config_t config =
{
    kFLEXIO_DigitalInput,
    0U,
    kFLEXIO_FlagRisingEdgeEnable | kFLEXIO_InputInterruptEnable,
}
Define a digital output pin configuration,
flexio_gpio_config_t config =
{
    kFLEXIO_DigitalOutput,
    0U,
    0U
}
```

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.
- config – FLEXIO pin configuration pointer.

```
FLEXIO_TIMER_TRIGGER_SEL_PININPUT(x)
```

Calculate FlexIO timer trigger.

```
FLEXIO_TIMER_TRIGGER_SEL_SHIFTrnSTAT(x)
```

```
FLEXIO_TIMER_TRIGGER_SEL_TIMn(x)
```

```
struct _flexio_config_
```

#include <fsl_flexio.h> Define FlexIO user configuration structure.

Public Members

```
bool enableFlexio
```

Enable/disable FlexIO module

```
bool enableInDoze
```

Enable/disable FlexIO operation in doze mode

```
bool enableInDebug
```

Enable/disable FlexIO operation in debug mode

```
bool enableFastAccess
```

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

```
struct _flexio_timer_config
```

#include <fsl_flexio.h> Define FlexIO timer configuration structure.

Public Members

`uint32_t triggerSelect`

The internal trigger selection number using MACROs.

`flexio_timer_trigger_polarity_t triggerPolarity`

Trigger Polarity.

`flexio_timer_trigger_source_t triggerSource`

Trigger Source, internal (see 'trgsel') or external.

`flexio_pin_config_t pinConfig`

Timer Pin Configuration.

`uint32_t pinSelect`

Timer Pin number Select.

`flexio_pin_polarity_t pinPolarity`

Timer Pin Polarity.

`flexio_timer_mode_t timerMode`

Timer work Mode.

`flexio_timer_output_t timerOutput`

Configures the initial state of the Timer Output and whether it is affected by the Timer reset.

`flexio_timer_decrement_source_t timerDecrement`

Configures the source of the Timer decrement and the source of the Shift clock.

`flexio_timer_reset_condition_t timerReset`

Configures the condition that causes the timer counter (and optionally the timer output) to be reset.

`flexio_timer_disable_condition_t timerDisable`

Configures the condition that causes the Timer to be disabled and stop decrementing.

`flexio_timer_enable_condition_t timerEnable`

Configures the condition that causes the Timer to be enabled and start decrementing.

`flexio_timer_stop_bit_condition_t timerStop`

Timer STOP Bit generation.

`flexio_timer_start_bit_condition_t timerStart`

Timer STRAT Bit generation.

`uint32_t timerCompare`

Value for Timer Compare N Register.

`struct _flexio_shifter_config`

`#include <fsl_flexio.h>` Define FlexIO shifter configuration structure.

Public Members

`uint32_t timerSelect`

Selects which Timer is used for controlling the logic/shift register and generating the Shift clock.

`flexio_shifter_timer_polarity_t timerPolarity`

Timer Polarity.

flexio_pin_config_t pinConfig
Shifter Pin Configuration.

uint32_t pinSelect
Shifter Pin number Select.

flexio_pin_polarity_t pinPolarity
Shifter Pin Polarity.

flexio_shifter_mode_t shifterMode
Configures the mode of the Shifter.

uint32_t parallelWidth
Configures the parallel width when using parallel mode.

flexio_shifter_input_source_t inputSource
Selects the input source for the shifter.

flexio_shifter_stop_bit_t shifterStop
Shifter STOP bit.

flexio_shifter_start_bit_t shifterStart
Shifter START bit.

struct __flexio_gpio_config

#include <fsl_flexio.h> The FLEXIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, use inputConfig param. If configured as an output pin, use outputLogic.

Public Members

flexio_gpio_direction_t pinDirection
FLEXIO pin direction, input or output

uint8_t outputLogic
Set a default output logic, which has no use in input

uint8_t inputConfig
Set an input config

2.17 FlexIO eDMA I2S Driver

```
void FLEXIO_I2S_TransferTxCreateHandleEDMA(FLEXIO_I2S_Type *base,
                                           flexio_i2s_edma_handle_t *handle,
                                           flexio_i2s_edma_callback_t callback, void
                                           *userData, edma_handle_t *dmaHandle)
```

Initializes the FlexIO I2S eDMA handle.

This function initializes the FlexIO I2S master DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer.

- callback – FlexIO I2S eDMA callback function called while finished a block.
- userData – User parameter for callback.
- dmaHandle – eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

```
void FLEXIO_I2S_TransferRxCreateHandleEDMA(FLEXIO_I2S_Type *base,  
                                           flexio_i2s_edma_handle_t *handle,  
                                           flexio_i2s_edma_callback_t callback, void  
                                           *userData, edma_handle_t *dmaHandle)
```

Initializes the FlexIO I2S Rx eDMA handle.

This function initializes the FlexIO I2S slave DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer.
- callback – FlexIO I2S eDMA callback function called while finished a block.
- userData – User parameter for callback.
- dmaHandle – eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

```
void FLEXIO_I2S_TransferSetFormatEDMA(FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t  
                                       *handle, flexio_i2s_format_t *format, uint32_t  
                                       srcClock_Hz)
```

Configures the FlexIO I2S Tx audio format.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to format.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer
- format – Pointer to FlexIO I2S audio data format structure.
- srcClock_Hz – FlexIO I2S clock source frequency in Hz, it should be 0 while in slave mode.

```
status_t FLEXIO_I2S_TransferSendEDMA(FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t  
                                       *handle, flexio_i2s_transfer_t *xfer)
```

Performs a non-blocking FlexIO I2S transfer using DMA.

Note: This interface returned immediately after transfer initiates. Users should call FLEXIO_I2S_GetTransferStatus to poll the transfer status and check whether the FlexIO I2S transfer is finished.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- `kStatus__Success` – Start a FlexIO I2S eDMA send successfully.
- `kStatus__InvalidArgument` – The input arguments is invalid.
- `kStatus__TxBusy` – FlexIO I2S is busy sending data.

`status_t` FLEXIO_I2S_TransferReceiveEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_transfer_t* *xfer)

Performs a non-blocking FlexIO I2S receive using eDMA.

Note: This interface returned immediately after transfer initiates. Users should call `FLEXIO_I2S_GetReceiveRemainingBytes` to poll the transfer status and check whether the FlexIO I2S transfer is finished.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- `kStatus__Success` – Start a FlexIO I2S eDMA receive successfully.
- `kStatus__InvalidArgument` – The input arguments is invalid.
- `kStatus__RxBusy` – FlexIO I2S is busy receiving data.

`void` FLEXIO_I2S_TransferAbortSendEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle)

Aborts a FlexIO I2S transfer using eDMA.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.

`void` FLEXIO_I2S_TransferAbortReceiveEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle)

Aborts a FlexIO I2S receive using eDMA.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.

`status_t` FLEXIO_I2S_TransferGetSendCountEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *size_t* *count)

Gets the remaining bytes to be sent.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- count – Bytes sent.

Return values

- `kStatus__Success` – Succeed get the transfer count.

- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

`status_t` FLEXIO_I2S_TransferGetReceiveCountEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *size_t* *count)

Get the remaining bytes to be received.

Parameters

- `base` – FlexIO I2S peripheral base address.
- `handle` – FlexIO I2S DMA handle pointer.
- `count` – Bytes received.

Return values

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

FSL_FLEXIO_I2S_EDMA_DRIVER_VERSION

FlexIO I2S EDMA driver version 2.1.9.

`typedef struct _flexio_i2s_edma_handle` *flexio_i2s_edma_handle_t*

`typedef void (*flexio_i2s_edma_callback_t)(FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t *handle, status_t status, void *userData)`

FlexIO I2S eDMA transfer callback function for finish and error.

`struct _flexio_i2s_edma_handle`

`#include <fsl_flexio_i2s_edma.h>` FlexIO I2S DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *dmaHandle

DMA handler for FlexIO I2S send

uint8_t bytesPerFrame

Bytes in a frame

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint32_t state

Internal state for FlexIO I2S eDMA transfer

flexio_i2s_edma_callback_t callback

Callback for users while transfer finish or error occurred

void *userData

User callback parameter

edma_tcd_t tcd[(4U) + 1U]

TCD pool for eDMA transfer.

flexio_i2s_transfer_t queue[(4U)]

Transfer queue storing queued transfer.

size_t transferSize[(4U)]

Data bytes need to transfer

volatile uint8_t queueUser

Index for user to queue transfer.

volatile uint8_t queueDriver

Index for driver to get the transfer data and size

2.18 FlexIO eDMA SPI Driver

```
status_t FLEXIO_SPI_MasterTransferCreateHandleEDMA(FLEXIO_SPI_Type *base,  
                                                    flexio_spi_master_edma_handle_t  
                                                    *handle,  
                                                    flexio_spi_master_edma_transfer_callback_t  
                                                    callback, void *userData,  
                                                    edma_handle_t *txHandle,  
                                                    edma_handle_t *rxHandle)
```

Initializes the FlexIO SPI master eDMA handle.

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

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state.
- callback – SPI callback, NULL means no callback.
- userData – callback function parameter.
- txHandle – User requested eDMA handle for FlexIO SPI RX eDMA transfer.
- rxHandle – User requested eDMA handle for FlexIO SPI TX eDMA transfer.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO SPI eDMA type/handle table out of range.

```
status_t FLEXIO_SPI_MasterTransferEDMA(FLEXIO_SPI_Type *base,  
                                         flexio_spi_master_edma_handle_t *handle,  
                                         flexio_spi_transfer_t *xfer)
```

Performs a non-blocking FlexIO SPI transfer using eDMA.

Note: This interface returns immediately after transfer initiates. Call FLEXIO_SPI_MasterGetTransferCountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state.
- xfer – Pointer to FlexIO SPI transfer structure.

Return values

- `kStatus__Success` – Successfully start a transfer.
- `kStatus__InvalidArgument` – Input argument is invalid.
- `kStatus_FLEXIO_SPI_Busy` – FlexIO SPI is not idle, is running another transfer.

```
void FLEXIO_SPI_MasterTransferAbortEDMA(FLEXIO_SPI_Type *base,  
                                         flexio_spi_master_edma_handle_t *handle)
```

Aborts a FlexIO SPI transfer using eDMA.

Parameters

- `base` – Pointer to `FLEXIO_SPI_Type` structure.
- `handle` – FlexIO SPI eDMA handle pointer.

```
status_t FLEXIO_SPI_MasterTransferGetCountEDMA(FLEXIO_SPI_Type *base,  
                                                flexio_spi_master_edma_handle_t *handle,  
                                                size_t *count)
```

Gets the number of bytes transferred so far using FlexIO SPI master eDMA.

Parameters

- `base` – Pointer to `FLEXIO_SPI_Type` structure.
- `handle` – FlexIO SPI eDMA handle pointer.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

```
static inline void FLEXIO_SPI_SlaveTransferCreateHandleEDMA(FLEXIO_SPI_Type *base,  
                                                           flexio_spi_slave_edma_handle_t  
                                                           *handle,  
                                                           flexio_spi_slave_edma_transfer_callback_t  
                                                           callback, void *userData,  
                                                           edma_handle_t *txHandle,  
                                                           edma_handle_t *rxHandle)
```

Initializes the FlexIO SPI slave eDMA handle.

This function initializes the FlexIO SPI slave eDMA handle.

Parameters

- `base` – Pointer to `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to `flexio_spi_slave_edma_handle_t` structure to store the transfer state.
- `callback` – SPI callback, NULL means no callback.
- `userData` – callback function parameter.
- `txHandle` – User requested eDMA handle for FlexIO SPI TX eDMA transfer.
- `rxHandle` – User requested eDMA handle for FlexIO SPI RX eDMA transfer.

```
status_t FLEXIO_SPI_SlaveTransferEDMA(FLEXIO_SPI_Type *base,  
                                       flexio_spi_slave_edma_handle_t *handle,  
                                       flexio_spi_transfer_t *xfer)
```

Performs a non-blocking FlexIO SPI transfer using eDMA.

Note: This interface returns immediately after transfer initiates. Call `FLEXIO_SPI_SlaveGetTransferCountEDMA` to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

- `base` – Pointer to `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to `flexio_spi_slave_edma_handle_t` structure to store the transfer state.
- `xfer` – Pointer to FlexIO SPI transfer structure.

Return values

- `kStatus_Success` – Successfully start a transfer.
- `kStatus_InvalidArgument` – Input argument is invalid.
- `kStatus_FLEXIO_SPI_Busy` – FlexIO SPI is not idle, is running another transfer.

```
static inline void FLEXIO_SPI_SlaveTransferAbortEDMA(FLEXIO_SPI_Type *base,
                                                    flexio_spi_slave_edma_handle_t
                                                    *handle)
```

Aborts a FlexIO SPI transfer using eDMA.

Parameters

- `base` – Pointer to `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to `flexio_spi_slave_edma_handle_t` structure to store the transfer state.

```
static inline status_t FLEXIO_SPI_SlaveTransferGetCountEDMA(FLEXIO_SPI_Type *base,
                                                           flexio_spi_slave_edma_handle_t
                                                           *handle, size_t *count)
```

Gets the number of bytes transferred so far using FlexIO SPI slave eDMA.

Parameters

- `base` – Pointer to `FLEXIO_SPI_Type` structure.
- `handle` – FlexIO SPI eDMA handle pointer.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

`FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION`

FlexIO SPI EDMA driver version.

```
typedef struct flexio_spi_master_edma_handle flexio_spi_master_edma_handle_t
typedef for flexio_spi_master_edma_handle_t in advance.
```

```
typedef flexio_spi_master_edma_handle_t flexio_spi_slave_edma_handle_t
Slave handle is the same with master handle.
```

```
typedef void (*flexio_spi_master_edma_transfer_callback_t)(FLEXIO_SPI_Type *base,
                                                           flexio_spi_master_edma_handle_t *handle, status_t status, void *userData)
```

FlexIO SPI master callback for finished transmit.

```
typedef void (*flexio_spi_slave_edma_transfer_callback_t)(FLEXIO_SPI_Type *base,
                                                           flexio_spi_slave_edma_handle_t *handle, status_t status, void *userData)
```

FlexIO SPI slave callback for finished transmit.

```
struct flexio_spi_master_edma_handle
```

`#include <fsl_flexio_spi_edma.h>` FlexIO SPI eDMA transfer handle, users should not touch the content of the handle.

Public Members

`size_t` transferSize
Total bytes to be transferred.

`uint8_t` nbytes
eDMA minor byte transfer count initially configured.

`bool` txInProgress
Send transfer in progress

`bool` rxInProgress
Receive transfer in progress

`edma_handle_t` *txHandle
DMA handler for SPI send

`edma_handle_t` *rxHandle
DMA handler for SPI receive

`flexio_spi_master_edma_transfer_callback_t` callback
Callback for SPI DMA transfer

`void` *userData
User Data for SPI DMA callback

2.19 FlexIO eDMA UART Driver

`status_t` FLEXIO_UART_TransferCreateHandleEDMA(*FLEXIO_UART_Type* *base,
 flexio_uart_edma_handle_t *handle,
 flexio_uart_edma_transfer_callback_t
 callback, void *userData, *edma_handle_t*
 *txEdmaHandle, *edma_handle_t*
 *rxEdmaHandle)

Initializes the UART handle which is used in transactional functions.

Parameters

- base – Pointer to *FLEXIO_UART_Type*.
- handle – Pointer to *flexio_uart_edma_handle_t* structure.
- callback – The callback function.
- userData – The parameter of the callback function.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.

Return values

- `kStatus_Success` – Successfully create the handle.
- `kStatus_OutOfRange` – The FlexIO SPI eDMA type/handle table out of range.

`status_t` FLEXIO_UART_TransferSendEDMA(*FLEXIO_UART_Type* *base,
 flexio_uart_edma_handle_t *handle,
 flexio_uart_transfer_t *xfer)

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent out, the send callback function is called.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – UART handle pointer.
- xfer – UART eDMA transfer structure, see flexio_uart_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXIO_UART_TxBusy – Previous transfer on going.

```
status_t FLEXIO_UART_TransferReceiveEDMA(FLEXIO_UART_Type *base,
                                         flexio_uart_edma_handle_t *handle,
                                         flexio_uart_transfer_t *xfer)
```

Receives data using eDMA.

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- xfer – UART eDMA transfer structure, see flexio_uart_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_UART_RxBusy – Previous transfer on going.

```
void FLEXIO_UART_TransferAbortSendEDMA(FLEXIO_UART_Type *base,
                                       flexio_uart_edma_handle_t *handle)
```

Aborts the sent data which using eDMA.

This function aborts sent data which using eDMA.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure

```
void FLEXIO_UART_TransferAbortReceiveEDMA(FLEXIO_UART_Type *base,
                                          flexio_uart_edma_handle_t *handle)
```

Aborts the receive data which using eDMA.

This function aborts the receive data which using eDMA.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure

```
status_t FLEXIO_UART_TransferGetSendCountEDMA(FLEXIO_UART_Type *base,
                                              flexio_uart_edma_handle_t *handle,
                                              size_t *count)
```

Gets the number of bytes sent out.

This function gets the number of bytes sent out.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- count – Number of bytes sent so far by the non-blocking transaction.

Return values

- `kStatus_NoTransferInProgress` – transfer has finished or no transfer in progress.
- `kStatus_Success` – Successfully return the count.

`status_t FLEXIO_UART_TransferGetReceiveCountEDMA(FLEXIO_UART_Type *base,
flexio_uart_edma_handle_t *handle,
size_t *count)`

Gets the number of bytes received.

This function gets the number of bytes received.

Parameters

- `base` – Pointer to `FLEXIO_UART_Type`
- `handle` – Pointer to `flexio_uart_edma_handle_t` structure
- `count` – Number of bytes received so far by the non-blocking transaction.

Return values

- `kStatus_NoTransferInProgress` – transfer has finished or no transfer in progress.
- `kStatus_Success` – Successfully return the count.

`FSL_FLEXIO_UART_EDMA_DRIVER_VERSION`

FlexIO UART EDMA driver version.

`typedef struct flexio_uart_edma_handle flexio_uart_edma_handle_t`

`typedef void (*flexio_uart_edma_transfer_callback_t)(FLEXIO_UART_Type *base,
flexio_uart_edma_handle_t *handle, status_t status, void *userData)`

UART transfer callback function.

`struct flexio_uart_edma_handle`

`#include <fsl_flexio_uart_edma.h>` UART eDMA handle.

Public Members

`flexio_uart_edma_transfer_callback_t` callback

Callback function.

`void *userData`

UART callback function parameter.

`size_t txDataSizeAll`

Total bytes to be sent.

`size_t rxDataSizeAll`

Total bytes to be received.

`edma_handle_t *txEdmaHandle`

The eDMA TX channel used.

`edma_handle_t *rxEdmaHandle`

The eDMA RX channel used.

`uint8_t nbytes`

eDMA minor byte transfer count initially configured.

volatile uint8_t txState
TX transfer state.

volatile uint8_t rxState
RX transfer state

2.20 FlexIO I2C Master Driver

status_t FLEXIO_I2C_CheckForBusyBus(*FLEXIO_I2C_Type* *base)

Make sure the bus isn't already pulled down.

Check the FLEXIO pin status to see whether either of SDA and SCL pin is pulled down.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure..

Return values

- kStatus_Success –
- kStatus_FLEXIO_I2C_Busy –

status_t FLEXIO_I2C_MasterInit(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO I2C hardware configuration.

Example

```
FLEXIO_I2C_Type base = {
    .flexioBase = FLEXIO,
    .SDAPinIndex = 0,
    .SCLPinIndex = 1,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_i2c_master_config_t config = {
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .baudRate_Bps = 100000
};
FLEXIO_I2C_MasterInit(base, &config, srcClock_Hz);
```

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- masterConfig – Pointer to flexio_i2c_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Initialization successful
- kStatus_InvalidArgument – The source clock exceed upper range limitation

void FLEXIO_I2C_MasterDeinit(*FLEXIO_I2C_Type* *base)

De-initializes the FlexIO I2C master peripheral. Calling this API Resets the FlexIO I2C master shifer and timer config, module can't work unless the FLEXIO_I2C_MasterInit is called.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterGetDefaultConfig(*flexio_i2c_master_config_t* *masterConfig)

Gets the default configuration to configure the FlexIO module. The configuration can be used directly for calling the FLEXIO_I2C_MasterInit().

Example:

```
flexio_i2c_master_config_t config;  
FLEXIO_I2C_MasterGetDefaultConfig(&config);
```

Parameters

- masterConfig – Pointer to flexio_i2c_master_config_t structure.

static inline void FLEXIO_I2C_MasterEnable(*FLEXIO_I2C_Type* *base, bool enable)

Enables/disables the FlexIO module operation.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- enable – Pass true to enable module, false does not have any effect.

uint32_t FLEXIO_I2C_MasterGetStatusFlags(*FLEXIO_I2C_Type* *base)

Gets the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure

Returns

Status flag, use status flag to AND `_flexio_i2c_master_status_flags` can get the related status.

void FLEXIO_I2C_MasterClearStatusFlags(*FLEXIO_I2C_Type* *base, uint32_t mask)

Clears the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Status flag. The parameter can be any combination of the following values:
 - kFLEXIO_I2C_RxFullFlag
 - kFLEXIO_I2C_ReceiveNakFlag

void FLEXIO_I2C_MasterEnableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)

Enables the FlexIO i2c master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source. Currently only one interrupt request source:
 - kFLEXIO_I2C_TransferCompleteInterruptEnable

void FLEXIO_I2C_MasterDisableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)

Disables the FlexIO I2C master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source.

```
void FLEXIO_I2C_MasterSetBaudRate(FLEXIO_I2C_Type *base, uint32_t baudRate_Bps,  
                                uint32_t srcClock_Hz)
```

Sets the FlexIO I2C master transfer baudrate.

Parameters

- base – Pointer to *FLEXIO_I2C_Type* structure
- baudRate_Bps – the baud rate value in HZ
- srcClock_Hz – source clock in HZ

```
void FLEXIO_I2C_MasterStart(FLEXIO_I2C_Type *base, uint8_t address, flexio_i2c_direction_t  
                           direction)
```

Sends START + 7-bit address to the bus.

Note: This API should be called when the transfer configuration is ready to send a START signal and 7-bit address to the bus. This is a non-blocking API, which returns directly after the address is put into the data register but the address transfer is not finished on the bus. Ensure that the *kFLEXIO_I2C_RxFullFlag* status is asserted before calling this API.

Parameters

- base – Pointer to *FLEXIO_I2C_Type* structure.
- address – 7-bit address.
- direction – transfer direction. This parameter is one of the values in *flexio_i2c_direction_t*:
 - *kFLEXIO_I2C_Write*: Transmit
 - *kFLEXIO_I2C_Read*: Receive

```
void FLEXIO_I2C_MasterStop(FLEXIO_I2C_Type *base)
```

Sends the stop signal on the bus.

Parameters

- base – Pointer to *FLEXIO_I2C_Type* structure.

```
void FLEXIO_I2C_MasterRepeatedStart(FLEXIO_I2C_Type *base)
```

Sends the repeated start signal on the bus.

Parameters

- base – Pointer to *FLEXIO_I2C_Type* structure.

```
void FLEXIO_I2C_MasterAbortStop(FLEXIO_I2C_Type *base)
```

Sends the stop signal when transfer is still on-going.

Parameters

- base – Pointer to *FLEXIO_I2C_Type* structure.

```
void FLEXIO_I2C_MasterEnableAck(FLEXIO_I2C_Type *base, bool enable)
```

Configures the sent ACK/NAK for the following byte.

Parameters

- base – Pointer to *FLEXIO_I2C_Type* structure.
- enable – True to configure send ACK, false configure to send NAK.

status_t FLEXIO_I2C_MasterSetTransferCount(*FLEXIO_I2C_Type* *base, uint16_t count)

Sets the number of bytes to be transferred from a start signal to a stop signal.

Note: Call this API before a transfer begins because the timer generates a number of clocks according to the number of bytes that need to be transferred.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- count – Number of bytes need to be transferred from a start signal to a re-start/stop signal

Return values

- kStatus_Success – Successfully configured the count.
- kStatus_InvalidArgument – Input argument is invalid.

static inline void FLEXIO_I2C_MasterWriteByte(*FLEXIO_I2C_Type* *base, uint32_t data)

Writes one byte of data to the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- data – a byte of data.

static inline uint8_t FLEXIO_I2C_MasterReadByte(*FLEXIO_I2C_Type* *base)

Reads one byte of data from the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the data is ready in the register.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

Returns

data byte read.

status_t FLEXIO_I2C_MasterWriteBlocking(*FLEXIO_I2C_Type* *base, const uint8_t *txBuff, uint8_t txSize)

Sends a buffer of data in bytes.

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

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- txBuff – The data bytes to send.
- txSize – The number of data bytes to send.

Return values

- kStatus_Success – Successfully write data.
- kStatus_FLEXIO_I2C_Nak – Receive NAK during writing data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterReadBlocking(*FLEXIO_I2C_Type* *base, uint8_t *rxBuff, uint8_t rxSize)

Receives a buffer of bytes.

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

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- rxBuff – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus_Success – Successfully read data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterTransferBlocking(*FLEXIO_I2C_Type* *base,
flexio_i2c_master_transfer_t *xfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to receiving NAK.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.
- xfer – pointer to flexio_i2c_master_transfer_t structure.

Returns

status of status_t.

status_t FLEXIO_I2C_MasterTransferCreateHandle(*FLEXIO_I2C_Type* *base,
flexio_i2c_master_handle_t *handle,
flexio_i2c_master_transfer_callback_t
callback, void *userData)

Initializes the I2C handle which is used in transactional functions.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- handle – Pointer to flexio_i2c_master_handle_t structure to store the transfer state.
- callback – Pointer to user callback function.
- userData – User param passed to the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/isr table out of range.

```
status_t FLEXIO_I2C_MasterTransferNonBlocking(FLEXIO_I2C_Type *base,  
                                              flexio_i2c_master_handle_t *handle,  
                                              flexio_i2c_master_transfer_t *xfer)
```

Performs a master interrupt non-blocking transfer on the I2C bus.

Note: The API returns immediately after the transfer initiates. Call `FLEXIO_I2C_MasterTransferGetCount` to poll the transfer status to check whether the transfer is finished. If the return status is not `kStatus_FLEXIO_I2C_Busy`, the transfer is finished.

Parameters

- base – Pointer to `FLEXIO_I2C_Type` structure
- handle – Pointer to `flexio_i2c_master_handle_t` structure which stores the transfer state
- xfer – pointer to `flexio_i2c_master_transfer_t` structure

Return values

- `kStatus_Success` – Successfully start a transfer.
- `kStatus_FLEXIO_I2C_Busy` – FlexIO I2C is not idle, is running another transfer.

```
status_t FLEXIO_I2C_MasterTransferGetCount(FLEXIO_I2C_Type *base,  
                                           flexio_i2c_master_handle_t *handle, size_t  
                                           *count)
```

Gets the master transfer status during a interrupt non-blocking transfer.

Parameters

- base – Pointer to `FLEXIO_I2C_Type` structure.
- handle – Pointer to `flexio_i2c_master_handle_t` structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_InvalidArgument` – count is Invalid.
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.
- `kStatus_Success` – Successfully return the count.

```
void FLEXIO_I2C_MasterTransferAbort(FLEXIO_I2C_Type *base, flexio_i2c_master_handle_t  
                                   *handle)
```

Aborts an interrupt non-blocking transfer early.

Note: This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – Pointer to `FLEXIO_I2C_Type` structure
- handle – Pointer to `flexio_i2c_master_handle_t` structure which stores the transfer state

void FLEXIO_I2C_MasterTransferHandleIRQ(void *i2cType, void *i2cHandle)

Master interrupt handler.

Parameters

- i2cType – Pointer to FLEXIO_I2C_Type structure
- i2cHandle – Pointer to flexio_i2c_master_transfer_t structure

FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION

FlexIO I2C transfer status.

Values:

enumerator kStatus_FLEXIO_I2C_Busy

I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Idle

I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Nak

NAK received during transfer.

enumerator kStatus_FLEXIO_I2C_Timeout

Timeout polling status flags.

enum _flexio_i2c_master_interrupt

Define FlexIO I2C master interrupt mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyInterruptEnable

Tx buffer empty interrupt enable.

enumerator kFLEXIO_I2C_RxFullInterruptEnable

Rx buffer full interrupt enable.

enum _flexio_i2c_master_status_flags

Define FlexIO I2C master status mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyFlag

Tx shifter empty flag.

enumerator kFLEXIO_I2C_RxFullFlag

Rx shifter full/Transfer complete flag.

enumerator kFLEXIO_I2C_ReceiveNakFlag

Receive NAK flag.

enum _flexio_i2c_direction

Direction of master transfer.

Values:

enumerator kFLEXIO_I2C_Write

Master send to slave.

enumerator kFLEXIO_I2C_Read

Master receive from slave.

typedef enum _flexio_i2c_direction flexio_i2c_direction_t

Direction of master transfer.

```
typedef struct _flexio_i2c_type FLEXIO_I2C_Type
    Define FlexIO I2C master access structure typedef.
typedef struct _flexio_i2c_master_config flexio_i2c_master_config_t
    Define FlexIO I2C master user configuration structure.
typedef struct _flexio_i2c_master_transfer flexio_i2c_master_transfer_t
    Define FlexIO I2C master transfer structure.
typedef struct _flexio_i2c_master_handle flexio_i2c_master_handle_t
    FlexIO I2C master handle typedef.
typedef void (*flexio_i2c_master_transfer_callback_t)(FLEXIO_I2C_Type *base,
flexio_i2c_master_handle_t *handle, status_t status, void *userData)
    FlexIO I2C master transfer callback typedef.
I2C_RETRY_TIMES
    Retry times for waiting flag.
struct _flexio_i2c_type
    #include <fsl_flexio_i2c_master.h> Define FlexIO I2C master access structure typedef.
```

Public Members

```
FLEXIO_Type *flexioBase
    FlexIO base pointer.
uint8_t SDAPinIndex
    Pin select for I2C SDA.
uint8_t SCLPinIndex
    Pin select for I2C SCL.
uint8_t shifterIndex[2]
    Shifter index used in FlexIO I2C.
uint8_t timerIndex[3]
    Timer index used in FlexIO I2C.
uint32_t baudrate
    Master transfer baudrate, used to calculate delay time.
struct _flexio_i2c_master_config
    #include <fsl_flexio_i2c_master.h> Define FlexIO I2C master user configuration structure.
```

Public Members

```
bool enableMaster
    Enables the FlexIO I2C peripheral at initialization time.
bool enableInDoze
    Enable/disable FlexIO operation in doze mode.
bool enableInDebug
    Enable/disable FlexIO operation in debug mode.
bool enableFastAccess
    Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to
    be at least twice the frequency of the bus clock.
```

uint32_t baudRate_Bps

Baud rate in Bps.

struct _flexio_i2c_master_transfer

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master transfer structure.

Public Members

uint32_t flags

Transfer flag which controls the transfer, reserved for FlexIO I2C.

uint8_t slaveAddress

7-bit slave address.

flexio_i2c_direction_t direction

Transfer direction, read or write.

uint32_t subaddress

Sub address. Transferred MSB first.

uint8_t subaddressSize

Size of sub address.

uint8_t volatile *data

Transfer buffer.

volatile size_t dataSize

Transfer size.

struct _flexio_i2c_master_handle

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master handle structure.

Public Members

flexio_i2c_master_transfer_t transfer

FlexIO I2C master transfer copy.

size_t transferSize

Total bytes to be transferred.

uint8_t state

Transfer state maintained during transfer.

flexio_i2c_master_transfer_callback_t completionCallback

Callback function called at transfer event. Callback function called at transfer event.

void *userData

Callback parameter passed to callback function.

bool needRestart

Whether master needs to send re-start signal.

2.21 FlexIO I2S Driver

void FLEXIO_I2S_Init(*FLEXIO_I2S_Type* *base, const *flexio_i2s_config_t* *config)

Initializes the FlexIO I2S.

This API configures FlexIO pins and shifter to I2S and configures the FlexIO I2S with a configuration structure. The configuration structure can be filled by the user, or be set with default values by FLEXIO_I2S_GetDefaultConfig().

Note: This API should be called at the beginning of the application to use the FlexIO I2S driver. Otherwise, any access to the FlexIO I2S module can cause hard fault because the clock is not enabled.

Parameters

- base – FlexIO I2S base pointer
- config – FlexIO I2S configure structure.

void FLEXIO_I2S_GetDefaultConfig(*flexio_i2s_config_t* *config)

Sets the FlexIO I2S configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in FLEXIO_I2S_Init(). Users may use the initialized structure unchanged in FLEXIO_I2S_Init() or modify some fields of the structure before calling FLEXIO_I2S_Init().

Parameters

- config – pointer to master configuration structure

void FLEXIO_I2S_Deinit(*FLEXIO_I2S_Type* *base)

De-initializes the FlexIO I2S.

Calling this API resets the FlexIO I2S shifter and timer config. After calling this API, call the FLEXIO_I2S_Init to use the FlexIO I2S module.

Parameters

- base – FlexIO I2S base pointer

static inline void FLEXIO_I2S_Enable(*FLEXIO_I2S_Type* *base, bool enable)

Enables/disables the FlexIO I2S module operation.

Parameters

- base – Pointer to FLEXIO_I2S_Type
- enable – True to enable, false dose not have any effect.

uint32_t FLEXIO_I2S_GetStatusFlags(*FLEXIO_I2S_Type* *base)

Gets the FlexIO I2S status flags.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure

Returns

Status flag, which are ORed by the enumerators in the `_flexio_i2s_status_flags`.

void FLEXIO_I2S_EnableInterrupts(*FLEXIO_I2S_Type* *base, uint32_t mask)

Enables the FlexIO I2S interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure
- mask – interrupt source

```
void FLEXIO_I2S_DisableInterrupts(FLEXIO_I2S_Type *base, uint32_t mask)
```

Disables the FlexIO I2S interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – pointer to *FLEXIO_I2S_Type* structure
- mask – interrupt source

```
static inline void FLEXIO_I2S_TxEnableDMA(FLEXIO_I2S_Type *base, bool enable)
```

Enables/disables the FlexIO I2S Tx DMA requests.

Parameters

- base – FlexIO I2S base pointer
- enable – True means enable DMA, false means disable DMA.

```
static inline void FLEXIO_I2S_RxEnableDMA(FLEXIO_I2S_Type *base, bool enable)
```

Enables/disables the FlexIO I2S Rx DMA requests.

Parameters

- base – FlexIO I2S base pointer
- enable – True means enable DMA, false means disable DMA.

```
static inline uint32_t FLEXIO_I2S_TxGetDataRegisterAddress(FLEXIO_I2S_Type *base)
```

Gets the FlexIO I2S send data register address.

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure

Returns

FlexIO i2s send data register address.

```
static inline uint32_t FLEXIO_I2S_RxGetDataRegisterAddress(FLEXIO_I2S_Type *base)
```

Gets the FlexIO I2S receive data register address.

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure

Returns

FlexIO i2s receive data register address.

```
void FLEXIO_I2S_MasterSetFormat(FLEXIO_I2S_Type *base, flexio_i2s_format_t *format,  
                                uint32_t srcClock_Hz)
```

Configures the FlexIO I2S audio format in master mode.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure
- format – Pointer to FlexIO I2S audio data format structure.
- srcClock_Hz – I2S master clock source frequency in Hz.

```
void FLEXIO_I2S_SlaveSetFormat(FLEXIO_I2S_Type *base, flexio_i2s_format_t *format)
```

Configures the FlexIO I2S audio format in slave mode.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure
- format – Pointer to FlexIO I2S audio data format structure.

```
status_t FLEXIO_I2S_WriteBlocking(FLEXIO_I2S_Type *base, uint8_t bitWidth, uint8_t *txData,  
                                size_t size)
```

Sends data using a blocking method.

Note: This function blocks via polling until data is ready to be sent.

Parameters

- base – FlexIO I2S base pointer.
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- txData – Pointer to the data to be written.
- size – Bytes to be written.

Return values

- kStatus_Success – Successfully write data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

```
static inline void FLEXIO_I2S_WriteData(FLEXIO_I2S_Type *base, uint8_t bitWidth, uint32_t  
                                       data)
```

Writes data into a data register.

Parameters

- base – FlexIO I2S base pointer.
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- data – Data to be written.

```
status_t FLEXIO_I2S_ReadBlocking(FLEXIO_I2S_Type *base, uint8_t bitWidth, uint8_t *rxData,  
                                size_t size)
```

Receives a piece of data using a blocking method.

Note: This function blocks via polling until data is ready to be sent.

Parameters

- base – FlexIO I2S base pointer
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- rxData – Pointer to the data to be read.
- size – Bytes to be read.

Return values

- kStatus_Success – Successfully read data.

- `kStatus_FLEXIO_I2C_Timeout` – Timeout polling status flags.

`static inline uint32_t FLEXIO_I2S_ReadData(FLEXIO_I2S_Type *base)`

Reads a data from the data register.

Parameters

- `base` – FlexIO I2S base pointer

Returns

Data read from data register.

`void FLEXIO_I2S_TransferTxCreateHandle(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, flexio_i2s_callback_t callback, void *userData)`

Initializes the FlexIO I2S handle.

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

Parameters

- `base` – Pointer to `FLEXIO_I2S_Type` structure
- `handle` – Pointer to `flexio_i2s_handle_t` structure to store the transfer state.
- `callback` – FlexIO I2S callback function, which is called while finished a block.
- `userData` – User parameter for the FlexIO I2S callback.

`void FLEXIO_I2S_TransferSetFormat(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, flexio_i2s_format_t *format, uint32_t srcClock_Hz)`

Configures the FlexIO I2S audio format.

Audio format can be changed at run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- `base` – Pointer to `FLEXIO_I2S_Type` structure.
- `handle` – FlexIO I2S handle pointer.
- `format` – Pointer to audio data format structure.
- `srcClock_Hz` – FlexIO I2S bit clock source frequency in Hz. This parameter should be 0 while in slave mode.

`void FLEXIO_I2S_TransferRxCreateHandle(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, flexio_i2s_callback_t callback, void *userData)`

Initializes the FlexIO I2S receive handle.

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

Parameters

- `base` – Pointer to `FLEXIO_I2S_Type` structure.
- `handle` – Pointer to `flexio_i2s_handle_t` structure to store the transfer state.
- `callback` – FlexIO I2S callback function, which is called while finished a block.
- `userData` – User parameter for the FlexIO I2S callback.

status_t FLEXIO_I2S_TransferSendNonBlocking(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_transfer_t* *xfer)

Performs an interrupt non-blocking send transfer on FlexIO I2S.

Note: The API returns immediately after transfer initiates. Call FLEXIO_I2S_GetRemainingBytes to poll the transfer status and check whether the transfer is finished. If the return status is 0, the transfer is finished.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- xfer – Pointer to flexio_i2s_transfer_t structure

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_FLEXIO_I2S_TxBusy – Previous transmission still not finished, data not all written to TX register yet.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t FLEXIO_I2S_TransferReceiveNonBlocking(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_transfer_t* *xfer)

Performs an interrupt non-blocking receive transfer on FlexIO I2S.

Note: The API returns immediately after transfer initiates. Call FLEXIO_I2S_GetRemainingBytes to poll the transfer status to check whether the transfer is finished. If the return status is 0, the transfer is finished.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- xfer – Pointer to flexio_i2s_transfer_t structure

Return values

- kStatus_Success – Successfully start the data receive.
- kStatus_FLEXIO_I2S_RxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

void FLEXIO_I2S_TransferAbortSend(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle)

Aborts the current send.

Note: This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.

- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state

void FLEXIO_I2S_TransferAbortReceive(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle)

Aborts the current receive.

Note: This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state

status_t FLEXIO_I2S_TransferGetSendCount(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *size_t* *count)

Gets the remaining bytes to be sent.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- count – Bytes sent.

Return values

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

status_t FLEXIO_I2S_TransferGetReceiveCount(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *size_t* *count)

Gets the remaining bytes to be received.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- count – Bytes recieved.

Return values

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

Returns

count Bytes received.

void FLEXIO_I2S_TransferTxHandleIRQ(void *i2sBase, void *i2sHandle)

Tx interrupt handler.

Parameters

- i2sBase – Pointer to FLEXIO_I2S_Type structure.
- i2sHandle – Pointer to flexio_i2s_handle_t structure

void FLEXIO_I2S_TransferRxHandleIRQ(void *i2sBase, void *i2sHandle)

Rx interrupt handler.

Parameters

- i2sBase – Pointer to FLEXIO_I2S_Type structure.
- i2sHandle – Pointer to flexio_i2s_handle_t structure.

FSL_FLEXIO_I2S_DRIVER_VERSION

FlexIO I2S driver version 2.2.2.

FlexIO I2S transfer status.

Values:

enumerator kStatus_FLEXIO_I2S_Idle

FlexIO I2S is in idle state

enumerator kStatus_FLEXIO_I2S_TxBusy

FlexIO I2S Tx is busy

enumerator kStatus_FLEXIO_I2S_RxBusy

FlexIO I2S Rx is busy

enumerator kStatus_FLEXIO_I2S_Error

FlexIO I2S error occurred

enumerator kStatus_FLEXIO_I2S_QueueFull

FlexIO I2S transfer queue is full.

enumerator kStatus_FLEXIO_I2S_Timeout

FlexIO I2S timeout polling status flags.

enum flexio_i2s_master_slave

Master or slave mode.

Values:

enumerator kFLEXIO_I2S_Master

Master mode

enumerator kFLEXIO_I2S_Slave

Slave mode

_flexio_i2s_interrupt_enable Define FlexIO FlexIO I2S interrupt mask.

Values:

enumerator kFLEXIO_I2S_TxDataRegEmptyInterruptEnable

Transmit buffer empty interrupt enable.

enumerator kFLEXIO_I2S_RxDataRegFullInterruptEnable

Receive buffer full interrupt enable.

_flexio_i2s_status_flags Define FlexIO FlexIO I2S status mask.

Values:

enumerator kFLEXIO_I2S_TxDataRegEmptyFlag

Transmit buffer empty flag.

enumerator kFLEXIO_I2S_RxDataRegFullFlag
Receive buffer full flag.

enum _flexio_i2s_sample_rate
Audio sample rate.

Values:

enumerator kFLEXIO_I2S_SampleRate8KHz
Sample rate 8000Hz

enumerator kFLEXIO_I2S_SampleRate11025Hz
Sample rate 11025Hz

enumerator kFLEXIO_I2S_SampleRate12KHz
Sample rate 12000Hz

enumerator kFLEXIO_I2S_SampleRate16KHz
Sample rate 16000Hz

enumerator kFLEXIO_I2S_SampleRate22050Hz
Sample rate 22050Hz

enumerator kFLEXIO_I2S_SampleRate24KHz
Sample rate 24000Hz

enumerator kFLEXIO_I2S_SampleRate32KHz
Sample rate 32000Hz

enumerator kFLEXIO_I2S_SampleRate44100Hz
Sample rate 44100Hz

enumerator kFLEXIO_I2S_SampleRate48KHz
Sample rate 48000Hz

enumerator kFLEXIO_I2S_SampleRate96KHz
Sample rate 96000Hz

enum _flexio_i2s_word_width
Audio word width.

Values:

enumerator kFLEXIO_I2S_WordWidth8bits
Audio data width 8 bits

enumerator kFLEXIO_I2S_WordWidth16bits
Audio data width 16 bits

enumerator kFLEXIO_I2S_WordWidth24bits
Audio data width 24 bits

enumerator kFLEXIO_I2S_WordWidth32bits
Audio data width 32 bits

typedef struct _flexio_i2s_type FLEXIO_I2S_Type
Define FlexIO I2S access structure typedef.

typedef enum _flexio_i2s_master_slave flexio_i2s_master_slave_t
Master or slave mode.

typedef struct _flexio_i2s_config flexio_i2s_config_t
FlexIO I2S configure structure.

```
typedef struct _flexio_i2s_format flexio_i2s_format_t
    FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx.
typedef enum _flexio_i2s_sample_rate flexio_i2s_sample_rate_t
    Audio sample rate.
typedef enum _flexio_i2s_word_width flexio_i2s_word_width_t
    Audio word width.
typedef struct _flexio_i2s_transfer flexio_i2s_transfer_t
    Define FlexIO I2S transfer structure.
typedef struct _flexio_i2s_handle flexio_i2s_handle_t
typedef void (*flexio_i2s_callback_t)(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle,
status_t status, void *userData)
    FlexIO I2S xfer callback prototype.
I2S__RETRY_TIMES
    Retry times for waiting flag.
FLEXIO_I2S_XFER_QUEUE_SIZE
    FlexIO I2S transfer queue size, user can refine it according to use case.
struct _flexio_i2s_type
    #include <fsl_flexio_i2s.h> Define FlexIO I2S access structure typedef.
```

Public Members

```
FLEXIO_Type *flexioBase
    FlexIO base pointer
uint8_t txPinIndex
    Tx data pin index in FlexIO pins
uint8_t rxPinIndex
    Rx data pin index
uint8_t bclkPinIndex
    Bit clock pin index
uint8_t fsPinIndex
    Frame sync pin index
uint8_t txShifterIndex
    Tx data shifter index
uint8_t rxShifterIndex
    Rx data shifter index
uint8_t bclkTimerIndex
    Bit clock timer index
uint8_t fsTimerIndex
    Frame sync timer index
struct _flexio_i2s_config
    #include <fsl_flexio_i2s.h> FlexIO I2S configure structure.
```

Public Members

bool enableI2S

Enable FlexIO I2S

flexio_i2s_master_slave_t masterSlave

Master or slave

flexio_pin_polarity_t txPinPolarity

Tx data pin polarity, active high or low

flexio_pin_polarity_t rxPinPolarity

Rx data pin polarity

flexio_pin_polarity_t bclkPinPolarity

Bit clock pin polarity

flexio_pin_polarity_t fsPinPolarity

Frame sync pin polarity

flexio_shifter_timer_polarity_t txTimerPolarity

Tx data valid on bclk rising or falling edge

flexio_shifter_timer_polarity_t rxTimerPolarity

Rx data valid on bclk rising or falling edge

struct *_flexio_i2s_format*

#include <fsl_flexio_i2s.h> FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx.

Public Members

uint8_t bitWidth

Bit width of audio data, always 8/16/24/32 bits

uint32_t sampleRate_Hz

Sample rate of the audio data

struct *_flexio_i2s_transfer*

#include <fsl_flexio_i2s.h> Define FlexIO I2S transfer structure.

Public Members

uint8_t *data

Data buffer start pointer

size_t dataSize

Bytes to be transferred.

struct *_flexio_i2s_handle*

#include <fsl_flexio_i2s.h> Define FlexIO I2S handle structure.

Public Members

uint32_t state

Internal state

flexio_i2s_callback_t callback

Callback function called at transfer event

`void *userData`
 Callback parameter passed to callback function

`uint8_t bitWidth`
 Bit width for transfer, 8/16/24/32bits

`flexio_i2s_transfer_t queue[(4U)]`
 Transfer queue storing queued transfer

`size_t transferSize[(4U)]`
 Data bytes need to transfer

`volatile uint8_t queueUser`
 Index for user to queue transfer

`volatile uint8_t queueDriver`
 Index for driver to get the transfer data and size

2.22 FlexIO SPI Driver

`void FLEXIO_SPI_MasterInit(FLEXIO_SPI_Type *base, flexio_spi_master_config_t *masterConfig, uint32_t srcClock_Hz)`

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI master hardware, and configures the FlexIO SPI with FlexIO SPI master configuration. The configuration structure can be filled by the user, or be set with default values by the `FLEXIO_SPI_MasterGetDefaultConfig()`.

Example

```
FLEXIO_SPI_Type spiDev = {
    .flexioBase = FLEXIO,
    .SDOPinIndex = 0,
    .SDIPinIndex = 1,
    .SCKPinIndex = 2,
    .CSnPinIndex = 3,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_spi_master_config_t config = {
    .enableMaster = true,
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .baudRate_Bps = 500000,
    .phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
    .direction = kFLEXIO_SPI_MsbFirst,
    .dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_MasterInit(&spiDev, &config, srcClock_Hz);
```

Note: 1.FlexIO SPI master only support CPOL = 0, which means clock inactive low. 2.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI master communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by 2*2=4. If FlexIO SPI master communicates with FlexIO SPI slave, the maximum baud rate is FlexIO clock frequency divided by (1.5+2.5)*2=8.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- masterConfig – Pointer to the flexio_spi_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

void FLEXIO_SPI_MasterDeinit(*FLEXIO_SPI_Type* *base)

Resets the FlexIO SPI timer and shifter config.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.

void FLEXIO_SPI_MasterGetDefaultConfig(*flexio_spi_master_config_t* *masterConfig)

Gets the default configuration to configure the FlexIO SPI master. The configuration can be used directly by calling the FLEXIO_SPI_MasterConfigure(). Example:

```
flexio_spi_master_config_t masterConfig;
FLEXIO_SPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – Pointer to the flexio_spi_master_config_t structure.

void FLEXIO_SPI_SlaveInit(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_config_t* *slaveConfig)

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI slave hardware configuration, and configures the FlexIO SPI with FlexIO SPI slave configuration. The configuration structure can be filled by the user, or be set with default values by the FLEXIO_SPI_SlaveGetDefaultConfig().

Note: 1. Only one timer is needed in the FlexIO SPI slave. As a result, the second timer index is ignored. 2. FlexIO SPI slave only support CPOL = 0, which means clock inactive low. 3. For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI slave communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by $3 \times 2 = 6$. If FlexIO SPI slave communicates with FlexIO SPI master, the maximum baud rate is FlexIO clock frequency divided by $(1.5 + 2.5) \times 2 = 8$. Example

```
FLEXIO_SPI_Type spiDev = {
    .flexioBase = FLEXIO,
    .SDOPinIndex = 0,
    .SDIPinIndex = 1,
    .SCKPinIndex = 2,
    .CSnPinIndex = 3,
    .shifterIndex = {0,1},
    .timerIndex = {0}
};
flexio_spi_slave_config_t config = {
    .enableSlave = true,
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
    .direction = kFLEXIO_SPI_MsbFirst,
    .dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_SlaveInit(&spiDev, &config);
```

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- slaveConfig – Pointer to the flexio_spi_slave_config_t structure.

void FLEXIO_SPI_SlaveDeinit(*FLEXIO_SPI_Type* *base)

Gates the FlexIO clock.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.

void FLEXIO_SPI_SlaveGetDefaultConfig(*flexio_spi_slave_config_t* *slaveConfig)

Gets the default configuration to configure the FlexIO SPI slave. The configuration can be used directly for calling the FLEXIO_SPI_SlaveConfigure(). Example:

```
flexio_spi_slave_config_t slaveConfig;  
FLEXIO_SPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – Pointer to the flexio_spi_slave_config_t structure.

uint32_t FLEXIO_SPI_GetStatusFlags(*FLEXIO_SPI_Type* *base)

Gets FlexIO SPI status flags.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

Returns

status flag; Use the status flag to AND the following flag mask and get the status.

- kFLEXIO_SPI_TxEmptyFlag
- kFLEXIO_SPI_RxEmptyFlag

void FLEXIO_SPI_ClearStatusFlags(*FLEXIO_SPI_Type* *base, uint32_t mask)

Clears FlexIO SPI status flags.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – status flag The parameter can be any combination of the following values:
 - kFLEXIO_SPI_TxEmptyFlag
 - kFLEXIO_SPI_RxEmptyFlag

void FLEXIO_SPI_EnableInterrupts(*FLEXIO_SPI_Type* *base, uint32_t mask)

Enables the FlexIO SPI interrupt.

This function enables the FlexIO SPI interrupt.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – interrupt source. The parameter can be any combination of the following values:
 - kFLEXIO_SPI_RxFullInterruptEnable
 - kFLEXIO_SPI_TxEmptyInterruptEnable


```
void FLEXIO_SPI_DisableInterrupts(FLEXIO_SPI_Type *base, uint32_t mask)
```

Disables the FlexIO SPI interrupt.

This function disables the FlexIO SPI interrupt.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- mask – interrupt source The parameter can be any combination of the following values:
 - kFLEXIO_SPI_RxFullInterruptEnable
 - kFLEXIO_SPI_TxEmptyInterruptEnable

```
void FLEXIO_SPI_EnableDMA(FLEXIO_SPI_Type *base, uint32_t mask, bool enable)
```

Enables/disables the FlexIO SPI transmit DMA. This function enables/disables the FlexIO SPI Tx DMA, which means that asserting the kFLEXIO_SPI_TxEmptyFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- mask – SPI DMA source.
- enable – True means enable DMA, false means disable DMA.

```
static inline uint32_t FLEXIO_SPI_GetTxDataRegisterAddress(FLEXIO_SPI_Type *base,  
                                                         flexio_spi_shift_direction_t  
                                                         direction)
```

Gets the FlexIO SPI transmit data register address for MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- direction – Shift direction of MSB first or LSB first.

Returns

FlexIO SPI transmit data register address.

```
static inline uint32_t FLEXIO_SPI_GetRxDataRegisterAddress(FLEXIO_SPI_Type *base,  
                                                         flexio_spi_shift_direction_t  
                                                         direction)
```

Gets the FlexIO SPI receive data register address for the MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- direction – Shift direction of MSB first or LSB first.

Returns

FlexIO SPI receive data register address.

```
static inline void FLEXIO_SPI_Enable(FLEXIO_SPI_Type *base, bool enable)
```

Enables/disables the FlexIO SPI module operation.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type*.
- enable – True to enable, false does not have any effect.

```
void FLEXIO_SPI_MasterSetBaudRate(FLEXIO_SPI_Type *base, uint32_t baudRate_Bps,  
                                uint32_t srcClockHz)
```

Sets baud rate for the FlexIO SPI transfer, which is only used for the master.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- baudRate_Bps – Baud Rate needed in Hz.
- srcClockHz – SPI source clock frequency in Hz.

```
static inline void FLEXIO_SPI_WriteData(FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t  
                                       direction, uint32_t data)
```

Writes one byte of data, which is sent using the MSB method.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- direction – Shift direction of MSB first or LSB first.
- data – 8/16/32 bit data.

```
static inline uint32_t FLEXIO_SPI_ReadData(FLEXIO_SPI_Type *base,  
                                          flexio_spi_shift_direction_t direction)
```

Reads 8 bit/16 bit data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- direction – Shift direction of MSB first or LSB first.

Returns

8 bit/16 bit data received.

```
status_t FLEXIO_SPI_WriteBlocking(FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t  
                                 direction, const uint8_t *buffer, size_t size)
```

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The data bytes to send.
- size – The number of data bytes to send.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

status_t FLEXIO_SPI_ReadBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, *uint8_t* *buffer, *size_t* size)

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The buffer to store the received bytes.
- size – The number of data bytes to be received.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

status_t FLEXIO_SPI_MasterTransferBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_transfer_t* *xfer)

Receives a buffer of bytes.

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

Parameters

- base – pointer to FLEXIO_SPI_Type structure
- xfer – FlexIO SPI transfer structure, see flexio_spi_transfer_t.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

void FLEXIO_SPI_FlushShifters(*FLEXIO_SPI_Type* *base)

Flush tx/rx shifters.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

status_t FLEXIO_SPI_MasterTransferCreateHandle(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, *flexio_spi_master_transfer_callback_t* callback, *void* *userData)

Initializes the FlexIO SPI Master handle, which is used in transactional functions.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- callback – The callback function.

- `userData` – The parameter of the callback function.

Return values

- `kStatus_Success` – Successfully create the handle.
- `kStatus_OutOfRange` – The FlexIO type/handle/ISR table out of range.

`status_t` FLEXIO_SPI_MasterTransferNonBlocking(*FLEXIO_SPI_Type* *base,
 flexio_spi_master_handle_t *handle,
 flexio_spi_transfer_t *xfer)

Master transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to the `flexio_spi_master_handle_t` structure to store the transfer state.
- `xfer` – FlexIO SPI transfer structure. See `flexio_spi_transfer_t`.

Return values

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

`void` FLEXIO_SPI_MasterTransferAbort(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t*
 *handle)

Aborts the master data transfer, which used IRQ.

Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to the `flexio_spi_master_handle_t` structure to store the transfer state.

`status_t` FLEXIO_SPI_MasterTransferGetCount(*FLEXIO_SPI_Type* *base,
 flexio_spi_master_handle_t *handle, *size_t*
 *count)

Gets the data transfer status which used IRQ.

Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to the `flexio_spi_master_handle_t` structure to store the transfer state.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_InvalidArgument` – count is Invalid.
- `kStatus_Success` – Successfully return the count.

`void` FLEXIO_SPI_MasterTransferHandleIRQ(*void* *spiType, *void* *spiHandle)

FlexIO SPI master IRQ handler function.

Parameters

- `spiType` – Pointer to the `FLEXIO_SPI_Type` structure.

- spiHandle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

```
status_t FLEXIO_SPI_SlaveTransferCreateHandle(FLEXIO_SPI_Type *base,
                                              flexio_spi_slave_handle_t *handle,
                                              flexio_spi_slave_transfer_callback_t callback,
                                              void *userData)
```

Initializes the FlexIO SPI Slave handle, which is used in transactional functions.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

```
status_t FLEXIO_SPI_SlaveTransferNonBlocking(FLEXIO_SPI_Type *base,
                                              flexio_spi_slave_handle_t *handle,
                                              flexio_spi_transfer_t *xfer)
```

Slave transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- base – Pointer to the FLEXIO_SPI_Type structure.
- xfer – FlexIO SPI transfer structure. See flexio_spi_transfer_t.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – SPI is not idle; it is running another transfer.

```
static inline void FLEXIO_SPI_SlaveTransferAbort(FLEXIO_SPI_Type *base,
                                                flexio_spi_slave_handle_t *handle)
```

Aborts the slave data transfer which used IRQ, share same API with master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

```
static inline status_t FLEXIO_SPI_SlaveTransferGetCount(FLEXIO_SPI_Type *base,
                                                       flexio_spi_slave_handle_t *handle,
                                                       size_t *count)
```

Gets the data transfer status which used IRQ, share same API with master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

- `handle` – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_InvalidArgument` – `count` is Invalid.
- `kStatus_Success` – Successfully return the count.

`void FLEXIO_SPI_SlaveTransferHandleIRQ(void *spiType, void *spiHandle)`

FlexIO SPI slave IRQ handler function.

Parameters

- `spiType` – Pointer to the `FLEXIO_SPI_Type` structure.
- `spiHandle` – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.

`FSL_FLEXIO_SPI_DRIVER_VERSION`

FlexIO SPI driver version.

Error codes for the FlexIO SPI driver.

Values:

enumerator `kStatus_FLEXIO_SPI_Busy`

FlexIO SPI is busy.

enumerator `kStatus_FLEXIO_SPI_Idle`

SPI is idle

enumerator `kStatus_FLEXIO_SPI_Error`

FlexIO SPI error.

enumerator `kStatus_FLEXIO_SPI_Timeout`

FlexIO SPI timeout polling status flags.

enum `_flexio_spi_clock_phase`

FlexIO SPI clock phase configuration.

Values:

enumerator `kFLEXIO_SPI_ClockPhaseFirstEdge`

First edge on SPSCCK occurs at the middle of the first cycle of a data transfer.

enumerator `kFLEXIO_SPI_ClockPhaseSecondEdge`

First edge on SPSCCK occurs at the start of the first cycle of a data transfer.

enum `_flexio_spi_shift_direction`

FlexIO SPI data shifter direction options.

Values:

enumerator `kFLEXIO_SPI_MsbFirst`

Data transfers start with most significant bit.

enumerator `kFLEXIO_SPI_LsbFirst`

Data transfers start with least significant bit.

enum `_flexio_spi_data_bitcount_mode`

FlexIO SPI data length mode options.

Values:

enumerator kFLEXIO_SPI_8BitMode
8-bit data transmission mode.

enumerator kFLEXIO_SPI_16BitMode
16-bit data transmission mode.

enumerator kFLEXIO_SPI_32BitMode
32-bit data transmission mode.

enum _flexio_spi_interrupt_enable
Define FlexIO SPI interrupt mask.

Values:

enumerator kFLEXIO_SPI_TxEmptyInterruptEnable
Transmit buffer empty interrupt enable.

enumerator kFLEXIO_SPI_RxFullInterruptEnable
Receive buffer full interrupt enable.

enum _flexio_spi_status_flags
Define FlexIO SPI status mask.

Values:

enumerator kFLEXIO_SPI_TxBufferEmptyFlag
Transmit buffer empty flag.

enumerator kFLEXIO_SPI_RxBufferFullFlag
Receive buffer full flag.

enum _flexio_spi_dma_enable
Define FlexIO SPI DMA mask.

Values:

enumerator kFLEXIO_SPI_TxDmaEnable
Tx DMA request source

enumerator kFLEXIO_SPI_RxDmaEnable
Rx DMA request source

enumerator kFLEXIO_SPI_DmaAllEnable
All DMA request source

enum _flexio_spi_transfer_flags
Define FlexIO SPI transfer flags.

Note: Use kFLEXIO_SPI_csContinuous and one of the other flags to OR together to form the transfer flag.

Values:

enumerator kFLEXIO_SPI_8bitMsb
FlexIO SPI 8-bit MSB first

enumerator kFLEXIO_SPI_8bitLsb
FlexIO SPI 8-bit LSB first

enumerator kFLEXIO_SPI_16bitMsb
FlexIO SPI 16-bit MSB first

enumerator `kFLEXIO_SPI_16bitLsb`
FlexIO SPI 16-bit LSB first

enumerator `kFLEXIO_SPI_32bitMsb`
FlexIO SPI 32-bit MSB first

enumerator `kFLEXIO_SPI_32bitLsb`
FlexIO SPI 32-bit LSB first

enumerator `kFLEXIO_SPI_csContinuous`
Enable the CS signal continuous mode

typedef enum `_flexio_spi_clock_phase` `flexio_spi_clock_phase_t`
FlexIO SPI clock phase configuration.

typedef enum `_flexio_spi_shift_direction` `flexio_spi_shift_direction_t`
FlexIO SPI data shifter direction options.

typedef enum `_flexio_spi_data_bitcount_mode` `flexio_spi_data_bitcount_mode_t`
FlexIO SPI data length mode options.

typedef struct `_flexio_spi_type` `FLEXIO_SPI_Type`
Define FlexIO SPI access structure typedef.

typedef struct `_flexio_spi_master_config` `flexio_spi_master_config_t`
Define FlexIO SPI master configuration structure.

typedef struct `_flexio_spi_slave_config` `flexio_spi_slave_config_t`
Define FlexIO SPI slave configuration structure.

typedef struct `_flexio_spi_transfer` `flexio_spi_transfer_t`
Define FlexIO SPI transfer structure.

typedef struct `_flexio_spi_master_handle` `flexio_spi_master_handle_t`
typedef for `flexio_spi_master_handle_t` in advance.

typedef `flexio_spi_master_handle_t` `flexio_spi_slave_handle_t`
Slave handle is the same with master handle.

typedef void (*`flexio_spi_master_transfer_callback_t`)(`FLEXIO_SPI_Type` *base,
`flexio_spi_master_handle_t` *handle, `status_t` status, void *userData)
FlexIO SPI master callback for finished transmit.

typedef void (*`flexio_spi_slave_transfer_callback_t`)(`FLEXIO_SPI_Type` *base,
`flexio_spi_slave_handle_t` *handle, `status_t` status, void *userData)
FlexIO SPI slave callback for finished transmit.

`FLEXIO_SPI_DUMMYDATA`
FlexIO SPI dummy transfer data, the data is sent while txData is NULL.

`SPI_RETRY_TIMES`
Retry times for waiting flag.

`FLEXIO_SPI_XFER_DATA_FORMAT(flag)`
Get the transfer data format of width and bit order.

struct `_flexio_spi_type`
`#include <fsl_flexio_spi.h>` Define FlexIO SPI access structure typedef.

Public Members

FLEXIO_Type *flexioBase

FlexIO base pointer.

uint8_t SDOPinIndex

Pin select for data output. To set SDO pin in Hi-Z state, user needs to mux the pin as GPIO input and disable all pull up/down in application.

uint8_t SDIPinIndex

Pin select for data input.

uint8_t SCKPinIndex

Pin select for clock.

uint8_t CSnPinIndex

Pin select for enable.

uint8_t shifterIndex[2]

Shifter index used in FlexIO SPI.

uint8_t timerIndex[2]

Timer index used in FlexIO SPI.

struct _flexio_spi_master_config

#include <fsl_flexio_spi.h> Define FlexIO SPI master configuration structure.

Public Members

bool enableMaster

Enable/disable FlexIO SPI master after configuration.

bool enableInDoze

Enable/disable FlexIO operation in doze mode.

bool enableInDebug

Enable/disable FlexIO operation in debug mode.

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps

Baud rate in Bps.

flexio_spi_clock_phase_t phase

Clock phase.

flexio_spi_data_bitcount_mode_t dataMode

8bit or 16bit mode.

struct _flexio_spi_slave_config

#include <fsl_flexio_spi.h> Define FlexIO SPI slave configuration structure.

Public Members

bool enableSlave

Enable/disable FlexIO SPI slave after configuration.

bool enableInDoze

Enable/disable FlexIO operation in doze mode.

bool enableInDebug

Enable/disable FlexIO operation in debug mode.

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

flexio_spi_clock_phase_t phase

Clock phase.

flexio_spi_data_bitcount_mode_t dataMode

8bit or 16bit mode.

struct *_flexio_spi_transfer*

#include <fsl_flexio_spi.h> Define FlexIO SPI transfer structure.

Public Members

const uint8_t *txData

Send buffer.

uint8_t *rxData

Receive buffer.

size_t dataSize

Transfer bytes.

uint8_t flags

FlexIO SPI control flag, MSB first or LSB first.

struct *_flexio_spi_master_handle*

#include <fsl_flexio_spi.h> Define FlexIO SPI handle structure.

Public Members

const uint8_t *txData

Transfer buffer.

uint8_t *rxData

Receive buffer.

size_t transferSize

Total bytes to be transferred.

volatile size_t txRemainingBytes

Send data remaining in bytes.

volatile size_t rxRemainingBytes

Receive data remaining in bytes.

volatile uint32_t state

FlexIO SPI internal state.

uint8_t bytePerFrame

SPI mode, 2bytes or 1byte in a frame

flexio_spi_shift_direction_t direction

Shift direction.

flexio_spi_master_transfer_callback_t callback

FlexIO SPI callback.

void *userData

Callback parameter.

bool isCsContinuous

Is current transfer using CS continuous mode.

uint32_t timer1Cfg

TIMER1 TIMCFG regiser value backup.

2.23 FlexIO UART Driver

status_t FLEXIO_UART_Init(*FLEXIO_UART_Type* *base, const *flexio_uart_config_t* *userConfig, uint32_t srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, configures FlexIO UART hardware, and configures the FlexIO UART with FlexIO UART configuration. The configuration structure can be filled by the user or be set with default values by FLEXIO_UART_GetDefaultConfig().

Example

```
FLEXIO_UART_Type base = {
    .flexioBase = FLEXIO,
    .TxPinIndex = 0,
    .RxPinIndex = 1,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_uart_config_t config = {
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .baudRate_Bps = 115200U,
    .bitCountPerChar = 8
};
FLEXIO_UART_Init(base, &config, srcClock_Hz);
```

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- userConfig – Pointer to the flexio_uart_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Configuration success.
- kStatus_FLEXIO_UART_BaudrateNotSupport – Baudrate is not supported for current clock source frequency.

void FLEXIO_UART_Deinit(*FLEXIO_UART_Type* *base)

Resets the FlexIO UART shifter and timer config.

Note: After calling this API, call the FLEXIO_UART_Init to use the FlexIO UART module.

Parameters

- base – Pointer to FLEXIO_UART_Type structure

void FLEXIO_UART_GetDefaultConfig(*flexio_uart_config_t* *userConfig)

Gets the default configuration to configure the FlexIO UART. The configuration can be used directly for calling the FLEXIO_UART_Init(). Example:

```
flexio_uart_config_t config;  
FLEXIO_UART_GetDefaultConfig(&userConfig);
```

Parameters

- userConfig – Pointer to the flexio_uart_config_t structure.

uint32_t FLEXIO_UART_GetStatusFlags(*FLEXIO_UART_Type* *base)

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART status flags.

void FLEXIO_UART_ClearStatusFlags(*FLEXIO_UART_Type* *base, uint32_t mask)

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Status flag. The parameter can be any combination of the following values:
 - kFLEXIO_UART_TxDataRegEmptyFlag
 - kFLEXIO_UART_RxEmptyFlag
 - kFLEXIO_UART_RxOverRunFlag

void FLEXIO_UART_EnableInterrupts(*FLEXIO_UART_Type* *base, uint32_t mask)

Enables the FlexIO UART interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Interrupt source.

void FLEXIO_UART_DisableInterrupts(*FLEXIO_UART_Type* *base, uint32_t mask)

Disables the FlexIO UART interrupt.

This function disables the FlexIO UART interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Interrupt source.

static inline uint32_t FLEXIO_UART_GetTxDataRegisterAddress(*FLEXIO_UART_Type* *base)

Gets the FlexIO UART transmit data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART transmit data register address.

```
static inline uint32_t FLEXIO_UART_GetRxDataRegisterAddress(FLEXIO_UART_Type *base)
```

Gets the FlexIO UART receive data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART receive data register address.

```
static inline void FLEXIO_UART_EnableTxDMA(FLEXIO_UART_Type *base, bool enable)
```

Enables/disables the FlexIO UART transmit DMA. This function enables/disables the FlexIO UART Tx DMA, which means asserting the kFLEXIO_UART_TxDataRegEmptyFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- enable – True to enable, false to disable.

```
static inline void FLEXIO_UART_EnableRxDMA(FLEXIO_UART_Type *base, bool enable)
```

Enables/disables the FlexIO UART receive DMA. This function enables/disables the FlexIO UART Rx DMA, which means asserting kFLEXIO_UART_RxDataRegFullFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- enable – True to enable, false to disable.

```
static inline void FLEXIO_UART_Enable(FLEXIO_UART_Type *base, bool enable)
```

Enables/disables the FlexIO UART module operation.

Parameters

- base – Pointer to the FLEXIO_UART_Type.
- enable – True to enable, false does not have any effect.

```
static inline void FLEXIO_UART_WriteByte(FLEXIO_UART_Type *base, const uint8_t *buffer)
```

Writes one byte of data.

Note: This is a non-blocking API, which returns directly after the data is put into the data register. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- buffer – The data bytes to send.

```
static inline void FLEXIO_UART_ReadByte(FLEXIO_UART_Type *base, uint8_t *buffer)
```

Reads one byte of data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- buffer – The buffer to store the received bytes.

status_t FLEXIO_UART_WriteBlocking(*FLEXIO_UART_Type* *base, const uint8_t *txData, size_t txSize)

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- txData – The data bytes to send.
- txSize – The number of data bytes to send.

Return values

- kStatus_FLEXIO_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t FLEXIO_UART_ReadBlocking(*FLEXIO_UART_Type* *base, uint8_t *rxData, size_t rxSize)

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- rxData – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus_FLEXIO_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t FLEXIO_UART_TransferCreateHandle(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_callback_t* callback, void *userData)

Initializes the UART handle.

This function initializes the FlexIO UART handle, which can be used for other FlexIO UART transactional APIs. Call this API once to get the initialized handle.

The UART driver supports the “background” receiving, which means that users can set up a RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn’t call the FLEXIO_UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as ringBuffer.

Parameters

- base – to FLEXIO_UART_Type structure.

- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

```
void FLEXIO_UART_TransferStartRingBuffer(FLEXIO_UART_Type *base, flexio_uart_handle_t
                                         *handle, uint8_t *ringBuffer, size_t
                                         ringBufferSize)
```

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_ReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly.

Note: When using the RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, only 31 bytes are used for saving data.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – Size of the ring buffer.

```
void FLEXIO_UART_TransferStopRingBuffer(FLEXIO_UART_Type *base, flexio_uart_handle_t
                                         *handle)
```

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

```
status_t FLEXIO_UART_TransferSendNonBlocking(FLEXIO_UART_Type *base,
                                              flexio_uart_handle_t *handle,
                                              flexio_uart_transfer_t *xfer)
```

Transmits a buffer of data using the interrupt method.

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in ISR, the FlexIO UART driver calls the callback function and passes the kStatus_FLEXIO_UART_TxIdle as status parameter.

Note: The kStatus_FLEXIO_UART_TxIdle is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out.

Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.
- `xfer` – FlexIO UART transfer structure. See `flexio_uart_transfer_t`.

Return values

- `kStatus_Success` – Successfully starts the data transmission.
- `kStatus_UART_TxBusy` – Previous transmission still not finished, data not written to the TX register.

```
void FLEXIO_UART_TransferAbortSend(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle)
```

Aborts the interrupt-driven data transmit.

This function aborts the interrupt-driven data sending. Get the `remainBytes` to find out how many bytes are still not sent out.

Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.

```
status_t FLEXIO_UART_TransferGetSendCount(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, size_t *count)
```

Gets the number of bytes sent.

This function gets the number of bytes sent driven by interrupt.

Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.
- `count` – Number of bytes sent so far by the non-blocking transaction.

Return values

- `kStatus_NoTransferInProgress` – transfer has finished or no transfer in progress.
- `kStatus_Success` – Successfully return the count.

```
status_t FLEXIO_UART_TransferReceiveNonBlocking(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, flexio_uart_transfer_t *xfer, size_t *receivedBytes)
```

Receives a buffer of data using the interrupt method.

This function receives data using the interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in ring buffer is not enough to read, the receive request is saved by the UART driver. When new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter `kStatus_UART_RxIdle`. For example, if the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer, the 5 bytes are copied to `xfer->data`. This function returns with the parameter `receivedBytes` set to 5. For the last 5 bytes, newly arrived data is saved from the

xfer->data[5]. When 5 bytes are received, the UART driver notifies upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- xfer – UART transfer structure. See flexio_uart_transfer_t.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into the transmit queue.
- kStatus_FLEXIO_UART_RxBusy – Previous receive request is not finished.

```
void FLEXIO_UART_TransferAbortReceive(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle)
```

Aborts the receive data which was using IRQ.

This function aborts the receive data which was using IRQ.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

```
status_t FLEXIO_UART_TransferGetReceiveCount(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, size_t *count)
```

Gets the number of bytes received.

This function gets the number of bytes received driven by interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- count – Number of bytes received so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

```
void FLEXIO_UART_TransferHandleIRQ(void *uartType, void *uartHandle)
```

FlexIO UART IRQ handler function.

This function processes the FlexIO UART transmit and receives the IRQ request.

Parameters

- uartType – Pointer to the FLEXIO_UART_Type structure.
- uartHandle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

void FLEXIO_UART_FlushShifters(*FLEXIO_UART_Type* *base)

Flush tx/rx shifters.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

FSL_FLEXIO_UART_DRIVER_VERSION

FlexIO UART driver version.

Error codes for the UART driver.

Values:

enumerator kStatus_FLEXIO_UART_TxBusy

Transmitter is busy.

enumerator kStatus_FLEXIO_UART_RxBusy

Receiver is busy.

enumerator kStatus_FLEXIO_UART_TxIdle

UART transmitter is idle.

enumerator kStatus_FLEXIO_UART_RxIdle

UART receiver is idle.

enumerator kStatus_FLEXIO_UART_ERROR

ERROR happens on UART.

enumerator kStatus_FLEXIO_UART_RxRingBufferOverrun

UART RX software ring buffer overrun.

enumerator kStatus_FLEXIO_UART_RxHardwareOverrun

UART RX receiver overrun.

enumerator kStatus_FLEXIO_UART_Timeout

UART times out.

enumerator kStatus_FLEXIO_UART_BaudrateNotSupport

Baudrate is not supported in current clock source

enum _flexio_uart_bit_count_per_char

FlexIO UART bit count per char.

Values:

enumerator kFLEXIO_UART_7BitsPerChar

7-bit data characters

enumerator kFLEXIO_UART_8BitsPerChar

8-bit data characters

enumerator kFLEXIO_UART_9BitsPerChar

9-bit data characters

enum _flexio_uart_interrupt_enable

Define FlexIO UART interrupt mask.

Values:

enumerator kFLEXIO_UART_TxDataRegEmptyInterruptEnable

Transmit buffer empty interrupt enable.

```

    enumerator kFLEXIO_UART_RxDataRegFullInterruptEnable
        Receive buffer full interrupt enable.

enum _flexio_uart_status_flags
    Define FlexIO UART status mask.

    Values:

    enumerator kFLEXIO_UART_TxDataRegEmptyFlag
        Transmit buffer empty flag.

    enumerator kFLEXIO_UART_RxDataRegFullFlag
        Receive buffer full flag.

    enumerator kFLEXIO_UART_RxOverRunFlag
        Receive buffer over run flag.

typedef enum _flexio_uart_bit_count_per_char flexio_uart_bit_count_per_char_t
    FlexIO UART bit count per char.

typedef struct _flexio_uart_type FLEXIO_UART_Type
    Define FlexIO UART access structure typedef.

typedef struct _flexio_uart_config flexio_uart_config_t
    Define FlexIO UART user configuration structure.

typedef struct _flexio_uart_transfer flexio_uart_transfer_t
    Define FlexIO UART transfer structure.

typedef struct _flexio_uart_handle flexio_uart_handle_t

typedef void (*flexio_uart_transfer_callback_t)(FLEXIO_UART_Type *base, flexio_uart_handle_t
*handle, status_t status, void *userData)
    FlexIO UART transfer callback function.

UART_RETRY_TIMES
    Retry times for waiting flag.

struct _flexio_uart_type
    #include <fsl_flexio_uart.h> Define FlexIO UART access structure typedef.

```

Public Members

```

FLEXIO_Type *flexioBase
    FlexIO base pointer.

uint8_t TxPinIndex
    Pin select for UART_Tx.

uint8_t RxPinIndex
    Pin select for UART_Rx.

uint8_t shifterIndex[2]
    Shifter index used in FlexIO UART.

uint8_t timerIndex[2]
    Timer index used in FlexIO UART.

struct _flexio_uart_config
    #include <fsl_flexio_uart.h> Define FlexIO UART user configuration structure.

```

Public Members

`bool enableUart`
Enable/disable FlexIO UART TX & RX.

`bool enableInDoze`
Enable/disable FlexIO operation in doze mode

`bool enableInDebug`
Enable/disable FlexIO operation in debug mode

`bool enableFastAccess`
Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

`uint32_t baudRate_Bps`
Baud rate in Bps.

`flexio_uart_bit_count_per_char_t bitCountPerChar`
number of bits, 7/8/9 -bit

`struct _flexio_uart_transfer`
#include <fsl_flexio_uart.h> Define FlexIO UART transfer structure.

Public Members

`size_t dataSize`
Transfer size

`struct _flexio_uart_handle`
#include <fsl_flexio_uart.h> Define FLEXIO UART handle structure.

Public Members

`const uint8_t *volatile txData`
Address of remaining data to send.

`volatile size_t txDataSize`
Size of the remaining data to send.

`uint8_t *volatile rxData`
Address of remaining data to receive.

`volatile size_t rxDataSize`
Size of the remaining data to receive.

`size_t txDataSizeAll`
Total bytes to be sent.

`size_t rxDataSizeAll`
Total bytes to be received.

`uint8_t *rxRingBuffer`
Start address of the receiver ring buffer.

`size_t rxRingBufferSize`
Size of the ring buffer.

`volatile uint16_t rxRingBufferHead`
Index for the driver to store received data into ring buffer.

`volatile uint16_t rxRingBufferTail`
Index for the user to get data from the ring buffer.

`flexio_uart_transfer_callback_t callback`
Callback function.

`void *userData`
UART callback function parameter.

`volatile uint8_t txState`
TX transfer state.

`volatile uint8_t rxState`
RX transfer state

`union __unnamed228__`

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.

2.24 FLEXSPI: Flexible Serial Peripheral Interface Driver

`uint32_t FLEXSPI_GetInstance(FLEXSPI_Type *base)`
Get the instance number for FLEXSPI.

Parameters

- `base` – FLEXSPI base pointer.

`status_t FLEXSPI_CheckAndClearError(FLEXSPI_Type *base, uint32_t status)`
Check and clear IP command execution errors.

Parameters

- `base` – FLEXSPI base pointer.
- `status` – interrupt status.

`void FLEXSPI_Init(FLEXSPI_Type *base, const flexspi_config_t *config)`
Initializes the FLEXSPI module and internal state.

This function enables the clock for FLEXSPI and also configures the FLEXSPI with the input configure parameters. Users should call this function before any FLEXSPI operations.

Parameters

- `base` – FLEXSPI peripheral base address.
- `config` – FLEXSPI configure structure.

void FLEXSPI_GetDefaultConfig(*flexspi_config_t* *config)

Gets default settings for FLEXSPI.

Parameters

- config – FLEXSPI configuration structure.

void FLEXSPI_Deinit(FLEXSPI_Type *base)

Deinitializes the FLEXSPI module.

Clears the FLEXSPI state and FLEXSPI module registers.

Parameters

- base – FLEXSPI peripheral base address.

void FLEXSPI_UpdateDllValue(FLEXSPI_Type *base, *flexspi_device_config_t* *config,
flexspi_port_t port)

Update FLEXSPI DLL value depending on currently flexspi root clock.

Parameters

- base – FLEXSPI peripheral base address.
- config – Flash configuration parameters.
- port – FLEXSPI Operation port.

void FLEXSPI_SetFlashConfig(FLEXSPI_Type *base, *flexspi_device_config_t* *config,
flexspi_port_t port)

Configures the connected device parameter.

This function configures the connected device relevant parameters, such as the size, command, and so on. The flash configuration value cannot have a default value. The user needs to configure it according to the connected device.

Parameters

- base – FLEXSPI peripheral base address.
- config – Flash configuration parameters.
- port – FLEXSPI Operation port.

void FLEXSPI_SoftwareReset(FLEXSPI_Type *base)

Software reset for the FLEXSPI logic.

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

Parameters

- base – FLEXSPI peripheral base address.

static inline void FLEXSPI_Enable(FLEXSPI_Type *base, bool enable)

Enables or disables the FLEXSPI module.

Parameters

- base – FLEXSPI peripheral base address.
- enable – True means enable FLEXSPI, false means disable.

void FLEXSPI_UpdateAhbBuffersSettings(FLEXSPI_Type *base, *flexspi_ahbBuffers_ctrl_t*
**ptrAhbBufferCtrl*)

Update all AHB buffers' settings, including buffer size, master ID.

Parameters

- base – FLEXSPI peripheral base address.

- ptrAhbBufferCtrl – Pointer to structure flexspi_ahbBuffers_ctrl_t which store all AHB buffers' settings.

static inline void FLEXSPI_EnableInterrupts(FLEXSPI_Type *base, uint32_t mask)

Enables the FLEXSPI interrupts.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

static inline void FLEXSPI_DisableInterrupts(FLEXSPI_Type *base, uint32_t mask)

Disable the FLEXSPI interrupts.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

static inline void FLEXSPI_EnableTxDMA(FLEXSPI_Type *base, bool enable)

Enables or disables FLEXSPI IP Tx FIFO DMA requests.

Parameters

- base – FLEXSPI peripheral base address.
- enable – Enable flag for transmit DMA request. Pass true for enable, false for disable.

static inline void FLEXSPI_EnableRxDMA(FLEXSPI_Type *base, bool enable)

Enables or disables FLEXSPI IP Rx FIFO DMA requests.

Parameters

- base – FLEXSPI peripheral base address.
- enable – Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t FLEXSPI_GetTxFifoAddress(FLEXSPI_Type *base)

Gets FLEXSPI IP tx fifo address for DMA transfer.

Parameters

- base – FLEXSPI peripheral base address.

Return values

The – tx fifo address.

static inline uint32_t FLEXSPI_GetRxFifoAddress(FLEXSPI_Type *base)

Gets FLEXSPI IP rx fifo address for DMA transfer.

Parameters

- base – FLEXSPI peripheral base address.

Return values

The – rx fifo address.

static inline void FLEXSPI_ResetFifos(FLEXSPI_Type *base, bool txFifo, bool rxFifo)

Clears the FLEXSPI IP FIFO logic.

Parameters

- base – FLEXSPI peripheral base address.
- txFifo – Pass true to reset TX FIFO.
- rxFifo – Pass true to reset RX FIFO.

```
static inline void FLEXSPI_GetFifoCounts(FLEXSPI_Type *base, size_t *txCount, size_t *rxCount)
```

Gets the valid data entries in the FLEXSPI FIFOs.

Parameters

- base – FLEXSPI peripheral base address.
- txCount – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

```
static inline uint32_t FLEXSPI_GetInterruptStatusFlags(FLEXSPI_Type *base)
```

Get the FLEXSPI interrupt status flags.

Parameters

- base – FLEXSPI peripheral base address.

Return values

interrupt – status flag, use status flag to AND flexspi_flags_t could get the related status.

```
static inline void FLEXSPI_ClearInterruptStatusFlags(FLEXSPI_Type *base, uint32_t mask)
```

Get the FLEXSPI interrupt status flags.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

```
static inline flexspi_arb_command_source_t FLEXSPI_GetArbitratorCommandSource(FLEXSPI_Type *base)
```

Gets the trigger source of current command sequence granted by arbitrator.

Parameters

- base – FLEXSPI peripheral base address.

Return values

trigger – source of current command sequence.

```
static inline flexspi_ip_error_code_t FLEXSPI_GetIPCommandErrorCode(FLEXSPI_Type *base, uint8_t *index)
```

Gets the error code when IP command error detected.

Parameters

- base – FLEXSPI peripheral base address.
- index – Pointer to a uint8_t type variable to receive the sequence index when error detected.

Return values

error – code when IP command error detected.

```
static inline flexspi_ahb_error_code_t FLEXSPI_GetAHBCommandErrorCode(FLEXSPI_Type *base, uint8_t *index)
```

Gets the error code when AHB command error detected.

Parameters

- base – FLEXSPI peripheral base address.

- `index` – Pointer to a `uint8_t` type variable to receive the sequence index when error detected.

Return values

`error` – code when AHB command error detected.

```
static inline bool FLEXSPI_GetBusIdleStatus(FLEXSPI_Type *base)
```

Returns whether the bus is idle.

Parameters

- `base` – FLEXSPI peripheral base address.

Return values

- `true` – Bus is idle.
- `false` – Bus is busy.

```
void FLEXSPI_UpdateRxSampleClock(FLEXSPI_Type *base, flexspi_read_sample_clock_t  
clockSource)
```

Update read sample clock source.

Parameters

- `base` – FLEXSPI peripheral base address.
- `clockSource` – `clockSource` of type `flexspi_read_sample_clock_t`

```
void FLEXSPI_UpdateLUT(FLEXSPI_Type *base, uint32_t index, const uint32_t *cmd, uint32_t  
count)
```

Updates the LUT table.

Parameters

- `base` – FLEXSPI peripheral base address.
- `index` – From which index start to update. It could be any index of the LUT table, which also allows user to update command content inside a command. Each command consists of up to 8 instructions and occupy 4*32-bit memory.
- `cmd` – Command sequence array.
- `count` – Number of sequences.

```
static inline void FLEXSPI_WriteData(FLEXSPI_Type *base, uint32_t data, uint8_t fifoIndex)
```

Writes data into FIFO.

Parameters

- `base` – FLEXSPI peripheral base address
- `data` – The data bytes to send
- `fifoIndex` – Destination fifo index.

```
static inline uint32_t FLEXSPI_ReadData(FLEXSPI_Type *base, uint8_t fifoIndex)
```

Receives data from data FIFO.

Parameters

- `base` – FLEXSPI peripheral base address
- `fifoIndex` – Source fifo index.

Returns

The data in the FIFO.

status_t FLEXSPI_WriteBlocking(FLEXSPI_Type *base, uint8_t *buffer, size_t size)

Sends a buffer of data bytes using blocking method.

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

Parameters

- base – FLEXSPI peripheral base address
- buffer – The data bytes to send
- size – The number of data bytes to send

Return values

- kStatus_Success – write success without error
- kStatus_FLEXSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_FLEXSPI_IpCommandSequenceError – IP command sequence error detected
- kStatus_FLEXSPI_IpCommandGrantTimeout – IP command grant timeout detected

status_t FLEXSPI_ReadBlocking(FLEXSPI_Type *base, uint8_t *buffer, size_t size)

Receives a buffer of data bytes using a blocking method.

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

Parameters

- base – FLEXSPI peripheral base address
- buffer – The data bytes to send
- size – The number of data bytes to receive

Return values

- kStatus_Success – read success without error
- kStatus_FLEXSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_FLEXSPI_IpCommandSequenceError – IP command sequence error detected
- kStatus_FLEXSPI_IpCommandGrantTimeout – IP command grant timeout detected

status_t FLEXSPI_TransferBlocking(FLEXSPI_Type *base, *flexspi_transfer_t* *xfer)

Execute command to transfer a buffer data bytes using a blocking method.

Parameters

- base – FLEXSPI peripheral base address
- xfer – pointer to the transfer structure.

Return values

- kStatus_Success – command transfer success without error

- `kStatus_FLEXSPI_SequenceExecutionTimeout` – sequence execution timeout
- `kStatus_FLEXSPI_IpCommandSequenceError` – IP command sequence error detected
- `kStatus_FLEXSPI_IpCommandGrantTimeout` – IP command grant timeout detected

`void FLEXSPI_TransferCreateHandle(FLEXSPI_Type *base, flexspi_handle_t *handle, flexspi_transfer_callback_t callback, void *userData)`

Initializes the FLEXSPI handle which is used in transactional functions.

Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – pointer to `flexspi_handle_t` structure to store the transfer state.
- `callback` – pointer to user callback function.
- `userData` – user parameter passed to the callback function.

`status_t FLEXSPI_TransferNonBlocking(FLEXSPI_Type *base, flexspi_handle_t *handle, flexspi_transfer_t *xfer)`

Performs a interrupt non-blocking transfer on the FLEXSPI bus.

Note: Calling the API returns immediately after transfer initiates. The user needs to call `FLEXSPI_GetTransferCount` to poll the transfer status to check whether the transfer is finished. If the return status is not `kStatus_FLEXSPI_Busy`, the transfer is finished. For `FLEXSPI_Read`, the `dataSize` should be multiple of rx watermark level, or FLEXSPI could not read data properly.

Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – pointer to `flexspi_handle_t` structure which stores the transfer state.
- `xfer` – pointer to `flexspi_transfer_t` structure.

Return values

- `kStatus_Success` – Successfully start the data transmission.
- `kStatus_FLEXSPI_Busy` – Previous transmission still not finished.

`status_t FLEXSPI_TransferGetCount(FLEXSPI_Type *base, flexspi_handle_t *handle, size_t *count)`

Gets the master transfer status during a interrupt non-blocking transfer.

Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – pointer to `flexspi_handle_t` structure which stores the transfer state.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_InvalidArgument` – count is Invalid.
- `kStatus_Success` – Successfully return the count.

void FLEXSPI_TransferAbort(FLEXSPI_Type *base, *flexspi_handle_t* *handle)

Aborts an interrupt non-blocking transfer early.

Note: This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to *flexspi_handle_t* structure which stores the transfer state

void FLEXSPI_TransferHandleIRQ(FLEXSPI_Type *base, *flexspi_handle_t* *handle)

Master interrupt handler.

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to *flexspi_handle_t* structure.

FSL_FLEXSPI_DRIVER_VERSION

FLEXSPI driver version.

Status structure of FLEXSPI.

Values:

enumerator kStatus_FLEXSPI_Busy

FLEXSPI is busy

enumerator kStatus_FLEXSPI_SequenceExecutionTimeout

Sequence execution timeout error occurred during FLEXSPI transfer.

enumerator kStatus_FLEXSPI_IpCommandSequenceError

IP command Sequence execution timeout error occurred during FLEXSPI transfer.

enumerator kStatus_FLEXSPI_IpCommandGrantTimeout

IP command grant timeout error occurred during FLEXSPI transfer.

CMD definition of FLEXSPI, use to form LUT instruction, *_flexspi_command*.

Values:

enumerator kFLEXSPI_Command_STOP

Stop execution, deassert CS.

enumerator kFLEXSPI_Command_SDR

Transmit Command code to Flash, using SDR mode.

enumerator kFLEXSPI_Command_RADDR_SDR

Transmit Row Address to Flash, using SDR mode.

enumerator kFLEXSPI_Command_CADDR_SDR

Transmit Column Address to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE1_SDR

Transmit 1-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE2_SDR
Transmit 2-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE4_SDR
Transmit 4-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE8_SDR
Transmit 8-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_WRITE_SDR
Transmit Programming Data to Flash, using SDR mode.

enumerator kFLEXSPI_Command_READ_SDR
Receive Read Data from Flash, using SDR mode.

enumerator kFLEXSPI_Command_LEARN_SDR
Receive Read Data or Preamble bit from Flash, SDR mode.

enumerator kFLEXSPI_Command_DATSZ_SDR
Transmit Read/Program Data size (byte) to Flash, SDR mode.

enumerator kFLEXSPI_Command_DUMMY_SDR
Leave data lines undriven by FlexSPI controller.

enumerator kFLEXSPI_Command_DUMMY_RWDS_SDR
Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

enumerator kFLEXSPI_Command_DDR
Transmit Command code to Flash, using DDR mode.

enumerator kFLEXSPI_Command_RADDR_DDR
Transmit Row Address to Flash, using DDR mode.

enumerator kFLEXSPI_Command_CADDR_DDR
Transmit Column Address to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE1_DDR
Transmit 1-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE2_DDR
Transmit 2-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE4_DDR
Transmit 4-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE8_DDR
Transmit 8-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_WRITE_DDR
Transmit Programming Data to Flash, using DDR mode.

enumerator kFLEXSPI_Command_READ_DDR
Receive Read Data from Flash, using DDR mode.

enumerator kFLEXSPI_Command_LEARN_DDR
Receive Read Data or Preamble bit from Flash, DDR mode.

enumerator kFLEXSPI_Command_DATSZ_DDR
Transmit Read/Program Data size (byte) to Flash, DDR mode.

enumerator kFLEXSPI_Command_DUMMY_DDR
Leave data lines undriven by FlexSPI controller.

enumerator kFLEXSPI_Command_DUMMY_RWDS_DDR

Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

enumerator kFLEXSPI_Command_JUMP_ON_CS

Stop execution, deassert CS and save operand[7:0] as the instruction start pointer for next sequence

enum _flexspi_pad

pad definition of FLEXSPI, use to form LUT instruction.

Values:

enumerator kFLEXSPI_1PAD

Transmit command/address and transmit/receive data only through DATA0/DATA1.

enumerator kFLEXSPI_2PAD

Transmit command/address and transmit/receive data only through DATA[1:0].

enumerator kFLEXSPI_4PAD

Transmit command/address and transmit/receive data only through DATA[3:0].

enumerator kFLEXSPI_8PAD

Transmit command/address and transmit/receive data only through DATA[7:0].

enum _flexspi_flags

FLEXSPI interrupt status flags.

Values:

enumerator kFLEXSPI_SequenceExecutionTimeoutFlag

Sequence execution timeout.

enumerator kFLEXSPI_AhbBusErrorFlag

AHB Bus error flag.

enumerator kFLEXSPI_SckStoppedBecauseTxEmptyFlag

SCK is stopped during command sequence because Async TX FIFO empty.

enumerator kFLEXSPI_SckStoppedBecauseRxFullFlag

SCK is stopped during command sequence because Async RX FIFO full.

enumerator kFLEXSPI_IpTxFifoWatermarkEmptyFlag

IP TX FIFO WaterMark empty.

enumerator kFLEXSPI_IpRxFifoWatermarkAvailableFlag

IP RX FIFO WaterMark available.

enumerator kFLEXSPI_AhbCommandSequenceErrorFlag

AHB triggered Command Sequences Error.

enumerator kFLEXSPI_IpCommandSequenceErrorFlag

IP triggered Command Sequences Error.

enumerator kFLEXSPI_AhbCommandGrantTimeoutFlag

AHB triggered Command Sequences Grant Timeout.

enumerator kFLEXSPI_IpCommandGrantTimeoutFlag

IP triggered Command Sequences Grant Timeout.

enumerator kFLEXSPI_IpCommandExecutionDoneFlag

IP triggered Command Sequences Execution finished.

enumerator kFLEXSPI_AllInterruptFlags

All flags.

enum _flexspi_read_sample_clock

FLEXSPI sample clock source selection for Flash Reading.

Values:

enumerator kFLEXSPI_ReadSampleClkLoopbackInternally

Dummy Read strobe generated by FlexSPI Controller and loopback internally.

enumerator kFLEXSPI_ReadSampleClkLoopbackFromDqsPad

Dummy Read strobe generated by FlexSPI Controller and loopback from DQS pad.

enumerator kFLEXSPI_ReadSampleClkLoopbackFromSckPad

SCK output clock and loopback from SCK pad.

enumerator kFLEXSPI_ReadSampleClkExternalInputFromDqsPad

Flash provided Read strobe and input from DQS pad.

enum _flexspi_cs_interval_cycle_unit

FLEXSPI interval unit for flash device select.

Values:

enumerator kFLEXSPI_CsIntervalUnit1SckCycle

Chip selection interval: CSINTERVAL * 1 serial clock cycle.

enumerator kFLEXSPI_CsIntervalUnit256SckCycle

Chip selection interval: CSINTERVAL * 256 serial clock cycle.

enum _flexspi_ahb_write_wait_unit

FLEXSPI AHB wait interval unit for writing.

Values:

enumerator kFLEXSPI_AhbWriteWaitUnit2AhbCycle

AWRWAIT unit is 2 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit8AhbCycle

AWRWAIT unit is 8 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit32AhbCycle

AWRWAIT unit is 32 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit128AhbCycle

AWRWAIT unit is 128 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit512AhbCycle

AWRWAIT unit is 512 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit2048AhbCycle

AWRWAIT unit is 2048 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit8192AhbCycle

AWRWAIT unit is 8192 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit32768AhbCycle

AWRWAIT unit is 32768 ahb clock cycle.

enum _flexspi_ip_error_code

Error Code when IP command Error detected.

Values:

enumerator kFLEXSPI_IpCmdErrorNoError

No error.

enumerator kFLEXSPI_IpCmdErrorJumpOnCsInIpCmd

IP command with JMP_ON_CS instruction used.

enumerator kFLEXSPI_IpCmdErrorUnknownOpCode

Unknown instruction opcode in the sequence.

enumerator kFLEXSPI_IpCmdErrorSdrDummyInDdrSequence

Instruction DUMMY_SDR/DUMMY_RWDS_SDR used in DDR sequence.

enumerator kFLEXSPI_IpCmdErrorDdrDummyInSdrSequence

Instruction DUMMY_DDR/DUMMY_RWDS_DDR used in SDR sequence.

enumerator kFLEXSPI_IpCmdErrorInvalidAddress

Flash access start address exceed the whole flash address range (A1/A2/B1/B2).

enumerator kFLEXSPI_IpCmdErrorSequenceExecutionTimeout

Sequence execution timeout.

enumerator kFLEXSPI_IpCmdErrorFlashBoundaryAcrosss

Flash boundary crossed.

enum _flexspi_ahb_error_code

Error Code when AHB command Error detected.

Values:

enumerator kFLEXSPI_AhbCmdErrorNoError

No error.

enumerator kFLEXSPI_AhbCmdErrorJumpOnCsInWriteCmd

AHB Write command with JMP_ON_CS instruction used in the sequence.

enumerator kFLEXSPI_AhbCmdErrorUnknownOpCode

Unknown instruction opcode in the sequence.

enumerator kFLEXSPI_AhbCmdErrorSdrDummyInDdrSequence

Instruction DUMMY_SDR/DUMMY_RWDS_SDR used in DDR sequence.

enumerator kFLEXSPI_AhbCmdErrorDdrDummyInSdrSequence

Instruction DUMMY_DDR/DUMMY_RWDS_DDR used in SDR sequence.

enumerator kFLEXSPI_AhbCmdSequenceExecutionTimeout

Sequence execution timeout.

enum _flexspi_port

FLEXSPI operation port select.

Values:

enumerator kFLEXSPI_PortA1

Access flash on A1 port.

enumerator kFLEXSPI_PortA2

Access flash on A2 port.

enumerator kFLEXSPI_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_ahbBuffers_ctrl* flexspi_ahbBuffers_ctrl_t

Structure to control all AHB buffers.

```
typedef struct _flexspi_config flexspi_config_t
```

FLEXSPI configuration structure.

```
typedef struct _flexspi_device_config flexspi_device_config_t
```

External device configuration items.

```
typedef struct _flexspi_transfer flexspi_transfer_t
```

Transfer structure for FLEXSPI.

```
typedef struct _flexspi_handle flexspi_handle_t
```

```
typedef void (*flexspi_transfer_callback_t)(FLEXSPI_Type *base, flexspi_handle_t *handle,  
status_t status, void *userData)
```

FLEXSPI transfer callback function.

```
typedef struct _flexspi_addr_map_config flexspi_addr_map_config_t
```

Address mapping configuration structure.

```
FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNT
```

```
FLEXSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)
```

Formula to form FLEXSPI instructions in LUT table.

```
struct _flexspi_ahbBuffer_config
```

```
#include <fsl_flexspi.h>
```

Public Members

```
uint8_t priority
```

This priority for AHB Master Read which this AHB RX Buffer is assigned.

```
uint8_t masterIndex
```

AHB Master ID the AHB RX Buffer is assigned.

```
uint16_t bufferSize
```

AHB buffer size in byte.

```
bool enablePrefetch
```

AHB Read Prefetch Enable for current AHB RX Buffer corresponding Master, allows prefetch disable/enable separately for each master.

```
struct _flexspi_ahbBuffers_ctrl
```

```
#include <fsl_flexspi.h> Structure to control all AHB buffers.
```

Public Members

```
flexspi_ahbBuffer_config_t buffer[FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNTn(0)]
```

Configurations of all AHB buffers.

```
struct _flexspi_config
```

```
#include <fsl_flexspi.h> FLEXSPI configuration structure.
```

Public Members

```
flexspi_read_sample_clock_t rxSampleClock
```

Sample Clock source selection for Flash Reading.

```
bool enableSckFreeRunning
```

Enable/disable SCK output free-running.

`bool enableDoze`
Enable/disable doze mode support.

`bool enableHalfSpeedAccess`
Enable/disable divide by 2 of the clock for half speed commands.

`bool enableSameConfigForAll`
Enable/disable same configuration for all connected devices when enabled, same configuration in FLASHA1CRx is applied to all.

`uint16_t seqTimeoutCycle`
Timeout wait cycle for command sequence execution, timeout after `ahbGrantTimeoutCycle*1024` serial root clock cycles.

`uint8_t ipGrantTimeoutCycle`
Timeout wait cycle for IP command grant, timeout after `ipGrantTimeoutCycle*1024` AHB clock cycles.

`uint8_t txWatermark`
FLEXSPI IP transmit watermark value.

`uint8_t rxWatermark`
FLEXSPI receive watermark value.

`struct _flexspi_device_config`
#include <fsl_flexspi.h> External device configuration items.

Public Members

`uint32_t flexspiRootClk`
FLEXSPI serial root clock.

`bool isSck2Enabled`
FLEXSPI use SCK2.

`uint32_t flashSize`
Flash size in KByte.

`bool addressShift`
Address shift.

flexspi_cs_interval_cycle_unit_t CSIntervalUnit
CS interval unit, 1 or 256 cycle.

`uint16_t CSInterval`
CS line assert interval, multiply CS interval unit to get the CS line assert interval cycles.

`uint8_t CSHoldTime`
CS line hold time.

`uint8_t CSSetupTime`
CS line setup time.

`uint8_t dataValidTime`
Data valid time for external device.

`uint8_t columnSpace`
Column space size.

`bool enableWordAddress`
If enable word address.

uint8_t AWRSeqIndex

Sequence ID for AHB write command.

uint8_t AWRSeqNumber

Sequence number for AHB write command.

uint8_t ARDSeqIndex

Sequence ID for AHB read command.

uint8_t ARDSeqNumber

Sequence number for AHB read command.

flexspi_ahb_write_wait_unit_t AHBWriteWaitUnit

AHB write wait unit.

uint16_t AHBWriteWaitInterval

AHB write wait interval, multiply AHB write interval unit to get the AHB write wait cycles.

bool enableWriteMask

Enable/Disable FLEXSPI drive DQS pin as write mask when writing to external device.

struct *_flexspi_transfer*

#include <fsl_flexspi.h> Transfer structure for FLEXSPI.

Public Members

uint32_t deviceAddress

Operation device address.

flexspi_port_t port

Operation port.

flexspi_command_type_t cmdType

Execution command type.

uint8_t seqIndex

Sequence ID for command.

uint8_t SeqNumber

Sequence number for command.

uint32_t *data

Data buffer.

size_t dataSize

Data size in bytes.

struct *_flexspi_handle*

#include <fsl_flexspi.h> Transfer handle structure for FLEXSPI.

Public Members

uint32_t state

Internal state for FLEXSPI transfer

uint8_t *data

Data buffer.

size_t dataSize

Remaining Data size in bytes.

size_t transferTotalSize

Total Data size in bytes.

flexspi_transfer_callback_t completionCallback

Callback for users while transfer finish or error occurred

void *userData

FLEXSPI callback function parameter.

struct *_flexspi_addr_map_config*

#include <fsl_flexspi.h> Address mapping configuration structure.

Public Members

uint32_t addrStart

Remapping start address.

uint32_t addrEnd

Remapping end address.

uint32_t addrOffset

Address offset.

bool remapEnable

Enable address remapping.

struct ahbConfig

Public Members

uint8_t ahbGrantTimeoutCycle

Timeout wait cycle for AHB command grant, timeout after ahbGrantTimeoutCycle*1024 AHB clock cycles.

uint16_t ahbBusTimeoutCycle

Timeout wait cycle for AHB read/write access, timeout after ahbBusTimeoutCycle*1024 AHB clock cycles.

uint8_t resumeWaitCycle

Wait cycle for idle state before suspended command sequence resume, timeout after ahbBusTimeoutCycle AHB clock cycles.

flexspi_ahbBuffer_config_t buffer[FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNTn(0)]

AHB buffer size.

bool enableClearAHBBufferOpt

Enable/disable automatically clean AHB RX Buffer and TX Buffer when FLEXSPI returns STOP mode ACK.

bool enableReadAddressOpt

Enable/disable remove AHB read burst start address alignment limitation. when enable, there is no AHB read burst start address alignment limitation.

bool enableAHBPrefetch

Enable/disable AHB read prefetch feature, when enabled, FLEXSPI will fetch more data than current AHB burst.

bool enableAHBBufferable

Enable/disable AHB bufferable write access support, when enabled, FLEXSPI return before waiting for command execution finished.

bool enableAHBCachable

Enable AHB bus cachable read access support.

2.25 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_EDMASize1Bytes(one byte).

```
status_t FLEXSPI_TransferEDMA(FLEXSPI_Type *base, flexspi_edma_handle_t *handle,
                              flexspi_transfer_t *xfer)
```

Transfers FLEXSPI data using an eDMA non-blocking method.

This function writes/receives data to/from the FLEXSPI transmit/receive FIFO. This function is non-blocking.

Parameters

- base – FLEXSPI peripheral base address.
- handle – Pointer to flexspi_edma_handle_t structure
- xfer – FLEXSPI transfer structure.

Return values

- `kStatus_FLEXSPI_Busy` – FLEXSPI is busy transfer.
- `kStatus_InvalidArgument` – The watermark configuration is invalid, the watermark should be power of 2 to do successfully EDMA transfer.
- `kStatus_Success` – FLEXSPI successfully start edma transfer.

`void FLEXSPI_TransferAbortEDMA(FLEXSPI_Type *base, flexspi_edma_handle_t *handle)`
 Aborts the transfer data using eDMA.

This function aborts the transfer data using eDMA.

Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – Pointer to `flexspi_edma_handle_t` structure

`status_t FLEXSPI_TransferGetTransferCountEDMA(FLEXSPI_Type *base, flexspi_edma_handle_t *handle, size_t *count)`

Gets the transferred counts of transfer.

Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – Pointer to `flexspi_edma_handle_t` structure.
- `count` – Bytes transfer.

Return values

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

`FSL_FLEXSPI_EDMA_DRIVER_VERSION`
 FLEXSPI EDMA driver version.

FLEXSPI EDMA driver.

`FSL_FLEXSPI_EDMA_DRIVER_VERSION`
 FLEXSPI EDMA driver.

`enum _flexspi_edma_ntransfer_size`
 eDMA transfer configuration

Values:

enumerator `kFLEXPSI_EDMAAnSize1Bytes`

Source/Destination data transfer size is 1 byte every time

enumerator `kFLEXPSI_EDMAAnSize2Bytes`

Source/Destination data transfer size is 2 bytes every time

enumerator `kFLEXPSI_EDMAAnSize4Bytes`

Source/Destination data transfer size is 4 bytes every time

enumerator `kFLEXPSI_EDMAAnSize8Bytes`

Source/Destination data transfer size is 8 bytes every time

enumerator `kFLEXPSI_EDMAAnSize32Bytes`

Source/Destination data transfer size is 32 bytes every time

`enum _flexspi_edma_ntransfer_size`
 eDMA transfer configuration

Values:

enumerator kFLEXPSI_EDMA_nSize1Bytes

Source/Destination data transfer size is 1 byte every time

enumerator kFLEXPSI_EDMA_nSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kFLEXPSI_EDMA_nSize4Bytes

Source/Destination data transfer size is 4 bytes every time

enumerator kFLEXPSI_EDMA_nSize8Bytes

Source/Destination data transfer size is 8 bytes every time

enumerator kFLEXPSI_EDMA_nSize32Bytes

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

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

2.26 I3C: I3C Driver

FSL_I3C_DRIVER_VERSION

I3C driver version.

I3C status return codes.

Values:

enumerator kStatus_I3C_Busy

The master is already performing a transfer.

enumerator kStatus_I3C_Idle

The slave driver is idle.

enumerator kStatus_I3C_Nak

The slave device sent a NAK in response to an address.

enumerator kStatus_I3C_WriteAbort

The slave device sent a NAK in response to a write.

enumerator kStatus_I3C_Term

The master terminates slave read.

enumerator kStatus_I3C_HdrParityError

Parity error from DDR read.

enumerator kStatus_I3C_CrcError

CRC error from DDR read.

enumerator kStatus_I3C_ReadFifoError

Read from M/SRDATAB register when FIFO empty.

enumerator kStatus_I3C_WriteFifoError

Write to M/SWDATAB register when FIFO full.

enumerator kStatus_I3C_MsgError

Message SDR/DDR mismatch or read/write message in wrong state

enumerator kStatus_I3C_InvalidReq

Invalid use of request.

enumerator kStatus_I3C_Timeout

The module has stalled too long in a frame.

enumerator kStatus_I3C_SlaveCountExceed

The I3C slave count has exceed the definition in I3C_MAX_DEVCNT.

enumerator kStatus_I3C_IBIWon

The I3C slave event IBI or MR or HJ won the arbitration on a header address.

enumerator kStatus_I3C_OverrunError

Slave internal from-bus buffer/FIFO overrun.

enumerator kStatus_I3C_UnderrunError

Slave internal to-bus buffer/FIFO underrun

enumerator kStatus_I3C_UnderrunNak

Slave internal from-bus buffer/FIFO underrun and NACK error

enumerator kStatus_I3C_InvalidStart

Slave invalid start flag

enumerator kStatus_I3C_SdrParityError

SDR parity error

enumerator kStatus_I3C_S0S1Error

S0 or S1 error

enum _i3c_hdr_mode

I3C HDR modes.

Values:

enumerator kI3C_HDRModeNone

enumerator kI3C_HDRModeDDR

enumerator kI3C_HDRModeTSP

enumerator kI3C_HDRModeTSL

typedef enum _i3c_hdr_mode i3c_hdr_mode_t

I3C HDR modes.

typedef struct _i3c_device_info i3c_device_info_t

I3C device information.

I3C_RETRY_TIMES

Max loops to wait for I3C operation status complete.

This is the maximum number of loops to wait for I3C operation status complete. If set to 0, it will wait indefinitely.

I3C_MAX_DEVCNT

I3C_IBI_BUFF_SIZE

struct _i3c_device_info

#include <fsl_i3c.h> I3C device information.

Public Members

uint8_t dynamicAddr

Device dynamic address.

uint8_t staticAddr

Static address.

uint8_t dcr

Device characteristics register information.

uint8_t bcr

Bus characteristics register information.

uint16_t vendorID

Device vendor ID(manufacture ID).

uint32_t partNumber
Device part number info

uint16_t maxReadLength
Maximum read length.

uint16_t maxWriteLength
Maximum write length.

uint8_t hdrMode
Support hdr mode, could be OR logic in i3c_hdr_mode.

2.27 I3C Common Driver

typedef struct *i3c_config* i3c_config_t

Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

uint32_t I3C_GetInstance(I3C_Type *base)

Get which instance current I3C is used.

Parameters

- base – The I3C peripheral base address.

void I3C_GetDefaultConfig(*i3c_config_t* *config)

Provides a default configuration for the I3C peripheral, the configuration covers both master functionality and slave functionality.

This function provides the following default configuration for I3C:

```
config->enableMaster      = kI3C_MasterCapable;
config->disableTimeout    = false;
config->hKeep             = kI3C_MasterHighKeeperNone;
config->enableOpenDrainStop = true;
config->enableOpenDrainHigh = true;
config->baudRate_Hz.i2cBaud = 400000U;
config->baudRate_Hz.i3cPushPullBaud = 12500000U;
config->baudRate_Hz.i3cOpenDrainBaud = 2500000U;
config->masterDynamicAddress = 0x0AU;
config->slowClock_Hz       = 1000000U;
config->enableSlave        = true;
config->vendorID           = 0x11BU;
config->enableRandomPart   = false;
config->partNumber         = 0;
config->dcr                = 0;
config->bcr = 0;
config->hdrMode            = (uint8_t)kI3C_HDRModeDDR;
config->nakAllRequest       = false;
config->ignoreS0S1Error    = false;
config->offline            = false;
config->matchSlaveStartStop = false;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the common I3C driver with I3C_Init().

Parameters

- `config` – **[out]** User provided configuration structure for default values. Refer to `i3c_config_t`.

`void I3C_Init(I3C_Type *base, const i3c_config_t *config, uint32_t sourceClock_Hz)`

Initializes the I3C peripheral. This function enables the peripheral clock and initializes the I3C peripheral as described by the user provided configuration. This will initialize both the master peripheral and slave peripheral so that I3C module could work as pure master, pure slave or secondary master, etc. A software reset is performed prior to configuration.

Parameters

- `base` – The I3C peripheral base address.
- `config` – User provided peripheral configuration. Use `I3C_GetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

`struct _i3c_config`

#include <fsl_i3c.h> Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_GetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

i3c_master_enable_t enableMaster

Enable master mode.

`bool` disableTimeout

Whether to disable timeout to prevent the ERRWARN.

i3c_master_hkeep_t hKeep

High keeper mode setting.

`bool` enableOpenDrainStop

Whether to emit open-drain speed STOP.

`bool` enableOpenDrainHigh

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

i3c_baudrate_hz_t baudRate_Hz

Desired baud rate settings.

i3c_start_scl_delay_t startSclDelay

I3C SCL delay after START.

i3c_start_scl_delay_t restartSclDelay

I3C SCL delay after Repeated START.

`uint8_t` masterDynamicAddress

Main master dynamic address configuration.

`uint32_t` maxWriteLength

Maximum write length.

`uint32_t` `maxReadLength`
Maximum read length.

`bool` `enableSlave`
Whether to enable slave.

`uint8_t` `staticAddr`
Static address.

`uint16_t` `vendorID`
Device vendor ID(manufacture ID).

`uint32_t` `partNumber`
Device part number info

`uint8_t` `dcr`
Device characteristics register information.

`uint8_t` `bcr`
Bus characteristics register information.

`uint8_t` `hdrMode`
Support hdr mode, could be OR logic in enumeration:`i3c_hdr_mode_t`.

`bool` `nakAllRequest`
Whether to reply NAK to all requests except broadcast CCC.

`bool` `ignoreS0S1Error`
Whether to ignore S0/S1 error in SDR mode.

`bool` `offline`
Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

`bool` `matchSlaveStartStop`
Whether to assert start/stop status only the time slave is addressed.

2.28 I3C Master Driver

`void` `I3C_MasterGetDefaultConfig(i3c_master_config_t *masterConfig)`

Provides a default configuration for the I3C master peripheral.

This function provides the following default configuration for the I3C master peripheral:

```
masterConfig->enableMaster      = kI3C_MasterOn;
masterConfig->disableTimeout    = false;
masterConfig->hKeep             = kI3C_MasterHighKeeperNone;
masterConfig->enableOpenDrainStop = true;
masterConfig->enableOpenDrainHigh = true;
masterConfig->baudRate_Hz       = 100000U;
masterConfig->busType            = kI3C_TypeI2C;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with `I3C_MasterInit()`.

Parameters

- `masterConfig` – **[out]** User provided configuration structure for default values. Refer to `i3c_master_config_t`.

```
void I3C_MasterInit(I3C_Type *base, const i3c_master_config_t *masterConfig, uint32_t
                    sourceClock_Hz)
```

Initializes the I3C master peripheral.

This function enables the peripheral clock and initializes the I3C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- `base` – The I3C peripheral base address.
- `masterConfig` – User provided peripheral configuration. Use `I3C_MasterGetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

```
void I3C_MasterDeinit(I3C_Type *base)
```

Deinitializes the I3C master peripheral.

This function disables the I3C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- `base` – The I3C peripheral base address.

```
status_t I3C_MasterCheckAndClearError(I3C_Type *base, uint32_t status)
```

```
status_t I3C_MasterWaitForCtrlDone(I3C_Type *base, bool waitIdle)
```

```
status_t I3C_CheckForBusyBus(I3C_Type *base)
```

```
static inline void I3C_MasterEnable(I3C_Type *base, i3c_master_enable_t enable)
```

Set I3C module master mode.

Parameters

- `base` – The I3C peripheral base address.
- `enable` – Enable master mode.

```
void I3C_SlaveGetDefaultConfig(i3c_slave_config_t *slaveConfig)
```

Provides a default configuration for the I3C slave peripheral.

This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableslave = true;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with `I3C_SlaveInit()`.

Parameters

- `slaveConfig` – **[out]** User provided configuration structure for default values. Refer to `i3c_slave_config_t`.

```
void I3C_SlaveInit(I3C_Type *base, const i3c_slave_config_t *slaveConfig, uint32_t
                  slowClock_Hz)
```

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

Parameters

- `base` – The I3C peripheral base address.
- `slaveConfig` – User provided peripheral configuration. Use `I3C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- `slowClock_Hz` – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If `FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH` defines as 1, this parameter is useless.

`void I3C_SlaveDeinit(I3C_Type *base)`

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

Parameters

- `base` – The I3C peripheral base address.

`static inline void I3C_SlaveEnable(I3C_Type *base, bool isEnabled)`

Enable/Disable Slave.

Parameters

- `base` – The I3C peripheral base address.
- `isEnabled` – Enable or disable.

`static inline uint32_t I3C_MasterGetStatusFlags(I3C_Type *base)`

Gets the I3C master status flags.

A bit mask with the state of all I3C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_master_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

`static inline void I3C_MasterClearStatusFlags(I3C_Type *base, uint32_t statusMask)`

Clears the I3C master status flag state.

The following status register flags can be cleared:

- `kI3C_MasterSlaveStartFlag`
- `kI3C_MasterControlDoneFlag`
- `kI3C_MasterCompleteFlag`
- `kI3C_MasterArbitrationWonFlag`
- `kI3C_MasterSlave2MasterFlag`

Attempts to clear other flags has no effect.

See also:

`_i3c_master_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_master_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_MasterGetStatusFlags()`.

```
static inline uint32_t I3C_MasterGetErrorStatusFlags(I3C_Type *base)
```

Gets the I3C master error status flags.

A bit mask with the state of all I3C master error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_master_error_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_MasterClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C master error status flag state.

See also:

`_i3c_master_error_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_master_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_MasterGetStatusFlags()`.

```
i3c_master_state_t I3C_MasterGetState(I3C_Type *base)
```

Gets the I3C master state.

Parameters

- `base` – The I3C peripheral base address.

Returns

I3C master state.

```
static inline uint32_t I3C_SlaveGetStatusFlags(I3C_Type *base)
```

Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_slave_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_SlaveClearStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C slave status flag state.

The following status register flags can be cleared:

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`

Attempts to clear other flags has no effect.

See also:

`_i3c_slave_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetStatusFlags()`.

```
static inline uint32_t I3C_SlaveGetErrorStatusFlags(I3C_Type *base)
```

Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_slave_error_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_SlaveClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C slave error status flag state.

See also:

`_i3c_slave_error_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_slave_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetErrorStatusFlags()`.

`i3c_slave_activity_state_t` `I3C_SlaveGetActivityState(I3C_Type *base)`

Gets the I3C slave state.

Parameters

- `base` – The I3C peripheral base address.

Returns

I3C slave activity state, refer `i3c_slave_activity_state_t`.

`status_t` `I3C_SlaveCheckAndClearError(I3C_Type *base, uint32_t status)`

`static inline void` `I3C_MasterEnableInterrupts(I3C_Type *base, uint32_t interruptMask)`

Enables the I3C master interrupt requests.

All flags except `kI3C_MasterBetweenFlag` and `kI3C_MasterNackDetectFlag` can be enabled as interrupts.

Parameters

- `base` – The I3C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_i3c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline void` `I3C_MasterDisableInterrupts(I3C_Type *base, uint32_t interruptMask)`

Disables the I3C master interrupt requests.

All flags except `kI3C_MasterBetweenFlag` and `kI3C_MasterNackDetectFlag` can be enabled as interrupts.

Parameters

- `base` – The I3C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_i3c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline uint32_t` `I3C_MasterGetEnabledInterrupts(I3C_Type *base)`

Returns the set of currently enabled I3C master interrupt requests.

Parameters

- `base` – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_master_flags` enumerators OR'd together to indicate the set of enabled interrupts.

`static inline uint32_t` `I3C_MasterGetPendingInterrupts(I3C_Type *base)`

Returns the set of pending I3C master interrupt requests.

Parameters

- `base` – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_master_flags` enumerators OR'd together to indicate the set of pending interrupts.

static inline void I3C_SlaveEnableInterrupts(I3C_Type *base, uint32_t interruptMask)

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`
- `kI3C_SlaveErrorFlag`
- `kI3C_SlaveHDRCommandMatchFlag`
- `kI3C_SlaveCCCHandledFlag`
- `kI3C_SlaveEventSentFlag`

Parameters

- `base` – The I3C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline void I3C_SlaveDisableInterrupts(I3C_Type *base, uint32_t interruptMask)

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`
- `kI3C_SlaveErrorFlag`
- `kI3C_SlaveHDRCommandMatchFlag`
- `kI3C_SlaveCCCHandledFlag`
- `kI3C_SlaveEventSentFlag`

Parameters

- `base` – The I3C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline uint32_t I3C_SlaveGetEnabledInterrupts(I3C_Type *base)
```

Returns the set of currently enabled I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

```
static inline uint32_t I3C_SlaveGetPendingInterrupts(I3C_Type *base)
```

Returns the set of pending I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of pending interrupts.

```
static inline void I3C_MasterEnableDMA(I3C_Type *base, bool enableTx, bool enableRx,  
                                       uint32_t width)
```

Enables or disables I3C master DMA requests.

Parameters

- base – The I3C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.
- width – DMA read/write unit in bytes.

```
static inline uint32_t I3C_MasterGetTxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C master transmit data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Master Transmit Data Register address.

```
static inline uint32_t I3C_MasterGetRxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C master receive data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Master Receive Data Register address.

```
static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t  
                                       width)
```

Enables or disables I3C slave DMA requests.

Parameters

- base – The I3C peripheral base address.

- `enableTx` – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- `enableRx` – Enable flag for receive DMA request. Pass true for enable, false for disable.
- `width` – DMA read/write unit in bytes.

static inline uint32_t I3C_SlaveGetTxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C slave transmit data register address for DMA transfer.

Parameters

- `base` – The I3C peripheral base address.
- `width` – DMA read/write unit in bytes.

Returns

The I3C Slave Transmit Data Register address.

static inline uint32_t I3C_SlaveGetRxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C slave receive data register address for DMA transfer.

Parameters

- `base` – The I3C peripheral base address.
- `width` – DMA read/write unit in bytes.

Returns

The I3C Slave Receive Data Register address.

static inline void I3C_MasterSetWatermarks(I3C_Type *base, *i3c_tx_trigger_level_t* txLvl,
i3c_rx_trigger_level_t rxLvl, bool flushTx, bool flushRx)

Sets the watermarks for I3C master FIFOs.

Parameters

- `base` – The I3C peripheral base address.
- `txLvl` – Transmit FIFO watermark level. The `kI3C_MasterTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches `txLvl`.
- `rxLvl` – Receive FIFO watermark level. The `kI3C_MasterRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches `rxLvl`.
- `flushTx` – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- `flushRx` – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

static inline void I3C_MasterGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of bytes in the I3C master FIFOs.

Parameters

- `base` – The I3C peripheral base address.
- `txCount` – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

static inline void I3C_SlaveSetWatermarks(I3C_Type *base, *i3c_tx_trigger_level_t* txLvl,
i3c_rx_trigger_level_t rxLvl, bool flushTx, bool flushRx)

Sets the watermarks for I3C slave FIFOs.

Parameters

- `base` – The I3C peripheral base address.
- `txLvl` – Transmit FIFO watermark level. The `kI3C_SlaveTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches `txLvl`.
- `rxLvl` – Receive FIFO watermark level. The `kI3C_SlaveRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches `rxLvl`.
- `flushTx` – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- `flushRx` – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

```
static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of bytes in the I3C slave FIFOs.

Parameters

- `base` – The I3C peripheral base address.
- `txCount` – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

```
void I3C_MasterSetBaudRate(I3C_Type *base, const i3c_baudrate_hz_t *baudRate_Hz, uint32_t  
sourceClock_Hz)
```

Sets the I3C bus frequency for master transactions.

The I3C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Parameters

- `base` – The I3C peripheral base address.
- `baudRate_Hz` – Pointer to structure of requested bus frequency in Hertz.
- `sourceClock_Hz` – I3C functional clock frequency in Hertz.

```
static inline bool I3C_MasterGetBusIdleState(I3C_Type *base)
```

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- `base` – The I3C peripheral base address.

Return values

- `true` – Bus is busy.
- `false` – Bus is idle.

```
status_t I3C_MasterStartWithRxSize(I3C_Type *base, i3c_bus_type_t type, uint8_t address,  
i3c_direction_t dir, uint8_t rxSize)
```

Sends a START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the `address` parameter. Note that this function

does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- `base` – The I3C peripheral base address.
- `type` – The bus type to use in this transaction.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- `rxSize` – Read terminate size for the followed read transfer, limit to 255 bytes.

Return values

- `kStatus_Success` – START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.

status_t I3C_MasterStart(I3C_Type *base, *i3c_bus_type_t* type, uint8_t address, *i3c_direction_t* dir)

Sends a START signal and slave address on the I2C/I3C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- `base` – The I3C peripheral base address.
- `type` – The bus type to use in this transaction.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- `kStatus_Success` – START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.

status_t I3C_MasterRepeatedStartWithRxSize(I3C_Type *base, *i3c_bus_type_t* type, uint8_t address, *i3c_direction_t* dir, uint8_t rxSize)

Sends a repeated START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to send a Repeated START signal when a transfer is already in progress. Like `I3C_MasterStart()`, it also sends the specified 7-bit address. Call this API also configures the read terminate size for the following read transfer. For example, set the `rxSize = 2`, the following read transfer will be terminated after two bytes of data received. Write transfer will not be affected by the `rxSize` configuration.

Note: This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- `base` – The I3C peripheral base address.
- `type` – The bus type to use in this transaction.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- `rxSize` – Read terminate size for the followed read transfer, limit to 255 bytes.

Return values

`kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.

```
static inline status_t I3C_MasterRepeatedStart(I3C_Type *base, i3c_bus_type_t type, uint8_t  
address, i3c_direction_t dir)
```

Sends a repeated START signal and slave address on the I2C/I3C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like `I3C_MasterStart()`, it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- `base` – The I3C peripheral base address.
- `type` – The bus type to use in this transaction.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

`kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.

```
status_t I3C_MasterSend(I3C_Type *base, const void *txBuff, size_t txSize, uint32_t flags)
```

Performs a polling send transfer on the I2C/I3C bus.

Sends up to `txSize` number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns `kStatus_I3C_Nak`.

Parameters

- `base` – The I3C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.
- `flags` – Bit mask of options for the transfer. See enumeration `_i3c_master_transfer_flags` for available options.

Return values

- `kStatus_Success` – Data was sent successfully.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.

- `kStatus_I3C_Nak` – The slave device sent a NAK in response to an address.
- `kStatus_I3C_WriteAbort` – The slave device sent a NAK in response to a write.
- `kStatus_I3C_MsgError` – Message SDR/DDR mismatch or read/write message in wrong state.
- `kStatus_I3C_WriteFifoError` – Write to M/SWDATAB register when FIFO full.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

status_t I3C_MasterReceive(I3C_Type *base, void *rxBuff, size_t rxSize, uint32_t flags)

Performs a polling receive transfer on the I2C/I3C bus.

Parameters

- `base` – The I3C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.
- `flags` – Bit mask of options for the transfer. See enumeration `_i3c_master_transfer_flags` for available options.

Return values

- `kStatus_Success` – Data was received successfully.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.
- `kStatus_I3C_Term` – The master terminates slave read.
- `kStatus_I3C_HdrParityError` – Parity error from DDR read.
- `kStatus_I3C_CrcError` – CRC error from DDR read.
- `kStatus_I3C_MsgError` – Message SDR/DDR mismatch or read/write message in wrong state.
- `kStatus_I3C_ReadFifoError` – Read from M/SRDATAB register when FIFO empty.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

status_t I3C_MasterStop(I3C_Type *base)

Sends a STOP signal on the I2C/I3C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- `base` – The I3C peripheral base address.

Return values

- `kStatus_Success` – The STOP signal was successfully sent on the bus and the transaction terminated.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

void I3C_MasterEmitRequest(I3C_Type *base, *i3c_bus_request_t* masterReq)

I3C master emit request.

Parameters

- base – The I3C peripheral base address.
- masterReq – I3C master request of type *i3c_bus_request_t*

static inline void I3C_MasterEmitIBIResponse(I3C_Type *base, *i3c_ibi_response_t* ibiResponse)

I3C master emit request.

Parameters

- base – The I3C peripheral base address.
- ibiResponse – I3C master emit IBI response of type *i3c_ibi_response_t*

void I3C_MasterRegisterIBI(I3C_Type *base, *i3c_register_ibi_addr_t* *ibiRule)

I3C master register IBI rule.

Parameters

- base – The I3C peripheral base address.
- ibiRule – Pointer to ibi rule description of type *i3c_register_ibi_addr_t*

void I3C_MasterGetIBIRules(I3C_Type *base, *i3c_register_ibi_addr_t* *ibiRule)

I3C master get IBI rule.

Parameters

- base – The I3C peripheral base address.
- ibiRule – Pointer to store the read out ibi rule description.

i3c_ibi_type_t I3C_GetIBIType(I3C_Type *base)

I3C master get IBI Type.

Parameters

- base – The I3C peripheral base address.

Return values

i3c_ibi_type_t – Type of *i3c_ibi_type_t*.

static inline uint8_t I3C_GetIBIAddress(I3C_Type *base)

I3C master get IBI Address.

Parameters

- base – The I3C peripheral base address.

Return values

The – 8-bit IBI address.

status_t I3C_MasterProcessDAASpecifiedBaudrate(I3C_Type *base, uint8_t *addressList, uint32_t count, *i3c_master_daa_baudrate_t* *daaBaudRate)

Performs a DAA in the i3c bus with specified temporary baud rate.

Parameters

- base – The I3C peripheral base address.
- addressList – The pointer for address list which is used to do DAA.
- count – The address count in the address list.

- `daaBaudRate` – The temporary baud rate in DAA process, NULL for using initial setting. The initial setting is set back between the completion of the DAA and the return of this function.

Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- `kStatus_I3C_SlaveCountExceed` – The I3C slave count has exceed the definition in `I3C_MAX_DEVCNT`.

```
static inline status_t I3C_MasterProcessDAA(I3C_Type *base, uint8_t *addressList, uint32_t count)
```

Performs a DAA in the i3c bus.

Parameters

- `base` – The I3C peripheral base address.
- `addressList` – The pointer for address list which is used to do DAA.
- `count` – The address count in the address list. The initial setting is set back between the completion of the DAA and the return of this function.

Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- `kStatus_I3C_SlaveCountExceed` – The I3C slave count has exceed the definition in `I3C_MAX_DEVCNT`.

```
i3c_device_info_t *I3C_MasterGetDeviceListAfterDAA(I3C_Type *base, uint8_t *count)
```

Get device information list after DAA process is done.

Parameters

- `base` – The I3C peripheral base address.
- `count` – **[out]** The pointer to store the available device count.

Returns

Pointer to the `i3c_device_info_t` array.

```
void I3C_MasterClearDeviceCount(I3C_Type *base)
```

Clear the global device count which represents current devices number on the bus. When user resets all dynamic addresses on the bus, should call this API.

Parameters

- `base` – The I3C peripheral base address.

```
status_t I3C_MasterTransferBlocking(I3C_Type *base, i3c_master_transfer_t *transfer)
```

Performs a master polling transfer on the I2C/I3C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- `base` – The I3C peripheral base address.
- `transfer` – Pointer to the transfer structure.

Return values

- `kStatus_I3C_Success` – Data was received successfully.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_IBIWon` – The I3C slave event IBI or MR or HJ won the arbitration on a header address.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.
- `kStatus_I3C_Nak` – The slave device sent a NAK in response to an address.
- `kStatus_I3C_WriteAbort` – The slave device sent a NAK in response to a write.
- `kStatus_I3C_Term` – The master terminates slave read.
- `kStatus_I3C_HdrParityError` – Parity error from DDR read.
- `kStatus_I3C_CrcError` – CRC error from DDR read.
- `kStatus_I3C_MsgError` – Message SDR/DDR mismatch or read/write message in wrong state.
- `kStatus_I3C_ReadFifoError` – Read from M/SRDATAB register when FIFO empty.
- `kStatus_I3C_WriteFifoError` – Write to M/SWDATAB register when FIFO full.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

status_t I3C_SlaveSend(I3C_Type *base, const void *txBuff, size_t txSize)

Performs a polling send transfer on the I3C bus.

Parameters

- `base` – The I3C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

status_t I3C_SlaveReceive(I3C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I3C bus.

Parameters

- `base` – The I3C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

`void` I3C_MasterTransferCreateHandle(I3C_Type *base, *i3c_master_handle_t* *handle, const *i3c_master_transfer_callback_t* *callback, void *userData)

Creates a new handle for the I3C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_MasterTransferAbort()` API shall be called.

Note: The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The I3C peripheral base address.
- handle – **[out]** Pointer to the I3C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t I3C_MasterTransferNonBlocking(I3C_Type *base, *i3c_master_handle_t* *handle, *i3c_master_transfer_t* *transfer)

Performs a non-blocking transaction on the I2C/I3C bus.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_I3C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t I3C_MasterTransferGetCount(I3C_Type *base, *i3c_master_handle_t* *handle, *size_t* *count)

Returns number of bytes transferred so far.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void I3C_MasterTransferAbort(I3C_Type *base, *i3c_master_handle_t* *handle)

Terminates a non-blocking I3C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the I3C peripheral's IRQ priority.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.

void I3C_MasterTransferHandleIRQ(I3C_Type *base, void *intHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The I3C peripheral base address.
- intHandle – Pointer to the I3C master driver handle.

enum _i3c_master_flags

I3C master peripheral flags.

The following status register flags can be cleared:

- kI3C_MasterSlaveStartFlag
- kI3C_MasterControlDoneFlag
- kI3C_MasterCompleteFlag
- kI3C_MasterArbitrationWonFlag
- kI3C_MasterSlave2MasterFlag

All flags except kI3C_MasterBetweenFlag and kI3C_MasterNackDetectFlag can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_MasterBetweenFlag

Between messages/DAA's flag

enumerator kI3C_MasterNackDetectFlag

NACK detected flag

enumerator kI3C_MasterSlaveStartFlag

Slave request start flag

enumerator kI3C_MasterControlDoneFlag

Master request complete flag

enumerator kI3C_MasterCompleteFlag

Transfer complete flag

enumerator kI3C_MasterRxReadyFlag

Rx data ready in Rx buffer flag

enumerator kI3C_MasterTxReadyFlag

Tx buffer ready for Tx data flag

enumerator kI3C_MasterArbitrationWonFlag

Header address won arbitration flag

enumerator kI3C_MasterErrorFlag

Error occurred flag

enumerator kI3C_MasterSlave2MasterFlag

Switch from slave to master flag

enumerator kI3C_MasterClearFlags

enum _i3c_master_error_flags

I3C master error flags to indicate the causes.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_MasterErrorNackFlag

Slave NACKed the last address

enumerator kI3C_MasterErrorWriteAbortFlag

Slave NACKed the write data

enumerator kI3C_MasterErrorParityFlag

Parity error from DDR read

enumerator kI3C_MasterErrorCrcFlag

CRC error from DDR read

enumerator kI3C_MasterErrorReadFlag

Read from MRDATAB register when FIFO empty

enumerator kI3C_MasterErrorWriteFlag

Write to MWDATAB register when FIFO full

enumerator kI3C_MasterErrorMsgFlag

Message SDR/DDR mismatch or read/write message in wrong state

enumerator kI3C_MasterErrorInvalidReqFlag

Invalid use of request

enumerator kI3C_MasterErrorTimeoutFlag

The module has stalled too long in a frame

enumerator kI3C_MasterAllErrorFlags

All error flags

enum _i3c_master_state

I3C working master state.

Values:

enumerator kI3C_MasterStateIdle

Bus stopped.

enumerator kI3C_MasterStateSlvReq

Bus stopped but slave holding SDA low.

enumerator kI3C_MasterStateMsgSdr

In SDR Message mode from using MWMSG_SDR.

enumerator kI3C_MasterStateNormAct

In normal active SDR mode.

enumerator kI3C_MasterStateDdr

In DDR Message mode.

enumerator kI3C_MasterStateDaa

In ENTDAAs mode.

enumerator kI3C_MasterStateIbiAck

Waiting on IBI ACK/NACK decision.

enumerator kI3C_MasterStateIbiRcv

Receiving IBI.

enum _i3c_master_enable

I3C master enable configuration.

Values:

enumerator kI3C_MasterOff

Master off.

enumerator kI3C_MasterOn

Master on.

enumerator kI3C_MasterCapable

Master capable.

enum _i3c_master_hkeep

I3C high keeper configuration.

Values:

enumerator kI3C_MasterHighKeeperNone

Use PUR to hold SCL high.

enumerator kI3C_MasterHighKeeperWiredIn

Use pin_HK controls.

enumerator kI3C_MasterPassiveSDA

Hi-Z for Bus Free and hold SDA.

enumerator kI3C_MasterPassiveSDASCL

Hi-Z both for Bus Free, and can Hi-Z SDA for hold.

enum _i3c_bus_request

Emits the requested operation when doing in pieces vs. by message.

Values:

enumerator kI3C_RequestNone

No request.

enumerator kI3C_RequestEmitStartAddr

Request to emit start and address on bus.

enumerator kI3C_RequestEmitStop

Request to emit stop on bus.

enumerator kI3C_RequestIbiAckNack

Manual IBI ACK or NACK.

enumerator kI3C_RequestProcessDAA

Process DAA.

enumerator kI3C_RequestForceExit

Request to force exit.

enumerator kI3C_RequestAutoIbi

Hold in stopped state, but Auto-emit START,7E.

enum _i3c_bus_type

Bus type with EmitStartAddr.

Values:

enumerator kI3C_TypeI3CSdr

SDR mode of I3C.

enumerator kI3C_TypeI2C

Standard i2c protocol.

enumerator kI3C_TypeI3CDdr

HDR-DDR mode of I3C.

enum _i3c_ibi_response

IBI response.

Values:

enumerator kI3C_IbiRespAck

ACK with no mandatory byte.

enumerator kI3C_IbiRespNack

NACK.

enumerator kI3C_IbiRespAckMandatory

ACK with mandatory byte.

enumerator kI3C_IbiRespManual

Reserved.

enum _i3c_ibi_type

IBI type.

Values:

enumerator kI3C_IbiNormal

In-band interrupt.

enumerator kI3C_IbiHotJoin

slave hot join.

enumerator kI3C_IbiMasterRequest

slave master ship request.

enum _i3c_ibi_state

IBI state.

Values:

enumerator kI3C_IbiReady

In-band interrupt ready state, ready for user to handle.

enumerator kI3C_IbiDataBuffNeed

In-band interrupt need data buffer for data receive.

enumerator kI3C_IbiAckNackPending

In-band interrupt Ack/Nack pending for decision.

enum `_i3c_direction`

Direction of master and slave transfers.

Values:

enumerator `kI3C_Write`

Master transmit.

enumerator `kI3C_Read`

Master receive.

enum `_i3c_tx_trigger_level`

Watermark of TX int/dma trigger level.

Values:

enumerator `kI3C_TxTriggerOnEmpty`

Trigger on empty.

enumerator `kI3C_TxTriggerUntilOneQuarterOrLess`

Trigger on 1/4 full or less.

enumerator `kI3C_TxTriggerUntilOneHalfOrLess`

Trigger on 1/2 full or less.

enumerator `kI3C_TxTriggerUntilOneLessThanFull`

Trigger on 1 less than full or less.

enum `_i3c_rx_trigger_level`

Watermark of RX int/dma trigger level.

Values:

enumerator `kI3C_RxTriggerOnNotEmpty`

Trigger on not empty.

enumerator `kI3C_RxTriggerUntilOneQuarterOrMore`

Trigger on 1/4 full or more.

enumerator `kI3C_RxTriggerUntilOneHalfOrMore`

Trigger on 1/2 full or more.

enumerator `kI3C_RxTriggerUntilThreeQuarterOrMore`

Trigger on 3/4 full or more.

enum `_i3c_rx_term_ops`

I3C master read termination operations.

Values:

enumerator `kI3C_RxTermDisable`

Master doesn't terminate read, used for CCC transfer.

enumerator `kI3C_RxAutoTerm`

Master auto terminate read after receiving specified bytes(<=255).

enumerator `kI3C_RxTermLastByte`

Master terminates read at any time after START, no length limitation.

enum `_i3c_start_scl_delay`

I3C start SCL delay options.

Values:

enumerator kI3C_NoDelay

No delay.

enumerator kI3C_IncreaseSclHalfPeriod

Increases SCL clock period by 1/2.

enumerator kI3C_IncreaseSclOnePeriod

Increases SCL clock period by 1.

enumerator kI3C_IncreaseSclOneAndHalfPeriod

Increases SCL clock period by 1 1/2

enum _i3c_master_transfer_flags

Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the `_i3c_master_transfer::flags` field.

Values:

enumerator kI3C_TransferDefaultFlag

Transfer starts with a start signal, stops with a stop signal.

enumerator kI3C_TransferNoStartFlag

Don't send a start condition, address, and sub address

enumerator kI3C_TransferRepeatedStartFlag

Send a repeated start condition

enumerator kI3C_TransferNoStopFlag

Don't send a stop condition.

enumerator kI3C_TransferWordsFlag

Transfer in words, else transfer in bytes.

enumerator kI3C_TransferDisableRxTermFlag

Disable Rx termination. Note: It's for I3C CCC transfer.

enumerator kI3C_TransferRxAutoTermFlag

Set Rx auto-termination. Note: It's adaptive based on Rx size(<=255 bytes) except in I3C_MasterReceive.

enumerator kI3C_TransferStartWithBroadcastAddr

Start transfer with 0x7E, then read/write data with device address.

typedef enum _i3c_master_state i3c_master_state_t

I3C working master state.

typedef enum _i3c_master_enable i3c_master_enable_t

I3C master enable configuration.

typedef enum _i3c_master_hkeep i3c_master_hkeep_t

I3C high keeper configuration.

typedef enum _i3c_bus_request i3c_bus_request_t

Emits the requested operation when doing in pieces vs. by message.

typedef enum _i3c_bus_type i3c_bus_type_t

Bus type with EmitStartAddr.

```
typedef enum _i3c_ibi_response i3c_ibi_response_t
    IBI response.
```

```
typedef enum _i3c_ibi_type i3c_ibi_type_t
    IBI type.
```

```
typedef enum _i3c_ibi_state i3c_ibi_state_t
    IBI state.
```

```
typedef enum _i3c_direction i3c_direction_t
    Direction of master and slave transfers.
```

```
typedef enum _i3c_tx_trigger_level i3c_tx_trigger_level_t
    Watermark of TX int/dma trigger level.
```

```
typedef enum _i3c_rx_trigger_level i3c_rx_trigger_level_t
    Watermark of RX int/dma trigger level.
```

```
typedef enum _i3c_rx_term_ops i3c_rx_term_ops_t
    I3C master read termination operations.
```

```
typedef enum _i3c_start_scl_delay i3c_start_scl_delay_t
    I3C start SCL delay options.
```

```
typedef struct _i3c_register_ibi_addr i3c_register_ibi_addr_t
    Structure with setting master IBI rules and slave registry.
```

```
typedef struct _i3c_baudrate i3c_baudrate_hz_t
    Structure with I3C baudrate settings.
```

```
typedef struct _i3c_master_daa_baudrate i3c_master_daa_baudrate_t
    I3C DAA baud rate configuration.
```

```
typedef struct _i3c_master_config i3c_master_config_t
    Structure with settings to initialize the I3C master module.
```

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_MasterGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

```
typedef struct _i3c_master_transfer i3c_master_transfer_t
```

```
typedef struct _i3c_master_handle i3c_master_handle_t
```

```
typedef struct _i3c_master_transfer_callback i3c_master_transfer_callback_t
    i3c master callback functions.
```

```
typedef void (*i3c_master_isr_t)(I3C_Type *base, void *handle)
    Typedef for master interrupt handler.
```

```
struct _i3c_register_ibi_addr
    #include <fsl_i3c.h> Structure with setting master IBI rules and slave registry.
```

Public Members

```
uint8_t address[5]
    Address array for registry.
```

```
bool i3cFastStart
    Allow the START header to run as push-pull speed if all dynamic addresses take MSB 0.
```

bool ibiHasPayload

Whether the address array has mandatory IBI byte.

struct __i3c__baudrate

#include <fsl_i3c.h> Structure with I3C baudrate settings.

Public Members

uint32_t i2cBaud

Desired I2C baud rate in Hertz.

uint32_t i3cPushPullBaud

Desired I3C push-pull baud rate in Hertz.

uint32_t i3cOpenDrainBaud

Desired I3C open-drain baud rate in Hertz.

struct __i3c__master__daa__baudrate

#include <fsl_i3c.h> I3C DAA baud rate configuration.

Public Members

uint32_t sourceClock_Hz

FCLK, function clock in Hertz.

uint32_t i3cPushPullBaud

Desired I3C push-pull baud rate in Hertz.

uint32_t i3cOpenDrainBaud

Desired I3C open-drain baud rate in Hertz.

struct __i3c__master__config

#include <fsl_i3c.h> Structure with settings to initialize the I3C master module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

i3c_master_enable_t enableMaster

Enable master mode.

bool disableTimeout

Whether to disable timeout to prevent the ERRWARN.

i3c_master_hkeep_t hKeep

High keeper mode setting.

bool enableOpenDrainStop

Whether to emit open-drain speed STOP.

bool enableOpenDrainHigh

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

i3c_baudrate_hz_t baudRate_Hz

Desired baud rate settings.

i3c_start_scl_delay_t startSclDelay

I3C SCL delay after START.

i3c_start_scl_delay_t restartSclDelay

I3C SCL delay after Repeated START.

struct *_i3c_master_transfer_callback*

#include <fsl_i3c.h> i3c master callback functions.

Public Members

void (*slave2Master)(I3C_Type *base, void *userData)

Transfer complete callback

void (*ibiCallback)(I3C_Type *base, *i3c_master_handle_t* *handle, *i3c_ibi_type_t* ibiType, *i3c_ibi_state_t* ibiState)

IBI event callback

void (*transferComplete)(I3C_Type *base, *i3c_master_handle_t* *handle, *status_t* completionStatus, void *userData)

Transfer complete callback

struct *_i3c_master_transfer*

#include <fsl_i3c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the I3C_MasterTransferNonBlocking() API.

Public Members

uint32_t flags

Bit mask of options for the transfer. See enumeration *_i3c_master_transfer_flags* for available options. Set to 0 or *kI3C_TransferDefaultFlag* for normal transfers.

uint8_t slaveAddress

The 7-bit slave address.

i3c_direction_t direction

Either *kI3C_Read* or *kI3C_Write*.

uint32_t subaddress

Sub address. Transferred MSB first.

size_t subaddressSize

Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data

Pointer to data to transfer.

size_t dataSize

Number of bytes to transfer.

i3c_bus_type_t busType

bus type.

i3c_ibi_response_t ibiResponse

ibi response during transfer.

```
struct _i3c_master_handle
    #include <fsl_i3c.h> Driver handle for master non-blocking APIs.
```

Note: The contents of this structure are private and subject to change.

Public Members

```
uint8_t state
    Transfer state machine current state.

uint32_t remainingBytes
    Remaining byte count in current state.

i3c_rx_term_ops_t rxTermOps
    Read termination operation.

i3c_master_transfer_t transfer
    Copy of the current transfer info.

uint8_t ibiAddress
    Slave address which request IBI.

uint8_t *ibiBuff
    Pointer to IBI buffer to keep ibi bytes.

size_t ibiPayloadSize
    IBI payload size.

i3c_ibi_type_t ibiType
    IBI type.

i3c_master_transfer_callback_t callback
    Callback functions pointer.

void *userData
    Application data passed to callback.
```

2.29 I3C Master DMA Driver

```
void I3C_MasterTransferCreateHandleEDMA(I3C_Type *base, i3c_master_edma_handle_t
                                         *handle, const i3c_master_edma_callback_t
                                         *callback, void *userData, edma_handle_t
                                         *rxDmaHandle, edma_handle_t *txDmaHandle)
```

Create a new handle for the I3C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_MasterTransferAbortDMA()` API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C master driver handle.
- *callback* – User provided pointer to the asynchronous callback function.

- `userData` – User provided pointer to the application callback data.
- `rxDmaHandle` – Handle for the DMA receive channel. Created by the user prior to calling this function.
- `txDmaHandle` – Handle for the DMA transmit channel. Created by the user prior to calling this function.

`status_t I3C_MasterTransferEDMA(I3C_Type *base, i3c_master_edma_handle_t *handle, i3c_master_transfer_t *transfer)`

Performs a non-blocking DMA-based transaction on the I3C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.
- `transfer` – The pointer to the transfer descriptor.

Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

`status_t I3C_MasterTransferGetCountEDMA(I3C_Type *base, i3c_master_edma_handle_t *handle, size_t *count)`

Returns number of bytes transferred so far.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.
- `count` – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_Success` –
- `kStatus_NoTransferInProgress` – There is not a DMA transaction currently in progress.

`void I3C_MasterTransferAbortEDMA(I3C_Type *base, i3c_master_edma_handle_t *handle)`

Terminates a non-blocking I3C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the DMA peripheral's IRQ priority.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.

`void I3C_MasterTransferEDMAHandleIRQ(I3C_Type *base, void *i3cHandle)`

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- `base` – The I3C peripheral base address.
- `i3cHandle` – Pointer to the I3C master DMA driver handle.

```
typedef struct _i3c_master_edma_handle i3c_master_edma_handle_t
```

```
typedef struct _i3c_master_edma_callback i3c_master_edma_callback_t
```

i3c master callback functions.

```
struct _i3c_master_edma_callback
```

```
#include <fsl_i3c_edma.h> i3c master callback functions.
```

Public Members

```
void (*slave2Master)(I3C_Type *base, void *userData)
```

Target asks for controller request.

```
void (*ibiCallback)(I3C_Type *base, i3c_master_edma_handle_t *handle, i3c_ibi_type_t ibiType, i3c_ibi_state_t ibiState)
```

IBI event callback.

```
void (*transferComplete)(I3C_Type *base, i3c_master_edma_handle_t *handle, status_t status, void *userData)
```

Transfer complete callback.

```
struct _i3c_master_edma_handle
```

```
#include <fsl_i3c_edma.h> Driver handle for master EDMA APIs.
```

Note: The contents of this structure are private and subject to change.

Public Members

```
I3C_Type *base
```

I3C base pointer.

```
uint8_t state
```

Transfer state machine current state.

```
uint32_t transferCount
```

Indicates progress of the transfer

```
uint8_t subaddressBuffer[4]
```

Saving subaddress command.

```
uint8_t subaddressCount
```

Saving command count.

```
i3c_master_transfer_t transfer
```

Copy of the current transfer info.

```
i3c_master_edma_callback_t callback
```

Callback function pointer.

`void *userData`
Application data passed to callback.

`edma_handle_t *rxDmaHandle`
Handle for receive DMA channel.

`edma_handle_t *txDmaHandle`
Handle for transmit DMA channel.

`bool ibiFlag`
IBIWON flag.

`uint8_t ibiAddress`
Slave address which request IBI.

`uint8_t *ibiBuff`
Pointer to IBI buffer to keep ibi bytes.

`size_t ibiPayloadSize`
IBI payload size.

`i3c_ibi_type_t ibiType`
IBI type.

`status_t result`
Transfer result.

2.30 I3C Slave Driver

`void I3C_SlaveGetDefaultConfig(i3c_slave_config_t *slaveConfig)`

Provides a default configuration for the I3C slave peripheral.

This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableslave = true;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with `I3C_SlaveInit()`.

Parameters

- `slaveConfig` – **[out]** User provided configuration structure for default values. Refer to `i3c_slave_config_t`.

`void I3C_SlaveInit(I3C_Type *base, const i3c_slave_config_t *slaveConfig, uint32_t slowClock_Hz)`

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

Parameters

- `base` – The I3C peripheral base address.
- `slaveConfig` – User provided peripheral configuration. Use `I3C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- `slowClock_Hz` – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If `FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH` defines as 1, this parameter is useless.

`void I3C_SlaveDeinit(I3C_Type *base)`

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

Parameters

- `base` – The I3C peripheral base address.

`static inline void I3C_SlaveEnable(I3C_Type *base, bool isEnabled)`

Enable/Disable Slave.

Parameters

- `base` – The I3C peripheral base address.
- `isEnabled` – Enable or disable.

`static inline uint32_t I3C_SlaveGetStatusFlags(I3C_Type *base)`

Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_slave_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

`static inline void I3C_SlaveClearStatusFlags(I3C_Type *base, uint32_t statusMask)`

Clears the I3C slave status flag state.

The following status register flags can be cleared:

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`

Attempts to clear other flags has no effect.

See also:

`_i3c_slave_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetStatusFlags()`.

`static inline uint32_t I3C_SlaveGetErrorStatusFlags(I3C_Type *base)`

Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_slave_error_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

`static inline void I3C_SlaveClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)`

Clears the I3C slave error status flag state.

See also:

`_i3c_slave_error_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_slave_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetErrorStatusFlags()`.

`i3c_slave_activity_state_t I3C_SlaveGetActivityState(I3C_Type *base)`

Gets the I3C slave state.

Parameters

- `base` – The I3C peripheral base address.

Returns

I3C slave activity state, refer `i3c_slave_activity_state_t`.

`status_t I3C_SlaveCheckAndClearError(I3C_Type *base, uint32_t status)`

`static inline void I3C_SlaveEnableInterrupts(I3C_Type *base, uint32_t interruptMask)`

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`

- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline void I3C_SlaveDisableInterrupts(I3C_Type *base, uint32_t interruptMask)

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t I3C_SlaveGetEnabledInterrupts(I3C_Type *base)

Returns the set of currently enabled I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

static inline uint32_t I3C_SlaveGetPendingInterrupts(I3C_Type *base)

Returns the set of pending I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of pending interrupts.

```
static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)
```

Enables or disables I3C slave DMA requests.

Parameters

- *base* – The I3C peripheral base address.
- *enableTx* – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- *enableRx* – Enable flag for receive DMA request. Pass true for enable, false for disable.
- *width* – DMA read/write unit in bytes.

```
static inline uint32_t I3C_SlaveGetTxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave transmit data register address for DMA transfer.

Parameters

- *base* – The I3C peripheral base address.
- *width* – DMA read/write unit in bytes.

Returns

The I3C Slave Transmit Data Register address.

```
static inline uint32_t I3C_SlaveGetRxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave receive data register address for DMA transfer.

Parameters

- *base* – The I3C peripheral base address.
- *width* – DMA read/write unit in bytes.

Returns

The I3C Slave Receive Data Register address.

```
static inline void I3C_SlaveSetWatermarks(I3C_Type *base, i3c_tx_trigger_level_t txLvl,
                                          i3c_rx_trigger_level_t rxLvl, bool flushTx, bool flushRx)
```

Sets the watermarks for I3C slave FIFOs.

Parameters

- *base* – The I3C peripheral base address.
- *txLvl* – Transmit FIFO watermark level. The `kI3C_SlaveTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
- *rxLvl* – Receive FIFO watermark level. The `kI3C_SlaveRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
- *flushTx* – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- *flushRx* – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

```
static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of bytes in the I3C slave FIFOs.

Parameters

- *base* – The I3C peripheral base address.
- *txCount* – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.

- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

`status_t I3C_SlaveSend(I3C_Type *base, const void *txBuff, size_t txSize)`

Performs a polling send transfer on the I3C bus.

Parameters

- `base` – The I3C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

`status_t I3C_SlaveReceive(I3C_Type *base, void *rxBuff, size_t rxSize)`

Performs a polling receive transfer on the I3C bus.

Parameters

- `base` – The I3C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

`void I3C_SlaveTransferCreateHandle(I3C_Type *base, i3c_slave_handle_t *handle, i3c_slave_transfer_callback_t callback, void *userData)`

Creates a new handle for the I3C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_SlaveTransferAbort()` API shall be called.

Note: The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – **[out]** Pointer to the I3C slave driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

`status_t I3C_SlaveTransferNonBlocking(I3C_Type *base, i3c_slave_handle_t *handle, uint32_t eventMask)`

Starts accepting slave transfers.

Call this API after calling `I2C_SlaveInit()` and `I3C_SlaveTransferCreateHandle()` to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to `I3C_SlaveTransferCreateHandle()`. The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the `eventMask` parameter to the OR'd combination of `i3c_slave_transfer_event_t` enumerators for the events you wish to receive. The `kI3C_SlaveTransmitEvent` and `kI3C_SlaveReceiveEvent` events are always enabled and do not need to be included in the mask. Alternatively, you can pass

0 to get a default set of only the transmit and receive events that are always enabled. In addition, the `kI3C_SlaveAllEvents` constant is provided as a convenient way to enable all events.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to struct: `_i3c_slave_handle` structure which stores the transfer state.
- `eventMask` – Bit mask formed by OR'ing together `i3c_slave_transfer_event_t` enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and `kI3C_SlaveAllEvents` to enable all events.

Return values

- `kStatus__Success` – Slave transfers were successfully started.
- `kStatus__I3C__Busy` – Slave transfers have already been started on this handle.

`status_t I3C_SlaveTransferGetCount(I3C_Type *base, i3c_slave_handle_t *handle, size_t *count)`

Gets the slave transfer status during a non-blocking transfer.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to `i2c_slave_handle_t` structure.
- `count` – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

- `kStatus__Success` –
- `kStatus__NoTransferInProgress` –

`void I3C_SlaveTransferAbort(I3C_Type *base, i3c_slave_handle_t *handle)`

Aborts the slave non-blocking transfers.

Note: This API could be called at any time to stop slave for handling the bus events.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to struct: `_i3c_slave_handle` structure which stores the transfer state.

`void I3C_SlaveTransferHandleIRQ(I3C_Type *base, void *intHandle)`

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

- `base` – The I3C peripheral base address.
- `intHandle` – Pointer to struct: `_i3c_slave_handle` structure which stores the transfer state.

enum _i3c_slave_flags

I3C slave peripheral flags.

The following status register flags can be cleared:

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag

Only below flags can be enabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_SlaveNotStopFlag

Slave status not stop flag

enumerator kI3C_SlaveMessageFlag

Slave status message, indicating slave is listening to the bus traffic or responding

enumerator kI3C_SlaveRequiredReadFlag

Slave status required, either is master doing SDR read from slave, or is IBI pushing out.

enumerator kI3C_SlaveRequiredWriteFlag

Slave status request write, master is doing SDR write to slave, except slave in ENTDAAMode

enumerator kI3C_SlaveBusDAAModeFlag

I3C bus is in ENTDAAMode

enumerator kI3C_SlaveBusHDRModeFlag

I3C bus is in HDR mode

enumerator kI3C_SlaveBusStartFlag

Start/Re-start event is seen since the bus was last cleared

enumerator kI3C_SlaveMatchedFlag

Slave address(dynamic/static) matched since last cleared

enumerator kI3C_SlaveBusStopFlag

Stop event is seen since the bus was last cleared

enumerator kI3C_SlaveRxReadyFlag
Rx data ready in rx buffer flag

enumerator kI3C_SlaveTxReadyFlag
Tx buffer ready for Tx data flag

enumerator kI3C_SlaveDynamicAddrChangedFlag
Slave dynamic address has been assigned, re-assigned, or lost

enumerator kI3C_SlaveReceivedCCCFlag
Slave received Common command code

enumerator kI3C_SlaveErrorFlag
Error occurred flag

enumerator kI3C_SlaveHDRCommandMatchFlag
High data rate command match

enumerator kI3C_SlaveCCCHandledFlag
Slave received Common command code is handled by I3C module

enumerator kI3C_SlaveEventSentFlag
Slave IBI/P2P/MR/HJ event has been sent

enumerator kI3C_SlaveIbiDisableFlag
Slave in band interrupt is disabled.

enumerator kI3C_SlaveMasterRequestDisabledFlag
Slave master request is disabled.

enumerator kI3C_SlaveHotJoinDisabledFlag
Slave Hot-Join is disabled.

enumerator kI3C_SlaveClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kI3C_SlaveAllIrqFlags

enum _i3c_slave_error_flags
I3C slave error flags to indicate the causes.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_SlaveErrorOverrunFlag
Slave internal from-bus buffer/FIFO overrun.

enumerator kI3C_SlaveErrorUnderrunFlag
Slave internal to-bus buffer/FIFO underrun

enumerator kI3C_SlaveErrorUnderrunNakFlag
Slave internal from-bus buffer/FIFO underrun and NACK error

enumerator kI3C_SlaveErrorTermFlag
Terminate error from master

enumerator kI3C_SlaveErrorInvalidStartFlag
Slave invalid start flag

enumerator kI3C_SlaveErrorSdrParityFlag
SDR parity error

enumerator kI3C_SlaveErrorHdrParityFlag

HDR parity error

enumerator kI3C_SlaveErrorHdrCRCFlag

HDR-DDR CRC error

enumerator kI3C_SlaveErrorS0S1Flag

S0 or S1 error

enumerator kI3C_SlaveErrorOverreadFlag

Over-read error

enumerator kI3C_SlaveErrorOverwriteFlag

Over-write error

enum _i3c_slave_event

I3C slave.event.

Values:

enumerator kI3C_SlaveEventNormal

Normal mode.

enumerator kI3C_SlaveEventIBI

In band interrupt event.

enumerator kI3C_SlaveEventMasterReq

Master request event.

enumerator kI3C_SlaveEventHotJoinReq

Hot-join event.

enum _i3c_slave_activity_state

I3C slave.activity state.

Values:

enumerator kI3C_SlaveNoLatency

Normal bus operation

enumerator kI3C_SlaveLatency1Ms

1ms of latency.

enumerator kI3C_SlaveLatency100Ms

100ms of latency.

enumerator kI3C_SlaveLatency10S

10s latency.

enum _i3c_slave_transfer_event

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to `I3C_SlaveTransferNonBlocking()` in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

Values:

enumerator `kI3C_SlaveAddressMatchEvent`

Received the slave address after a start or repeated start.

enumerator `kI3C_SlaveTransmitEvent`

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator `kI3C_SlaveReceiveEvent`

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator `kI3C_SlaveRequiredTransmitEvent`

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator `kI3C_SlaveStartEvent`

A start/repeated start was detected.

enumerator `kI3C_SlaveHDRCommandMatchEvent`

Slave Match HDR Command.

enumerator `kI3C_SlaveCompletionEvent`

A stop was detected, completing the transfer.

enumerator `kI3C_SlaveRequestSentEvent`

Slave request event sent.

enumerator `kI3C_SlaveReceivedCCCEvent`

Slave received CCC event, need to handle by application.

enumerator `kI3C_SlaveAllEvents`

Bit mask of all available events.

typedef enum `_i3c_slave_event` `i3c_slave_event_t`

I3C slave.event.

typedef enum `_i3c_slave_activity_state` `i3c_slave_activity_state_t`

I3C slave.activity state.

typedef struct `_i3c_slave_config` `i3c_slave_config_t`

Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_SlaveGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum `_i3c_slave_transfer_event` `i3c_slave_transfer_event_t`

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to `I3C_SlaveTransferNonBlocking()` in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

typedef struct `_i3c_slave_transfer` `i3c_slave_transfer_t`

I3C slave transfer structure.

typedef struct `_i3c_slave_handle` `i3c_slave_handle_t`

```
typedef void (*i3c_slave_transfer_callback_t)(I3C_Type *base, i3c_slave_transfer_t *transfer, void *userData)
```

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the I3C_SlaveSetCallback() function after you have created a handle.

Param base

Base address for the I3C instance on which the event occurred.

Param transfer

Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData

Arbitrary pointer-sized value passed from the application.

```
typedef void (*i3c_slave_isr_t)(I3C_Type *base, void *handle)
```

Typedef for slave interrupt handler.

```
struct _i3c_slave_config
```

#include <fsl_i3c.h> Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableSlave

Whether to enable slave.

uint8_t staticAddr

Static address.

uint16_t vendorID

Device vendor ID(manufacture ID).

uint32_t partNumber

Device part number info

uint8_t dcr

Device characteristics register information.

uint8_t bcr

Bus characteristics register information.

uint8_t hdrMode

Support hdr mode, could be OR logic in enumeration:i3c_hdr_mode_t.

bool nakAllRequest

Whether to reply NAK to all requests except broadcast CCC.

bool ignoreS0S1Error

Whether to ignore S0/S1 error in SDR mode.

bool offline

Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

bool matchSlaveStartStop

Whether to assert start/stop status only the time slave is addressed.

uint32_t maxWriteLength

Maximum write length.

uint32_t maxReadLength

Maximum read length.

struct __i3c_slave_transfer

#include <fsl_i3c.h> I3C slave transfer structure.

Public Members

uint32_t event

Reason the callback is being invoked.

uint8_t *txData

Transfer buffer

size_t txDataSize

Transfer size

uint8_t *rxData

Transfer buffer

size_t rxDataSize

Transfer size

status_t completionStatus

Success or error code describing how the transfer completed. Only applies for kI3C_SlaveCompletionEvent.

size_t transferredCount

Number of bytes actually transferred since start or last repeated start.

struct __i3c_slave_handle

#include <fsl_i3c.h> I3C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

i3c_slave_transfer_t transfer

I3C slave transfer copy.

bool isBusy

Whether transfer is busy.

bool wasTransmit

Whether the last transfer was a transmit.

uint32_t eventMask

Mask of enabled events.

uint32_t transferredCount

Count of bytes transferred.

i3c_slave_transfer_callback_t callback

Callback function called at transfer event.

void *userData

Callback parameter passed to callback.

size_t txFifoSize

Tx Fifo size

2.31 I3C Slave DMA Driver

```
void I3C_SlaveTransferCreateHandleEDMA(I3C_Type *base, i3c_slave_edma_handle_t *handle,
                                       i3c_slave_edma_callback_t callback, void *userData,
                                       edma_handle_t *rxDmaHandle, edma_handle_t
                                       *txDmaHandle)
```

Create a new handle for the I3C slave DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_SlaveTransferAbortDMA()` API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C slave driver handle.
- *callback* – User provided pointer to the asynchronous callback function.
- *userData* – User provided pointer to the application callback data.
- *rxDmaHandle* – Handle for the DMA receive channel. Created by the user prior to calling this function.
- *txDmaHandle* – Handle for the DMA transmit channel. Created by the user prior to calling this function.

```
status_t I3C_SlaveTransferEDMA(I3C_Type *base, i3c_slave_edma_handle_t *handle,
                               i3c_slave_edma_transfer_t *transfer, uint32_t eventMask)
```

Prepares for a non-blocking DMA-based transaction on the I3C bus.

The API will do DMA configuration according to the input transfer descriptor, and the data will be transferred when there's bus master requesting transfer from/to this slave. So the timing of call to this API need be aligned with master application to ensure the transfer is executed as expected. Callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C slave driver handle.
- *transfer* – The pointer to the transfer descriptor.
- *eventMask* – Bit mask formed by OR'ing together `i3c_slave_transfer_event_t` enumerators to specify which events to send to the callback. The transmit and receive events is not allowed to be enabled.

Return values

- `kStatus_Success` – The transaction was started successfully.

- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.
- `kStatus_Fail` – The transaction can't be set.

`void I3C_SlaveTransferAbortEDMA(I3C_Type *base, i3c_slave_edma_handle_t *handle)`

Abort a slave edma non-blocking transfer in a early time.

Parameters

- `base` – I3C peripheral base address
- `handle` – pointer to `i3c_slave_edma_handle_t` structure

`void I3C_SlaveTransferEDMAHandleIRQ(I3C_Type *base, void *i3CHandle)`

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- `base` – The I3C peripheral base address.
- `i3CHandle` – Pointer to the I3C slave DMA driver handle.

`typedef struct _i3c_slave_edma_handle i3c_slave_edma_handle_t`

`typedef struct _i3c_slave_edma_transfer i3c_slave_edma_transfer_t`

I3C slave transfer structure.

`typedef void (*i3c_slave_edma_callback_t)(I3C_Type *base, i3c_slave_edma_transfer_t *transfer, void *userData)`

Slave event callback function pointer type.

This callback is used only for the slave DMA transfer API.

Param base

Base address for the I3C instance on which the event occurred.

Param handle

Pointer to slave DMA transfer handle.

Param transfer

Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData

Arbitrary pointer-sized value passed from the application.

`struct _i3c_slave_edma_transfer`

`#include <fsl_i3c_edma.h>` I3C slave transfer structure.

Public Members

`uint32_t event`

Reason the callback is being invoked.

`uint8_t *txData`

Transfer buffer

`size_t txDataSize`

Transfer size


```

uint8_t *rxData
    Transfer buffer

size_t rxDataSize
    Transfer size

status_t completionStatus
    Success or error code describing how the transfer completed. Only applies for
    kI3C_SlaveCompletionEvent.

struct _i3c_slave_edma_handle
    #include <fsl_i3c_edma.h> I3C slave edma handle structure.

```

Note: The contents of this structure are private and subject to change.

Public Members

```

I3C_Type *base
    I3C base pointer.

i3c_slave_edma_transfer_t transfer
    I3C slave transfer copy.

bool isBusy
    Whether transfer is busy.

bool wasTransmit
    Whether the last transfer was a transmit.

uint32_t eventMask
    Mask of enabled events.

i3c_slave_edma_callback_t callback
    Callback function called at transfer event.

edma_handle_t *rxDmaHandle
    Handle for receive DMA channel.

edma_handle_t *txDmaHandle
    Handle for transmit DMA channel.

void *userData
    Callback parameter passed to callback.

```

2.32 INTM: Interrupt Monitor Driver

```

FSL_INTM_DRIVER_VERSION
    INTM driver version.

enum _intm_monitor
    Interrupt monitors.

    Values:

    enumerator kINTM_Monitor1

    enumerator kINTM_Monitor2

```

enumerator kINTM_Monitor3

enumerator kINTM_Monitor4

typedef enum *_intm_monitor* intm_monitor_t

Interrupt monitors.

typedef struct *_intm_monitor_config* intm_monitor_config_t

INTM interrupt source configuration structure.

typedef struct *_intm_config* intm_config_t

INTM configuration structure.

void INTM_GetDefaultConfig(*intm_config_t* *config)

Fill in the INTM config struct with the default settings.

The default values are:

```
config[0].irqnumber = NotAvail_IRQn;
config[0].maxtimer = 1000U;
config[1].irqnumber = NotAvail_IRQn;
config[1].maxtimer = 1000U;
config[2].irqnumber = NotAvail_IRQn;
config[2].maxtimer = 1000U;
config[3].irqnumber = NotAvail_IRQn;
config[3].maxtimer = 1000U;
config->enable = false;
```

Parameters

- config – Pointer to user's INTM config structure.

void INTM_Init(INTM_Type *base, const *intm_config_t* *config)

Ungates the INTM clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the INTM driver.

Parameters

- base – INTM peripheral base address
- config – Pointer to user's INTM config structure.

void INTM_Deinit(INTM_Type *base)

Disables the INTM module.

Parameters

- base – INTM peripheral base address

static inline void INTM_EnableCycleCount(INTM_Type *base, bool enable)

Enable the cycle count timer mode.

Monitor mode enables the cycle count timer on a monitored interrupt request for comparison to the latency register.

Parameters

- base – INTM peripheral base address.
- enable – Enable the cycle count or not.

```
static inline void INTM_AckIrq(INTM_Type *base, IRQn_Type irq)
```

Interrupt Acknowledge.

Call this function in ISR to acknowledge interrupt.

Parameters

- base – INTM peripheral base address.
- irq – Handle interrupt number.

```
static inline void INTM_SetInterruptRequestNumber(INTM_Type *base, intm_monitor_t intms,  
                                                IRQn_Type irq)
```

Interrupt Request Select.

This function is used to set the interrupt request number to monitor or check.

Parameters

- base – INTM peripheral base address.
- intms – Programmable interrupt monitors.
- irq – Interrupt request number to monitor.

Returns

Select the interrupt request number to monitor.

```
static inline void INTM_SetMaxTime(INTM_Type *base, intm_monitor_t intms, uint32_t count)
```

Set the maximum count time.

This function is to set the maximum time from interrupt generation to confirmation.

Parameters

- base – INTM peripheral base address.
- intms – Programmable interrupt monitors.
- count – Timer maximum count.

```
static inline void INTM_ClearTimeCount(INTM_Type *base, intm_monitor_t intms)
```

Clear the timer period in units of count.

This function is used to clear the INTM_TIMERa register.

Parameters

- base – INTM peripheral base address.
- intms – Programmable interrupt monitors.

```
static inline uint32_t INTM_GetTimeCount(INTM_Type *base, intm_monitor_t intms)
```

Gets the timer period in units of count.

This function is used to get the number of INTM clock cycles from interrupt request to confirmation interrupt processing. If this number exceeds the set maximum time, will be an error signal.

Parameters

- base – INTM peripheral base address.
- intms – Programmable interrupt monitors.

```
static inline bool INTM_GetStatusFlags(INTM_Type *base, intm_monitor_t intms)
```

Interrupt monitor status.

This function indicates whether the INTM_TIMERa value has exceeded the INTM_LATENCYa value. If any interrupt source in INTM_TIMERa exceeds the programmed

delay value, the monitor state can be cleared by calling the INTM_ClearTimeCount() API to clear the corresponding INTM_TIMERa register.

Parameters

- base – INTM peripheral base address.
- intms – Programmable interrupt monitors.

Returns

Whether INTM_TIMER value has exceeded INTM_LATENCY value.
false:INTM_TIMER value has not exceeded the INTM_LATENCY value;
true:INTM_TIMER value has exceeded the INTM_LATENCY value.

struct __intm_monitor_config

#include <fsl_intm.h> INTM interrupt source configuration structure.

Public Members

uint32_t maxtimer

Set the maximum timer

IRQn_Type irqnumber

Select the interrupt request number to monitor.

struct __intm_config

#include <fsl_intm.h> INTM configuration structure.

Public Members

bool enable

Interrupt source monitor config. enables the cycle count timer on a monitored interrupt request for comparison to the latency register.

2.33 IOMUXC: IOMUX Controller

```
static inline void IOMUXC_SetPinMux(uint32_t muxRegister, uint32_t muxMode, uint32_t
                                     inputRegister, uint32_t inputDaisy, uint32_t
                                     configRegister, uint32_t inputOnfield)
```

Sets the IOMUXC pin mux mode.

Note: The first five parameters can be filled with the pin function ID macros.

Parameters

- muxRegister – The pin mux register
- muxMode – The pin mux mode
- inputRegister – The select input register
- inputDaisy – The input daisy
- configRegister – The config register
- inputOn – The software input on

```
static inline void IOMUXC_SetPinConfig(uint32_t muxRegister, uint32_t muxMode, uint32_t
                                     inputRegister, uint32_t inputDaisy, uint32_t
                                     configRegister, uint32_t configValue)
```

Sets the IOMUXC pin configuration.

Note: The previous five parameters can be filled with the pin function ID macros.

Parameters

- muxRegister – The pin mux register
- muxMode – The pin mux mode
- inputRegister – The select input register
- inputDaisy – The input daisy
- configRegister – The config register
- configValue – The pin config value

FSL_IOMUXC_DRIVER_VERSION

IOMUXC driver version 1.0.0.

IOMUXC_PAD_DAP_TDI__JTAG_MUX_TDI

IOMUXC_PAD_DAP_TDI__MQS2_LEFT

IOMUXC_PAD_DAP_TDI__NETC_TMR_1588_ALARM1

IOMUXC_PAD_DAP_TDI__CAN2_TX

IOMUXC_PAD_DAP_TDI__FLEXIO2_FLEXIO_BIT30

IOMUXC_PAD_DAP_TDI__GPIO3_IO_BIT28

IOMUXC_PAD_DAP_TDI__LPUART5_RX

IOMUXC_PAD_DAP_TMS_SWDIO__JTAG_MUX_TMS

IOMUXC_PAD_DAP_TMS_SWDIO__CAN4_TX

IOMUXC_PAD_DAP_TMS_SWDIO__FLEXIO2_FLEXIO_BIT31

IOMUXC_PAD_DAP_TMS_SWDIO__GPIO3_IO_BIT29

IOMUXC_PAD_DAP_TMS_SWDIO__LPUART5_RTS_B

IOMUXC_PAD_DAP_TCLK_SWCLK__JTAG_MUX_TCK

IOMUXC_PAD_DAP_TCLK_SWCLK__CAN4_RX

IOMUXC_PAD_DAP_TCLK_SWCLK__FLEXIO1_FLEXIO_BIT30

IOMUXC_PAD_DAP_TCLK_SWCLK__GPIO3_IO_BIT30

IOMUXC_PAD_DAP_TCLK_SWCLK__LPUART5_CTS_B

IOMUXC_PAD_DAP_TDO_TRACESWO__JTAG_MUX_TDO

IOMUXC_PAD_DAP_TDO_TRACESWO__MQS2_RIGHT

IOMUXC_PAD_DAP_TDO_TRACESWO__NETC_TMR_1588_ALARM2

IOMUXC_PAD_DAP_TDO_TRACESWO__CAN2_RX
IOMUXC_PAD_DAP_TDO_TRACESWO__FLEXIO1_FLEXIO_BIT31
IOMUXC_PAD_DAP_TDO_TRACESWO__GPIO3_IO_BIT31
IOMUXC_PAD_DAP_TDO_TRACESWO__LPUART5_TX
IOMUXC_PAD_GPIO_IO00__GPIO2_IO_BIT0
IOMUXC_PAD_GPIO_IO00__LPI2C3_SDA
IOMUXC_PAD_GPIO_IO00__LPSPI6_PCS0
IOMUXC_PAD_GPIO_IO00__LPUART5_TX
IOMUXC_PAD_GPIO_IO00__LPI2C5_SDA
IOMUXC_PAD_GPIO_IO00__FLEXIO1_FLEXIO_BIT0
IOMUXC_PAD_GPIO_IO01__GPIO2_IO_BIT1
IOMUXC_PAD_GPIO_IO01__LPI2C3_SCL
IOMUXC_PAD_GPIO_IO01__LPSPI6_SIN
IOMUXC_PAD_GPIO_IO01__LPUART5_RX
IOMUXC_PAD_GPIO_IO01__LPI2C5_SCL
IOMUXC_PAD_GPIO_IO01__FLEXIO1_FLEXIO_BIT1
IOMUXC_PAD_GPIO_IO02__GPIO2_IO_BIT2
IOMUXC_PAD_GPIO_IO02__LPI2C4_SDA
IOMUXC_PAD_GPIO_IO02__LPSPI6_SOUT
IOMUXC_PAD_GPIO_IO02__LPUART5_CTS_B
IOMUXC_PAD_GPIO_IO02__LPI2C6_SDA
IOMUXC_PAD_GPIO_IO02__FLEXIO1_FLEXIO_BIT2
IOMUXC_PAD_GPIO_IO03__GPIO2_IO_BIT3
IOMUXC_PAD_GPIO_IO03__LPI2C4_SCL
IOMUXC_PAD_GPIO_IO03__LPSPI6_SCK
IOMUXC_PAD_GPIO_IO03__LPUART5_RTS_B
IOMUXC_PAD_GPIO_IO03__LPI2C6_SCL
IOMUXC_PAD_GPIO_IO03__FLEXIO1_FLEXIO_BIT3
IOMUXC_PAD_GPIO_IO04__GPIO2_IO_BIT4
IOMUXC_PAD_GPIO_IO04__TPM3_CH0
IOMUXC_PAD_GPIO_IO04__PDM_CLK
IOMUXC_PAD_GPIO_IO04__CAN4_TX
IOMUXC_PAD_GPIO_IO04__LPSPI7_PCS0

IOMUXC_PAD_GPIO_IO04__LPUART6_TX
IOMUXC_PAD_GPIO_IO04__LPI2C6_SDA
IOMUXC_PAD_GPIO_IO04__FLEXIO1_FLEXIO_BIT4
IOMUXC_PAD_GPIO_IO05__GPIO2_IO_BIT5
IOMUXC_PAD_GPIO_IO05__TPM4_CH0
IOMUXC_PAD_GPIO_IO05__PDM_BIT_STREAM_BIT0
IOMUXC_PAD_GPIO_IO05__CAN4_RX
IOMUXC_PAD_GPIO_IO05__LPSPI7_SIN
IOMUXC_PAD_GPIO_IO05__LPUART6_RX
IOMUXC_PAD_GPIO_IO05__LPI2C6_SCL
IOMUXC_PAD_GPIO_IO05__FLEXIO1_FLEXIO_BIT5
IOMUXC_PAD_GPIO_IO06__GPIO2_IO_BIT6
IOMUXC_PAD_GPIO_IO06__TPM5_CH0
IOMUXC_PAD_GPIO_IO06__PDM_BIT_STREAM_BIT1
IOMUXC_PAD_GPIO_IO06__LPSPI7_SOUT
IOMUXC_PAD_GPIO_IO06__LPUART6_CTS_B
IOMUXC_PAD_GPIO_IO06__LPI2C7_SDA
IOMUXC_PAD_GPIO_IO06__FLEXIO1_FLEXIO_BIT6
IOMUXC_PAD_GPIO_IO07__GPIO2_IO_BIT7
IOMUXC_PAD_GPIO_IO07__LPSPI3_PCS1
IOMUXC_PAD_GPIO_IO07__LPSPI7_SCK
IOMUXC_PAD_GPIO_IO07__LPUART6_RTS_B
IOMUXC_PAD_GPIO_IO07__LPI2C7_SCL
IOMUXC_PAD_GPIO_IO07__FLEXIO1_FLEXIO_BIT7
IOMUXC_PAD_GPIO_IO08__GPIO2_IO_BIT8
IOMUXC_PAD_GPIO_IO08__LPSPI3_PCS0
IOMUXC_PAD_GPIO_IO08__TPM6_CH0
IOMUXC_PAD_GPIO_IO08__LPUART7_TX
IOMUXC_PAD_GPIO_IO08__LPI2C7_SDA
IOMUXC_PAD_GPIO_IO08__FLEXIO1_FLEXIO_BIT8
IOMUXC_PAD_GPIO_IO09__GPIO2_IO_BIT9
IOMUXC_PAD_GPIO_IO09__LPSPI3_SIN
IOMUXC_PAD_GPIO_IO09__TPM3_EXTCLK

IOMUXC_PAD_GPIO_IO09__LPUART7_RX
IOMUXC_PAD_GPIO_IO09__LPI2C7_SCL
IOMUXC_PAD_GPIO_IO09__FLEXIO1_FLEXIO_BIT9
IOMUXC_PAD_GPIO_IO10__GPIO2_IO_BIT10
IOMUXC_PAD_GPIO_IO10__LPSPI3_SOUT
IOMUXC_PAD_GPIO_IO10__TPM4_EXTCLK
IOMUXC_PAD_GPIO_IO10__LPUART7_CTS_B
IOMUXC_PAD_GPIO_IO10__LPI2C8_SDA
IOMUXC_PAD_GPIO_IO10__FLEXIO1_FLEXIO_BIT10
IOMUXC_PAD_GPIO_IO11__GPIO2_IO_BIT11
IOMUXC_PAD_GPIO_IO11__LPSPI3_SCK
IOMUXC_PAD_GPIO_IO11__TPM5_EXTCLK
IOMUXC_PAD_GPIO_IO11__LPUART7_RTS_B
IOMUXC_PAD_GPIO_IO11__LPI2C8_SCL
IOMUXC_PAD_GPIO_IO11__FLEXIO1_FLEXIO_BIT11
IOMUXC_PAD_GPIO_IO12__GPIO2_IO_BIT12
IOMUXC_PAD_GPIO_IO12__TPM3_CH2
IOMUXC_PAD_GPIO_IO12__PDM_BIT_STREAM_BIT2
IOMUXC_PAD_GPIO_IO12__FLEXIO1_FLEXIO_BIT12
IOMUXC_PAD_GPIO_IO12__LPSPI8_PCS0
IOMUXC_PAD_GPIO_IO12__LPUART8_TX
IOMUXC_PAD_GPIO_IO12__LPI2C8_SDA
IOMUXC_PAD_GPIO_IO12__SAI3_RX_SYNC
IOMUXC_PAD_GPIO_IO13__GPIO2_IO_BIT13
IOMUXC_PAD_GPIO_IO13__TPM4_CH2
IOMUXC_PAD_GPIO_IO13__PDM_BIT_STREAM_BIT3
IOMUXC_PAD_GPIO_IO13__LPSPI8_SIN
IOMUXC_PAD_GPIO_IO13__LPUART8_RX
IOMUXC_PAD_GPIO_IO13__LPI2C8_SCL
IOMUXC_PAD_GPIO_IO13__FLEXIO1_FLEXIO_BIT13
IOMUXC_PAD_GPIO_IO14__GPIO2_IO_BIT14
IOMUXC_PAD_GPIO_IO14__LPUART3_TX
IOMUXC_PAD_GPIO_IO14__LPSPI8_SOUT

IOMUXC_PAD_GPIO_IO14__LPUART8_CTS_B
IOMUXC_PAD_GPIO_IO14__LPUART4_TX
IOMUXC_PAD_GPIO_IO14__FLEXIO1_FLEXIO_BIT14
IOMUXC_PAD_GPIO_IO15__GPIO2_IO_BIT15
IOMUXC_PAD_GPIO_IO15__LPUART3_RX
IOMUXC_PAD_GPIO_IO15__LPSPI8_SCK
IOMUXC_PAD_GPIO_IO15__LPUART8_RTS_B
IOMUXC_PAD_GPIO_IO15__LPUART4_RX
IOMUXC_PAD_GPIO_IO15__FLEXIO1_FLEXIO_BIT15
IOMUXC_PAD_GPIO_IO16__GPIO2_IO_BIT16
IOMUXC_PAD_GPIO_IO16__SAI3_TX_BCLK
IOMUXC_PAD_GPIO_IO16__PDM_BIT_STREAM_BIT2
IOMUXC_PAD_GPIO_IO16__LPUART3_CTS_B
IOMUXC_PAD_GPIO_IO16__LPSPI4_PCS2
IOMUXC_PAD_GPIO_IO16__LPUART4_CTS_B
IOMUXC_PAD_GPIO_IO16__FLEXIO1_FLEXIO_BIT16
IOMUXC_PAD_GPIO_IO17__GPIO2_IO_BIT17
IOMUXC_PAD_GPIO_IO17__SAI3_MCLK
IOMUXC_PAD_GPIO_IO17__LPUART3_RTS_B
IOMUXC_PAD_GPIO_IO17__LPSPI4_PCS1
IOMUXC_PAD_GPIO_IO17__LPUART4_RTS_B
IOMUXC_PAD_GPIO_IO17__FLEXIO1_FLEXIO_BIT17
IOMUXC_PAD_GPIO_IO18__GPIO2_IO_BIT18
IOMUXC_PAD_GPIO_IO18__SAI3_RX_BCLK
IOMUXC_PAD_GPIO_IO18__LPSPI5_PCS0
IOMUXC_PAD_GPIO_IO18__LPSPI4_PCS0
IOMUXC_PAD_GPIO_IO18__TPM5_CH2
IOMUXC_PAD_GPIO_IO18__FLEXIO1_FLEXIO_BIT18
IOMUXC_PAD_GPIO_IO19__GPIO2_IO_BIT19
IOMUXC_PAD_GPIO_IO19__SAI3_RX_SYNC
IOMUXC_PAD_GPIO_IO19__PDM_BIT_STREAM_BIT3
IOMUXC_PAD_GPIO_IO19__FLEXIO1_FLEXIO_BIT19
IOMUXC_PAD_GPIO_IO19__LPSPI5_SIN

IOMUXC_PAD_GPIO_IO19__LPSPI4_SIN
IOMUXC_PAD_GPIO_IO19__TPM6_CH2
IOMUXC_PAD_GPIO_IO19__SAI3_TX_DATA_BIT0
IOMUXC_PAD_GPIO_IO20__GPIO2_IO_BIT20
IOMUXC_PAD_GPIO_IO20__SAI3_RX_DATA_BIT0
IOMUXC_PAD_GPIO_IO20__PDM_BIT_STREAM_BIT0
IOMUXC_PAD_GPIO_IO20__LPSPI5_SOUT
IOMUXC_PAD_GPIO_IO20__LPSPI4_SOUT
IOMUXC_PAD_GPIO_IO20__TPM3_CH1
IOMUXC_PAD_GPIO_IO20__FLEXIO1_FLEXIO_BIT20
IOMUXC_PAD_GPIO_IO21__GPIO2_IO_BIT21
IOMUXC_PAD_GPIO_IO21__SAI3_TX_DATA_BIT0
IOMUXC_PAD_GPIO_IO21__PDM_CLK
IOMUXC_PAD_GPIO_IO21__FLEXIO1_FLEXIO_BIT21
IOMUXC_PAD_GPIO_IO21__LPSPI5_SCK
IOMUXC_PAD_GPIO_IO21__LPSPI4_SCK
IOMUXC_PAD_GPIO_IO21__TPM4_CH1
IOMUXC_PAD_GPIO_IO21__SAI3_RX_BCLK
IOMUXC_PAD_GPIO_IO22__GPIO2_IO_BIT22
IOMUXC_PAD_GPIO_IO22__USDHC3_CLK
IOMUXC_PAD_GPIO_IO22__SPDIF_IN
IOMUXC_PAD_GPIO_IO22__CAN5_TX
IOMUXC_PAD_GPIO_IO22__TPM5_CH1
IOMUXC_PAD_GPIO_IO22__TPM6_EXTCLK
IOMUXC_PAD_GPIO_IO22__LPI2C5_SDA
IOMUXC_PAD_GPIO_IO22__FLEXIO1_FLEXIO_BIT22
IOMUXC_PAD_GPIO_IO23__GPIO2_IO_BIT23
IOMUXC_PAD_GPIO_IO23__USDHC3_CMD
IOMUXC_PAD_GPIO_IO23__SPDIF_OUT
IOMUXC_PAD_GPIO_IO23__CAN5_RX
IOMUXC_PAD_GPIO_IO23__TPM6_CH1
IOMUXC_PAD_GPIO_IO23__LPI2C5_SCL
IOMUXC_PAD_GPIO_IO23__FLEXIO1_FLEXIO_BIT23

IOMUXC_PAD_GPIO_IO24___GPIO2_IO_BIT24
IOMUXC_PAD_GPIO_IO24___USDHC3_DATA0
IOMUXC_PAD_GPIO_IO24___TPM3_CH3
IOMUXC_PAD_GPIO_IO24___JTAG_MUX_TDO
IOMUXC_PAD_GPIO_IO24___LPSPI6_PCS1
IOMUXC_PAD_GPIO_IO24___FLEXIO1_FLEXIO_BIT24
IOMUXC_PAD_GPIO_IO25___GPIO2_IO_BIT25
IOMUXC_PAD_GPIO_IO25___USDHC3_DATA1
IOMUXC_PAD_GPIO_IO25___CAN2_TX
IOMUXC_PAD_GPIO_IO25___TPM4_CH3
IOMUXC_PAD_GPIO_IO25___JTAG_MUX_TCK
IOMUXC_PAD_GPIO_IO25___LPSPI7_PCS1
IOMUXC_PAD_GPIO_IO25___FLEXIO1_FLEXIO_BIT25
IOMUXC_PAD_GPIO_IO26___GPIO2_IO_BIT26
IOMUXC_PAD_GPIO_IO26___USDHC3_DATA2
IOMUXC_PAD_GPIO_IO26___PDM_BIT_STREAM_BIT1
IOMUXC_PAD_GPIO_IO26___FLEXIO1_FLEXIO_BIT26
IOMUXC_PAD_GPIO_IO26___TPM5_CH3
IOMUXC_PAD_GPIO_IO26___JTAG_MUX_TDI
IOMUXC_PAD_GPIO_IO26___LPSPI8_PCS1
IOMUXC_PAD_GPIO_IO26___SAI3_TX_SYNC
IOMUXC_PAD_GPIO_IO27___GPIO2_IO_BIT27
IOMUXC_PAD_GPIO_IO27___USDHC3_DATA3
IOMUXC_PAD_GPIO_IO27___CAN2_RX
IOMUXC_PAD_GPIO_IO27___TPM6_CH3
IOMUXC_PAD_GPIO_IO27___JTAG_MUX_TMS
IOMUXC_PAD_GPIO_IO27___LPSPI5_PCS1
IOMUXC_PAD_GPIO_IO27___FLEXIO1_FLEXIO_BIT27
IOMUXC_PAD_GPIO_IO28___GPIO2_IO_BIT28
IOMUXC_PAD_GPIO_IO28___LPI2C3_SDA
IOMUXC_PAD_GPIO_IO28___CAN3_TX
IOMUXC_PAD_GPIO_IO28___FLEXIO1_FLEXIO_BIT28
IOMUXC_PAD_GPIO_IO29___GPIO2_IO_BIT29

IOMUXC_PAD_GPIO_IO29__LPI2C3_SCL
IOMUXC_PAD_GPIO_IO29__CAN3_RX
IOMUXC_PAD_GPIO_IO29__FLEXIO1_FLEXIO_BIT29
IOMUXC_PAD_GPIO_IO30__GPIO2_IO_BIT30
IOMUXC_PAD_GPIO_IO30__LPI2C4_SDA
IOMUXC_PAD_GPIO_IO30__CAN5_TX
IOMUXC_PAD_GPIO_IO30__FLEXIO1_FLEXIO_BIT30
IOMUXC_PAD_GPIO_IO31__GPIO2_IO_BIT31
IOMUXC_PAD_GPIO_IO31__LPI2C4_SCL
IOMUXC_PAD_GPIO_IO31__CAN5_RX
IOMUXC_PAD_GPIO_IO31__FLEXIO1_FLEXIO_BIT31
IOMUXC_PAD_GPIO_IO32__GPIO5_IO_BIT12
IOMUXC_PAD_GPIO_IO32__PCIE1_CLKREQ_B
IOMUXC_PAD_GPIO_IO32__LPUART6_TX
IOMUXC_PAD_GPIO_IO32__LPSPI4_PCS2
IOMUXC_PAD_GPIO_IO33__GPIO5_IO_BIT13
IOMUXC_PAD_GPIO_IO33__LPUART6_RX
IOMUXC_PAD_GPIO_IO33__LPSPI4_PCS1
IOMUXC_PAD_GPIO_IO34__GPIO5_IO_BIT14
IOMUXC_PAD_GPIO_IO34__LPUART6_CTS_B
IOMUXC_PAD_GPIO_IO34__LPSPI4_PCS0
IOMUXC_PAD_GPIO_IO35__GPIO5_IO_BIT15
IOMUXC_PAD_GPIO_IO35__PCIE2_CLKREQ_B
IOMUXC_PAD_GPIO_IO35__LPUART6_RTS_B
IOMUXC_PAD_GPIO_IO35__LPSPI4_SIN
IOMUXC_PAD_GPIO_IO36__LPSPI4_SOUT
IOMUXC_PAD_GPIO_IO36__GPIO5_IO_BIT16
IOMUXC_PAD_GPIO_IO36__LPUART7_TX
IOMUXC_PAD_GPIO_IO37__GPIO5_IO_BIT17
IOMUXC_PAD_GPIO_IO37__LPUART7_RX
IOMUXC_PAD_GPIO_IO37__LPSPI4_SCK
IOMUXC_PAD_CCM_CLKO1__CLKO_1
IOMUXC_PAD_CCM_CLKO1__NETC_TMR_1588_TRIG1

IOMUXC_PAD_CCM_CLKO1__FLEXIO1_FLEXIO_BIT26
IOMUXC_PAD_CCM_CLKO1__GPIO3_IO_BIT26
IOMUXC_PAD_CCM_CLKO2__GPIO3_IO_BIT27
IOMUXC_PAD_CCM_CLKO2__CLKO_2
IOMUXC_PAD_CCM_CLKO2__NETC_TMR_1588_PP1
IOMUXC_PAD_CCM_CLKO2__FLEXIO1_FLEXIO_BIT27
IOMUXC_PAD_CCM_CLKO3__CLKO_3
IOMUXC_PAD_CCM_CLKO3__NETC_TMR_1588_TRIG2
IOMUXC_PAD_CCM_CLKO3__CAN3_TX
IOMUXC_PAD_CCM_CLKO3__FLEXIO2_FLEXIO_BIT28
IOMUXC_PAD_CCM_CLKO3__GPIO4_IO_BIT28
IOMUXC_PAD_CCM_CLKO4__CLKO_4
IOMUXC_PAD_CCM_CLKO4__NETC_TMR_1588_PP2
IOMUXC_PAD_CCM_CLKO4__CAN3_RX
IOMUXC_PAD_CCM_CLKO4__FLEXIO2_FLEXIO_BIT29
IOMUXC_PAD_CCM_CLKO4__GPIO4_IO_BIT29
IOMUXC_PAD_ENET1_MDC__NETC_MDC
IOMUXC_PAD_ENET1_MDC__LPUART3_DCD_B
IOMUXC_PAD_ENET1_MDC__I3C2_SCL
IOMUXC_PAD_ENET1_MDC__USB1_OTG_ID
IOMUXC_PAD_ENET1_MDC__FLEXIO2_FLEXIO_BIT0
IOMUXC_PAD_ENET1_MDC__GPIO4_IO_BIT0
IOMUXC_PAD_ENET1_MDIO__NETC_MDIO
IOMUXC_PAD_ENET1_MDIO__LPUART3_RIN_B
IOMUXC_PAD_ENET1_MDIO__I3C2_SDA
IOMUXC_PAD_ENET1_MDIO__USB1_OTG_PWR
IOMUXC_PAD_ENET1_MDIO__FLEXIO2_FLEXIO_BIT1
IOMUXC_PAD_ENET1_MDIO__GPIO4_IO_BIT1
IOMUXC_PAD_ENET1_TD3__ETH0_RGMII_TD3
IOMUXC_PAD_ENET1_TD3__CAN2_TX
IOMUXC_PAD_ENET1_TD3__USB2_OTG_ID
IOMUXC_PAD_ENET1_TD3__FLEXIO2_FLEXIO_BIT2
IOMUXC_PAD_ENET1_TD3__GPIO4_IO_BIT2

IOMUXC_PAD_ENET1_TD2__ETH0_RGMII_TD2
IOMUXC_PAD_ENET1_TD2__ETH0_RMII_REF50_CLK
IOMUXC_PAD_ENET1_TD2__CAN2_RX
IOMUXC_PAD_ENET1_TD2__USB2_OTG_OC
IOMUXC_PAD_ENET1_TD2__FLEXIO2_FLEXIO_BIT3
IOMUXC_PAD_ENET1_TD2__GPIO4_IO_BIT3
IOMUXC_PAD_ENET1_TD1__ETH0_RGMII_TD1
IOMUXC_PAD_ENET1_TD1__LPUART3_RTS_B
IOMUXC_PAD_ENET1_TD1__I3C2_PUR
IOMUXC_PAD_ENET1_TD1__USB1_OTG_OC
IOMUXC_PAD_ENET1_TD1__FLEXIO2_FLEXIO_BIT4
IOMUXC_PAD_ENET1_TD1__GPIO4_IO_BIT4
IOMUXC_PAD_ENET1_TD1__I3C2_PUR_B
IOMUXC_PAD_ENET1_TD1__ETH0_RMII_TXD1
IOMUXC_PAD_ENET1_TD0__ETH0_RGMII_TD0
IOMUXC_PAD_ENET1_TD0__LPUART3_TX
IOMUXC_PAD_ENET1_TD0__ETH0_RMII_TXD0
IOMUXC_PAD_ENET1_TD0__FLEXIO2_FLEXIO_BIT5
IOMUXC_PAD_ENET1_TD0__GPIO4_IO_BIT5
IOMUXC_PAD_ENET1_TX_CTL__ETH0_RGMII_TX_CTL
IOMUXC_PAD_ENET1_TX_CTL__LPUART3_DTR_B
IOMUXC_PAD_ENET1_TX_CTL__ETH0_RMII_TX_EN
IOMUXC_PAD_ENET1_TX_CTL__FLEXIO2_FLEXIO_BIT6
IOMUXC_PAD_ENET1_TX_CTL__GPIO4_IO_BIT6
IOMUXC_PAD_ENET1_TXC__ETH0_RGMII_TX_CLK
IOMUXC_PAD_ENET1_TXC__ENET_CLK_ROOT
IOMUXC_PAD_ENET1_TXC__FLEXIO2_FLEXIO_BIT7
IOMUXC_PAD_ENET1_TXC__GPIO4_IO_BIT7
IOMUXC_PAD_ENET1_RX_CTL__ETH0_RGMII_RX_CTL
IOMUXC_PAD_ENET1_RX_CTL__LPUART3_DSR_B
IOMUXC_PAD_ENET1_RX_CTL__ETH0_RMII_CRS_DV
IOMUXC_PAD_ENET1_RX_CTL__USB2_OTG_PWR
IOMUXC_PAD_ENET1_RX_CTL__FLEXIO2_FLEXIO_BIT8

IOMUXC_PAD_ENET1_RX_CTL__GPIO4_IO_BIT8
IOMUXC_PAD_ENET1_RXC__ETH0_RGMII_RX_CLK
IOMUXC_PAD_ENET1_RXC__ETH0_RMII_RX_ER
IOMUXC_PAD_ENET1_RXC__FLEXIO2_FLEXIO_BIT9
IOMUXC_PAD_ENET1_RXC__GPIO4_IO_BIT9
IOMUXC_PAD_ENET1_RD0__ETH0_RGMII_RD0
IOMUXC_PAD_ENET1_RD0__LPUART3_RX
IOMUXC_PAD_ENET1_RD0__ETH0_RMII_RXD0
IOMUXC_PAD_ENET1_RD0__FLEXIO2_FLEXIO_BIT10
IOMUXC_PAD_ENET1_RD0__GPIO4_IO_BIT10
IOMUXC_PAD_ENET1_RD1__ETH0_RGMII_RD1
IOMUXC_PAD_ENET1_RD1__LPUART3_CTS_B
IOMUXC_PAD_ENET1_RD1__ETH0_RMII_RXD1
IOMUXC_PAD_ENET1_RD1__LPTMR2_ALT1
IOMUXC_PAD_ENET1_RD1__FLEXIO2_FLEXIO_BIT11
IOMUXC_PAD_ENET1_RD1__GPIO4_IO_BIT11
IOMUXC_PAD_ENET1_RD2__ETH0_RGMII_RD2
IOMUXC_PAD_ENET1_RD2__ETH0_RMII_RX_ER
IOMUXC_PAD_ENET1_RD2__LPTMR2_ALT2
IOMUXC_PAD_ENET1_RD2__FLEXIO2_FLEXIO_BIT12
IOMUXC_PAD_ENET1_RD2__GPIO4_IO_BIT12
IOMUXC_PAD_ENET1_RD3__ETH0_RGMII_RD3
IOMUXC_PAD_ENET1_RD3__LPTMR2_ALT3
IOMUXC_PAD_ENET1_RD3__FLEXIO2_FLEXIO_BIT13
IOMUXC_PAD_ENET1_RD3__GPIO4_IO_BIT13
IOMUXC_PAD_ENET2_MDC__NETC_MDC
IOMUXC_PAD_ENET2_MDC__LPUART4_DCD_B
IOMUXC_PAD_ENET2_MDC__SAI2_RX_SYNC
IOMUXC_PAD_ENET2_MDC__FLEXIO2_FLEXIO_BIT14
IOMUXC_PAD_ENET2_MDC__GPIO4_IO_BIT14
IOMUXC_PAD_ENET2_MDIO__NETC_MDIO
IOMUXC_PAD_ENET2_MDIO__LPUART4_RIN_B
IOMUXC_PAD_ENET2_MDIO__SAI2_RX_BCLK

IOMUXC_PAD_ENET2_MDIO__FLEXIO2_FLEXIO_BIT15
IOMUXC_PAD_ENET2_MDIO__GPIO4_IO_BIT15
IOMUXC_PAD_ENET2_TD3__SAI2_RX_DATA_BIT0
IOMUXC_PAD_ENET2_TD3__FLEXIO2_FLEXIO_BIT16
IOMUXC_PAD_ENET2_TD3__GPIO4_IO_BIT16
IOMUXC_PAD_ENET2_TD3__ETH1_RGMII_TD3
IOMUXC_PAD_ENET2_TD2__ETH1_RGMII_TD2
IOMUXC_PAD_ENET2_TD2__ETH1_RMII_REF50_CLK
IOMUXC_PAD_ENET2_TD2__SAI2_RX_DATA_BIT1
IOMUXC_PAD_ENET2_TD2__SAI4_TX_SYNC
IOMUXC_PAD_ENET2_TD2__FLEXIO2_FLEXIO_BIT17
IOMUXC_PAD_ENET2_TD2__GPIO4_IO_BIT17
IOMUXC_PAD_ENET2_TD1__ETH1_RGMII_TD1
IOMUXC_PAD_ENET2_TD1__LPUART4_RTS_B
IOMUXC_PAD_ENET2_TD1__SAI2_RX_DATA_BIT2
IOMUXC_PAD_ENET2_TD1__SAI4_TX_BCLK
IOMUXC_PAD_ENET2_TD1__FLEXIO2_FLEXIO_BIT18
IOMUXC_PAD_ENET2_TD1__GPIO4_IO_BIT18
IOMUXC_PAD_ENET2_TD1__ETH1_RMII_TXD1
IOMUXC_PAD_ENET2_TD0__ETH1_RGMII_TD0
IOMUXC_PAD_ENET2_TD0__LPUART4_TX
IOMUXC_PAD_ENET2_TD0__SAI2_RX_DATA_BIT3
IOMUXC_PAD_ENET2_TD0__SAI4_TX_DATA_BIT0
IOMUXC_PAD_ENET2_TD0__FLEXIO2_FLEXIO_BIT19
IOMUXC_PAD_ENET2_TD0__GPIO4_IO_BIT19
IOMUXC_PAD_ENET2_TD0__ETH1_RMII_TXD0
IOMUXC_PAD_ENET2_TX_CTL__ETH1_RGMII_TX_CTL
IOMUXC_PAD_ENET2_TX_CTL__LPUART4_DTR_B
IOMUXC_PAD_ENET2_TX_CTL__SAI2_TX_SYNC
IOMUXC_PAD_ENET2_TX_CTL__ETH1_RMII_TX_EN
IOMUXC_PAD_ENET2_TX_CTL__FLEXIO2_FLEXIO_BIT20
IOMUXC_PAD_ENET2_TX_CTL__GPIO4_IO_BIT20
IOMUXC_PAD_ENET2_TXC__ETH1_RGMII_TX_CLK

IOMUXC_PAD_ENET2_TXC__ENET_CLK_ROOT
IOMUXC_PAD_ENET2_TXC__SAI2_TX_BCLK
IOMUXC_PAD_ENET2_TXC__FLEXIO2_FLEXIO_BIT21
IOMUXC_PAD_ENET2_TXC__GPIO4_IO_BIT21
IOMUXC_PAD_ENET2_RX_CTL__ETH1_RGMII_RX_CTL
IOMUXC_PAD_ENET2_RX_CTL__LPUART4_DSR_B
IOMUXC_PAD_ENET2_RX_CTL__SAI2_TX_DATA_BIT0
IOMUXC_PAD_ENET2_RX_CTL__FLEXIO2_FLEXIO_BIT22
IOMUXC_PAD_ENET2_RX_CTL__GPIO4_IO_BIT22
IOMUXC_PAD_ENET2_RX_CTL__ETH1_RMII_CRS_DV
IOMUXC_PAD_ENET2_RXC__ETH1_RGMII_RX_CLK
IOMUXC_PAD_ENET2_RXC__ETH1_RMII_RX_ER
IOMUXC_PAD_ENET2_RXC__SAI2_TX_DATA_BIT1
IOMUXC_PAD_ENET2_RXC__SAI4_RX_SYNC
IOMUXC_PAD_ENET2_RXC__FLEXIO2_FLEXIO_BIT23
IOMUXC_PAD_ENET2_RXC__GPIO4_IO_BIT23
IOMUXC_PAD_ENET2_RD0__ETH1_RGMII_RD0
IOMUXC_PAD_ENET2_RD0__LPUART4_RX
IOMUXC_PAD_ENET2_RD0__SAI2_TX_DATA_BIT2
IOMUXC_PAD_ENET2_RD0__SAI4_RX_BCLK
IOMUXC_PAD_ENET2_RD0__FLEXIO2_FLEXIO_BIT24
IOMUXC_PAD_ENET2_RD0__GPIO4_IO_BIT24
IOMUXC_PAD_ENET2_RD0__ETH1_RMII_RXD0
IOMUXC_PAD_ENET2_RD1__ETH1_RGMII_RD1
IOMUXC_PAD_ENET2_RD1__SPDIF_IN
IOMUXC_PAD_ENET2_RD1__SAI2_TX_DATA_BIT3
IOMUXC_PAD_ENET2_RD1__SAI4_RX_DATA_BIT0
IOMUXC_PAD_ENET2_RD1__FLEXIO2_FLEXIO_BIT25
IOMUXC_PAD_ENET2_RD1__GPIO4_IO_BIT25
IOMUXC_PAD_ENET2_RD1__ETH1_RMII_RXD1
IOMUXC_PAD_ENET2_RD2__ETH1_RGMII_RD2
IOMUXC_PAD_ENET2_RD2__LPUART4_CTS_B
IOMUXC_PAD_ENET2_RD2__SAI2_MCLK

IOMUXC_PAD_ENET2_RD2__MQS2_RIGHT
IOMUXC_PAD_ENET2_RD2__FLEXIO2_FLEXIO_BIT26
IOMUXC_PAD_ENET2_RD2__GPIO4_IO_BIT26
IOMUXC_PAD_ENET2_RD2__ETH1_RMII_RX_ER
IOMUXC_PAD_ENET2_RD3__ETH1_RGMII_RD3
IOMUXC_PAD_ENET2_RD3__SPDIF_OUT
IOMUXC_PAD_ENET2_RD3__SPDIF_IN
IOMUXC_PAD_ENET2_RD3__MQS2_LEFT
IOMUXC_PAD_ENET2_RD3__FLEXIO2_FLEXIO_BIT27
IOMUXC_PAD_ENET2_RD3__GPIO4_IO_BIT27
IOMUXC_PAD_SD1_CLK__FLEXIO1_FLEXIO_BIT8
IOMUXC_PAD_SD1_CLK__GPIO3_IO_BIT8
IOMUXC_PAD_SD1_CLK__USDHC1_CLK
IOMUXC_PAD_SD1_CMD__USDHC1_CMD
IOMUXC_PAD_SD1_CMD__FLEXIO1_FLEXIO_BIT9
IOMUXC_PAD_SD1_CMD__GPIO3_IO_BIT9
IOMUXC_PAD_SD1_DATA0__USDHC1_DATA0
IOMUXC_PAD_SD1_DATA0__FLEXIO1_FLEXIO_BIT10
IOMUXC_PAD_SD1_DATA0__GPIO3_IO_BIT10
IOMUXC_PAD_SD1_DATA1__USDHC1_DATA1
IOMUXC_PAD_SD1_DATA1__FLEXIO1_FLEXIO_BIT11
IOMUXC_PAD_SD1_DATA1__GPIO3_IO_BIT11
IOMUXC_PAD_SD1_DATA2__USDHC1_DATA2
IOMUXC_PAD_SD1_DATA2__FLEXIO1_FLEXIO_BIT12
IOMUXC_PAD_SD1_DATA2__GPIO3_IO_BIT12
IOMUXC_PAD_SD1_DATA2__PMIC_READY
IOMUXC_PAD_SD1_DATA3__USDHC1_DATA3
IOMUXC_PAD_SD1_DATA3__FLEXSPI1_A_SS1_B
IOMUXC_PAD_SD1_DATA3__FLEXIO1_FLEXIO_BIT13
IOMUXC_PAD_SD1_DATA3__GPIO3_IO_BIT13
IOMUXC_PAD_SD1_DATA4__USDHC1_DATA4
IOMUXC_PAD_SD1_DATA4__FLEXSPI1_A_DATA_BIT4
IOMUXC_PAD_SD1_DATA4__FLEXIO1_FLEXIO_BIT14

IOMUXC_PAD_SD1_DATA4___GPIO3_IO_BIT14
IOMUXC_PAD_SD1_DATA4___XSPI_DATA_BIT4
IOMUXC_PAD_SD1_DATA5___USDHC1_DATA5
IOMUXC_PAD_SD1_DATA5___FLEXSPI1_A_DATA_BIT5
IOMUXC_PAD_SD1_DATA5___USDHC1_RESET_B
IOMUXC_PAD_SD1_DATA5___FLEXIO1_FLEXIO_BIT15
IOMUXC_PAD_SD1_DATA5___GPIO3_IO_BIT15
IOMUXC_PAD_SD1_DATA5___XSPI_DATA_BIT5
IOMUXC_PAD_SD1_DATA6___USDHC1_DATA6
IOMUXC_PAD_SD1_DATA6___FLEXSPI1_A_DATA_BIT6
IOMUXC_PAD_SD1_DATA6___USDHC1_CD_B
IOMUXC_PAD_SD1_DATA6___FLEXIO1_FLEXIO_BIT16
IOMUXC_PAD_SD1_DATA6___GPIO3_IO_BIT16
IOMUXC_PAD_SD1_DATA6___XSPI_DATA_BIT6
IOMUXC_PAD_SD1_DATA7___USDHC1_DATA7
IOMUXC_PAD_SD1_DATA7___FLEXSPI1_A_DATA_BIT7
IOMUXC_PAD_SD1_DATA7___USDHC1_WP
IOMUXC_PAD_SD1_DATA7___FLEXIO1_FLEXIO_BIT17
IOMUXC_PAD_SD1_DATA7___GPIO3_IO_BIT17
IOMUXC_PAD_SD1_DATA7___XSPI_DATA_BIT7
IOMUXC_PAD_SD1_STROBE___USDHC1_STROBE
IOMUXC_PAD_SD1_STROBE___FLEXSPI1_A_DQS
IOMUXC_PAD_SD1_STROBE___FLEXIO1_FLEXIO_BIT18
IOMUXC_PAD_SD1_STROBE___GPIO3_IO_BIT18
IOMUXC_PAD_SD1_STROBE___XSPI_DQS
IOMUXC_PAD_SD2_VSELECT___USDHC2_VSELECT
IOMUXC_PAD_SD2_VSELECT___USDHC2_WP
IOMUXC_PAD_SD2_VSELECT___LPTMR2_ALT3
IOMUXC_PAD_SD2_VSELECT___FLEXIO1_FLEXIO_BIT19
IOMUXC_PAD_SD2_VSELECT___GPIO3_IO_BIT19
IOMUXC_PAD_SD2_VSELECT___EXT_CLK1
IOMUXC_PAD_SD3_CLK___USDHC3_CLK
IOMUXC_PAD_SD3_CLK___FLEXSPI1_A_SCLK

IOMUXC_PAD_SD3_CLK__SAI5_TX_DATA_BIT1
IOMUXC_PAD_SD3_CLK__SAI5_RX_DATA_BIT0
IOMUXC_PAD_SD3_CLK__FLEXIO1_FLEXIO_BIT20
IOMUXC_PAD_SD3_CLK__GPIO3_IO_BIT20
IOMUXC_PAD_SD3_CLK__XSPI_CLK
IOMUXC_PAD_SD3_CMD__USDHC3_CMD
IOMUXC_PAD_SD3_CMD__FLEXSPI1_A_SS0_B
IOMUXC_PAD_SD3_CMD__SAI5_TX_DATA_BIT2
IOMUXC_PAD_SD3_CMD__SAI5_RX_SYNC
IOMUXC_PAD_SD3_CMD__FLEXIO1_FLEXIO_BIT21
IOMUXC_PAD_SD3_CMD__GPIO3_IO_BIT21
IOMUXC_PAD_SD3_CMD__XSPI_CS
IOMUXC_PAD_SD3_DATA0__USDHC3_DATA0
IOMUXC_PAD_SD3_DATA0__FLEXSPI1_A_DATA_BIT0
IOMUXC_PAD_SD3_DATA0__SAI5_TX_DATA_BIT3
IOMUXC_PAD_SD3_DATA0__SAI5_RX_BCLK
IOMUXC_PAD_SD3_DATA0__FLEXIO1_FLEXIO_BIT22
IOMUXC_PAD_SD3_DATA0__GPIO3_IO_BIT22
IOMUXC_PAD_SD3_DATA0__XSPI_DATA_BIT0
IOMUXC_PAD_SD3_DATA1__USDHC3_DATA1
IOMUXC_PAD_SD3_DATA1__FLEXSPI1_A_DATA_BIT1
IOMUXC_PAD_SD3_DATA1__SAI5_RX_DATA_BIT1
IOMUXC_PAD_SD3_DATA1__SAI5_TX_DATA_BIT0
IOMUXC_PAD_SD3_DATA1__FLEXIO1_FLEXIO_BIT23
IOMUXC_PAD_SD3_DATA1__GPIO3_IO_BIT23
IOMUXC_PAD_SD3_DATA1__XSPI_DATA_BIT1
IOMUXC_PAD_SD3_DATA2__USDHC3_DATA2
IOMUXC_PAD_SD3_DATA2__FLEXSPI1_A_DATA_BIT2
IOMUXC_PAD_SD3_DATA2__SAI5_RX_DATA_BIT2
IOMUXC_PAD_SD3_DATA2__SAI5_TX_SYNC
IOMUXC_PAD_SD3_DATA2__FLEXIO1_FLEXIO_BIT24
IOMUXC_PAD_SD3_DATA2__GPIO3_IO_BIT24
IOMUXC_PAD_SD3_DATA2__XSPI_DATA_BIT2

IOMUXC_PAD_SD3_DATA3__USDHC3_DATA3
IOMUXC_PAD_SD3_DATA3__FLEXSPI1_A_DATA_BIT3
IOMUXC_PAD_SD3_DATA3__SAI5_RX_DATA_BIT3
IOMUXC_PAD_SD3_DATA3__SAI5_TX_BCLK
IOMUXC_PAD_SD3_DATA3__FLEXIO1_FLEXIO_BIT25
IOMUXC_PAD_SD3_DATA3__GPIO3_IO_BIT25
IOMUXC_PAD_SD3_DATA3__XSPI_DATA_BIT3
IOMUXC_PAD_XSPI1_DATA0__FLEXSPI1_A_DATA_BIT0
IOMUXC_PAD_XSPI1_DATA0__SAI2_TX_DATA_BIT4
IOMUXC_PAD_XSPI1_DATA0__SAI4_TX_BCLK
IOMUXC_PAD_XSPI1_DATA0__SAI4_RX_DATA_BIT1
IOMUXC_PAD_XSPI1_DATA0__XSPI_DATA_BIT0
IOMUXC_PAD_XSPI1_DATA0__GPIO5_IO_BIT0
IOMUXC_PAD_XSPI1_DATA1__FLEXSPI1_A_DATA_BIT1
IOMUXC_PAD_XSPI1_DATA1__SAI2_TX_DATA_BIT5
IOMUXC_PAD_XSPI1_DATA1__SAI4_TX_SYNC
IOMUXC_PAD_XSPI1_DATA1__SAI4_TX_DATA_BIT1
IOMUXC_PAD_XSPI1_DATA1__XSPI_DATA_BIT1
IOMUXC_PAD_XSPI1_DATA1__GPIO5_IO_BIT1
IOMUXC_PAD_XSPI1_DATA2__FLEXSPI1_A_DATA_BIT2
IOMUXC_PAD_XSPI1_DATA2__SAI2_TX_DATA_BIT6
IOMUXC_PAD_XSPI1_DATA2__SAI4_TX_DATA_BIT0
IOMUXC_PAD_XSPI1_DATA2__XSPI_DATA_BIT2
IOMUXC_PAD_XSPI1_DATA2__GPIO5_IO_BIT2
IOMUXC_PAD_XSPI1_DATA3__FLEXSPI1_A_DATA_BIT3
IOMUXC_PAD_XSPI1_DATA3__SAI2_TX_DATA_BIT7
IOMUXC_PAD_XSPI1_DATA3__SAI4_RX_DATA_BIT0
IOMUXC_PAD_XSPI1_DATA3__XSPI_DATA_BIT3
IOMUXC_PAD_XSPI1_DATA3__GPIO5_IO_BIT3
IOMUXC_PAD_XSPI1_DATA4__FLEXSPI1_A_DATA_BIT4
IOMUXC_PAD_XSPI1_DATA4__SAI5_TX_DATA_BIT0
IOMUXC_PAD_XSPI1_DATA4__SAI5_RX_DATA_BIT1
IOMUXC_PAD_XSPI1_DATA4__XSPI_DATA_BIT4

IOMUXC_PAD_XSPI1_DATA4__GPIO5_IO_BIT4
IOMUXC_PAD_XSPI1_DATA5__FLEXSPI1_A_DATA_BIT5
IOMUXC_PAD_XSPI1_DATA5__SAI5_TX_SYNC
IOMUXC_PAD_XSPI1_DATA5__SAI5_RX_DATA_BIT2
IOMUXC_PAD_XSPI1_DATA5__SAI2_RX_DATA_BIT6
IOMUXC_PAD_XSPI1_DATA5__XSPI_DATA_BIT5
IOMUXC_PAD_XSPI1_DATA5__GPIO5_IO_BIT5
IOMUXC_PAD_XSPI1_DATA6__FLEXSPI1_A_DATA_BIT6
IOMUXC_PAD_XSPI1_DATA6__SAI5_TX_BCLK
IOMUXC_PAD_XSPI1_DATA6__SAI5_RX_DATA_BIT3
IOMUXC_PAD_XSPI1_DATA6__SAI2_RX_DATA_BIT7
IOMUXC_PAD_XSPI1_DATA6__XSPI_DATA_BIT6
IOMUXC_PAD_XSPI1_DATA6__GPIO5_IO_BIT6
IOMUXC_PAD_XSPI1_DATA7__FLEXSPI1_A_DATA_BIT7
IOMUXC_PAD_XSPI1_DATA7__SAI5_RX_DATA_BIT0
IOMUXC_PAD_XSPI1_DATA7__SAI5_TX_DATA_BIT1
IOMUXC_PAD_XSPI1_DATA7__XSPI_DATA_BIT7
IOMUXC_PAD_XSPI1_DATA7__GPIO5_IO_BIT7
IOMUXC_PAD_XSPI1_DQS__FLEXSPI1_A_DQS
IOMUXC_PAD_XSPI1_DQS__SAI5_RX_SYNC
IOMUXC_PAD_XSPI1_DQS__SAI5_TX_DATA_BIT2
IOMUXC_PAD_XSPI1_DQS__SAI2_RX_DATA_BIT6
IOMUXC_PAD_XSPI1_DQS__XSPI_DQS
IOMUXC_PAD_XSPI1_DQS__GPIO5_IO_BIT8
IOMUXC_PAD_XSPI1_SCLK__FLEXSPI1_A_SCLK
IOMUXC_PAD_XSPI1_SCLK__SAI2_RX_DATA_BIT4
IOMUXC_PAD_XSPI1_SCLK__SAI4_RX_SYNC
IOMUXC_PAD_XSPI1_SCLK__EARC_DC_HPD_IN
IOMUXC_PAD_XSPI1_SCLK__XSPI_CLK
IOMUXC_PAD_XSPI1_SCLK__GPIO5_IO_BIT9
IOMUXC_PAD_XSPI1_SS0_B__FLEXSPI1_A_SS0_B
IOMUXC_PAD_XSPI1_SS0_B__SAI2_RX_DATA_BIT5
IOMUXC_PAD_XSPI1_SS0_B__SAI4_RX_BCLK

IOMUXC_PAD_XSPI1_SS0_B__EARC_CEC_OUT
IOMUXC_PAD_XSPI1_SS0_B__XSPI_CS
IOMUXC_PAD_XSPI1_SS0_B__GPIO5_IO_BIT10
IOMUXC_PAD_XSPI1_SS1_B__FLEXSPI1_A_SS1_B
IOMUXC_PAD_XSPI1_SS1_B__SAI5_RX_BCLK
IOMUXC_PAD_XSPI1_SS1_B__SAI5_TX_DATA_BIT3
IOMUXC_PAD_XSPI1_SS1_B__SAI2_RX_DATA_BIT7
IOMUXC_PAD_XSPI1_SS1_B__GPIO5_IO_BIT11
IOMUXC_PAD_SD2_CD_B__USDHC2_CD_B
IOMUXC_PAD_SD2_CD_B__NETC_TMR_1588_TRIG1
IOMUXC_PAD_SD2_CD_B__I3C2_SCL
IOMUXC_PAD_SD2_CD_B__FLEXIO1_FLEXIO_BIT0
IOMUXC_PAD_SD2_CD_B__GPIO3_IO_BIT0
IOMUXC_PAD_SD2_CLK__USDHC2_CLK
IOMUXC_PAD_SD2_CLK__NETC_TMR_1588_PP1
IOMUXC_PAD_SD2_CLK__I3C2_SDA
IOMUXC_PAD_SD2_CLK__FLEXIO1_FLEXIO_BIT1
IOMUXC_PAD_SD2_CLK__GPIO3_IO_BIT1
IOMUXC_PAD_SD2_CLK__OBSERVE_0
IOMUXC_PAD_SD2_CMD__USDHC2_CMD
IOMUXC_PAD_SD2_CMD__NETC_TMR_1588_TRIG2
IOMUXC_PAD_SD2_CMD__I3C2_PUR
IOMUXC_PAD_SD2_CMD__I3C2_PUR_B
IOMUXC_PAD_SD2_CMD__FLEXIO1_FLEXIO_BIT2
IOMUXC_PAD_SD2_CMD__GPIO3_IO_BIT2
IOMUXC_PAD_SD2_CMD__OBSERVE_1
IOMUXC_PAD_SD2_DATA0__USDHC2_DATA0
IOMUXC_PAD_SD2_DATA0__NETC_TMR_1588_PP2
IOMUXC_PAD_SD2_DATA0__CAN2_TX
IOMUXC_PAD_SD2_DATA0__FLEXIO1_FLEXIO_BIT3
IOMUXC_PAD_SD2_DATA0__GPIO3_IO_BIT3
IOMUXC_PAD_SD2_DATA0__OBSERVE_2
IOMUXC_PAD_SD2_DATA1__USDHC2_DATA1

IOMUXC_PAD_SD2_DATA1__NETC_TMR_1588_CLK
IOMUXC_PAD_SD2_DATA1__CAN2_RX
IOMUXC_PAD_SD2_DATA1__FLEXIO1_FLEXIO_BIT4
IOMUXC_PAD_SD2_DATA1__GPIO3_IO_BIT4
IOMUXC_PAD_SD2_DATA2__USDHC2_DATA2
IOMUXC_PAD_SD2_DATA2__NETC_TMR_1588_PP3
IOMUXC_PAD_SD2_DATA2__MQS2_RIGHT
IOMUXC_PAD_SD2_DATA2__FLEXIO1_FLEXIO_BIT5
IOMUXC_PAD_SD2_DATA2__GPIO3_IO_BIT5
IOMUXC_PAD_SD2_DATA3__USDHC2_DATA3
IOMUXC_PAD_SD2_DATA3__LPTMR2_ALT1
IOMUXC_PAD_SD2_DATA3__MQS2_LEFT
IOMUXC_PAD_SD2_DATA3__NETC_TMR_1588_ALARM1
IOMUXC_PAD_SD2_DATA3__FLEXIO1_FLEXIO_BIT6
IOMUXC_PAD_SD2_DATA3__GPIO3_IO_BIT6
IOMUXC_PAD_SD2_RESET_B__USDHC2_RESET_B
IOMUXC_PAD_SD2_RESET_B__LPTMR2_ALT2
IOMUXC_PAD_SD2_RESET_B__NETC_TMR_1588_GCLK
IOMUXC_PAD_SD2_RESET_B__FLEXIO1_FLEXIO_BIT7
IOMUXC_PAD_SD2_RESET_B__GPIO3_IO_BIT7
IOMUXC_PAD_I2C1_SCL__LPI2C1_SCL
IOMUXC_PAD_I2C1_SCL__I3C1_SCL
IOMUXC_PAD_I2C1_SCL__LPUART1_DCD_B
IOMUXC_PAD_I2C1_SCL__TPM2_CH0
IOMUXC_PAD_I2C1_SCL__UART_RX
IOMUXC_PAD_I2C1_SCL__GPIO1_IO_BIT0
IOMUXC_PAD_I2C1_SDA__LPI2C1_SDA
IOMUXC_PAD_I2C1_SDA__I3C1_SDA
IOMUXC_PAD_I2C1_SDA__LPUART1_RIN_B
IOMUXC_PAD_I2C1_SDA__TPM2_CH1
IOMUXC_PAD_I2C1_SDA__UART_TX
IOMUXC_PAD_I2C1_SDA__GPIO1_IO_BIT1
IOMUXC_PAD_I2C2_SCL__LPI2C2_SCL

IOMUXC_PAD_I2C2_SCL___I3C1_PUR
IOMUXC_PAD_I2C2_SCL___LPUART2_DCD_B
IOMUXC_PAD_I2C2_SCL___TPM2_CH2
IOMUXC_PAD_I2C2_SCL___SAI1_RX_SYNC
IOMUXC_PAD_I2C2_SCL___GPIO1_IO_BIT2
IOMUXC_PAD_I2C2_SCL___I3C1_PUR_B
IOMUXC_PAD_I2C2_SDA___LPI2C2_SDA
IOMUXC_PAD_I2C2_SDA___LPUART2_RIN_B
IOMUXC_PAD_I2C2_SDA___TPM2_CH3
IOMUXC_PAD_I2C2_SDA___SAI1_RX_BCLK
IOMUXC_PAD_I2C2_SDA___GPIO1_IO_BIT3
IOMUXC_PAD_UART1_RXD___LPUART1_RX
IOMUXC_PAD_UART1_RXD___S400_UART_RX
IOMUXC_PAD_UART1_RXD___LPSPI2_SIN
IOMUXC_PAD_UART1_RXD___TPM1_CH0
IOMUXC_PAD_UART1_RXD___GPIO1_IO_BIT4
IOMUXC_PAD_UART1_TXD___LPUART1_TX
IOMUXC_PAD_UART1_TXD___S400_UART_TX
IOMUXC_PAD_UART1_TXD___LPSPI2_PCS0
IOMUXC_PAD_UART1_TXD___TPM1_CH1
IOMUXC_PAD_UART1_TXD___GPIO1_IO_BIT5
IOMUXC_PAD_UART2_RXD___LPUART2_RX
IOMUXC_PAD_UART2_RXD___LPUART1_CTS_B
IOMUXC_PAD_UART2_RXD___LPSPI2_SOUT
IOMUXC_PAD_UART2_RXD___TPM1_CH2
IOMUXC_PAD_UART2_RXD___SAI1_MCLK
IOMUXC_PAD_UART2_RXD___GPIO1_IO_BIT6
IOMUXC_PAD_UART2_TXD___LPUART2_TX
IOMUXC_PAD_UART2_TXD___LPUART1_RTS_B
IOMUXC_PAD_UART2_TXD___LPSPI2_SCK
IOMUXC_PAD_UART2_TXD___TPM1_CH3
IOMUXC_PAD_UART2_TXD___GPIO1_IO_BIT7
IOMUXC_PAD_PDM_CLK___PDM_CLK

IOMUXC_PAD_PDM_CLK__MQS1_LEFT
IOMUXC_PAD_PDM_CLK__LPTMR1_ALT1
IOMUXC_PAD_PDM_CLK__GPIO1_IO_BIT8
IOMUXC_PAD_PDM_CLK__CAN1_TX
IOMUXC_PAD_PDM_BIT_STREAM0__PDM_BIT_STREAM_BIT0
IOMUXC_PAD_PDM_BIT_STREAM0__MQS1_RIGHT
IOMUXC_PAD_PDM_BIT_STREAM0__LPSPI1_PCS1
IOMUXC_PAD_PDM_BIT_STREAM0__TPM1_EXTCLK
IOMUXC_PAD_PDM_BIT_STREAM0__LPTMR1_ALT2
IOMUXC_PAD_PDM_BIT_STREAM0__GPIO1_IO_BIT9
IOMUXC_PAD_PDM_BIT_STREAM0__CAN1_RX
IOMUXC_PAD_PDM_BIT_STREAM1__PDM_BIT_STREAM_BIT1
IOMUXC_PAD_PDM_BIT_STREAM1__NMI_GLUE_NMI
IOMUXC_PAD_PDM_BIT_STREAM1__LPSPI2_PCS1
IOMUXC_PAD_PDM_BIT_STREAM1__TPM2_EXTCLK
IOMUXC_PAD_PDM_BIT_STREAM1__LPTMR1_ALT3
IOMUXC_PAD_PDM_BIT_STREAM1__GPIO1_IO_BIT10
IOMUXC_PAD_PDM_BIT_STREAM1__EXT_CLK1
IOMUXC_PAD_SAI1_TXFS__SAI1_TX_SYNC
IOMUXC_PAD_SAI1_TXFS__SAI1_TX_DATA_BIT1
IOMUXC_PAD_SAI1_TXFS__LPSPI1_PCS0
IOMUXC_PAD_SAI1_TXFS__LPUART2_DTR_B
IOMUXC_PAD_SAI1_TXFS__MQS1_LEFT
IOMUXC_PAD_SAI1_TXFS__GPIO1_IO_BIT11
IOMUXC_PAD_SAI1_TXC__SAI1_TX_BCLK
IOMUXC_PAD_SAI1_TXC__LPUART2_CTS_B
IOMUXC_PAD_SAI1_TXC__LPSPI1_SIN
IOMUXC_PAD_SAI1_TXC__LPUART1_DSR_B
IOMUXC_PAD_SAI1_TXC__CAN1_RX
IOMUXC_PAD_SAI1_TXC__GPIO1_IO_BIT12
IOMUXC_PAD_SAI1_TXD0__SAI1_TX_DATA_BIT0
IOMUXC_PAD_SAI1_TXD0__LPUART2_RTS_B
IOMUXC_PAD_SAI1_TXD0__LPSPI1_SCK

IOMUXC_PAD_SAI1_TXD0__LPUART1_DTR_B
IOMUXC_PAD_SAI1_TXD0__CAN1_TX
IOMUXC_PAD_SAI1_TXD0__GPIO1_IO_BIT13
IOMUXC_PAD_SAI1_RXD0__SAI1_RX_DATA_BIT0
IOMUXC_PAD_SAI1_RXD0__SAI1_MCLK
IOMUXC_PAD_SAI1_RXD0__LPSPI1_SOUT
IOMUXC_PAD_SAI1_RXD0__LPUART2_DSR_B
IOMUXC_PAD_SAI1_RXD0__MQS1_RIGHT
IOMUXC_PAD_SAI1_RXD0__GPIO1_IO_BIT14
IOMUXC_PAD_WDOG_ANY__WDOG_ANY
IOMUXC_PAD_WDOG_ANY__FCCU_EOUT1
IOMUXC_PAD_WDOG_ANY__GPIO1_IO_BIT15
IOMUXC_PAD_MUX_MODE_MASK
IOMUXC_PAD_MUX_MODE_SHIFT
IOMUXC_PAD_MUX_MODE(x)
IOMUXC_PAD_SION_MASK
IOMUXC_PAD_SION_SHIFT
IOMUXC_PAD_SION(x)
IOMUXC_PAD_DSE_MASK
IOMUXC_PAD_DSE_SHIFT
IOMUXC_PAD_DSE(x)
IOMUXC_PAD_FSEL1_MASK
IOMUXC_PAD_FSEL1_SHIFT
IOMUXC_PAD_FSEL1(x)
IOMUXC_PAD_PU_MASK
IOMUXC_PAD_PU_SHIFT
IOMUXC_PAD_PU(x)
IOMUXC_PAD_PD_MASK
IOMUXC_PAD_PD_SHIFT
IOMUXC_PAD_PD(x)
IOMUXC_PAD_OD_MASK
IOMUXC_PAD_OD_SHIFT
IOMUXC_PAD_OD(x)

IOMUXC_PAD_HYS_MASK
IOMUXC_PAD_HYS_SHIFT
IOMUXC_PAD_HYS(x)
IOMUXC_PAD_APC_MASK
IOMUXC_PAD_APC_SHIFT
IOMUXC_PAD_APC(x)
FSL_COMPONENT_ID

2.34 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.

2.35 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->addr = 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.

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 aligned 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).

enum _isi_roi_index

ISI roi index number.

Values:

enumerator ISI_ROI_INDEX_0

ISI ROI index 0

enumerator ISI_ROI_INDEX_1

ISI ROI index 1

enumerator ISI_ROI_INDEX_2

ISI ROI index 2

enumerator ISI_ROI_INDEX_3

ISI ROI index 3

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

- $Y = (A1 \times R) + (A2 \times G) + (A3 \times B) + D1$
- $U = (B1 \times R) + (B2 \times G) + (B3 \times B) + D2$
- $V = (C1 \times R) + (C2 \times G) + (C3 \times B) + D3$

(b) YUV (or YCbCr) to RGB conversion

- $R = (A1 \times (Y + D1)) + (A2 \times (U + D2)) + (A3 \times (V + D3))$
- $G = (B1 \times (Y + D1)) + (B2 \times (U + D2)) + (B3 \times (V + D3))$

- $B = (C1 \times (Y + D1)) + (C2 \times (U + D2)) + (C3 \times (V + D3))$

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_regain_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.

```
typedef enum _isi_roi_index isi_roi_index_t
```

ISI roi index number.

```
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 cropping, 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 occurs.

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

- $Y = (A1 \times R) + (A2 \times G) + (A3 \times B) + D1$
- $U = (B1 \times R) + (B2 \times G) + (B3 \times B) + D2$
- $V = (C1 \times R) + (C2 \times G) + (C3 \times B) + D3$

(b) YUV (or YCbCr) to RGB conversion

- $R = (A1 \times (Y + D1)) + (A2 \times (U + D2)) + (A3 \times (V + D3))$
- $G = (B1 \times (Y + D1)) + (B2 \times (U + D2)) + (B3 \times (V + D3))$
- $B = (C1 \times (Y + D1)) + (C2 \times (U + D2)) + (C3 \times (V + D3))$

Overflow for the three channels are saturated at 0x255 and underflow is saturated at 0x00.

Public Members

isi_csc_mode_t mode

Conversion 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 pitch in bytes.

uint16_t framePitchBytes
Frame pitch in bytes.

isi_input_mem_format_t format
Image format of the input memory.

2.36 Common Driver

FSL_COMMON_DRIVER_VERSION
common driver version.

DEBUG_CONSOLE_DEVICE_TYPE_NONE
No debug console.

DEBUG_CONSOLE_DEVICE_TYPE_UART
Debug console based on UART.

DEBUG_CONSOLE_DEVICE_TYPE_LPUART
Debug console based on LPUART.

DEBUG_CONSOLE_DEVICE_TYPE_LPSCI
Debug console based on LPSCI.

DEBUG_CONSOLE_DEVICE_TYPE_USBCDC
Debug console based on USBCDC.

DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM
Debug console based on FLEXCOMM.

DEBUG_CONSOLE_DEVICE_TYPE_IUART
Debug console based on i.MX UART.

DEBUG_CONSOLE_DEVICE_TYPE_VUSART

Debug console based on LPC_VUSART.

DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART

Debug console based on LPC_USART.

DEBUG_CONSOLE_DEVICE_TYPE_SWO

Debug console based on SWO.

DEBUG_CONSOLE_DEVICE_TYPE_QSCI

Debug console based on QSCI.

MIN(*a*, *b*)

Computes the minimum of *a* and *b*.

MAX(*a*, *b*)

Computes the maximum of *a* and *b*.

UINT16_MAX

Max value of uint16_t type.

UINT32_MAX

Max value of uint32_t type.

SDK_ATOMIC_LOCAL_ADD(addr, val)

Add value *val* from the variable at address *address*.

SDK_ATOMIC_LOCAL_SUB(addr, val)

Subtract value *val* to the variable at address *address*.

SDK_ATOMIC_LOCAL_SET(addr, bits)

Set the bits specified by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR(addr, bits)

Clear the bits specified by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_TOGGLE(addr, bits)

Toggle the bits specified by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR_AND_SET(addr, clearBits, setBits)

For the variable at address *address*, clear the bits specified by *clearBits* and set the bits specified by *setBits*.

SDK_ATOMIC_LOCAL_COMPARE_AND_SET(addr, expected, newValue)

For the variable at address *address*, check whether the value equal to *expected*. If value same as *expected* then update *newValue* to address and return **true**, else return **false**.

SDK_ATOMIC_LOCAL_TEST_AND_SET(addr, newValue)

For the variable at address *address*, set as *newValue* value and return old value.

USEC_TO_COUNT(us, clockFreqInHz)

Macro to convert a microsecond period to raw count value

COUNT_TO_USEC(count, clockFreqInHz)

Macro to convert a raw count value to microsecond

MSEC_TO_COUNT(ms, clockFreqInHz)

Macro to convert a millisecond period to raw count value

COUNT_TO_MSEC(count, clockFreqInHz)

Macro to convert a raw count value to millisecond

SDK_ISR_EXIT_BARRIER

SDK_ALIGN(var, alignbytes)

Macro to define a variable with alignbytes alignment

SDK_L1DCACHE_ALIGN(var)

Macro to define a variable with L1 d-cache line size alignment

SDK_SIZEALIGN(var, alignbytes)

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*.

MCUX_CS

AT_CACHE_LINE_SECTION(var)

Define a variable *var*, which is cache line size aligned and be placed in CacheLineData section.

AT_CACHE_LINE_SECTION_INIT(var)

Define a variable *var* with initial value, which is cache line size aligned and be placed in CacheLineData.init section.

CACHE_LINE_DATA

AT_QUICKACCESS_SECTION_CODE(func)

Place function in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_DATA(var)

Place data in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_DATA_ALIGN(var, alignbytes)

Place data in a section which can be accessed quickly by core, and the variable address is set to align with *alignbytes*.

RAMFUNCTION_SECTION_CODE(func)

Place function in ram.

enum _status_groups

Status group numbers.

Values:

enumerator kStatusGroup_Generic

Group number for generic status codes.

enumerator kStatusGroup_FLASH

Group number for FLASH status codes.

enumerator kStatusGroup_LPSPI

Group number for LPSPI status codes.

enumerator kStatusGroup_FLEXIO_SPI
Group number for FLEXIO SPI status codes.

enumerator kStatusGroup_DSPI
Group number for DSPI status codes.

enumerator kStatusGroup_FLEXIO_UART
Group number for FLEXIO UART status codes.

enumerator kStatusGroup_FLEXIO_I2C
Group number for FLEXIO I2C status codes.

enumerator kStatusGroup_LPI2C
Group number for LPI2C status codes.

enumerator kStatusGroup_UART
Group number for UART status codes.

enumerator kStatusGroup_I2C
Group number for UART status codes.

enumerator kStatusGroup_LPSCI
Group number for LPSCI status codes.

enumerator kStatusGroup_LPUART
Group number for LPUART status codes.

enumerator kStatusGroup_SPI
Group number for SPI status code.

enumerator kStatusGroup_XRDC
Group number for XRDC status code.

enumerator kStatusGroup_SEMA42
Group number for SEMA42 status code.

enumerator kStatusGroup_SDHC
Group number for SDHC status code

enumerator kStatusGroup_SDMMC
Group number for SDMMC status code

enumerator kStatusGroup_SAI
Group number for SAI status code

enumerator kStatusGroup_MCG
Group number for MCG status codes.

enumerator kStatusGroup_SCG
Group number for SCG status codes.

enumerator kStatusGroup_SDSPI
Group number for SDSPI status codes.

enumerator kStatusGroup_FLEXIO_I2S
Group number for FLEXIO I2S status codes

enumerator kStatusGroup_FLEXIO_MCULCD
Group number for FLEXIO LCD status codes

enumerator kStatusGroup_FLASHIAP
Group number for FLASHIAP status codes

enumerator kStatusGroup_FLEXCOMM_I2C
Group number for FLEXCOMM I2C status codes

enumerator kStatusGroup_I2S
Group number for I2S status codes

enumerator kStatusGroup_IUART
Group number for IUART status codes

enumerator kStatusGroup_CSI
Group number for CSI status codes

enumerator kStatusGroup_MIPIDSI
Group number for MIPI DSI status codes

enumerator kStatusGroup_SDRAMC
Group number for SDRAMC status codes.

enumerator kStatusGroup_POWER
Group number for POWER status codes.

enumerator kStatusGroup_ENET
Group number for ENET status codes.

enumerator kStatusGroup_PHY
Group number for PHY status codes.

enumerator kStatusGroup_TRGMUX
Group number for TRGMUX status codes.

enumerator kStatusGroup_SMARTCARD
Group number for SMARTCARD status codes.

enumerator kStatusGroup_LMEM
Group number for LMEM status codes.

enumerator kStatusGroup_QSPI
Group number for QSPI status codes.

enumerator kStatusGroup_DMA
Group number for DMA status codes.

enumerator kStatusGroup_EDMA
Group number for EDMA status codes.

enumerator kStatusGroup_DMAMGR
Group number for DMAMGR status codes.

enumerator kStatusGroup_FLEXCAN
Group number for FlexCAN status codes.

enumerator kStatusGroup_LTC
Group number for LTC status codes.

enumerator kStatusGroup_FLEXIO_CAMERA
Group number for FLEXIO CAMERA status codes.

enumerator kStatusGroup_LPC_SPI
Group number for LPC_SPI status codes.

enumerator kStatusGroup_LPC_USART
Group number for LPC_USART status codes.

enumerator kStatusGroup_DMIC
Group number for DMIC status codes.

enumerator kStatusGroup_SDIF
Group number for SDIF status codes.

enumerator kStatusGroup_SPIFI
Group number for SPIFI status codes.

enumerator kStatusGroup_OTP
Group number for OTP status codes.

enumerator kStatusGroup_MCAN
Group number for MCAN status codes.

enumerator kStatusGroup_CAAM
Group number for CAAM status codes.

enumerator kStatusGroup_ECSPi
Group number for ECSPi status codes.

enumerator kStatusGroup_USDHC
Group number for USDHC status codes.

enumerator kStatusGroup_LPC_I2C
Group number for LPC_I2C status codes.

enumerator kStatusGroup_DCP
Group number for DCP status codes.

enumerator kStatusGroup_MSCAN
Group number for MSCAN status codes.

enumerator kStatusGroup_ESAI
Group number for ESAI status codes.

enumerator kStatusGroup_FLEXSPI
Group number for FLEXSPI status codes.

enumerator kStatusGroup_MMDC
Group number for MMDC status codes.

enumerator kStatusGroup_PDM
Group number for MIC status codes.

enumerator kStatusGroup_SDMA
Group number for SDMA status codes.

enumerator kStatusGroup_ICS
Group number for ICS status codes.

enumerator kStatusGroup_SPDIF
Group number for SPDIF status codes.

enumerator kStatusGroup_LPC_MINISPI
Group number for LPC_MINISPI status codes.

enumerator kStatusGroup_HASHCRYPT
Group number for Hashcrypt status codes

enumerator kStatusGroup_LPC_SPI_SSP
Group number for LPC_SPI_SSP status codes.

enumerator kStatusGroup_I3C
Group number for I3C status codes

enumerator kStatusGroup_LPC_I2C_1
Group number for LPC_I2C_1 status codes.

enumerator kStatusGroup_NOTIFIER
Group number for NOTIFIER status codes.

enumerator kStatusGroup_DebugConsole
Group number for debug console status codes.

enumerator kStatusGroup_SEMC
Group number for SEMC status codes.

enumerator kStatusGroup_ApplicationRangeStart
Starting number for application groups.

enumerator kStatusGroup_IAP
Group number for IAP status codes

enumerator kStatusGroup_SFA
Group number for SFA status codes

enumerator kStatusGroup_SPC
Group number for SPC status codes.

enumerator kStatusGroup_PUF
Group number for PUF status codes.

enumerator kStatusGroup_TOUCH_PANEL
Group number for touch panel status codes

enumerator kStatusGroup_VBAT
Group number for VBAT status codes

enumerator kStatusGroup_XSPI
Group number for XSPI status codes

enumerator kStatusGroup_PNGDEC
Group number for PNGDEC status codes

enumerator kStatusGroup_JPEGDEC
Group number for JPEGDEC status codes

enumerator kStatusGroup_AUDMIX
Group number for AUDMIX status codes

enumerator kStatusGroup_HAL_GPIO
Group number for HAL GPIO status codes.

enumerator kStatusGroup_HAL_UART
Group number for HAL UART status codes.

enumerator kStatusGroup_HAL_TIMER
Group number for HAL TIMER status codes.

enumerator kStatusGroup_HAL_SPI
Group number for HAL SPI status codes.

enumerator kStatusGroup_HAL_I2C
Group number for HAL I2C status codes.

enumerator kStatusGroup_HAL_FLASH
Group number for HAL FLASH status codes.

enumerator kStatusGroup_HAL_PWM
Group number for HAL PWM status codes.

enumerator kStatusGroup_HAL_RNG
Group number for HAL RNG status codes.

enumerator kStatusGroup_HAL_I2S
Group number for HAL I2S status codes.

enumerator kStatusGroup_HAL_ADC_SENSOR
Group number for HAL ADC SENSOR status codes.

enumerator kStatusGroup_TIMERMANAGER
Group number for TiMER MANAGER status codes.

enumerator kStatusGroup_SERIALMANAGER
Group number for SERIAL MANAGER status codes.

enumerator kStatusGroup_LED
Group number for LED status codes.

enumerator kStatusGroup_BUTTON
Group number for BUTTON status codes.

enumerator kStatusGroup_EXTERN_EEPROM
Group number for EXTERN EEPROM status codes.

enumerator kStatusGroup_SHELL
Group number for SHELL status codes.

enumerator kStatusGroup_MEM_MANAGER
Group number for MEM MANAGER status codes.

enumerator kStatusGroup_LIST
Group number for List status codes.

enumerator kStatusGroup_OSA
Group number for OSA status codes.

enumerator kStatusGroup_COMMON_TASK
Group number for Common task status codes.

enumerator kStatusGroup_MSG
Group number for messaging status codes.

enumerator kStatusGroup_SDK_OCOTP
Group number for OCOTP status codes.

enumerator kStatusGroup_SDK_FLEXSPINOR
Group number for FLEXSPINOR status codes.

enumerator kStatusGroup_CODEC
Group number for codec status codes.

enumerator kStatusGroup_ASRC
Group number for codec status ASRC.

enumerator kStatusGroup_OTFAD
Group number for codec status codes.

enumerator `kStatusGroup_SDIO_SLV`
Group number for SDIO_SLV status codes.

enumerator `kStatusGroup_MECC`
Group number for MECC status codes.

enumerator `kStatusGroup_ENET_QOS`
Group number for ENET_QOS status codes.

enumerator `kStatusGroup_LOG`
Group number for LOG status codes.

enumerator `kStatusGroup_I3CBUS`
Group number for I3CBUS status codes.

enumerator `kStatusGroup_QSCI`
Group number for QSCI status codes.

enumerator `kStatusGroup_ELEMU`
Group number for ELEMU status codes.

enumerator `kStatusGroup_QUEUEDSPI`
Group number for QSPI status codes.

enumerator `kStatusGroup_POWER_MANAGER`
Group number for POWER_MANAGER status codes.

enumerator `kStatusGroup_IPED`
Group number for IPED status codes.

enumerator `kStatusGroup_ELS_PKC`
Group number for ELS PKC status codes.

enumerator `kStatusGroup_CSS_PKC`
Group number for CSS PKC status codes.

enumerator `kStatusGroup_HOSTIF`
Group number for HOSTIF status codes.

enumerator `kStatusGroup_CLIF`
Group number for CLIF status codes.

enumerator `kStatusGroup_BMA`
Group number for BMA status codes.

enumerator `kStatusGroup_NETC`
Group number for NETC status codes.

enumerator `kStatusGroup_ELE`
Group number for ELE status codes.

enumerator `kStatusGroup_GLIKEY`
Group number for GLIKEY status codes.

enumerator `kStatusGroup_AON_POWER`
Group number for AON_POWER status codes.

enumerator `kStatusGroup_AON_COMMON`
Group number for AON_COMMON status codes.

enumerator `kStatusGroup_ENDAT3`
Group number for ENDAT3 status codes.

enumerator kStatusGroup_HIPERFACE

Group number for HIPERFACE status codes.

enumerator kStatusGroup_NPX

Group number for NPX status codes.

enumerator kStatusGroup_ELA_CSEC

Group number for ELA_CSEC status codes.

enumerator kStatusGroup_FLEXIO_T_FORMAT

Group number for T-format status codes.

enumerator kStatusGroup_FLEXIO_A_FORMAT

Group number for A-format status codes.

Generic status return codes.

Values:

enumerator kStatus_Success

Generic status for Success.

enumerator kStatus_Fail

Generic status for Fail.

enumerator kStatus_ReadOnly

Generic status for read only failure.

enumerator kStatus_OutOfRange

Generic status for out of range access.

enumerator kStatus_InvalidArgument

Generic status for invalid argument check.

enumerator kStatus_Timeout

Generic status for timeout.

enumerator kStatus_NoTransferInProgress

Generic status for no transfer in progress.

enumerator kStatus_Busy

Generic status for module is busy.

enumerator kStatus_NoData

Generic status for no data is found for the operation.

typedef int32_t status_t

Type used for all status and error return values.

void *SDK_Malloc(size_t size, size_t alignbytes)

Allocate memory with given alignment and aligned size.

This is provided to support the dynamically allocated memory used in cache-able region.

Parameters

- size – The length required to malloc.
- alignbytes – The alignment size.

Return values

The – allocated memory.

`void SDK_Free(void *ptr)`

Free memory.

Parameters

- `ptr` – The memory to be release.

`void SDK_DelayAtLeastUs(uint32_t delayTime_us, uint32_t coreClock_Hz)`

Delay at least for some time. Please note that, this API uses while loop for delay, different run-time environments make the time not precise, if precise delay count was needed, please implement a new delay function with hardware timer.

Parameters

- `delayTime_us` – Delay time in unit of microsecond.
- `coreClock_Hz` – Core clock frequency with Hz.

`static inline status_t EnableIRQ(IRQn_Type interrupt)`

Enable specific interrupt.

Enable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only enables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro `FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS`.

Parameters

- `interrupt` – The IRQ number.

Return values

- `kStatus_Success` – Interrupt enabled successfully
- `kStatus_Fail` – Failed to enable the interrupt

`static inline status_t DisableIRQ(IRQn_Type interrupt)`

Disable specific interrupt.

Disable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only disables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro `FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS`.

Parameters

- `interrupt` – The IRQ number.

Return values

- `kStatus_Success` – Interrupt disabled successfully
- `kStatus_Fail` – Failed to disable the interrupt

`static inline status_t EnableIRQWithPriority(IRQn_Type interrupt, uint8_t priNum)`

Enable the IRQ, and also set the interrupt priority.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro `FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS`.

Parameters

- interrupt – The IRQ to Enable.
- priNum – Priority number set to interrupt controller register.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline *status_t* IRQ_SetPriority(IRQn_Type interrupt, uint8_t priNum)

Set the IRQ priority.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ to set.
- priNum – Priority number set to interrupt controller register.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline *status_t* IRQ_ClearPendingIRQ(IRQn_Type interrupt)

Clear the pending IRQ flag.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The flag which IRQ to clear.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline uint32_t DisableGlobalIRQ(void)

Disable the global IRQ.

Disable the global interrupt and return the current primask register. User is required to provided the primask register for the EnableGlobalIRQ().

Returns

Current primask value.

static inline void EnableGlobalIRQ(uint32_t primask)

Enable the global IRQ.

Set the primask register with the provided primask value but not just enable the primask. The idea is for the convenience of integration of RTOS. some RTOS get its own management

mechanism of primask. User is required to use the EnableGlobalIRQ() and DisableGlobalIRQ() in pair.

Parameters

- primask – value of primask register to be restored. The primask value is supposed to be provided by the DisableGlobalIRQ().

```
static inline bool __SDK_AtomicLocalCompareAndSet(uint32_t *addr, uint32_t expected, uint32_t  
newValue)
```

```
static inline uint32_t __SDK_AtomicTestAndSet(uint32_t *addr, uint32_t newValue)
```

FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ

Macro to use the default weak IRQ handler in drivers.

MAKE_STATUS(group, code)

Construct a status code value from a group and code number.

MAKE_VERSION(major, minor, bugfix)

Construct the version number for drivers.

The driver version is a 32-bit number, for both 32-bit platforms(such as Cortex M) and 16-bit platforms(such as DSC).

Unused	Major Version		Minor Version		Bug Fix		
31	25	24	17	16	9	8	0

ARRAY_SIZE(x)

Computes the number of elements in an array.

UINT64_H(X)

Macro to get upper 32 bits of a 64-bit value

UINT64_L(X)

Macro to get lower 32 bits of a 64-bit value

SUPPRESS_FALL_THROUGH_WARNING()

For switch case code block, if case section ends without “break;” statement, there will be fallthrough warning with compiler flag -Wextra or -Wimplicit-fallthrough=n when using armgcc. To suppress this warning, “SUPPRESS_FALL_THROUGH_WARNING();” need to be added at the end of each case section which misses “break;”statement.

MSDK_REG_SECURE_ADDR(x)

Convert the register address to the one used in secure mode.

MSDK_REG_NONSECURE_ADDR(x)

Convert the register address to the one used in non-secure mode.

MSDK_HAS_DWT_CYCCNT

The chip supports DWT CYCCNT or not.

MSDK_INVALID_IRQ_HANDLER

Invalid IRQ handler address.

2.37 LDB: LVDS Display Bridge

void LDB_Init(LDB_Type *base, uint8_t diIndex, uint8_t dualpanelIndex, uint8_t datamap)

brief Initializes the LDB module for LVDS port panel.

param base LDB peripheral base address. param diIndex Display Input index. param dualpanelIndex dual panel index. param datamap 0 for SPWG 1 for JEIDA.

FSL_LDB_DRIVER_VERSION

LDB driver version.

FSL_LDB_DRIVER_VERSION

LDB driver version.

LVDS_DI_COUNT

LVDS_SPWG

LVDS_JEIDA

enum _ldb_output_bus

LDB output bus format.

Values:

enumerator kLDB_OutputRGB666_7Bit_SPWG

enumerator kLDB_OutputRGB888_7Bit_SPWG

enumerator kLDB_OutputRGB888_7Bit_JEIDA

enumerator kLDB_OutputRGB101010_10Bit_SPWG

enumerator kLDB_OutputRGB101010_10Bit_JEIDA

enum _ldb_input_flag

LDB input signal priority.

Values:

enumerator kLDB_InputVsyncActiveLow

VSYN active low.

enumerator kLDB_InputVsyncActiveHigh

VSYN active high.

enumerator kLDB_InputHsyncActiveLow

HSYN active low.

enumerator kLDB_InputHsyncActiveHigh

HSYN active high.

enumerator kLDB_InputDataLatchOnFallingEdge

Latch data on falling clock edge.

enumerator kLDB_InputDataLatchOnRisingEdge

Latch data on rising clock edge.

typedef enum _ldb_output_bus ldb_output_bus_t

LDB output bus format.

typedef struct _ldb_channel_config ldb_channel_config_t

LDB channel configuration.

`void LDB_Init(LDB_Type *base)`

Initializes the LDB module.

Parameters

- `base` – LDB peripheral base address.

`void LDB_Deinit(LDB_Type *base)`

De-initializes the LDB module.

Parameters

- `base` – LDB peripheral base address.

`status_t LDB_InitChannel(LDB_Type *base, uint8_t channel, const ldb_channel_config_t *config)`

Initializes the LDB channel.

Parameters

- `base` – LDB peripheral base address.
- `channel` – Channel index.
- `config` – Pointer to the configuration.

Returns

Return `kStatus_Success` if success.

`void LDB_DeinitChannel(LDB_Type *base, uint8_t channel)`

De-initializes the LDB channel.

Parameters

- `base` – LDB peripheral base address.
- `channel` – Channel index.

`struct _ldb_channel_config`

#include <fsl_ldb.h> LDB channel configuration.

Public Members

ldb_output_bus_t outputBus

Output bus format.

`uint32_t` inputFlag

Input flag, OR'ed value of `_ldb_input_flag`.

`uint32_t` pixelClock_Hz

Pixel clock in HZ.

2.38 Lin_lpuart_driver

`FSL_LIN_LPUART_DRIVER_VERSION`

LIN LPUART driver version.

`enum _lin_lpuart_stop_bit_count`

Values:

enumerator `kLPUART_OneStopBit`

One stop bit

enumerator kLPUART_TwoStopBit

Two stop bits

enum _lin_lpuart_flags

Values:

enumerator kLPUART_TxDataRegEmptyFlag

Transmit data register empty flag, sets when transmit buffer is empty

enumerator kLPUART_TransmissionCompleteFlag

Transmission complete flag, sets when transmission activity complete

enumerator kLPUART_RxDataRegFullFlag

Receive data register full flag, sets when the receive data buffer is full

enumerator kLPUART_IdleLineFlag

Idle line detect flag, sets when idle line detected

enumerator kLPUART_RxOverrunFlag

Receive Overrun, sets when new data is received before data is read from receive register

enumerator kLPUART_NoiseErrorFlag

Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets

enumerator kLPUART_FramingErrorFlag

Frame error flag, sets if logic 0 was detected where stop bit expected

enumerator kLPUART_ParityErrorFlag

If parity enabled, sets upon parity error detection

enumerator kLPUART_LinBreakFlag

LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled

enumerator kLPUART_RxActiveEdgeFlag

Receive pin active edge interrupt flag, sets when active edge detected

enumerator kLPUART_RxActiveFlag

Receiver Active Flag (RAF), sets at beginning of valid start bit

enumerator kLPUART_DataMatch1Flag

The next character to be read from LPUART_DATA matches MA1

enumerator kLPUART_DataMatch2Flag

The next character to be read from LPUART_DATA matches MA2

enumerator kLPUART_NoiseErrorInRxDataRegFlag

NOISY bit, sets if noise detected in current data word

enumerator kLPUART_ParityErrorInRxDataRegFlag

PARITY bit, sets if noise detected in current data word

enumerator kLPUART_TxFifoEmptyFlag

TXEMPT bit, sets if transmit buffer is empty

enumerator kLPUART_RxFifoEmptyFlag

RXEMPT bit, sets if receive buffer is empty

enumerator kLPUART_TxFifoOverflowFlag

TXOF bit, sets if transmit buffer overflow occurred

enumerator kLPUART_RxFifoUnderflowFlag
RXUF bit, sets if receive buffer underflow occurred

enum _lin_lpuart_interrupt_enable

Values:

enumerator kLPUART_LinBreakInterruptEnable
LIN break detect.

enumerator kLPUART_RxActiveEdgeInterruptEnable
Receive Active Edge.

enumerator kLPUART_TxDataRegEmptyInterruptEnable
Transmit data register empty.

enumerator kLPUART_TransmissionCompleteInterruptEnable
Transmission complete.

enumerator kLPUART_RxDataRegFullInterruptEnable
Receiver data register full.

enumerator kLPUART_IdleLineInterruptEnable
Idle line.

enumerator kLPUART_RxOverrunInterruptEnable
Receiver Overrun.

enumerator kLPUART_NoiseErrorInterruptEnable
Noise error flag.

enumerator kLPUART_FramingErrorInterruptEnable
Framing error flag.

enumerator kLPUART_ParityErrorInterruptEnable
Parity error flag.

enumerator kLPUART_TxFifoOverflowInterruptEnable
Transmit FIFO Overflow.

enumerator kLPUART_RxFifoUnderflowInterruptEnable
Receive FIFO Underflow.

enum _lin_lpuart_status

Values:

enumerator kStatus_LPUART_TxBusy
TX busy

enumerator kStatus_LPUART_RxBusy
RX busy

enumerator kStatus_LPUART_TxIdle
LPUART transmitter is idle.

enumerator kStatus_LPUART_RxIdle
LPUART receiver is idle.

enumerator kStatus_LPUART_TxWatermarkTooLarge
TX FIFO watermark too large

enumerator kStatus_LPUART_RxWatermarkTooLarge
RX FIFO watermark too large


```

enumerator kStatus_LPUART_FlagCannotClearManually
    Some flag can't manually clear
enumerator kStatus_LPUART_Error
    Error happens on LPUART.
enumerator kStatus_LPUART_RxRingBufferOverrun
    LPUART RX software ring buffer overrun.
enumerator kStatus_LPUART_RxHardwareOverrun
    LPUART RX receiver overrun.
enumerator kStatus_LPUART_NoiseError
    LPUART noise error.
enumerator kStatus_LPUART_FramingError
    LPUART framing error.
enumerator kStatus_LPUART_ParityError
    LPUART parity error.
enum lin_lpuart_bit_count_per_char_t
    Values:
    enumerator LPUART_8_BITS_PER_CHAR
        8-bit data characters
    enumerator LPUART_9_BITS_PER_CHAR
        9-bit data characters
    enumerator LPUART_10_BITS_PER_CHAR
        10-bit data characters
typedef enum _lin_lpuart_stop_bit_count lin_lpuart_stop_bit_count_t
static inline bool LIN_LPUART_GetRxDataPolarity(const LPUART_Type *base)
static inline void LIN_LPUART_SetRxDataPolarity(LPUART_Type *base, bool polarity)
static inline void LIN_LPUART_WriteByte(LPUART_Type *base, uint8_t data)
static inline void LIN_LPUART_ReadByte(const LPUART_Type *base, uint8_t *readData)
status_t LIN_LPUART_CalculateBaudRate(LPUART_Type *base, uint32_t baudRate_Bps,
                                     uint32_t srcClock_Hz, uint32_t *osr, uint16_t *sbr)
    Calculates the best osr and sbr value for configured baudrate.

Parameters
    • base – LPUART peripheral base address
    • baudRate_Bps – user configuration structure of type #lin_user_config_t
    • srcClock_Hz – pointer to the LIN_LPUART driver state structure
    • osr – pointer to osr value
    • sbr – pointer to sbr value

Returns
    An error code or lin_status_t

```

```
void LIN_LPUART_SetBaudRate(LPUART_Type *base, uint32_t *osr, uint16_t *sbr)
```

Configure baudrate according to osr and sbr value.

Parameters

- base – LPUART peripheral base address
- osr – pointer to osr value
- sbr – pointer to sbr value

```
lin_status_t LIN_LPUART_Init(LPUART_Type *base, lin_user_config_t *linUserConfig,  
                             lin_state_t *linCurrentState, uint32_t linSourceClockFreq)
```

Initializes an LIN_LPUART instance for LIN Network.

The caller provides memory for the driver state structures during initialization. The user must select the LIN_LPUART clock source in the application to initialize the LIN_LPUART. This function initializes a LPUART instance for operation. This function will initialize the run-time state structure to keep track of the on-going transfers, initialize the module to user defined settings and default settings, set break field length to be 13 bit times minimum, enable the break detect interrupt, Rx complete interrupt, frame error detect interrupt, and enable the LPUART module transmitter and receiver

Parameters

- base – LPUART peripheral base address
- linUserConfig – user configuration structure of type #lin_user_config_t
- linCurrentState – pointer to the LIN_LPUART driver state structure

Returns

An error code or lin_status_t

```
lin_status_t LIN_LPUART_Deinit(LPUART_Type *base)
```

Shuts down the LIN_LPUART by disabling interrupts and transmitter/receiver.

Parameters

- base – LPUART peripheral base address

Returns

An error code or lin_status_t

```
lin_status_t LIN_LPUART_SendFrameDataBlocking(LPUART_Type *base, const uint8_t *txBuff,  
                                               uint8_t txSize, uint32_t timeoutMSec)
```

Sends Frame data out through the LIN_LPUART module using blocking method. This function will calculate the checksum byte and send it with the frame data. Blocking means that the function does not return until the transmission is complete.

Parameters

- base – LPUART peripheral base address
- txBuff – source buffer containing 8-bit data chars to send
- txSize – the number of bytes to send
- timeoutMSec – timeout value in milli seconds

Returns

An error code or lin_status_t

```
lin_status_t LIN_LPUART_SendFrameData(LPUART_Type *base, const uint8_t *txBuff, uint8_t  
                                       txSize)
```

Sends frame data out through the LIN_LPUART module using non-blocking method. This

enables an a-sync method for transmitting data. Non-blocking means that the function returns immediately. The application has to get the transmit status to know when the transmit is complete. This function will calculate the checksum byte and send it with the frame data.

Parameters

- base – LPUART peripheral base address
- txBuff – source buffer containing 8-bit data chars to send
- txSize – the number of bytes to send

Returns

An error code or `lin_status_t`

`lin_status_t LIN_LPUART_GetTransmitStatus(LPUART_Type *base, uint8_t *bytesRemaining)`

Get status of an on-going non-blocking transmission. While sending frame data using non-blocking method, users can use this function to get status of that transmission. This function return `LIN_TX_BUSY` while sending, or `LIN_TIMEOUT` if timeout has occurred, or return `LIN_SUCCESS` when the transmission is complete. The `bytesRemaining` shows number of bytes that still needed to transmit.

Parameters

- base – LPUART peripheral base address
- bytesRemaining – Number of bytes still needed to transmit

Returns

`lin_status_t LIN_TX_BUSY`, `LIN_SUCCESS` or `LIN_TIMEOUT`

`lin_status_t LIN_LPUART_RecvFrmDataBlocking(LPUART_Type *base, uint8_t *rxBuff, uint8_t rxSize, uint32_t timeoutMSec)`

Receives frame data through the `LIN_LPUART` module using blocking method. This function will check the checksum byte. If the checksum is correct, it will receive the frame data. Blocking means that the function does not return until the reception is complete.

Parameters

- base – LPUART peripheral base address
- rxBuff – buffer containing 8-bit received data
- rxSize – the number of bytes to receive
- timeoutMSec – timeout value in milli seconds

Returns

An error code or `lin_status_t`

`lin_status_t LIN_LPUART_RecvFrmData(LPUART_Type *base, uint8_t *rxBuff, uint8_t rxSize)`

Receives frame data through the `LIN_LPUART` module using non-blocking method. This function will check the checksum byte. If the checksum is correct, it will receive it with the frame data. Non-blocking means that the function returns immediately. The application has to get the receive status to know when the reception is complete.

Parameters

- base – LPUART peripheral base address
- rxBuff – buffer containing 8-bit received data
- rxSize – the number of bytes to receive

Returns

An error code or `lin_status_t`

`lin_status_t LIN_LPUART_AbortTransferData(LPUART_Type *base)`

Aborts an on-going non-blocking transmission/reception. While performing a non-blocking transferring data, users can call this function to terminate immediately the transferring.

Parameters

- `base` – LPUART peripheral base address

Returns

An error code or `lin_status_t`

`lin_status_t LIN_LPUART_GetReceiveStatus(LPUART_Type *base, uint8_t *bytesRemaining)`

Get status of an on-going non-blocking reception. While receiving frame data using non-blocking method, users can use this function to get status of that receiving. This function return the current event ID, `LIN_RX_BUSY` while receiving and return `LIN_SUCCESS`, or timeout (`LIN_TIMEOUT`) when the reception is complete. The `bytesRemaining` shows number of bytes that still needed to receive.

Parameters

- `base` – LPUART peripheral base address
- `bytesRemaining` – Number of bytes still needed to receive

Returns

`lin_status_t` `LIN_RX_BUSY`, `LIN_TIMEOUT` or `LIN_SUCCESS`

`lin_status_t LIN_LPUART_GoToSleepMode(LPUART_Type *base)`

This function puts current node to sleep mode. This function changes current node state to `LIN_NODE_STATE_SLEEP_MODE`.

Parameters

- `base` – LPUART peripheral base address

Returns

An error code or `lin_status_t`

`lin_status_t LIN_LPUART_GotoIdleState(LPUART_Type *base)`

Puts current LIN node to Idle state. This function changes current node state to `LIN_NODE_STATE_IDLE`.

Parameters

- `base` – LPUART peripheral base address

Returns

An error code or `lin_status_t`

`lin_status_t LIN_LPUART_SendWakeupSignal(LPUART_Type *base)`

Sends a wakeup signal through the LIN_LPUART interface.

Parameters

- `base` – LPUART peripheral base address

Returns

An error code or `lin_status_t`

`lin_status_t LIN_LPUART_MasterSendHeader(LPUART_Type *base, uint8_t id)`

Sends frame header out through the LIN_LPUART module using a non-blocking method. This function sends LIN Break field, sync field then the ID with correct parity.

Parameters

- `base` – LPUART peripheral base address
- `id` – Frame Identifier

Returns

An error code or `lin_status_t`

`lin_status_t` LIN_LPUART_EnableIRQ(LPUART_Type *base)

Enables LIN_LPUART hardware interrupts.

Parameters

- base – LPUART peripheral base address

Returns

An error code or `lin_status_t`

`lin_status_t` LIN_LPUART_DisableIRQ(LPUART_Type *base)

Disables LIN_LPUART hardware interrupts.

Parameters

- base – LPUART peripheral base address

Returns

An error code or `lin_status_t`

`lin_status_t` LIN_LPUART_AutoBaudCapture(uint32_t instance)

This function capture bits time to detect break char, calculate baudrate from sync bits and enable transceiver if autobaud successful. This function should only be used in Slave. The timer should be in mode input capture of both rising and falling edges. The timer input capture pin should be externally connected to RXD pin.

Parameters

- instance – LPUART instance

Returns

`lin_status_t`

`void` LIN_LPUART_IRQHandler(LPUART_Type *base)

LIN_LPUART RX TX interrupt handler.

Parameters

- base – LPUART peripheral base address

Returns

`void`

LIN_LPUART_TRANSMISSION_COMPLETE_TIMEOUT

Max loops to wait for LPUART transmission complete.

When de-initializing the LIN LPUART module, the program shall wait for the previous transmission to complete. This parameter defines how many loops to check completion before return error. If defined as 0, driver will wait forever until completion.

AUTOBAUD_BAUDRATE_TOLERANCE

BIT_RATE_TOLERANCE_UNSYNC

BIT_DURATION_MAX_19200

BIT_DURATION_MIN_19200

BIT_DURATION_MAX_14400

BIT_DURATION_MIN_14400

BIT_DURATION_MAX_9600

BIT_DURATION_MIN_9600
BIT_DURATION_MAX_4800
BIT_DURATION_MIN_4800
BIT_DURATION_MAX_2400
BIT_DURATION_MIN_2400
TWO_BIT_DURATION_MAX_19200
TWO_BIT_DURATION_MIN_19200
TWO_BIT_DURATION_MAX_14400
TWO_BIT_DURATION_MIN_14400
TWO_BIT_DURATION_MAX_9600
TWO_BIT_DURATION_MIN_9600
TWO_BIT_DURATION_MAX_4800
TWO_BIT_DURATION_MIN_4800
TWO_BIT_DURATION_MAX_2400
TWO_BIT_DURATION_MIN_2400
AUTOBAUD_BREAK_TIME_MIN

2.39 LPI2C: Low Power Inter-Integrated Circuit Driver

`void LPI2C_DriverIRQHandler(uint32_t instance)`

LPI2C driver IRQ handler common entry.

This function provides the common IRQ request entry for LPI2C.

Parameters

- `instance` – LPI2C instance.

`FSL_LPI2C_DRIVER_VERSION`

LPI2C driver version.

LPI2C status return codes.

Values:

enumerator `kStatus_LPI2C_Busy`

The master is already performing a transfer.

enumerator `kStatus_LPI2C_Idle`

The slave driver is idle.

enumerator `kStatus_LPI2C_Nak`

The slave device sent a NAK in response to a byte.

enumerator `kStatus_LPI2C_FifoError`

FIFO under run or overrun.

enumerator kStatus_LPI2C_BitError

Transferred bit was not seen on the bus.

enumerator kStatus_LPI2C_ArbitrationLost

Arbitration lost error.

enumerator kStatus_LPI2C_PinLowTimeout

SCL or SDA were held low longer than the timeout.

enumerator kStatus_LPI2C_NoTransferInProgress

Attempt to abort a transfer when one is not in progress.

enumerator kStatus_LPI2C_DmaRequestFail

DMA request failed.

enumerator kStatus_LPI2C_Timeout

Timeout polling status flags.

IRQn_Type const kLpi2cIrqs[]

Array to map LPI2C instance number to IRQ number, used internally for LPI2C master interrupt and EDMA transactional APIs.

lpi2c_master_isr_t s_lpi2cMasterIsr

Pointer to master IRQ handler for each instance, used internally for LPI2C master interrupt and EDMA transactional APIs.

void *s_lpi2cMasterHandle[]

Pointers to master handles for each instance, used internally for LPI2C master interrupt and EDMA transactional APIs.

uint32_t LPI2C_GetInstance(LPI2C_Type *base)

Returns an instance number given a base address.

If an invalid base address is passed, debug builds will assert. Release builds will just return instance number 0.

Parameters

- base – The LPI2C peripheral base address.

Returns

LPI2C instance number starting from 0.

I2C_RETRY_TIMES

Retry times for waiting flag.

2.40 LPI2C Master Driver

void LPI2C_MasterGetDefaultConfig(*lpi2c_master_config_t* *masterConfig)

Provides a default configuration for the LPI2C master peripheral.

This function provides the following default configuration for the LPI2C master peripheral:

```
masterConfig->enableMaster      = true;
masterConfig->debugEnable       = false;
masterConfig->ignoreAck         = false;
masterConfig->pinConfig          = kLPI2C_2PinOpenDrain;
masterConfig->baudRate_Hz       = 100000U;
masterConfig->busIdleTimeout_ns = 0;
masterConfig->pinLowTimeout_ns  = 0;
masterConfig->sdaGlitchFilterWidth_ns = 0;
```

(continues on next page)

(continued from previous page)

```

masterConfig->sciGlitchFilterWidth_ns = 0;
masterConfig->hostRequest.enable      = false;
masterConfig->hostRequest.source      = kLPI2C_HostRequestExternalPin;
masterConfig->hostRequest.polarity    = kLPI2C_HostRequestPinActiveHigh;

```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with `LPI2C_MasterInit()`.

Parameters

- `masterConfig` – **[out]** User provided configuration structure for default values. Refer to `lpi2c_master_config_t`.

```
void LPI2C_MasterInit(LPI2C_Type *base, const lpi2c_master_config_t *masterConfig, uint32_t
                    sourceClock_Hz)
```

Initializes the LPI2C master peripheral.

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- `base` – The LPI2C peripheral base address.
- `masterConfig` – User provided peripheral configuration. Use `LPI2C_MasterGetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

```
void LPI2C_MasterDeinit(LPI2C_Type *base)
```

Deinitializes the LPI2C master peripheral.

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- `base` – The LPI2C peripheral base address.

```
void LPI2C_MasterConfigureDataMatch(LPI2C_Type *base, const lpi2c_data_match_config_t
                                    *matchConfig)
```

Configures LPI2C master data match feature.

Parameters

- `base` – The LPI2C peripheral base address.
- `matchConfig` – Settings for the data match feature.

```
status_t LPI2C_MasterCheckAndClearError(LPI2C_Type *base, uint32_t status)
```

Convert provided flags to status code, and clear any errors if present.

Parameters

- `base` – The LPI2C peripheral base address.
- `status` – Current status flags value that will be checked.

Return values

- `kStatus_Success` –
- `kStatus_LPI2C_PinLowTimeout` –
- `kStatus_LPI2C_ArbitrationLost` –

- kStatus_LPI2C_Nak –
- kStatus_LPI2C_FifoError –

`status_t` LPI2C_CheckForBusyBus(LPI2C_Type *base)

Make sure the bus isn't already busy.

A busy bus is allowed if we are the one driving it.

Parameters

- base – The LPI2C peripheral base address.

Return values

- kStatus_Success –
- kStatus_LPI2C_Busy –

`static inline void` LPI2C_MasterReset(LPI2C_Type *base)

Performs a software reset.

Restores the LPI2C master peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

`static inline void` LPI2C_MasterEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as master.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as master.

`static inline uint32_t` LPI2C_MasterGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C master status flags.

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_lpi2c_master_flags`

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

`static inline void` LPI2C_MasterClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C master status flag state.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag

- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

Attempts to clear other flags has no effect.

See also:

`_lpi2c_master_flags`.

Parameters

- `base` – The LPI2C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_lpi2c_master_flags` enumerators OR'd together. You may pass the result of a previous call to `LPI2C_MasterGetStatusFlags()`.

`static inline void LPI2C_MasterEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)`

Enables the LPI2C master interrupt requests.

All flags except `kLPI2C_MasterBusyFlag` and `kLPI2C_MasterBusBusyFlag` can be enabled as interrupts.

Parameters

- `base` – The LPI2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_lpi2c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline void LPI2C_MasterDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)`

Disables the LPI2C master interrupt requests.

All flags except `kLPI2C_MasterBusyFlag` and `kLPI2C_MasterBusBusyFlag` can be enabled as interrupts.

Parameters

- `base` – The LPI2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_lpi2c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline uint32_t LPI2C_MasterGetEnabledInterrupts(LPI2C_Type *base)`

Returns the set of currently enabled LPI2C master interrupt requests.

Parameters

- `base` – The LPI2C peripheral base address.

Returns

A bitmask composed of `_lpi2c_master_flags` enumerators OR'd together to indicate the set of enabled interrupts.

`static inline void LPI2C_MasterEnableDMA(LPI2C_Type *base, bool enableTx, bool enableRx)`

Enables or disables LPI2C master DMA requests.

Parameters

- `base` – The LPI2C peripheral base address.
- `enableTx` – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- `enableRx` – Enable flag for receive DMA request. Pass true for enable, false for disable.

```
static inline uint32_t LPI2C_MasterGetTxFifoAddress(LPI2C_Type *base)
```

Gets LPI2C master transmit data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Transmit Data Register address.

```
static inline uint32_t LPI2C_MasterGetRxFifoAddress(LPI2C_Type *base)
```

Gets LPI2C master receive data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Receive Data Register address.

```
static inline void LPI2C_MasterSetWatermarks(LPI2C_Type *base, size_t txWords, size_t rxWords)
```

Sets the watermarks for LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txWords – Transmit FIFO watermark value in words. The kLPI2C_MasterTxReadyFlag flag is set whenever the number of words in the transmit FIFO is equal or less than txWords. Writing a value equal or greater than the FIFO size is truncated.
- rxWords – Receive FIFO watermark value in words. The kLPI2C_MasterRxReadyFlag flag is set whenever the number of words in the receive FIFO is greater than rxWords. Writing a value equal or greater than the FIFO size is truncated.

```
static inline void LPI2C_MasterGetFifoCounts(LPI2C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of words in the LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txCount – **[out]** Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.

```
void LPI2C_MasterSetBaudRate(LPI2C_Type *base, uint32_t sourceClock_Hz, uint32_t baudRate_Hz)
```

Sets the I2C bus frequency for master transactions.

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Note: Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

Parameters

- `base` – The LPI2C peripheral base address.
- `sourceClock_Hz` – LPI2C functional clock frequency in Hertz.
- `baudRate_Hz` – Requested bus frequency in Hertz.

`static inline bool LPI2C_MasterGetBusIdleState(LPI2C_Type *base)`

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- `base` – The LPI2C peripheral base address.

Return values

- `true` – Bus is busy.
- `false` – Bus is idle.

`status_t LPI2C_MasterStart(LPI2C_Type *base, uint8_t address, lpi2c_direction_t dir)`

Sends a START signal and slave address on the I2C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- `base` – The LPI2C peripheral base address.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kLPI2C_Read` or `kLPI2C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- `kStatus_Success` – START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.

`static inline status_t LPI2C_MasterRepeatedStart(LPI2C_Type *base, uint8_t address, lpi2c_direction_t dir)`

Sends a repeated START signal and slave address on the I2C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like `LPI2C_MasterStart()`, it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- `base` – The LPI2C peripheral base address.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kLPI2C_Read` or `kLPI2C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- `kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.

status_t LPI2C_MasterSend(LPI2C_Type *base, void *txBuff, size_t txSize)

Performs a polling send transfer on the I2C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns `kStatus_LPI2C_Nak`.

Parameters

- *base* – The LPI2C peripheral base address.
- *txBuff* – The pointer to the data to be transferred.
- *txSize* – The length in bytes of the data to be transferred.

Return values

- `kStatus_Success` – Data was sent successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or over run.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- *base* – The LPI2C peripheral base address.
- *rxBuff* – The pointer to the data to be transferred.
- *rxSize* – The length in bytes of the data to be transferred.

Return values

- `kStatus_Success` – Data was received successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterStop(LPI2C_Type *base)

Sends a STOP signal on the I2C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- *base* – The LPI2C peripheral base address.

Return values

- `kStatus_Success` – The STOP signal was successfully sent on the bus and the transaction terminated.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

`status_t LPI2C_MasterTransferBlocking(LPI2C_Type *base, lpi2c_master_transfer_t *transfer)`
Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- `base` – The LPI2C peripheral base address.
- `transfer` – Pointer to the transfer structure.

Return values

- `kStatus_Success` – Data was received successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

`void LPI2C_MasterTransferCreateHandle(LPI2C_Type *base, lpi2c_master_handle_t *handle, lpi2c_master_transfer_callback_t callback, void *userData)`

Creates a new handle for the LPI2C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `LPI2C_MasterTransferAbort()` API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- `base` – The LPI2C peripheral base address.
- `handle` – **[out]** Pointer to the LPI2C master driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

```
status_t LPI2C_MasterTransferNonBlocking(LPI2C_Type *base, lpi2c_master_handle_t *handle,  
                                         lpi2c_master_transfer_t *transfer)
```

Performs a non-blocking transaction on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

```
status_t LPI2C_MasterTransferGetCount(LPI2C_Type *base, lpi2c_master_handle_t *handle,  
                                       size_t *count)
```

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

```
void LPI2C_MasterTransferAbort(LPI2C_Type *base, lpi2c_master_handle_t *handle)
```

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

```
void LPI2C_MasterTransferHandleIRQ(LPI2C_Type *base, void *lpi2cMasterHandle)
```

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The LPI2C peripheral base address.
- lpi2cMasterHandle – Pointer to the LPI2C master driver handle.

enum _lpi2c_master_flags

LPI2C master peripheral flags.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_MasterTxReadyFlag

Transmit data flag

enumerator kLPI2C_MasterRxReadyFlag

Receive data flag

enumerator kLPI2C_MasterEndOfPacketFlag

End Packet flag

enumerator kLPI2C_MasterStopDetectFlag

Stop detect flag

enumerator kLPI2C_MasterNackDetectFlag

NACK detect flag

enumerator kLPI2C_MasterArbitrationLostFlag

Arbitration lost flag

enumerator kLPI2C_MasterFifoErrFlag

FIFO error flag

enumerator kLPI2C_MasterPinLowTimeoutFlag

Pin low timeout flag

enumerator kLPI2C_MasterDataMatchFlag

Data match flag

enumerator kLPI2C_MasterBusyFlag

Master busy flag

enumerator kLPI2C_MasterBusBusyFlag

Bus busy flag

enumerator kLPI2C_MasterClearFlags

All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_MasterIrqFlags

IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_MasterErrorFlags

Errors to check for.

enum _lpi2c_direction

Direction of master and slave transfers.

Values:

enumerator kLPI2C_Write

Master transmit.

enumerator kLPI2C_Read

Master receive.

enum _lpi2c_master_pin_config

LPI2C pin configuration.

Values:

enumerator kLPI2C_2PinOpenDrain

LPI2C Configured for 2-pin open drain mode

enumerator kLPI2C_2PinOutputOnly

LPI2C Configured for 2-pin output only mode (ultra-fast mode)

enumerator kLPI2C_2PinPushPull

LPI2C Configured for 2-pin push-pull mode

enumerator kLPI2C_4PinPushPull

LPI2C Configured for 4-pin push-pull mode

enumerator kLPI2C_2PinOpenDrainWithSeparateSlave

LPI2C Configured for 2-pin open drain mode with separate LPI2C slave

enumerator kLPI2C_2PinOutputOnlyWithSeparateSlave

LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave

enumerator kLPI2C_2PinPushPullWithSeparateSlave

LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave

enumerator kLPI2C_4PinPushPullWithInvertedOutput

LPI2C Configured for 4-pin push-pull mode(inverted outputs)

enum _lpi2c_host_request_source

LPI2C master host request selection.

Values:

enumerator kLPI2C_HostRequestExternalPin

Select the LPI2C_HREQ pin as the host request input

enumerator kLPI2C_HostRequestInputTrigger

Select the input trigger as the host request input

enum _lpi2c_host_request_polarity

LPI2C master host request pin polarity configuration.

Values:

enumerator kLPI2C_HostRequestPinActiveLow

Configure the LPI2C_HREQ pin active low

enumerator kLPI2C_HostRequestPinActiveHigh
Configure the LPI2C_HREQ pin active high

enum _lpi2c_data_match_config_mode
LPI2C master data match configuration modes.

Values:

enumerator kLPI2C_MatchDisabled
LPI2C Match Disabled

enumerator kLPI2C_1stWordEqualsM0OrM1
LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1

enumerator kLPI2C_AnyWordEqualsM0OrM1
LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1

enumerator kLPI2C_1stWordEqualsM0And2ndWordEqualsM1
LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1

enumerator kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1
LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1

enumerator kLPI2C_1stWordAndM1EqualsM0AndM1
LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1

enumerator kLPI2C_AnyWordAndM1EqualsM0AndM1
LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1

enum _lpi2c_master_transfer_flags
Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the _lpi2c_master_transfer::flags field.

Values:

enumerator kLPI2C_TransferDefaultFlag
Transfer starts with a start signal, stops with a stop signal.

enumerator kLPI2C_TransferNoStartFlag
Don't send a start condition, address, and sub address

enumerator kLPI2C_TransferRepeatedStartFlag
Send a repeated start condition

enumerator kLPI2C_TransferNoStopFlag
Don't send a stop condition.

typedef enum _lpi2c_direction lpi2c_direction_t
Direction of master and slave transfers.

typedef enum _lpi2c_master_pin_config lpi2c_master_pin_config_t
LPI2C pin configuration.

typedef enum _lpi2c_host_request_source lpi2c_host_request_source_t
LPI2C master host request selection.

typedef enum _lpi2c_host_request_polarity lpi2c_host_request_polarity_t
LPI2C master host request pin polarity configuration.

```
typedef struct _lpi2c_master_config lpi2c_master_config_t
```

Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

```
typedef enum _lpi2c_data_match_config_mode lpi2c_data_match_config_mode_t
```

LPI2C master data match configuration modes.

```
typedef struct _lpi2c_match_config lpi2c_data_match_config_t
```

LPI2C master data match configuration structure.

```
typedef struct _lpi2c_master_transfer lpi2c_master_transfer_t
```

LPI2C master descriptor of the transfer.

```
typedef struct _lpi2c_master_handle lpi2c_master_handle_t
```

LPI2C master handle of the transfer.

```
typedef void (*lpi2c_master_transfer_callback_t)(LPI2C_Type *base, lpi2c_master_handle_t *handle, status_t completionStatus, void *userData)
```

Master completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C_MasterTransferCreateHandle().

Param base

The LPI2C peripheral base address.

Param handle

Pointer to the LPI2C master driver handle.

Param completionStatus

Either kStatus_Success or an error code describing how the transfer completed.

Param userData

Arbitrary pointer-sized value passed from the application.

```
typedef void (*lpi2c_master_isr_t)(LPI2C_Type *base, void *handle)
```

Typedef for master interrupt handler, used internally for LPI2C master interrupt and EDMA transactional APIs.

```
struct _lpi2c_master_config
```

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

```
bool enableMaster
```

Whether to enable master mode.

```
bool enableDoze
```

Whether master is enabled in doze mode.

bool debugEnable

Enable transfers to continue when halted in debug mode.

bool ignoreAck

Whether to ignore ACK/NACK.

lpi2c_master_pin_config_t pinConfig

The pin configuration option.

uint32_t baudRate_Hz

Desired baud rate in Hertz.

uint32_t busIdleTimeout_ns

Bus idle timeout in nanoseconds. Set to 0 to disable.

uint32_t pinLowTimeout_ns

Pin low timeout in nanoseconds. Set to 0 to disable.

uint8_t sdaGlitchFilterWidth_ns

Width in nanoseconds of glitch filter on SDA pin. Set to 0 to disable.

uint8_t sclGlitchFilterWidth_ns

Width in nanoseconds of glitch filter on SCL pin. Set to 0 to disable.

struct *_lpi2c_master_config* hostRequest

Host request options.

struct *_lpi2c_match_config*

#include <fsl_lpi2c.h> LPI2C master data match configuration structure.

Public Members

lpi2c_data_match_config_mode_t matchMode

Data match configuration setting.

bool rxDataMatchOnly

When set to true, received data is ignored until a successful match.

uint32_t match0

Match value 0.

uint32_t match1

Match value 1.

struct *_lpi2c_master_transfer*

#include <fsl_lpi2c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the LPI2C_MasterTransferNonBlocking() API.

Public Members

uint32_t flags

Bit mask of options for the transfer. See enumeration *_lpi2c_master_transfer_flags* for available options. Set to 0 or *kLPI2C_TransferDefaultFlag* for normal transfers.

uint16_t slaveAddress

The 7-bit slave address.

lpi2c_direction_t direction

Either kLPI2C_Read or kLPI2C_Write.

uint32_t subaddress

Sub address. Transferred MSB first.

size_t subaddressSize

Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data

Pointer to data to transfer.

size_t dataSize

Number of bytes to transfer.

struct _lpi2c_master_handle

#include <fsl_lpi2c.h> Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state

Transfer state machine current state.

uint16_t remainingBytes

Remaining byte count in current state.

uint8_t *buf

Buffer pointer for current state.

uint16_t commandBuffer[6]

LPI2C command sequence. When all 6 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word]

lpi2c_master_transfer_t transfer

Copy of the current transfer info.

lpi2c_master_transfer_callback_t completionCallback

Callback function pointer.

void *userData

Application data passed to callback.

struct hostRequest

Public Members

bool enable

Enable host request.

lpi2c_host_request_source_t source

Host request source.

lpi2c_host_request_polarity_t polarity

Host request pin polarity.

2.41 LPI2C Master DMA Driver

```
void LPI2C_MasterCreateEDMAHandle(LPI2C_Type *base, lpi2c_master_edma_handle_t *handle,  
                                edma_handle_t *rxDmaHandle, edma_handle_t  
                                *txDmaHandle, lpi2c_master_edma_transfer_callback_t  
                                callback, void *userData)
```

Create a new handle for the LPI2C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_MasterTransferAbortEDMA() API shall be called.

For devices where the LPI2C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – **[out]** Pointer to the LPI2C master driver handle.
- *rxDmaHandle* – Handle for the eDMA receive channel. Created by the user prior to calling this function.
- *txDmaHandle* – Handle for the eDMA transmit channel. Created by the user prior to calling this function.
- *callback* – User provided pointer to the asynchronous callback function.
- *userData* – User provided pointer to the application callback data.

```
status_t LPI2C_MasterTransferEDMA(LPI2C_Type *base, lpi2c_master_edma_handle_t *handle,  
                                lpi2c_master_transfer_t *transfer)
```

Performs a non-blocking DMA-based transaction on the I2C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – Pointer to the LPI2C master driver handle.
- *transfer* – The pointer to the transfer descriptor.

Return values

- *kStatus_Success* – The transaction was started successfully.
- *kStatus_LPI2C_Busy* – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

```
status_t LPI2C_MasterTransferGetCountEDMA(LPI2C_Type *base, lpi2c_master_edma_handle_t  
                                *handle, size_t *count)
```

Returns number of bytes transferred so far.

Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – Pointer to the LPI2C master driver handle.
- *count* – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- *kStatus_Success* –

- `kStatus_NoTransferInProgress` – There is not a DMA transaction currently in progress.

`status_t` LPI2C_MasterTransferAbortEDMA(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle)

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the eDMA peripheral's IRQ priority.

Parameters

- `base` – The LPI2C peripheral base address.
- `handle` – Pointer to the LPI2C master driver handle.

Return values

- `kStatus_Success` – A transaction was successfully aborted.
- `kStatus_LPI2C_Idle` – There is not a DMA transaction currently in progress.

`typedef struct _lpi2c_master_edma_handle` `lpi2c_master_edma_handle_t`
LPI2C master EDMA handle of the transfer.

`typedef void (*lpi2c_master_edma_transfer_callback_t)(LPI2C_Type *base,`
`lpi2c_master_edma_handle_t *handle, status_t completionStatus, void *userData)`

Master DMA completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to `LPI2C_MasterCreateEDMAHandle()`.

Param base

The LPI2C peripheral base address.

Param handle

Handle associated with the completed transfer.

Param completionStatus

Either `kStatus_Success` or an error code describing how the transfer completed.

Param userData

Arbitrary pointer-sized value passed from the application.

`struct _lpi2c_master_edma_handle`
`#include <fsl_lpi2c_edma.h>` Driver handle for master DMA APIs.

Note: The contents of this structure are private and subject to change.

Public Members

`LPI2C_Type *base`
LPI2C base pointer.

`bool isBusy`
Transfer state machine current state.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint16_t commandBuffer[20]

LPI2C command sequence. When all 10 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word] + receive&Size[4 words]

lpi2c_master_transfer_t transfer

Copy of the current transfer info.

lpi2c_master_edma_transfer_callback_t completionCallback

Callback function pointer.

void *userData

Application data passed to callback.

edma_handle_t *rx

Handle for receive DMA channel.

edma_handle_t *tx

Handle for transmit DMA channel.

edma_tcd_t tcds[3]

Software TCD. Three are allocated to provide enough room to align to 32-bytes.

2.42 LPI2C Slave Driver

void LPI2C_SlaveGetDefaultConfig(*lpi2c_slave_config_t* *slaveConfig)

Provides a default configuration for the LPI2C slave peripheral.

This function provides the following default configuration for the LPI2C slave peripheral:

```
slaveConfig->enableSlave      = true;
slaveConfig->address0         = 0U;
slaveConfig->address1         = 0U;
slaveConfig->addressMatchMode = kLPI2C_MatchAddress0;
slaveConfig->filterDozeEnable  = true;
slaveConfig->filterEnable     = true;
slaveConfig->enableGeneralCall = false;
slaveConfig->sclStall.enableAck = false;
slaveConfig->sclStall.enableTx  = true;
slaveConfig->sclStall.enableRx  = true;
slaveConfig->sclStall.enableAddress = true;
slaveConfig->ignoreAck         = false;
slaveConfig->enableReceivedAddressRead = false;
slaveConfig->sdaGlitchFilterWidth_ns = 0;
slaveConfig->sclGlitchFilterWidth_ns = 0;
slaveConfig->dataValidDelay_ns  = 0;
slaveConfig->clockHoldTime_ns   = 0;
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with LPI2C_SlaveInit(). Be sure to override at least the *address0* member of the configuration structure with the desired slave address.

Parameters

- slaveConfig – **[out]** User provided configuration structure that is set to default values. Refer to *lpi2c_slave_config_t*.


```
void LPI2C_SlaveInit(LPI2C_Type *base, const lpi2c_slave_config_t *slaveConfig, uint32_t  
                    sourceClock_Hz)
```

Initializes the LPI2C slave peripheral.

This function enables the peripheral clock and initializes the LPI2C slave peripheral as described by the user provided configuration.

Parameters

- `base` – The LPI2C peripheral base address.
- `slaveConfig` – User provided peripheral configuration. Use `LPI2C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the LPI2C functional clock. Used to calculate the filter widths, data valid delay, and clock hold time.

```
void LPI2C_SlaveDeinit(LPI2C_Type *base)
```

Deinitializes the LPI2C slave peripheral.

This function disables the LPI2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- `base` – The LPI2C peripheral base address.

```
static inline void LPI2C_SlaveReset(LPI2C_Type *base)
```

Performs a software reset of the LPI2C slave peripheral.

Parameters

- `base` – The LPI2C peripheral base address.

```
static inline void LPI2C_SlaveEnable(LPI2C_Type *base, bool enable)
```

Enables or disables the LPI2C module as slave.

Parameters

- `base` – The LPI2C peripheral base address.
- `enable` – Pass true to enable or false to disable the specified LPI2C as slave.

```
static inline uint32_t LPI2C_SlaveGetStatusFlags(LPI2C_Type *base)
```

Gets the LPI2C slave status flags.

A bit mask with the state of all LPI2C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_lpi2c_slave_flags`

Parameters

- `base` – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void LPI2C_SlaveClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)
```

Clears the LPI2C status flag state.

The following status register flags can be cleared:

- kLPI2C_SlaveRepeatedStartDetectFlag
- kLPI2C_SlaveStopDetectFlag
- kLPI2C_SlaveBitErrFlag
- kLPI2C_SlaveFifoErrFlag

Attempts to clear other flags has no effect.

See also:

`_lpi2c_slave_flags`.

Parameters

- `base` – The LPI2C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_lpi2c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `LPI2C_SlaveGetStatusFlags()`.

```
static inline void LPI2C_SlaveEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)
```

Enables the LPI2C slave interrupt requests.

All flags except `kLPI2C_SlaveBusyFlag` and `kLPI2C_SlaveBusBusyFlag` can be enabled as interrupts.

Parameters

- `base` – The LPI2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_lpi2c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline void LPI2C_SlaveDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)
```

Disables the LPI2C slave interrupt requests.

All flags except `kLPI2C_SlaveBusyFlag` and `kLPI2C_SlaveBusBusyFlag` can be enabled as interrupts.

Parameters

- `base` – The LPI2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_lpi2c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline uint32_t LPI2C_SlaveGetEnabledInterrupts(LPI2C_Type *base)
```

Returns the set of currently enabled LPI2C slave interrupt requests.

Parameters

- `base` – The LPI2C peripheral base address.

Returns

A bitmask composed of `_lpi2c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

```
static inline void LPI2C_SlaveEnableDMA(LPI2C_Type *base, bool enableAddressValid, bool enableRx, bool enableTx)
```

Enables or disables the LPI2C slave peripheral DMA requests.

Parameters

- `base` – The LPI2C peripheral base address.
- `enableAddressValid` – Enable flag for the address valid DMA request. Pass true for enable, false for disable. The address valid DMA request is shared with the receive data DMA request.
- `enableRx` – Enable flag for the receive data DMA request. Pass true for enable, false for disable.
- `enableTx` – Enable flag for the transmit data DMA request. Pass true for enable, false for disable.

```
static inline bool LPI2C_SlaveGetBusIdleState(LPI2C_Type *base)
```

Returns whether the bus is idle.

Requires the slave mode to be enabled.

Parameters

- `base` – The LPI2C peripheral base address.

Return values

- `true` – Bus is busy.
- `false` – Bus is idle.

```
static inline void LPI2C_SlaveTransmitAck(LPI2C_Type *base, bool ackOrNack)
```

Transmits either an ACK or NAK on the I2C bus in response to a byte from the master.

Use this function to send an ACK or NAK when the `kLPI2C_SlaveTransmitAckFlag` is asserted. This only happens if you enable the `sclStall.enableAck` field of the `lpi2c_slave_config_t` configuration structure used to initialize the slave peripheral.

Parameters

- `base` – The LPI2C peripheral base address.
- `ackOrNack` – Pass true for an ACK or false for a NAK.

```
static inline void LPI2C_SlaveEnableAckStall(LPI2C_Type *base, bool enable)
```

Enables or disables ACKSTALL.

When enables ACKSTALL, software can transmit either an ACK or NAK on the I2C bus in response to a byte from the master.

Parameters

- `base` – The LPI2C peripheral base address.
- `enable` – True will enable ACKSTALL, false will disable ACKSTALL.

```
static inline uint32_t LPI2C_SlaveGetReceivedAddress(LPI2C_Type *base)
```

Returns the slave address sent by the I2C master.

This function should only be called if the `kLPI2C_SlaveAddressValidFlag` is asserted.

Parameters

- `base` – The LPI2C peripheral base address.

Returns

The 8-bit address matched by the LPI2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

```
status_t LPI2C_SlaveSend(LPI2C_Type *base, void *txBuff, size_t txSize, size_t *actualTxSize)
```

Performs a polling send transfer on the I2C bus.

Parameters

- `base` – The LPI2C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.
- `actualTxSize` – **[out]**

Returns

Error or success status returned by API.

`status_t` LPI2C_SlaveReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize, size_t *actualRxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- `base` – The LPI2C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.
- `actualRxSize` – **[out]**

Returns

Error or success status returned by API.

`void` LPI2C_SlaveTransferCreateHandle(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, *lpi2c_slave_transfer_callback_t* callback, void *userData)

Creates a new handle for the LPI2C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_SlaveTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- `base` – The LPI2C peripheral base address.
- `handle` – **[out]** Pointer to the LPI2C slave driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

`status_t` LPI2C_SlaveTransferNonBlocking(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, uint32_t eventMask)

Starts accepting slave transfers.

Call this API after calling I2C_SlaveInit() and LPI2C_SlaveTransferCreateHandle() to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to LPI2C_SlaveTransferCreateHandle(). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of *lpi2c_slave_transfer_event_t* enumerators for the events you wish to receive. The *kLPI2C_SlaveTransmitEvent* and *kLPI2C_SlaveReceiveEvent* events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the *kLPI2C_SlaveAllEvents* constant is provided as a convenient way to enable all events.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to `lpi2c_slave_handle_t` structure which stores the transfer state.
- eventMask – Bit mask formed by OR'ing together `lpi2c_slave_transfer_event_t` enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and `kLPI2C_SlaveAllEvents` to enable all events.

Return values

- `kStatus__Success` – Slave transfers were successfully started.
- `kStatus__LPI2C_Busy` – Slave transfers have already been started on this handle.

`status_t LPI2C_SlaveTransferGetCount(LPI2C_Type *base, lpi2c_slave_handle_t *handle, size_t *count)`

Gets the slave transfer status during a non-blocking transfer.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to `i2c_slave_handle_t` structure.
- count – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

- `kStatus__Success` –
- `kStatus__NoTransferInProgress` –

`void LPI2C_SlaveTransferAbort(LPI2C_Type *base, lpi2c_slave_handle_t *handle)`

Aborts the slave non-blocking transfers.

Note: This API could be called at any time to stop slave for handling the bus events.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to `lpi2c_slave_handle_t` structure which stores the transfer state.

`void LPI2C_SlaveTransferHandleIRQ(LPI2C_Type *base, lpi2c_slave_handle_t *handle)`

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to `lpi2c_slave_handle_t` structure which stores the transfer state.

enum _lpi2c_slave_flags

LPI2C slave peripheral flags.

The following status register flags can be cleared:

- kLPI2C_SlaveRepeatedStartDetectFlag
- kLPI2C_SlaveStopDetectFlag
- kLPI2C_SlaveBitErrFlag
- kLPI2C_SlaveFifoErrFlag

All flags except kLPI2C_SlaveBusyFlag and kLPI2C_SlaveBusBusyFlag can be enabled as interrupts.

Note: These enumerations are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_SlaveTxReadyFlag

Transmit data flag

enumerator kLPI2C_SlaveRxReadyFlag

Receive data flag

enumerator kLPI2C_SlaveAddressValidFlag

Address valid flag

enumerator kLPI2C_SlaveTransmitAckFlag

Transmit ACK flag

enumerator kLPI2C_SlaveRepeatedStartDetectFlag

Repeated start detect flag

enumerator kLPI2C_SlaveStopDetectFlag

Stop detect flag

enumerator kLPI2C_SlaveBitErrFlag

Bit error flag

enumerator kLPI2C_SlaveFifoErrFlag

FIFO error flag

enumerator kLPI2C_SlaveAddressMatch0Flag

Address match 0 flag

enumerator kLPI2C_SlaveAddressMatch1Flag

Address match 1 flag

enumerator kLPI2C_SlaveGeneralCallFlag

General call flag

enumerator kLPI2C_SlaveBusyFlag

Master busy flag

enumerator kLPI2C_SlaveBusBusyFlag

Bus busy flag

enumerator kLPI2C_SlaveClearFlags

All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_SlaveIrqFlags

IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_SlaveErrorFlags

Errors to check for.

enum __lpi2c_slave_address_match

LPI2C slave address match options.

Values:

enumerator kLPI2C_MatchAddress0

Match only address 0.

enumerator kLPI2C_MatchAddress0OrAddress1

Match either address 0 or address 1.

enumerator kLPI2C_MatchAddress0ThroughAddress1

Match a range of slave addresses from address 0 through address 1.

enum __lpi2c_slave_transfer_event

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

Values:

enumerator kLPI2C_SlaveAddressMatchEvent

Received the slave address after a start or repeated start.

enumerator kLPI2C_SlaveTransmitEvent

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator kLPI2C_SlaveReceiveEvent

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kLPI2C_SlaveTransmitAckEvent

Callback needs to either transmit an ACK or NACK.

enumerator kLPI2C_SlaveRepeatedStartEvent

A repeated start was detected.

enumerator kLPI2C_SlaveCompletionEvent

A stop was detected, completing the transfer.

enumerator kLPI2C_SlaveAllEvents

Bit mask of all available events.

typedef enum __lpi2c_slave_address_match lpi2c_slave_address_match_t

LPI2C slave address match options.

typedef struct __lpi2c_slave_config lpi2c_slave_config_t

Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the LPI2C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

```
typedef enum _lpi2c_slave_transfer_event lpi2c_slave_transfer_event_t
```

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

```
typedef struct _lpi2c_slave_transfer lpi2c_slave_transfer_t
```

LPI2C slave transfer structure.

```
typedef struct _lpi2c_slave_handle lpi2c_slave_handle_t
```

LPI2C slave handle structure.

```
typedef void (*lpi2c_slave_transfer_callback_t)(LPI2C_Type *base, lpi2c_slave_transfer_t  
*transfer, void *userData)
```

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the LPI2C_SlaveSetCallback() function after you have created a handle.

Param base

Base address for the LPI2C instance on which the event occurred.

Param transfer

Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData

Arbitrary pointer-sized value passed from the application.

```
struct _lpi2c_slave_config
```

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the LPI2C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableSlave

Enable slave mode.

uint8_t address0

Slave's 7-bit address.

uint8_t address1

Alternate slave 7-bit address.

lpi2c_slave_address_match_t addressMatchMode

Address matching options.

bool filterDozeEnable

Enable digital glitch filter in doze mode.

`bool filterEnable`
Enable digital glitch filter.

`bool enableGeneralCall`
Enable general call address matching.

`struct _lpi2c_slave_config sclStall`
SCL stall enable options.

`bool ignoreAck`
Continue transfers after a NACK is detected.

`bool enableReceivedAddressRead`
Enable reading the address received address as the first byte of data.

`uint32_t sdaGlitchFilterWidth_ns`
Width in nanoseconds of the digital filter on the SDA signal. Set to 0 to disable.

`uint32_t sclGlitchFilterWidth_ns`
Width in nanoseconds of the digital filter on the SCL signal. Set to 0 to disable.

`uint32_t dataValidDelay_ns`
Width in nanoseconds of the data valid delay.

`uint32_t clockHoldTime_ns`
Width in nanoseconds of the clock hold time.

`struct _lpi2c_slave_transfer`
#include <fsl_lpi2c.h> LPI2C slave transfer structure.

Public Members

`lpi2c_slave_transfer_event_t event`
Reason the callback is being invoked.

`uint8_t receivedAddress`
Matching address send by master.

`uint8_t *data`
Transfer buffer

`size_t dataSize`
Transfer size

`status_t completionStatus`
Success or error code describing how the transfer completed. Only applies for kLPI2C_SlaveCompletionEvent.

`size_t transferredCount`
Number of bytes actually transferred since start or last repeated start.

`struct _lpi2c_slave_handle`
#include <fsl_lpi2c.h> LPI2C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

lpi2c_slave_transfer_t transfer
LPI2C slave transfer copy.

bool isBusy
Whether transfer is busy.

bool wasTransmit
Whether the last transfer was a transmit.

uint32_t eventMask
Mask of enabled events.

uint32_t transferredCount
Count of bytes transferred.

lpi2c_slave_transfer_callback_t callback
Callback function called at transfer event.

void *userData
Callback parameter passed to callback.

struct sclStall

Public Members

bool enableAck
Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted. Clock stretching occurs when transmitting the 9th bit. When enableAckSCLStall is enabled, there is no need to set either enableRxDatSCLStall or enableAddressSCLStall.

bool enableTx
Enables SCL clock stretching when the transmit data flag is set during a slave-transmit transfer.

bool enableRx
Enables SCL clock stretching when receive data flag is set during a slave-receive transfer.

bool enableAddress
Enables SCL clock stretching when the address valid flag is asserted.

2.43 LPIT: Low-Power Interrupt Timer

void LPIT_Init(LPIT_Type *base, const *lpit_config_t* *config)
Ungates the LPIT clock and configures the peripheral for a basic operation.
This function issues a software reset to reset all channels and registers except the Module Control register.

Note: This API should be called at the beginning of the application using the LPIT driver.

Parameters

- `base` – LPIT peripheral base address.
- `config` – Pointer to the user configuration structure.

`void LPIT_Deinit(LPIT_Type *base)`

Disables the module and gates the LPIT clock.

Parameters

- `base` – LPIT peripheral base address.

`void LPIT_GetDefaultConfig(lpit_config_t *config)`

Fills in the LPIT configuration structure with default settings.

The default values are:

```
config->enableRunInDebug = false;
config->enableRunInDoze = false;
```

Parameters

- `config` – Pointer to the user configuration structure.

`status_t LPIT_SetupChannel(LPIT_Type *base, lpit_chnl_t channel, const lpit_chnl_params_t *chnlSetup)`

Sets up an LPIT channel based on the user's preference.

This function sets up the operation mode to one of the options available in the enumeration `lpit_timer_modes_t`. It sets the trigger source as either internal or external, trigger selection and the timers behaviour when a timeout occurs. It also chains the timer if a prior timer if requested by the user.

Parameters

- `base` – LPIT peripheral base address.
- `channel` – Channel that is being configured.
- `chnlSetup` – Configuration parameters.

`static inline void LPIT_EnableInterrupts(LPIT_Type *base, uint32_t mask)`

Enables the selected PIT interrupts.

Parameters

- `base` – LPIT peripheral base address.
- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `lpit_interrupt_enable_t`

`static inline void LPIT_DisableInterrupts(LPIT_Type *base, uint32_t mask)`

Disables the selected PIT interrupts.

Parameters

- `base` – LPIT peripheral base address.
- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `lpit_interrupt_enable_t`

`static inline uint32_t LPIT_GetEnabledInterrupts(LPIT_Type *base)`

Gets the enabled LPIT interrupts.

Parameters

- `base` – LPIT peripheral base address.

Returns

The enabled interrupts. This is the logical OR of members of the enumeration `lpit_interrupt_enable_t`

```
static inline uint32_t LPIT_GetStatusFlags(LPIT_Type *base)
```

Gets the LPIT status flags.

Parameters

- `base` – LPIT peripheral base address.

Returns

The status flags. This is the logical OR of members of the enumeration `lpit_status_flags_t`

```
static inline void LPIT_ClearStatusFlags(LPIT_Type *base, uint32_t mask)
```

Clears the LPIT status flags.

Parameters

- `base` – LPIT peripheral base address.
- `mask` – The status flags to clear. This is a logical OR of members of the enumeration `lpit_status_flags_t`

```
static inline void LPIT_SetTimerPeriod(LPIT_Type *base, lpit_chnl_t channel, uint32_t ticks)
```

Sets the timer period in units of count.

Timers begin counting down from the value set by this function until it reaches 0, at which point it generates an interrupt and loads this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

Note: User can call the utility macros provided in `fsl_common.h` to convert to ticks.

Parameters

- `base` – LPIT peripheral base address.
- `channel` – Timer channel number.
- `ticks` – Timer period in units of ticks.

```
static inline void LPIT_SetTimerValue(LPIT_Type *base, lpit_chnl_t channel, uint32_t ticks)
```

Sets the timer period in units of count.

In the Dual 16-bit Periodic Counter mode, the counter will load and then the lower 16-bits will decrement down to zero, which will assert the output pre-trigger. The upper 16-bits will then decrement down to zero, which will negate the output pre-trigger and set the timer interrupt flag.

Note: Set TVAL register to 0 or 1 is invalid in compare mode.

Parameters

- `base` – LPIT peripheral base address.
- `channel` – Timer channel number.
- `ticks` – Timer period in units of ticks.

static inline uint32_t LPIT_GetCurrentTimerCount(LPIT_Type *base, *lpit_chnl_t* channel)

Reads the current timer counting value.

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note: User can call the utility macros provided in `fsl_common.h` to convert ticks to microseconds or milliseconds.

Parameters

- `base` – LPIT peripheral base address.
- `channel` – Timer channel number.

Returns

Current timer counting value in ticks.

static inline void LPIT_StartTimer(LPIT_Type *base, *lpit_chnl_t* channel)

Starts the timer counting.

After calling this function, timers load the period value and count down to 0. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

- `base` – LPIT peripheral base address.
- `channel` – Timer channel number.

static inline void LPIT_StopTimer(LPIT_Type *base, *lpit_chnl_t* channel)

Stops the timer counting.

Parameters

- `base` – LPIT peripheral base address.
- `channel` – Timer channel number.

FSL_LPIT_DRIVER_VERSION

Version 2.1.3

enum *_lpit_chnl*

List of LPIT channels.

Note: Actual number of available channels is SoC-dependent

Values:

enumerator `kLPIT_Chnl_0`
LPIT channel number 0

enumerator `kLPIT_Chnl_1`
LPIT channel number 1

enumerator `kLPIT_Chnl_2`
LPIT channel number 2

enumerator `kLPIT_Chnl_3`
LPIT channel number 3

enum *_lpit_timer_modes*

Mode options available for the LPIT timer.

Values:

enumerator kLPIT_PeriodicCounter

Use the all 32-bits, counter loads and decrements to zero

enumerator kLPIT_DualPeriodicCounter

Counter loads, lower 16-bits decrement to zero, then upper 16-bits decrement

enumerator kLPIT_TriggerAccumulator

Counter loads on first trigger and decrements on each trigger

enumerator kLPIT_InputCapture

Counter loads with 0xFFFFFFFF, decrements to zero. It stores the inverse of the current value when a input trigger is detected

enum _lpit_trigger_select

Trigger options available.

This is used for both internal and external trigger sources. The actual trigger options available is SoC-specific, user should refer to the reference manual.

Values:

enumerator kLPIT_Trigger_TimerChn0

Channel 0 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn1

Channel 1 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn2

Channel 2 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn3

Channel 3 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn4

Channel 4 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn5

Channel 5 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn6

Channel 6 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn7

Channel 7 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn8

Channel 8 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn9

Channel 9 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn10

Channel 10 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn11

Channel 11 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn12

Channel 12 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn13

Channel 13 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn14
Channel 14 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn15
Channel 15 is selected as a trigger source

enum _lpit_trigger_source
Trigger source options available.

Values:

enumerator kLPIT_TriggerSource_External
Use external trigger input

enumerator kLPIT_TriggerSource_Internal
Use internal trigger

enum _lpit_interrupt_enable
List of LPIT interrupts.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

Values:

enumerator kLPIT_Channel0TimerInterruptEnable
Channel 0 Timer interrupt

enumerator kLPIT_Channel1TimerInterruptEnable
Channel 1 Timer interrupt

enumerator kLPIT_Channel2TimerInterruptEnable
Channel 2 Timer interrupt

enumerator kLPIT_Channel3TimerInterruptEnable
Channel 3 Timer interrupt

enum _lpit_status_flags
List of LPIT status flags.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

Values:

enumerator kLPIT_Channel0TimerFlag
Channel 0 Timer interrupt flag

enumerator kLPIT_Channel1TimerFlag
Channel 1 Timer interrupt flag

enumerator kLPIT_Channel2TimerFlag
Channel 2 Timer interrupt flag

enumerator kLPIT_Channel3TimerFlag
Channel 3 Timer interrupt flag

typedef enum _lpit_chnl lpit_chnl_t
List of LPIT channels.

Note: Actual number of available channels is SoC-dependent

```
typedef enum _lpit_timer_modes lpit_timer_modes_t
```

Mode options available for the LPIT timer.

```
typedef enum _lpit_trigger_select lpit_trigger_select_t
```

Trigger options available.

This is used for both internal and external trigger sources. The actual trigger options available is SoC-specific, user should refer to the reference manual.

```
typedef enum _lpit_trigger_source lpit_trigger_source_t
```

Trigger source options available.

```
typedef enum _lpit_interrupt_enable lpit_interrupt_enable_t
```

List of LPIT interrupts.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

```
typedef enum _lpit_status_flags lpit_status_flags_t
```

List of LPIT status flags.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

```
typedef struct _lpit_chnl_params lpit_chnl_params_t
```

Structure to configure the channel timer.

```
typedef struct _lpit_config lpit_config_t
```

LPIT configuration structure.

This structure holds the configuration settings for the LPIT peripheral. To initialize this structure to reasonable defaults, call the LPIT_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

```
static void LPIT_ResetStateDelay(void)
```

Short wait for LPIT state reset.

After clear or set LPIT_EN, there should be delay longer than 4 LPIT functional clock.

```
static inline void LPIT_Reset(LPIT_Type *base)
```

Performs a software reset on the LPIT module.

This resets all channels and registers except the Module Control Register.

Parameters

- base – LPIT peripheral base address.

```
LPIT_RESET_STATE_DELAY
```

Delay used in LPIT_Reset.

The macro value should be larger than 4 * core clock / LPIT peripheral clock.

```
struct _lpit_chnl_params
```

#include <fsl_lpit.h> Structure to configure the channel timer.

Public Members

```
bool chainChannel
```

true: Timer chained to previous timer; false: Timer not chained

lpit_timer_modes_t timerMode

Timers mode of operation.

lpit_trigger_select_t triggerSelect

Trigger selection for the timer

lpit_trigger_source_t triggerSource

Decides if we use external or internal trigger.

bool enableReloadOnTrigger

true: Timer reloads when a trigger is detected; false: No effect

bool enableStopOnTimeout

true: Timer will stop after timeout; false: does not stop after timeout

bool enableStartOnTrigger

true: Timer starts when a trigger is detected; false: decrement immediately

struct *__lpit_config*

#include <fsl_lpit.h> LPIT configuration structure.

This structure holds the configuration settings for the LPIT peripheral. To initialize this structure to reasonable defaults, call the LPIT_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

Public Members

bool enableRunInDebug

true: Timers run in debug mode; false: Timers stop in debug mode

bool enableRunInDoze

true: Timers run in doze mode; false: Timers stop in doze mode

2.44 LPSPI: Low Power Serial Peripheral Interface

2.45 LPSPI Peripheral driver

void LPSPI_MasterInit(LPSPI_Type *base, const *lpspi_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Initializes the LPSPI master.

Parameters

- base – LPSPI peripheral address.
- masterConfig – Pointer to structure *lpspi_master_config_t*.
- srcClock_Hz – Module source input clock in Hertz

void LPSPI_MasterGetDefaultConfig(*lpspi_master_config_t* *masterConfig)

Sets the *lpspi_master_config_t* structure to default values.

This API initializes the configuration structure for LPSPI_MasterInit(). The initialized structure can remain unchanged in LPSPI_MasterInit(), or can be modified before calling the LPSPI_MasterInit(). Example:

```
lpspi_master_config_t masterConfig;  
LPSPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – pointer to lpspi_master_config_t structure

void LPSPI_SlaveInit(LPSPI_Type *base, const *lpspi_slave_config_t* *slaveConfig)
LPSPI slave configuration.

Parameters

- base – LPSPI peripheral address.
- slaveConfig – Pointer to a structure lpspi_slave_config_t.

void LPSPI_SlaveGetDefaultConfig(*lpspi_slave_config_t* *slaveConfig)
Sets the lpspi_slave_config_t structure to default values.

This API initializes the configuration structure for LPSPI_SlaveInit(). The initialized structure can remain unchanged in LPSPI_SlaveInit() or can be modified before calling the LPSPI_SlaveInit(). Example:

```
lpspi_slave_config_t slaveConfig;  
LPSPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – pointer to lpspi_slave_config_t structure.

void LPSPI_Deinit(LPSPI_Type *base)
De-initializes the LPSPI peripheral. Call this API to disable the LPSPI clock.

Parameters

- base – LPSPI peripheral address.

void LPSPI_Reset(LPSPI_Type *base)
Restores the LPSPI peripheral to reset state. Note that this function sets all registers to reset state. As a result, the LPSPI module can't work after calling this API.

Parameters

- base – LPSPI peripheral address.

uint32_t LPSPI_GetInstance(LPSPI_Type *base)
Get the LPSPI instance from peripheral base address.

Parameters

- base – LPSPI peripheral base address.

Returns

LPSPI instance.

static inline void LPSPI_Enable(LPSPI_Type *base, bool enable)
Enables the LPSPI peripheral and sets the MCR MDIS to 0.

Parameters

- base – LPSPI peripheral address.
- enable – Pass true to enable module, false to disable module.

```
static inline uint32_t LPSPI_GetStatusFlags(LPSPI_Type *base)
```

Gets the LPSPI status flag state.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI status(in SR register).

```
static inline uint8_t LPSPI_GetTxFifoSize(LPSPI_Type *base)
```

Gets the LPSPI Tx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Tx FIFO size.

```
static inline uint8_t LPSPI_GetRxFifoSize(LPSPI_Type *base)
```

Gets the LPSPI Rx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Rx FIFO size.

```
static inline uint32_t LPSPI_GetTxFifoCount(LPSPI_Type *base)
```

Gets the LPSPI Tx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns

The number of words in the transmit FIFO.

```
static inline uint32_t LPSPI_GetRxFifoCount(LPSPI_Type *base)
```

Gets the LPSPI Rx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns

The number of words in the receive FIFO.

```
static inline void LPSPI_ClearStatusFlags(LPSPI_Type *base, uint32_t statusFlags)
```

Clears the LPSPI status flag.

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status flag bit to clear. The list of status flags is defined in the `_lpspi_flags`. Example usage:

```
LPSPI_ClearStatusFlags(base, kLPSPI_TxDataRequestFlag|kLPSPI_RxDataReadyFlag);
```

Parameters

- base – LPSPI peripheral address.
- statusFlags – The status flag used from type `_lpspi_flags`.

```
static inline uint32_t LPSPI_GetTcr(LPSPI_Type *base)
```

```
static inline void LPSPI_EnableInterrupts(LPSPI_Type *base, uint32_t mask)
```

Enables the LPSPI interrupts.

This function configures the various interrupt masks of the LPSPI. The parameters are base and an interrupt mask. Note that, for Tx fill and Rx FIFO drain requests, enabling the interrupt request disables the DMA request.

```
LPSPI_EnableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_interrupt_enable`.

```
static inline void LPSPI_DisableInterrupts(LPSPI_Type *base, uint32_t mask)
```

Disables the LPSPI interrupts.

```
LPSPI_DisableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_interrupt_enable`.

```
static inline void LPSPI_EnableDMA(LPSPI_Type *base, uint32_t mask)
```

Enables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
LPSPI_EnableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_dma_enable`.

```
static inline void LPSPI_DisableDMA(LPSPI_Type *base, uint32_t mask)
```

Disables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
SPI_DisableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_dma_enable`.

```
static inline uint32_t LPSPI_GetTxRegisterAddress(LPSPI_Type *base)
```

Gets the LPSPI Transmit Data Register address for a DMA operation.

This function gets the LPSPI Transmit Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Transmit Data Register address.

```
static inline uint32_t LPSPI_GetRxRegisterAddress(LPSPI_Type *base)
```

Gets the LPSPI Receive Data Register address for a DMA operation.

This function gets the LPSPI Receive Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Receive Data Register address.

```
bool LPSPI_CheckTransferArgument(LPSPI_Type *base, lpspi_transfer_t *transfer, bool isEdma)
```

Check the argument for transfer .

Parameters

- base – LPSPI peripheral address.
- transfer – the transfer struct to be used.
- isEdma – True to check for EDMA transfer, false to check interrupt non-blocking transfer

Returns

Return true for right and false for wrong.

```
static inline void LPSPI_SetMasterSlaveMode(LPSPI_Type *base, lpspi_master_slave_mode_t mode)
```

Configures the LPSPI for either master or slave.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

Parameters

- base – LPSPI peripheral address.
- mode – Mode setting (master or slave) of type lpspi_master_slave_mode_t.

```
static inline void LPSPI_SelectTransferPCS(LPSPI_Type *base, lpspi_which_pcs_t select)
```

Configures the peripheral chip select used for the transfer.

Parameters

- base – LPSPI peripheral address.
- select – LPSPI Peripheral Chip Select (PCS) configuration.

```
static inline void LPSPI_SetPCSContinuous(LPSPI_Type *base, bool IsContinuous)
```

Set the PCS signal to continuous or uncontinuous mode.

Note: In master mode, continuous transfer will keep the PCS asserted at the end of the frame size, until a command word is received that starts a new frame. So PCS must be set back to uncontinuous when transfer finishes. In slave mode, when continuous transfer is enabled, the LPSPI will only transmit the first frame size bits, after that the LPSPI will transmit received data back (assuming a 32-bit shift register).

Parameters

- base – LPSPI peripheral address.
- IsContinuous – True to set the transfer PCS to continuous mode, false to set to uncontinuous mode.

```
static inline bool LPSPI_IsMaster(LPSPI_Type *base)
```

Returns whether the LPSPI module is in master mode.

Parameters

- base – LPSPI peripheral address.

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

```
static inline void LPSPI_FlushFifo(LPSPI_Type *base, bool flushTxFifo, bool flushRxFifo)
```

Flushes the LPSPI FIFOs.

Parameters

- base – LPSPI peripheral address.
- flushTxFifo – Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO.
- flushRxFifo – Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO.

```
static inline void LPSPI_SetFifoWatermarks(LPSPI_Type *base, uint32_t txWater, uint32_t rxWater)
```

Sets the transmit and receive FIFO watermark values.

This function allows the user to set the receive and transmit FIFO watermarks. The function does not compare the watermark settings to the FIFO size. The FIFO watermark should not be equal to or greater than the FIFO size. It is up to the higher level driver to make this check.

Parameters

- base – LPSPI peripheral address.
- txWater – The TX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.
- rxWater – The RX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.

```
static inline void LPSPI_SetAllPcsPolarity(LPSPI_Type *base, uint32_t mask)
```

Configures all LPSPI peripheral chip select polarities simultaneously.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

This is an example: PCS0 and PCS1 set to active low and other PCSs set to active high. Note that the number of PCS is device-specific.

```
LPSPI_SetAllPcsPolarity(base, kLPSPI_Pcs0ActiveLow | kLPSPI_Pcs1ActiveLow);
```

Parameters

- base – LPSPI peripheral address.
- mask – The PCS polarity mask; Use the enum `_lpspi_pcs_polarity`.

```
static inline void LPSPI_SetFrameSize(LPSPI_Type *base, uint32_t frameSize)
```

Configures the frame size.

The minimum frame size is 8-bits and the maximum frame size is 4096-bits. If the frame size is less than or equal to 32-bits, the word size and frame size are identical. If the frame size is greater than 32-bits, the word size is 32-bits for each word except the last (the last word contains the remainder bits if the frame size is not divisible by 32). The minimum word size is 2-bits. A frame size of 33-bits (or similar) is not supported.

Note 1: The transmit command register should be initialized before enabling the LPSPI in slave mode, although the command register does not update until after the LPSPI is enabled. After it is enabled, the transmit command register should only be changed if the LPSPI is idle.

Note 2: The transmit and command FIFO is a combined FIFO that includes both transmit data and command words. That means the TCR register should be written to when the Tx FIFO is not full.

Parameters

- base – LPSPI peripheral address.
- frameSize – The frame size in number of bits.

```
uint32_t LPSPI_MasterSetBaudRate(LPSPI_Type *base, uint32_t baudRate_Bps, uint32_t
                                srcClock_Hz, uint32_t *tcrPrescaleValue)
```

Sets the LPSPI baud rate in bits per second.

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate and returns the calculated baud rate in bits-per-second. It requires the caller to provide the frequency of the module source clock (in Hertz). Note that the baud rate does not go into effect until the Transmit Control Register (TCR) is programmed with the prescale value. Hence, this function returns the prescale tcrPrescaleValue parameter for later programming in the TCR. The higher level peripheral driver should alert the user of an out of range baud rate input.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- baudRate_Bps – The desired baud rate in bits per second.
- srcClock_Hz – Module source input clock in Hertz.
- tcrPrescaleValue – The TCR prescale value needed to program the TCR.

Returns

The actual calculated baud rate. This function may also return a “0” if the LPSPI is not configured for master mode or if the LPSPI module is not disabled.

```
void LPSPI_MasterSetDelayScaler(LPSPI_Type *base, uint32_t scaler, lpspi_delay_type_t
                                whichDelay)
```

Manually configures a specific LPSPI delay parameter (module must be disabled to change the delay values).

This function configures the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type `lpspi_delay_type_t`.

The user passes the desired delay along with the delay value. This allows the user to directly set the delay values if they have pre-calculated them or if they simply wish to manually increment the value.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- scaler – The 8-bit delay value 0x00 to 0xFF (255).

- `whichDelay` – The desired delay to configure, must be of type `lpspi_delay_type_t`.

`uint32_t LPSPI_MasterSetDelayTimes(LPSPI_Type *base, uint32_t delayTimeInNanoSec, lpspi_delay_type_t whichDelay, uint32_t srcClock_Hz)`

Calculates the delay based on the desired delay input in nanoseconds (module must be disabled to change the delay values).

This function calculates the values for the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type `lpspi_delay_type_t`.

The user passes the desired delay and the desired delay value in nano-seconds. The function calculates the value needed for the desired delay parameter and returns the actual calculated delay because an exact delay match may not be possible. In this case, the closest match is calculated without going below the desired delay value input. It is possible to input a very large delay value that exceeds the capability of the part, in which case the maximum supported delay is returned. It is up to the higher level peripheral driver to alert the user of an out of range delay input.

Note that the LPSPI module must be configured for master mode before configuring this. And note that the `delayTime = LPSPI_clockSource / (PRESCALE * Delay_scaler)`.

Parameters

- `base` – LPSPI peripheral address.
- `delayTimeInNanoSec` – The desired delay value in nano-seconds.
- `whichDelay` – The desired delay to configuration, which must be of type `lpspi_delay_type_t`.
- `srcClock_Hz` – Module source input clock in Hertz.

Returns

actual Calculated delay value in nano-seconds.

`static inline void LPSPI_WriteData(LPSPI_Type *base, uint32_t data)`

Writes data into the transmit data buffer.

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user has to manage sending the data one 32-bit word at a time. Any writes to the TDR result in an immediate push to the transmit FIFO. This function can be used for either master or slave modes.

Parameters

- `base` – LPSPI peripheral address.
- `data` – The data word to be sent.

`static inline uint32_t LPSPI_ReadData(LPSPI_Type *base)`

Reads data from the data buffer.

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

Parameters

- `base` – LPSPI peripheral address.

Returns

The data read from the data buffer.


```
void LPSPI_SetDummyData(LPSPI_Type *base, uint8_t dummyData)
```

Set up the dummy data.

Parameters

- `base` – LPSPI peripheral address.
- `dummyData` – Data to be transferred when tx buffer is NULL. Note: This API has no effect when LPSPI in slave interrupt mode, because driver will set the TXMSK bit to 1 if txData is NULL, no data is loaded from transmit FIFO and output pin is tristated.

```
void LPSPI_MasterTransferCreateHandle(LPSPI_Type *base, lpspi_master_handle_t *handle,
                                     lpspi_master_transfer_callback_t callback, void
                                     *userData)
```

Initializes the LPSPI master handle.

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

Parameters

- `base` – LPSPI peripheral address.
- `handle` – LPSPI handle pointer to `lpspi_master_handle_t`.
- `callback` – DSPI callback.
- `userData` – callback function parameter.

```
status_t LPSPI_MasterTransferBlocking(LPSPI_Type *base, lpspi_transfer_t *transfer)
```

LPSPI master transfer data using a polling method.

This function transfers data using a polling method. This is a blocking function, which does not return until all transfers have been completed.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- `base` – LPSPI peripheral address.
- `transfer` – pointer to `lpspi_transfer_t` structure.

Returns

status of `status_t`.

```
status_t LPSPI_MasterTransferNonBlocking(LPSPI_Type *base, lpspi_master_handle_t *handle,
                                         lpspi_transfer_t *transfer)
```

LPSPI master transfer data using an interrupt method.

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- `base` – LPSPI peripheral address.
- `handle` – pointer to `lpspi_master_handle_t` structure which stores the transfer state.

- transfer – pointer to `lpspi_transfer_t` structure.

Returns

status of `status_t`.

`status_t` LPSPI_MasterTransferGetCount(LPSPI_Type *base, *lpspi_master_handle_t* *handle, size_t *count)

Gets the master transfer remaining bytes.

This function gets the master transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_master_handle_t` structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns

status of `status_t`.

void LPSPI_MasterTransferAbort(LPSPI_Type *base, *lpspi_master_handle_t* *handle)

LPSPI master abort transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_master_handle_t` structure which stores the transfer state.

void LPSPI_MasterTransferHandleIRQ(LPSPI_Type *base, *lpspi_master_handle_t* *handle)

LPSPI Master IRQ handler function.

This function processes the LPSPI transmit and receive IRQ.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_master_handle_t` structure which stores the transfer state.

void LPSPI_SlaveTransferCreateHandle(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_slave_transfer_callback_t* callback, void *userData)

Initializes the LPSPI slave handle.

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

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to `lpspi_slave_handle_t`.
- callback – DSPI callback.
- userData – callback function parameter.

`status_t` LPSPI_SlaveTransferNonBlocking(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI slave transfer data using an interrupt method.

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.
- transfer – pointer to `lpspi_transfer_t` structure.

Returns

status of `status_t`.

`status_t` LPSPI_SlaveTransferGetCount(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, size_t *count)

Gets the slave transfer remaining bytes.

This function gets the slave transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns

status of `status_t`.

void LPSPI_SlaveTransferAbort(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI slave aborts a transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.

void LPSPI_SlaveTransferHandleIRQ(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI Slave IRQ handler function.

This function processes the LPSPI transmit and receives an IRQ.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.

bool LPSPI_WaitTxFifoEmpty(LPSPI_Type *base)

Wait for tx FIFO to be empty.

This function wait the tx fifo empty

Parameters

- base – LPSPI peripheral address.

Returns

true for the tx FIFO is ready, false is not.

void LPSPI_DriverIRQHandler(uint32_t instance)

LPSPI driver IRQ handler common entry.

This function provides the common IRQ request entry for LPSPI.

Parameters

- instance – LPSPI instance.

FSL_LPSPI_DRIVER_VERSION

LPSPI driver version.

Status for the LPSPI driver.

Values:

enumerator kStatus_LPSPI_Busy

LPSPI transfer is busy.

enumerator kStatus_LPSPI_Error

LPSPI driver error.

enumerator kStatus_LPSPI_Idle

LPSPI is idle.

enumerator kStatus_LPSPI_OutOfRange

LPSPI transfer out Of range.

enumerator kStatus_LPSPI_Timeout

LPSPI timeout polling status flags.

enum _lpspi_flags

LPSPI status flags in SPIx_SR register.

Values:

enumerator kLPSPI_TxDataRequestFlag

Transmit data flag

enumerator kLPSPI_RxDataReadyFlag

Receive data flag

enumerator kLPSPI_WordCompleteFlag

Word Complete flag

enumerator kLPSPI_FrameCompleteFlag

Frame Complete flag

enumerator kLPSPI_TransferCompleteFlag

Transfer Complete flag

enumerator kLPSPI_TransmitErrorFlag

Transmit Error flag (FIFO underrun)

enumerator kLPSPI_ReceiveErrorFlag

Receive Error flag (FIFO overrun)

enumerator kLPSPI_DataMatchFlag

Data Match flag

enumerator kLPSPI_ModuleBusyFlag

Module Busy flag

enumerator kLPSPI_AllStatusFlag

Used for clearing all w1c status flags

enum _lpspi_interrupt_enable

LPSPI interrupt source.

Values:

enumerator kLPSPI_TxInterruptEnable

Transmit data interrupt enable

enumerator kLPSPI_RxInterruptEnable

Receive data interrupt enable

enumerator kLPSPI_WordCompleteInterruptEnable

Word complete interrupt enable

enumerator kLPSPI_FrameCompleteInterruptEnable

Frame complete interrupt enable

enumerator kLPSPI_TransferCompleteInterruptEnable

Transfer complete interrupt enable

enumerator kLPSPI_TransmitErrorInterruptEnable

Transmit error interrupt enable(FIFO underrun)

enumerator kLPSPI_ReceiveErrorInterruptEnable

Receive Error interrupt enable (FIFO overrun)

enumerator kLPSPI_DataMatchInterruptEnable

Data Match interrupt enable

enumerator kLPSPI_AllInterruptEnable

All above interrupts enable.

enum _lpspi_dma_enable

LPSPI DMA source.

Values:

enumerator kLPSPI_TxDmaEnable

Transmit data DMA enable

enumerator kLPSPI_RxDmaEnable

Receive data DMA enable

enum _lpspi_master_slave_mode

LPSPI master or slave mode configuration.

Values:

enumerator kLPSPI_Master

LPSPI peripheral operates in master mode.

enumerator kLPSPI_Slave

LPSPI peripheral operates in slave mode.

enum _lpspi_which_pcs_config

LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

Values:

enumerator kLPSPI_Pcs0

PCS[0]

enumerator kLPSPi_Pcs1
PCS[1]

enumerator kLPSPi_Pcs2
PCS[2]

enumerator kLPSPi_Pcs3
PCS[3]

enum __lpspi_pcs_polarity_config
LPSPi Peripheral Chip Select (PCS) Polarity configuration.
Values:

enumerator kLPSPi_PcsActiveHigh
PCS Active High (idles low)

enumerator kLPSPi_PcsActiveLow
PCS Active Low (idles high)

enum __lpspi_pcs_polarity
LPSPi Peripheral Chip Select (PCS) Polarity.
Values:

enumerator kLPSPi_Pcs0ActiveLow
Pcs0 Active Low (idles high).

enumerator kLPSPi_Pcs1ActiveLow
Pcs1 Active Low (idles high).

enumerator kLPSPi_Pcs2ActiveLow
Pcs2 Active Low (idles high).

enumerator kLPSPi_Pcs3ActiveLow
Pcs3 Active Low (idles high).

enumerator kLPSPi_PcsAllActiveLow
Pcs0 to Pcs5 Active Low (idles high).

enum __lpspi_clock_polarity
LPSPi clock polarity configuration.
Values:

enumerator kLPSPi_ClockPolarityActiveHigh
CPOL=0. Active-high LPSPi clock (idles low)

enumerator kLPSPi_ClockPolarityActiveLow
CPOL=1. Active-low LPSPi clock (idles high)

enum __lpspi_clock_phase
LPSPi clock phase configuration.
Values:

enumerator kLPSPi_ClockPhaseFirstEdge
CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.

enumerator kLPSPi_ClockPhaseSecondEdge
CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

enum `_lpspi_shift_direction`

LPSPi data shifter direction options.

Values:

enumerator `kLPSPI_MsbFirst`

Data transfers start with most significant bit.

enumerator `kLPSPI_LsbFirst`

Data transfers start with least significant bit.

enum `_lpspi_host_request_select`

LPSPi Host Request select configuration.

Values:

enumerator `kLPSPI_HostReqExtPin`

Host Request is an ext pin.

enumerator `kLPSPI_HostReqInternalTrigger`

Host Request is an internal trigger.

enum `_lpspi_match_config`

LPSPi Match configuration options.

Values:

enumerator `kLPSI_MatchDisabled`

LPSPi Match Disabled.

enumerator `kLPSI_1stWordEqualsM0orM1`

LPSPi Match Enabled.

enumerator `kLPSI_AnyWordEqualsM0orM1`

LPSPi Match Enabled.

enumerator `kLPSI_1stWordEqualsM0and2ndWordEqualsM1`

LPSPi Match Enabled.

enumerator `kLPSI_AnyWordEqualsM0andNxtWordEqualsM1`

LPSPi Match Enabled.

enumerator `kLPSI_1stWordAndM1EqualsM0andM1`

LPSPi Match Enabled.

enumerator `kLPSI_AnyWordAndM1EqualsM0andM1`

LPSPi Match Enabled.

enum `_lpspi_pin_config`

LPSPi pin (SDO and SDI) configuration.

Values:

enumerator `kLPSPI_SdiInSdoOut`

LPSPi SDI input, SDO output.

enumerator `kLPSPI_SdiInSdiOut`

LPSPi SDI input, SDI output.

enumerator `kLPSPI_SdoInSdoOut`

LPSPi SDO input, SDO output.

enumerator `kLPSPI_SdoInSdiOut`

LPSPi SDO input, SDI output.

enum `_lpspi_data_out_config`

LPSPi data output configuration.

Values:

enumerator `kLpspiDataOutRetained`

Data out retains last value when chip select is de-asserted

enumerator `kLpspiDataOutTristate`

Data out is tristated when chip select is de-asserted

enum `_lpspi_transfer_width`

LPSPi transfer width configuration.

Values:

enumerator `kLPSPi_SingleBitXfer`

1-bit shift at a time, data out on SDO, in on SDI (normal mode)

enumerator `kLPSPi_TwoBitXfer`

2-bits shift out on SDO/SDI and in on SDO/SDI

enumerator `kLPSPi_FourBitXfer`

4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

enum `_lpspi_delay_type`

LPSPi delay type selection.

Values:

enumerator `kLPSPi_PcsToSck`

PCS-to-SCK delay.

enumerator `kLPSPi_LastSckToPcs`

Last SCK edge to PCS delay.

enumerator `kLPSPi_BetweenTransfer`

Delay between transfers.

enum `_lpspi_transfer_config_flag_for_master`

Use this enumeration for LPSPi master transfer configFlags.

Values:

enumerator `kLPSPi_MasterPcs0`

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS0 signal

enumerator `kLPSPi_MasterPcs1`

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS1 signal

enumerator `kLPSPi_MasterPcs2`

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS2 signal

enumerator `kLPSPi_MasterPcs3`

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS3 signal

enumerator `kLPSPi_MasterPcsContinuous`

Is PCS signal continuous

enumerator `kLPSPi_MasterByteSwap`

Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set `lpspi_shift_direction_t` to MSB).

- i. If you set `bitPerFrame = 8` , no matter the `kLPSPi_MasterByteSwap` you flag is used or not, the waveform is 1 2 3 4 5 6 7 8.

- ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPi_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi_MasterByteSwap flag.
- iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPi_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi_MasterByteSwap flag.

enum _lpspi_transfer_config_flag_for_slave

Use this enumeration for LPSPi slave transfer configFlags.

Values:

enumerator kLPSPi_SlavePcs0

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS0 signal

enumerator kLPSPi_SlavePcs1

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS1 signal

enumerator kLPSPi_SlavePcs2

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS2 signal

enumerator kLPSPi_SlavePcs3

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS3 signal

enumerator kLPSPi_SlaveByteSwap

Is slave swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

- i. If you set bitPerFrame = 8 , no matter the kLPSPi_SlaveByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
- ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPi_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi_SlaveByteSwap flag.
- iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPi_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi_SlaveByteSwap flag.

enum _lpspi_transfer_state

LPSPi transfer state, which is used for LPSPi transactional API state machine.

Values:

enumerator kLPSPi_Idle

Nothing in the transmitter/receiver.

enumerator kLPSPi_Busy

Transfer queue is not finished.

enumerator kLPSPi_Error

Transfer error.

typedef enum _lpspi_master_slave_mode lpspi_master_slave_mode_t

LPSPi master or slave mode configuration.

typedef enum _lpspi_which_pcs_config lpspi_which_pcs_t

LPSPi Peripheral Chip Select (PCS) configuration (which PCS to configure).

typedef enum _lpspi_pcs_polarity_config lpspi_pcs_polarity_config_t

LPSPi Peripheral Chip Select (PCS) Polarity configuration.

typedef enum *_lpspi_clock_polarity* lpspi_clock_polarity_t

LPSPI clock polarity configuration.

typedef enum *_lpspi_clock_phase* lpspi_clock_phase_t

LPSPI clock phase configuration.

typedef enum *_lpspi_shift_direction* lpspi_shift_direction_t

LPSPI data shifter direction options.

typedef enum *_lpspi_host_request_select* lpspi_host_request_select_t

LPSPI Host Request select configuration.

typedef enum *_lpspi_match_config* lpspi_match_config_t

LPSPI Match configuration options.

typedef enum *_lpspi_pin_config* lpspi_pin_config_t

LPSPI pin (SDO and SDI) configuration.

typedef enum *_lpspi_data_out_config* lpspi_data_out_config_t

LPSPI data output configuration.

typedef enum *_lpspi_transfer_width* lpspi_transfer_width_t

LPSPI transfer width configuration.

typedef enum *_lpspi_delay_type* lpspi_delay_type_t

LPSPI delay type selection.

typedef struct *_lpspi_master_config* lpspi_master_config_t

LPSPI master configuration structure.

typedef struct *_lpspi_slave_config* lpspi_slave_config_t

LPSPI slave configuration structure.

typedef struct *_lpspi_master_handle* lpspi_master_handle_t

Forward declaration of the *_lpspi_master_handle* typedefs.

typedef struct *_lpspi_slave_handle* lpspi_slave_handle_t

Forward declaration of the *_lpspi_slave_handle* typedefs.

typedef void (*lpspi_master_transfer_callback_t)(LP_SPI_Type *base, *lpspi_master_handle_t* *handle, *status_t* status, void *userData)

Master completion callback function pointer type.

Param base

LPSPI peripheral address.

Param handle

Pointer to the handle for the LPSPI master.

Param status

Success or error code describing whether the transfer is completed.

Param userData

Arbitrary pointer-dataSized value passed from the application.

typedef void (*lpspi_slave_transfer_callback_t)(LP_SPI_Type *base, *lpspi_slave_handle_t* *handle, *status_t* status, void *userData)

Slave completion callback function pointer type.

Param base

LPSPI peripheral address.

Param handle

Pointer to the handle for the LPSPI slave.

Param status

Success or error code describing whether the transfer is completed.

Param userData

Arbitrary pointer-dataSized value passed from the application.

```
typedef struct _lpspi_transfer lpspi_transfer_t
```

LPSPI master/slave transfer structure.

```
volatile uint8_t g_lpspiDummyData[]
```

Global variable for dummy data value setting.

```
LPSPI_DUMMY_DATA
```

LPSPI dummy data if no Tx data.

Dummy data used for tx if there is not txData.

```
SPI_RETRY_TIMES
```

Retry times for waiting flag.

```
LPSPI_MASTER_PCS_SHIFT
```

LPSPI master PCS shift macro , internal used.

```
LPSPI_MASTER_PCS_MASK
```

LPSPI master PCS shift macro , internal used.

```
LPSPI_SLAVE_PCS_SHIFT
```

LPSPI slave PCS shift macro , internal used.

```
LPSPI_SLAVE_PCS_MASK
```

LPSPI slave PCS shift macro , internal used.

```
struct _lpspi_master_config
```

#include <fsl_lpspi.h> LPSPI master configuration structure.

Public Members

```
uint32_t baudRate
```

Baud Rate for LPSPI.

```
uint32_t bitsPerFrame
```

Bits per frame, minimum 8, maximum 4096.

```
lpspi_clock_polarity_t cpol
```

Clock polarity.

```
lpspi_clock_phase_t cpha
```

Clock phase.

```
lpspi_shift_direction_t direction
```

MSB or LSB data shift direction.

```
uint32_t pcsToSckDelayInNanoSec
```

PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

```
uint32_t lastSckToPcsDelayInNanoSec
```

Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32_t betweenTransferDelayInNanoSec

After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

lpspi_which_pcs_t whichPcs

Desired Peripheral Chip Select (PCS).

lpspi_pcs_polarity_config_t pcsActiveHighOrLow

Desired PCS active high or low

lpspi_pin_config_t pinCfg

Configures which pins are used for input and output data during single bit transfers.

lpspi_data_out_config_t dataOutConfig

Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

bool enableInputDelay

Enable master to sample the input data on a delayed SCK. This can help improve slave setup time. Refer to device data sheet for specific time length.

struct __lpspi_slave_config

#include <fsl_lpspi.h> LPSPI slave configuration structure.

Public Members

uint32_t bitsPerFrame

Bits per frame, minimum 8, maximum 4096.

lpspi_clock_polarity_t cpol

Clock polarity.

lpspi_clock_phase_t cpha

Clock phase.

lpspi_shift_direction_t direction

MSB or LSB data shift direction.

lpspi_which_pcs_t whichPcs

Desired Peripheral Chip Select (pcs)

lpspi_pcs_polarity_config_t pcsActiveHighOrLow

Desired PCS active high or low

lpspi_pin_config_t pinCfg

Configures which pins are used for input and output data during single bit transfers.

lpspi_data_out_config_t dataOutConfig

Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

struct __lpspi_transfer

#include <fsl_lpspi.h> LPSPI master/slave transfer structure.

Public Members

const uint8_t *txData

Send buffer.

uint8_t *rxData

Receive buffer.

volatile size_t dataSize

Transfer bytes.

uint32_t configFlags

Transfer transfer configuration flags. Set from `_lpspi_transfer_config_flag_for_master` if the transfer is used for master or `_lpspi_transfer_config_flag_for_slave` enumeration if the transfer is used for slave.

struct `_lpspi_master_handle`

#include <fsl_lpspi.h> LPSPI master transfer handle structure used for transactional API.

Public Members

volatile bool isPcsContinuous

Is PCS continuous in transfer.

volatile bool writeTcrInIsr

A flag that whether should write TCR in ISR.

volatile bool isByteSwap

A flag that whether should byte swap.

volatile bool isTxMask

A flag that whether TCR[TXMSK] is set.

volatile uint16_t bytesPerFrame

Number of bytes in each frame

volatile uint16_t frameSize

Backup of TCR[FRAMESZ]

volatile uint8_t fifoSize

FIFO dataSize.

volatile uint8_t rxWatermark

Rx watermark.

volatile uint8_t bytesEachWrite

Bytes for each write TDR.

volatile uint8_t bytesEachRead

Bytes for each read RDR.

const uint8_t *volatile txData

Send buffer.

uint8_t *volatile rxData

Receive buffer.

volatile size_t txRemainingByteCount

Number of bytes remaining to send.

volatile size_t rxRemainingByteCount

Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes

Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes

Read RDR register remaining times.

uint32_t totalByteCount

Number of transfer bytes

uint32_t txBuffIfNull

Used if the txData is NULL.

volatile uint8_t state

LPSPi transfer state , _lpspi_transfer_state.

lpspi_master_transfer_callback_t callback

Completion callback.

void *userData

Callback user data.

struct _lpspi_slave_handle

#include <fsl_lpspi.h> LPSPi slave transfer handle structure used for transactional API.

Public Members

volatile bool isByteSwap

A flag that whether should byte swap.

volatile uint8_t fifoSize

FIFO dataSize.

volatile uint8_t rxWatermark

Rx watermark.

volatile uint8_t bytesEachWrite

Bytes for each write TDR.

volatile uint8_t bytesEachRead

Bytes for each read RDR.

const uint8_t *volatile txData

Send buffer.

uint8_t *volatile rxData

Receive buffer.

volatile size_t txRemainingByteCount

Number of bytes remaining to send.

volatile size_t rxRemainingByteCount

Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes

Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes

Read RDR register remaining times.

uint32_t totalByteCount

Number of transfer bytes

volatile uint8_t state

LPSPi transfer state , _lpspi_transfer_state.

volatile uint32_t errorCount

Error count for slave transfer.

lpspi_slave_transfer_callback_t callback

Completion callback.

void *userData

Callback user data.

2.46 LPSPI eDMA Driver

FSL_LPSPI_EDMA_DRIVER_VERSION

LPSPI EDMA driver version.

DMA_MAX_TRANSFER_COUNT

DMA max transfer size.

typedef struct *lpspi_master_edma_handle* lpspi_master_edma_handle_t

Forward declaration of the *lpspi_master_edma_handle* typedefs.

typedef struct *lpspi_slave_edma_handle* lpspi_slave_edma_handle_t

Forward declaration of the *lpspi_slave_edma_handle* typedefs.

typedef void (*lpspi_master_edma_transfer_callback_t)(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *status_t* status, void *userData)

Completion callback function pointer type.

Param base

LPSPI peripheral base address.

Param handle

Pointer to the handle for the LPSPI master.

Param status

Success or error code describing whether the transfer completed.

Param userData

Arbitrary pointer-dataSized value passed from the application.

typedef void (*lpspi_slave_edma_transfer_callback_t)(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *status_t* status, void *userData)

Completion callback function pointer type.

Param base

LPSPI peripheral base address.

Param handle

Pointer to the handle for the LPSPI slave.

Param status

Success or error code describing whether the transfer completed.

Param userData

Arbitrary pointer-dataSized value passed from the application.

void LPSPI_MasterTransferCreateHandleEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_master_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *edmaRxRegToRxDataHandle, *edma_handle_t* *edmaTxDataToTxRegHandle)

Initializes the LPSPI master eDMA handle.

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

Note that the LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx are the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for `edmaRxRegToRxDataHandle` and Tx DMAMUX source for `edmaTxDataToTxRegHandle`. (2) For a shared DMA request source, enable and set the Rx/Tx DMAMUX source for `edmaRxRegToRxDataHandle`.

Parameters

- `base` – LPSPI peripheral base address.
- `handle` – LPSPI handle pointer to `lpspi_master_edma_handle_t`.
- `callback` – LPSPI callback.
- `userData` – callback function parameter.
- `edmaRxRegToRxDataHandle` – `edmaRxRegToRxDataHandle` pointer to `edma_handle_t`.
- `edmaTxDataToTxRegHandle` – `edmaTxDataToTxRegHandle` pointer to `edma_handle_t`.

status_t LPSPI_MasterTransferEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of `bytesPerFrame` if `bytesPerFrame` is less than or equal to 4. For `bytesPerFrame` greater than 4: The transfer data size should be equal to `bytesPerFrame` if the `bytesPerFrame` is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of `bytesPerFrame`.

Parameters

- `base` – LPSPI peripheral base address.
- `handle` – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.
- `transfer` – pointer to `lpspi_transfer_t` structure.

Returns

status of `status_t`.

status_t LPSPI_MasterTransferPrepareEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, `uint32_t` configFlags)

LPSPI master config transfer parameter while using eDMA.

This function is preparing to transfer data using eDMA, work with `LPSPI_MasterTransferEDMALite`.

Parameters

- `base` – LPSPI peripheral base address.
- `handle` – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.
- `configFlags` – transfer configuration flags. `_lpspi_transfer_config_flag_for_master`.

Return values

- `kStatus_Success` – Execution successfully.
- `kStatus_LPSPI_Busy` – The LPSPI device is busy.

Returns

Indicates whether LPSPI master transfer was successful or not.

status_t LPSPI_MasterTransferEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using eDMA without configs.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: This API is only for transfer through DMA without configuration. Before calling this API, you must call LPSPI_MasterTransferPrepareEDMALite to configure it once. The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to *lpspi_master_edma_handle_t* structure which stores the transfer state.
- transfer – pointer to *lpspi_transfer_t* structure, config field is not used.

Return values

- kStatus_Success – Execution successfully.
- kStatus_LPSPI_Busy – The LPSPI device is busy.
- kStatus_InvalidArgument – The transfer structure is invalid.

Returns

Indicates whether LPSPI master transfer was successful or not.

void LPSPI_MasterTransferAbortEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle)

LPSPI master aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to *lpspi_master_edma_handle_t* structure which stores the transfer state.

status_t LPSPI_MasterTransferGetCountEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *size_t* *count)

Gets the master eDMA transfer remaining bytes.

This function gets the master eDMA transfer remaining bytes.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to *lpspi_master_edma_handle_t* structure which stores the transfer state.
- count – Number of bytes transferred so far by the EDMA transaction.

Returns

status of *status_t*.

```
void LPSPI_SlaveTransferCreateHandleEDMA(LPSPI_Type *base, lpspi_slave_edma_handle_t
                                         *handle, lpspi_slave_edma_transfer_callback_t
                                         callback, void *userData, edma_handle_t
                                         *edmaRxRegToRxDataHandle, edma_handle_t
                                         *edmaTxDataToTxRegHandle)
```

Initializes the LPSPI slave eDMA handle.

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

Note that LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx as the same source) DMA request source.

(1) For a separated DMA request source, enable and set the Rx DMAMUX source for *edmaRxRegToRxDataHandle* and Tx DMAMUX source for *edmaTxDataToTxRegHandle*. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for *edmaRxRegToRxDataHandle*.

Parameters

- *base* – LPSPI peripheral base address.
- *handle* – LPSPI handle pointer to *lpspi_slave_edma_handle_t*.
- *callback* – LPSPI callback.
- *userData* – callback function parameter.
- *edmaRxRegToRxDataHandle* – *edmaRxRegToRxDataHandle* pointer to *edma_handle_t*.
- *edmaTxDataToTxRegHandle* – *edmaTxDataToTxRegHandle* pointer to *edma_handle_t*.

```
status_t LPSPI_SlaveTransferEDMA(LPSPI_Type *base, lpspi_slave_edma_handle_t *handle,
                                  lpspi_transfer_t *transfer)
```

LPSPI slave transfers data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which return right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of *bytesPerFrame* if *bytesPerFrame* is less than or equal to 4. For *bytesPerFrame* greater than 4: The transfer data size should be equal to *bytesPerFrame* if the *bytesPerFrame* is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of *bytesPerFrame*.

Parameters

- *base* – LPSPI peripheral base address.
- *handle* – pointer to *lpspi_slave_edma_handle_t* structure which stores the transfer state.
- *transfer* – pointer to *lpspi_transfer_t* structure.

Returns

status of *status_t*.

```
void LPSPI_SlaveTransferAbortEDMA(LPSPI_Type *base, lpspi_slave_edma_handle_t *handle)
```

LPSPI slave aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- *base* – LPSPI peripheral base address.

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

`status_t` LPSPI_SlaveTransferGetCountEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, `size_t` *count)

Gets the slave eDMA transfer remaining bytes.

This function gets the slave eDMA transfer remaining bytes.

Parameters

- `base` – LPSPI peripheral base address.
- `handle` – pointer to `lpspi_slave_edma_handle_t` structure which stores the transfer state.
- `count` – Number of bytes transferred so far by the eDMA transaction.

Returns

status of `status_t`.

`struct _lpspi_master_edma_handle`

#include <fsl_lpspi_edma.h> LPSPI master eDMA transfer handle structure used for transactional API.

Public Members

`volatile bool` `isPcsContinuous`

Is PCS continuous in transfer.

`volatile bool` `isByteSwap`

A flag that whether should byte swap.

`volatile uint8_t` `fifoSize`

FIFO dataSize.

`volatile uint8_t` `rxWatermark`

Rx watermark.

`volatile uint8_t` `bytesEachWrite`

Bytes for each write TDR.

`volatile uint8_t` `bytesEachRead`

Bytes for each read RDR.

`volatile uint8_t` `bytesLastRead`

Bytes for last read RDR.

`volatile bool` `isThereExtraRxBytes`

Is there extra RX byte.

`const uint8_t *volatile` `txData`

Send buffer.

`uint8_t *volatile` `rxData`

Receive buffer.

`volatile size_t` `txRemainingByteCount`

Number of bytes remaining to send.

`volatile size_t` `rxRemainingByteCount`

Number of bytes remaining to receive.

`volatile uint32_t writeRegRemainingTimes`
Write TDR register remaining times.

`volatile uint32_t readRegRemainingTimes`
Read RDR register remaining times.

`uint32_t totalByteCount`
Number of transfer bytes

`edma_tcd_t *lastTimeTCD`
Pointer to the lastTime TCD

`bool isMultiDMATransmit`
Is there multi DMA transmit

`volatile uint8_t dmaTransmitTime`
DMA Transfer times.

`uint32_t lastTimeDataBytes`
DMA transmit last Time data Bytes

`uint32_t dataBytesEveryTime`
Bytes in a time for DMA transfer, default is DMA_MAX_TRANSFER_COUNT

`edma_transfer_config_t transferConfigRx`
Config of DMA rx channel.

`edma_transfer_config_t transferConfigTx`
Config of DMA tx channel.

`uint32_t txBuffIfNull`
Used if there is not txData for DMA purpose.

`uint32_t rxBuffIfNull`
Used if there is not rxData for DMA purpose.

`uint32_t transmitCommand`
Used to write TCR for DMA purpose.

`volatile uint8_t state`
LPSPI transfer state , `_lpspi_transfer_state`.

`uint8_t nbytes`
eDMA minor byte transfer count initially configured.

`lpspi_master_edma_transfer_callback_t callback`
Completion callback.

`void *userData`
Callback user data.

`edma_handle_t *edmaRxRegToRxDataHandle`
edma_handle_t handle point used for RxReg to RxData buff

`edma_handle_t *edmaTxDataToTxRegHandle`
edma_handle_t handle point used for TxData to TxReg buff

`edma_tcd_t lpspiSoftwareTCD[3]`
SoftwareTCD, internal used

`struct _lpspi_slave_edma_handle`
`#include <fsl_lpspi_edma.h>` LPSPI slave eDMA transfer handle structure used for transactional API.

Public Members

volatile bool isByteSwap

A flag that whether should byte swap.

volatile uint8_t fifoSize

FIFO dataSize.

volatile uint8_t rxWatermark

Rx watermark.

volatile uint8_t bytesEachWrite

Bytes for each write TDR.

volatile uint8_t bytesEachRead

Bytes for each read RDR.

volatile uint8_t bytesLastRead

Bytes for last read RDR.

volatile bool isThereExtraRxBytes

Is there extra RX byte.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

const uint8_t *volatile txData

Send buffer.

uint8_t *volatile rxData

Receive buffer.

volatile size_t txRemainingByteCount

Number of bytes remaining to send.

volatile size_t rxRemainingByteCount

Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes

Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes

Read RDR register remaining times.

uint32_t totalByteCount

Number of transfer bytes

uint32_t txBuffIfNull

Used if there is not txData for DMA purpose.

uint32_t rxBuffIfNull

Used if there is not rxData for DMA purpose.

volatile uint8_t state

LPSPi transfer state.

uint32_t errorCount

Error count for slave transfer.

lpspi_slave_edma_transfer_callback_t callback

Completion callback.

`void *userData`

Callback user data.

`edma_handle_t *edmaRxRegToRxDataHandle`

`edma_handle_t` handle point used for RxReg to RxData buff

`edma_handle_t *edmaTxDataToTxRegHandle`

`edma_handle_t` handle point used for TxData to TxReg

`edma_tcd_t` `lpspiSoftwareTCD[2]`

SoftwareTCD, internal used

2.47 LPTMR: Low-Power Timer

`void LPTMR_Init(LPTMR_Type *base, const lptmr_config_t *config)`

Ungates the LPTMR clock and configures the peripheral for a basic operation.

Note: This API should be called at the beginning of the application using the LPTMR driver.

Parameters

- `base` – LPTMR peripheral base address
- `config` – A pointer to the LPTMR configuration structure.

`void LPTMR_Deinit(LPTMR_Type *base)`

Gates the LPTMR clock.

Parameters

- `base` – LPTMR peripheral base address

`void LPTMR_GetDefaultConfig(lptmr_config_t *config)`

Fills in the LPTMR configuration structure with default settings.

The default values are as follows.

```
config->timerMode = kLPTMR_TimerModeTimeCounter;
config->pinSelect = kLPTMR_PinSelectInput_0;
config->pinPolarity = kLPTMR_PinPolarityActiveHigh;
config->enableFreeRunning = false;
config->bypassPrescaler = true;
config->prescalerClockSource = kLPTMR_PrescalerClock_1;
config->value = kLPTMR_Prescale_Glitch_0;
```

Parameters

- `config` – A pointer to the LPTMR configuration structure.

`static inline void LPTMR_EnableInterrupts(LPTMR_Type *base, uint32_t mask)`

Enables the selected LPTMR interrupts.

Parameters

- `base` – LPTMR peripheral base address
- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `lptmr_interrupt_enable_t`

```
static inline void LPTMR_DisableInterrupts(LPTMR_Type *base, uint32_t mask)
```

Disables the selected LPTMR interrupts.

Parameters

- `base` – LPTMR peripheral base address
- `mask` – The interrupts to disable. This is a logical OR of members of the enumeration `lptmr_interrupt_enable_t`.

```
static inline uint32_t LPTMR_GetEnabledInterrupts(LPTMR_Type *base)
```

Gets the enabled LPTMR interrupts.

Parameters

- `base` – LPTMR peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration `lptmr_interrupt_enable_t`

```
static inline uint32_t LPTMR_GetStatusFlags(LPTMR_Type *base)
```

Gets the LPTMR status flags.

Parameters

- `base` – LPTMR peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration `lptmr_status_flags_t`

```
static inline void LPTMR_ClearStatusFlags(LPTMR_Type *base, uint32_t mask)
```

Clears the LPTMR status flags.

Parameters

- `base` – LPTMR peripheral base address
- `mask` – The status flags to clear. This is a logical OR of members of the enumeration `lptmr_status_flags_t`.

```
static inline void LPTMR_SetTimerPeriod(LPTMR_Type *base, uint32_t ticks)
```

Sets the timer period in units of count.

Timers counts from 0 until it equals the count value set here. The count value is written to the CMR register.

Note:

- a. The TCF flag is set with the CNR equals the count provided here and then increments.
 - b. Call the utility macros provided in the `fsl_common.h` to convert to ticks.
-

Parameters

- `base` – LPTMR peripheral base address
- `ticks` – A timer period in units of ticks

```
static inline uint32_t LPTMR_GetCurrentTimerCount(LPTMR_Type *base)
```

Reads the current timer counting value.

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note: Call the utility macros provided in the `fsl_common.h` to convert ticks to usec or msec.

Parameters

- base – LPTMR peripheral base address

Returns

The current counter value in ticks

static inline void LPTMR_StartTimer(LPTMR_Type *base)

Starts the timer.

After calling this function, the timer counts up to the CMR register value. Each time the timer reaches the CMR value and then increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

Parameters

- base – LPTMR peripheral base address

static inline void LPTMR_StopTimer(LPTMR_Type *base)

Stops the timer.

This function stops the timer and resets the timer's counter register.

Parameters

- base – LPTMR peripheral base address

FSL_LPTMR_DRIVER_VERSION

Driver Version

enum _lptmr_pin_select

LPTMR pin selection used in pulse counter mode.

Values:

enumerator kLPTMR_PinSelectInput_0

Pulse counter input 0 is selected

enumerator kLPTMR_PinSelectInput_1

Pulse counter input 1 is selected

enumerator kLPTMR_PinSelectInput_2

Pulse counter input 2 is selected

enumerator kLPTMR_PinSelectInput_3

Pulse counter input 3 is selected

enum _lptmr_pin_polarity

LPTMR pin polarity used in pulse counter mode.

Values:

enumerator kLPTMR_PinPolarityActiveHigh

Pulse Counter input source is active-high

enumerator kLPTMR_PinPolarityActiveLow

Pulse Counter input source is active-low

enum _lptmr_timer_mode

LPTMR timer mode selection.

Values:

enumerator kLPTMR_TimerModeTimeCounter

Time Counter mode

enumerator kLPTMR_TimerModePulseCounter

Pulse Counter mode

enum _lptmr_prescaler_glitch_value

LPTMR prescaler/glitch filter values.

Values:

enumerator kLPTMR_Prescale_Glitch_0

Prescaler divide 2, glitch filter does not support this setting

enumerator kLPTMR_Prescale_Glitch_1

Prescaler divide 4, glitch filter 2

enumerator kLPTMR_Prescale_Glitch_2

Prescaler divide 8, glitch filter 4

enumerator kLPTMR_Prescale_Glitch_3

Prescaler divide 16, glitch filter 8

enumerator kLPTMR_Prescale_Glitch_4

Prescaler divide 32, glitch filter 16

enumerator kLPTMR_Prescale_Glitch_5

Prescaler divide 64, glitch filter 32

enumerator kLPTMR_Prescale_Glitch_6

Prescaler divide 128, glitch filter 64

enumerator kLPTMR_Prescale_Glitch_7

Prescaler divide 256, glitch filter 128

enumerator kLPTMR_Prescale_Glitch_8

Prescaler divide 512, glitch filter 256

enumerator kLPTMR_Prescale_Glitch_9

Prescaler divide 1024, glitch filter 512

enumerator kLPTMR_Prescale_Glitch_10

Prescaler divide 2048 glitch filter 1024

enumerator kLPTMR_Prescale_Glitch_11

Prescaler divide 4096, glitch filter 2048

enumerator kLPTMR_Prescale_Glitch_12

Prescaler divide 8192, glitch filter 4096

enumerator kLPTMR_Prescale_Glitch_13

Prescaler divide 16384, glitch filter 8192

enumerator kLPTMR_Prescale_Glitch_14

Prescaler divide 32768, glitch filter 16384

enumerator kLPTMR_Prescale_Glitch_15

Prescaler divide 65536, glitch filter 32768

enum _lptmr_prescaler_clock_select

LPTMR prescaler/glitch filter clock select.

Note: Clock connections are SoC-specific

Values:

enum `_lptmr_interrupt_enable`

List of the LPTMR interrupts.

Values:

enumerator `kLPTMR_TimerInterruptEnable`

Timer interrupt enable

enum `_lptmr_status_flags`

List of the LPTMR status flags.

Values:

enumerator `kLPTMR_TimerCompareFlag`

Timer compare flag

typedef enum `_lptmr_pin_select` `lptmr_pin_select_t`

LPTMR pin selection used in pulse counter mode.

typedef enum `_lptmr_pin_polarity` `lptmr_pin_polarity_t`

LPTMR pin polarity used in pulse counter mode.

typedef enum `_lptmr_timer_mode` `lptmr_timer_mode_t`

LPTMR timer mode selection.

typedef enum `_lptmr_prescaler_glitch_value` `lptmr_prescaler_glitch_value_t`

LPTMR prescaler/glitch filter values.

typedef enum `_lptmr_prescaler_clock_select` `lptmr_prescaler_clock_select_t`

LPTMR prescaler/glitch filter clock select.

Note: Clock connections are SoC-specific

typedef enum `_lptmr_interrupt_enable` `lptmr_interrupt_enable_t`

List of the LPTMR interrupts.

typedef enum `_lptmr_status_flags` `lptmr_status_flags_t`

List of the LPTMR status flags.

typedef struct `_lptmr_config` `lptmr_config_t`

LPTMR config structure.

This structure holds the configuration settings for the LPTMR peripheral. To initialize this structure to reasonable defaults, call the `LPTMR_GetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration struct can be made constant so it resides in flash.

static inline void `LPTMR_EnableTimerDMA(LPTMR_Type *base, bool enable)`

Enable or disable timer DMA request.

Parameters

- `base` – base LPTMR peripheral base address
- `enable` – Switcher of timer DMA feature. “true” means to enable, “false” means to disable.

struct `_lptmr_config`

`#include <fsl_lptmr.h>` LPTMR config structure.

This structure holds the configuration settings for the LPTMR peripheral. To initialize this structure to reasonable defaults, call the `LPTMR_GetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration struct can be made constant so it resides in flash.

Public Members

lptmr_timer_mode_t timerMode

Time counter mode or pulse counter mode

lptmr_pin_select_t pinSelect

LPTMR pulse input pin select; used only in pulse counter mode

lptmr_pin_polarity_t pinPolarity

LPTMR pulse input pin polarity; used only in pulse counter mode

bool enableFreeRunning

True: enable free running, counter is reset on overflow False: counter is reset when the compare flag is set

bool bypassPrescaler

True: bypass prescaler; false: use clock from prescaler

lptmr_prescaler_clock_select_t prescalerClockSource

LPTMR clock source

lptmr_prescaler_glitch_value_t value

Prescaler or glitch filter value

2.48 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

2.49 LPUART Driver

static inline void LPUART_SoftwareReset(LPUART_Type *base)

Resets the LPUART using software.

This function resets all internal logic and registers except the Global Register. Remains set until cleared by software.

Parameters

- base – LPUART peripheral base address.

status_t LPUART_Init(LPUART_Type *base, const *lpuart_config_t* *config, uint32_t srcClock_Hz)

Initializes an LPUART instance with the user configuration structure and the peripheral clock.

This function configures the LPUART module with user-defined settings. Call the LPUART_GetDefaultConfig() function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
lpuart_config_t lpuartConfig;
lpuartConfig.baudRate_Bps = 115200U;
lpuartConfig.parityMode = kLPUART_ParityDisabled;
lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
lpuartConfig.isMsb = false;
lpuartConfig.stopBitCount = kLPUART_OneStopBit;
lpuartConfig.txFifoWatermark = 0;
```

(continues on next page)

(continued from previous page)

```
lpuartConfig.rxFifoWatermark = 1;
LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
```

Parameters

- base – LPUART peripheral base address.
- config – Pointer to a user-defined configuration structure.
- srcClock_Hz – LPUART clock source frequency in HZ.

Return values

- kStatus_LPUART_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus_Success – LPUART initialize succeed

status_t LPUART_Deinit(LPUART_Type *base)

Deinitializes a LPUART instance.

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

- base – LPUART peripheral base address.

Return values

- kStatus_Success – Deinit is success.
- kStatus_LPUART_Timeout – Timeout during deinit.

void LPUART_GetDefaultConfig(*lpuart_config_t* *config)

Gets the default configuration structure.

This function initializes the LPUART configuration structure to a default value. The default values are: lpuartConfig->baudRate_Bps = 115200U; lpuartConfig->parityMode = kLPUART_ParityDisabled; lpuartConfig->dataBitsCount = kLPUART_EightDataBits; lpuartConfig->isMsb = false; lpuartConfig->stopBitCount = kLPUART_OneStopBit; lpuartConfig->txFifoWatermark = 0; lpuartConfig->rxFifoWatermark = 1; lpuartConfig->rxIdleType = kLPUART_IdleTypeStartBit; lpuartConfig->rxIdleConfig = kLPUART_IdleCharacter1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;

Parameters

- config – Pointer to a configuration structure.

status_t LPUART_SetBaudRate(LPUART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the LPUART instance baudrate.

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART_Init.

```
LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
```

Parameters

- base – LPUART peripheral base address.
- baudRate_Bps – LPUART baudrate to be set.
- srcClock_Hz – LPUART clock source frequency in HZ.

Return values

- kStatus_LPUART_BaudrateNotSupport – Baudrate is not supported in the current clock source.
- kStatus_Success – Set baudrate succeeded.

void LPUART_Enable9bitMode(LPUART_Type *base, bool enable)

Enable 9-bit data mode for LPUART.

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

- base – LPUART peripheral base address.
- enable – true to enable, false to disable.

static inline void LPUART_SetMatchAddress(LPUART_Type *base, uint16_t address1, uint16_t address2)

Set the LPUART address.

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receives with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer; otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note: Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

- base – LPUART peripheral base address.
- address1 – LPUART slave address1.
- address2 – LPUART slave address2.

static inline void LPUART_EnableMatchAddress(LPUART_Type *base, bool match1, bool match2)

Enable the LPUART match address feature.

Parameters

- base – LPUART peripheral base address.
- match1 – true to enable match address1, false to disable.
- match2 – true to enable match address2, false to disable.

static inline void LPUART_SetRxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the rx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Rx FIFO watermark.

static inline void LPUART_SetTxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the tx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Tx FIFO watermark.

static inline void LPUART_TransferEnable16Bit(*lpuart_handle_t* *handle, bool enable)

Sets the LPUART using 16bit transmit, only for 9bit or 10bit mode.

This function Enable 16bit Data transmit in *lpuart_handle_t*.

Parameters

- handle – LPUART handle pointer.
- enable – true to enable, false to disable.

uint32_t LPUART_GetStatusFlags(LPUART_Type *base)

Gets LPUART status flags.

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators *_lpuart_flags*. To check for a specific status, compare the return value with enumerators in the *_lpuart_flags*. For example, to check whether the TX is empty:

```
if (kLPUART_TxDataRegEmptyFlag & LPUART_GetStatusFlags(LPUART1))
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART status flags which are Ored by the enumerators in the *_lpuart_flags*.

status_t LPUART_ClearStatusFlags(LPUART_Type *base, uint32_t mask)

Clears status flags with a provided mask.

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only cleared or set by hardware are: *kLPUART_TxDataRegEmptyFlag*, *kLPUART_TransmissionCompleteFlag*, *kLPUART_RxDataRegFullFlag*, *kLPUART_RxActiveFlag*, *kLPUART_NoiseErrorFlag*, *kLPUART_ParityErrorFlag*, *kLPUART_TxFifoEmptyFlag*, *kLPUART_RxFifoEmptyFlag* Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

- base – LPUART peripheral base address.
- mask – the status flags to be cleared. The user can use the enumerators in the *_lpuart_status_flag_t* to do the OR operation and get the mask.

Return values

- *kStatus_LPUART_FlagCannotClearManually* – The flag can't be cleared by this function but it is cleared automatically by hardware.
- *kStatus_Success* – Status in the mask are cleared.

Returns

0 succeed, others failed.

void LPUART_EnableInterrupts(LPUART_Type *base, uint32_t mask)

Enables LPUART interrupts according to a provided mask.

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the *_lpuart_interrupt_enable*. This examples shows how to enable TX empty interrupt and RX full interrupt:

```
LPUART_EnableInterrupts(LPUART1, kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↳ RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to enable. Logical OR of `_lpuart_interrupt_enable`.

```
void LPUART_DisableInterrupts(LPUART_Type *base, uint32_t mask)
```

Disables LPUART interrupts according to a provided mask.

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See `_lpuart_interrupt_enable`. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
LPUART_DisableInterrupts(LPUART1, kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↳ RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to disable. Logical OR of `_lpuart_interrupt_enable`.

```
uint32_t LPUART_GetEnabledInterrupts(LPUART_Type *base)
```

Gets enabled LPUART interrupts.

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators `_lpuart_interrupt_enable`. To check a specific interrupt enable status, compare the return value with enumerators in `_lpuart_interrupt_enable`. For example, to check whether the TX empty interrupt is enabled:

```
uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);

if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART interrupt flags which are logical OR of the enumerators in `_lpuart_interrupt_enable`.

```
static inline uintptr_t LPUART_GetDataRegisterAddress(LPUART_Type *base)
```

Gets the LPUART data register address.

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART data register addresses which are used both by the transmitter and receiver.

```
static inline void LPUART_EnableTxDMA(LPUART_Type *base, bool enable)
```

Enables or disables the LPUART transmitter DMA request.

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

```
static inline void LPUART_EnableRxDMA(LPUART_Type *base, bool enable)
```

Enables or disables the LPUART receiver DMA.

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

```
uint32_t LPUART_GetInstance(LPUART_Type *base)
```

Get the LPUART instance from peripheral base address.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART instance.

```
static inline void LPUART_EnableTx(LPUART_Type *base, bool enable)
```

Enables or disables the LPUART transmitter.

This function enables or disables the LPUART transmitter.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

```
static inline void LPUART_EnableRx(LPUART_Type *base, bool enable)
```

Enables or disables the LPUART receiver.

This function enables or disables the LPUART receiver.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

```
static inline void LPUART_WriteByte(LPUART_Type *base, uint8_t data)
```

Writes to the transmitter register.

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

- base – LPUART peripheral base address.
- data – Data write to the TX register.


```
static inline uint8_t LPUART_ReadByte(LPUART_Type *base)
```

Reads the receiver register.

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

- base – LPUART peripheral base address.

Returns

Data read from data register.

```
static inline uint8_t LPUART_GetRxFifoCount(LPUART_Type *base)
```

Gets the rx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

rx FIFO data count.

```
static inline uint8_t LPUART_GetTxFifoCount(LPUART_Type *base)
```

Gets the tx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

tx FIFO data count.

```
void LPUART_SendAddress(LPUART_Type *base, uint8_t address)
```

Transmit an address frame in 9-bit data mode.

Parameters

- base – LPUART peripheral base address.
- address – LPUART slave address.

```
status_t LPUART_WriteBlocking(LPUART_Type *base, const uint8_t *data, size_t length)
```

Writes to the transmitter register using a blocking method.

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the data to be sent out to the bus.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

```
status_t LPUART_WriteBlocking16bit(LPUART_Type *base, const uint16_t *data, size_t length)
```

Writes to the transmitter register using a blocking method in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer. Please make sure only 10bit of data is valid and other bits are 0.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t LPUART_ReadBlocking(LPUART_Type *base, uint8_t *data, size_t length)

Reads the receiver data register using a blocking method.

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data.
- length – Size of the buffer.

Return values

- kStatus_LPUART_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus_LPUART_NoiseError – Noise error happened while receiving data.
- kStatus_LPUART_FramingError – Framing error happened while receiving data.
- kStatus_LPUART_ParityError – Parity error happened while receiving data.
- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t LPUART_ReadBlocking16bit(LPUART_Type *base, uint16_t *data, size_t length)

Reads the receiver data register in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data by 16bit, only 10bit is valid.
- length – Size of the buffer.

Return values

- kStatus_LPUART_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus_LPUART_NoiseError – Noise error happened while receiving data.
- kStatus_LPUART_FramingError – Framing error happened while receiving data.

- `kStatus_LPUART_ParityError` – Parity error happened while receiving data.
- `kStatus_LPUART_Timeout` – Transmission timed out and was aborted.
- `kStatus_Success` – Successfully received all data.

```
void LPUART__TransferCreateHandle(LPUART_Type *base, lpuart_handle_t *handle,  
                                lpuart_transfer_callback_t callback, void *userData)
```

Initializes the LPUART handle.

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

The LPUART driver supports the “background” receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn’t call the `LPUART_TransferReceiveNonBlocking()` API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as `ringBuffer`.

Parameters

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

```
status_t LPUART__TransferSendNonBlocking(LPUART_Type *base, lpuart_handle_t *handle,  
                                         lpuart_transfer_t *xfer)
```

Transmits a buffer of data using the interrupt method.

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the `kStatus_LPUART_TxIdle` as status parameter.

Note: The `kStatus_LPUART_TxIdle` is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the TX, check the `kLPUART_TransmissionCompleteFlag` to ensure that the transmit is finished.

Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.
- `xfer` – LPUART transfer structure, see `lpuart_transfer_t`.

Return values

- `kStatus_Success` – Successfully start the data transmission.
- `kStatus_LPUART_TxBusy` – Previous transmission still not finished, data not all written to the TX register.
- `kStatus_InvalidArgument` – Invalid argument.

```
void LPUART__TransferStartRingBuffer(LPUART_Type *base, lpuart_handle_t *handle, uint8_t  
                                     *ringBuffer, size_t ringBufferSize)
```

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the `UART_TransferReceiveNonBlocking()` API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note: When using RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, then only 31 bytes are used for saving data.

Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.
- `ringBuffer` – Start address of ring buffer for background receiving. Pass `NULL` to disable the ring buffer.
- `ringBufferSize` – size of the ring buffer.

`void LPUART__TransferStopRingBuffer(LPUART_Type *base, lpuart_handle_t *handle)`

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.

`size_t LPUART__TransferGetRxRingBufferLength(LPUART_Type *base, lpuart_handle_t *handle)`

Get the length of received data in RX ring buffer.

Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.

Returns

Length of received data in RX ring buffer.

`void LPUART__TransferAbortSend(LPUART_Type *base, lpuart_handle_t *handle)`

Aborts the interrupt-driven data transmit.

This function aborts the interrupt driven data sending. The user can get the `remainBtyes` to find out how many bytes are not sent out.

Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.

`status_t LPUART__TransferGetSendCount(LPUART_Type *base, lpuart_handle_t *handle, uint32_t *count)`

Gets the number of bytes that have been sent out to bus.

This function gets the number of bytes that have been sent out to bus by an interrupt method.

Parameters

- `base` – LPUART peripheral base address.

- `handle` – LPUART handle pointer.
- `count` – Send bytes count.

Return values

- `kStatus_NoTransferInProgress` – No send in progress.
- `kStatus_InvalidArgument` – Parameter is invalid.
- `kStatus_Success` – Get successfully through the parameter `count`;

`status_t` LPUART_TransferReceiveNonBlocking(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_t* *xfer, `size_t` *receivedBytes)

Receives a buffer of data using the interrupt method.

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter `kStatus_UART_RxIdle`. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to `xfer->data`, which returns with the parameter `receivedBytes` set to 5. For the remaining 5 bytes, the newly arrived data is saved from `xfer->data[5]`. When 5 bytes are received, the LPUART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to `xfer->data`. When all data is received, the upper layer is notified.

Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.
- `xfer` – LPUART transfer structure, see `uart_transfer_t`.
- `receivedBytes` – Bytes received from the ring buffer directly.

Return values

- `kStatus_Success` – Successfully queue the transfer into the transmit queue.
- `kStatus_LPUART_RxBusy` – Previous receive request is not finished.
- `kStatus_InvalidArgument` – Invalid argument.

`void` LPUART_TransferAbortReceive(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the interrupt-driven data receiving.

This function aborts the interrupt-driven data receiving. The user can get the `remainBytes` to find out how many bytes not received yet.

Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.

`status_t` LPUART_TransferGetReceiveCount(LPUART_Type *base, *lpuart_handle_t* *handle, `uint32_t` *count)

Gets the number of bytes that have been received.

This function gets the number of bytes that have been received.

Parameters

- `base` – LPUART peripheral base address.

- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- kStatus_NoTransferInProgress – No receive in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

void LPUART_TransferHandleIRQ(LPUART_Type *base, void *irqHandle)
LPUART IRQ handle function.

This function handles the LPUART transmit and receive IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

void LPUART_TransferHandleErrorIRQ(LPUART_Type *base, void *irqHandle)
LPUART Error IRQ handle function.

This function handles the LPUART error IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

void LPUART_DriverIRQHandler(uint32_t instance)
LPUART driver IRQ handler common entry.

This function provides the common IRQ request entry for LPUART.

Parameters

- instance – LPUART instance.

FSL_LPUART_DRIVER_VERSION
LPUART driver version.

Error codes for the LPUART driver.

Values:

enumerator kStatus_LPUART_TxBusy
TX busy

enumerator kStatus_LPUART_RxBusy
RX busy

enumerator kStatus_LPUART_TxIdle
LPUART transmitter is idle.

enumerator kStatus_LPUART_RxIdle
LPUART receiver is idle.

enumerator kStatus_LPUART_TxWatermarkTooLarge
TX FIFO watermark too large

enumerator kStatus_LPUART_RxWatermarkTooLarge
RX FIFO watermark too large

enumerator kStatus_LPUART_FlagCannotClearManually

Some flag can't manually clear

enumerator kStatus_LPUART_Error

Error happens on LPUART.

enumerator kStatus_LPUART_RxRingBufferOverrun

LPUART RX software ring buffer overrun.

enumerator kStatus_LPUART_RxHardwareOverrun

LPUART RX receiver overrun.

enumerator kStatus_LPUART_NoiseError

LPUART noise error.

enumerator kStatus_LPUART_FramingError

LPUART framing error.

enumerator kStatus_LPUART_ParityError

LPUART parity error.

enumerator kStatus_LPUART_BaudrateNotSupport

Baudrate is not support in current clock source

enumerator kStatus_LPUART_IdleLineDetected

IDLE flag.

enumerator kStatus_LPUART_Timeout

LPUART times out.

enum _lpuart_parity_mode

LPUART parity mode.

Values:

enumerator kLPUART_ParityDisabled

Parity disabled

enumerator kLPUART_ParityEven

Parity enabled, type even, bit setting: PE | PT = 10

enumerator kLPUART_ParityOdd

Parity enabled, type odd, bit setting: PE | PT = 11

enum _lpuart_data_bits

LPUART data bits count.

Values:

enumerator kLPUART_EightDataBits

Eight data bit

enumerator kLPUART_SevenDataBits

Seven data bit

enum _lpuart_stop_bit_count

LPUART stop bit count.

Values:

enumerator kLPUART_OneStopBit

One stop bit

enumerator kLPUART_TwoStopBit

Two stop bits

enum _lpuart_transmit_cts_source

LPUART transmit CTS source.

Values:

enumerator kLPUART_CtsSourcePin

CTS resource is the LPUART_CTS pin.

enumerator kLPUART_CtsSourceMatchResult

CTS resource is the match result.

enum _lpuart_transmit_cts_config

LPUART transmit CTS configure.

Values:

enumerator kLPUART_CtsSampleAtStart

CTS input is sampled at the start of each character.

enumerator kLPUART_CtsSampleAtIdle

CTS input is sampled when the transmitter is idle

enum _lpuart_idle_type_select

LPUART idle flag type defines when the receiver starts counting.

Values:

enumerator kLPUART_IdleTypeStartBit

Start counting after a valid start bit.

enumerator kLPUART_IdleTypeStopBit

Start counting after a stop bit.

enum _lpuart_idle_config

LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

Values:

enumerator kLPUART_IdleCharacter1

the number of idle characters.

enumerator kLPUART_IdleCharacter2

the number of idle characters.

enumerator kLPUART_IdleCharacter4

the number of idle characters.

enumerator kLPUART_IdleCharacter8

the number of idle characters.

enumerator kLPUART_IdleCharacter16

the number of idle characters.

enumerator kLPUART_IdleCharacter32

the number of idle characters.

enumerator kLPUART_IdleCharacter64

the number of idle characters.

enumerator kLPUART_IdleCharacter128
the number of idle characters.

enum _lpuart_interrupt_enable

LPUART interrupt configuration structure, default settings all disabled.

This structure contains the settings for all LPUART interrupt configurations.

Values:

enumerator kLPUART_LinBreakInterruptEnable
LIN break detect. bit 7

enumerator kLPUART_RxActiveEdgeInterruptEnable
Receive Active Edge. bit 6

enumerator kLPUART_TxDataRegEmptyInterruptEnable
Transmit data register empty. bit 23

enumerator kLPUART_TransmissionCompleteInterruptEnable
Transmission complete. bit 22

enumerator kLPUART_RxDataRegFullInterruptEnable
Receiver data register full. bit 21

enumerator kLPUART_IdleLineInterruptEnable
Idle line. bit 20

enumerator kLPUART_RxOverrunInterruptEnable
Receiver Overrun. bit 27

enumerator kLPUART_NoiseErrorInterruptEnable
Noise error flag. bit 26

enumerator kLPUART_FramingErrorInterruptEnable
Framing error flag. bit 25

enumerator kLPUART_ParityErrorInterruptEnable
Parity error flag. bit 24

enumerator kLPUART_Match1InterruptEnable
Parity error flag. bit 15

enumerator kLPUART_Match2InterruptEnable
Parity error flag. bit 14

enumerator kLPUART_TxFifoOverflowInterruptEnable
Transmit FIFO Overflow. bit 9

enumerator kLPUART_RxFifoUnderflowInterruptEnable
Receive FIFO Underflow. bit 8

enumerator kLPUART_AllInterruptEnable

enum _lpuart_flags

LPUART status flags.

This provides constants for the LPUART status flags for use in the LPUART functions.

Values:

enumerator kLPUART_TxDataRegEmptyFlag
Transmit data register empty flag, sets when transmit buffer is empty. bit 23

enumerator kLPUART_TransmissionCompleteFlag

Transmission complete flag, sets when transmission activity complete. bit 22

enumerator kLPUART_RxDataRegFullFlag

Receive data register full flag, sets when the receive data buffer is full. bit 21

enumerator kLPUART_IdleLineFlag

Idle line detect flag, sets when idle line detected. bit 20

enumerator kLPUART_RxOverrunFlag

Receive Overrun, sets when new data is received before data is read from receive register. bit 19

enumerator kLPUART_NoiseErrorFlag

Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18

enumerator kLPUART_FramingErrorFlag

Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17

enumerator kLPUART_ParityErrorFlag

If parity enabled, sets upon parity error detection. bit 16

enumerator kLPUART_LinBreakFlag

LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31

enumerator kLPUART_RxActiveEdgeFlag

Receive pin active edge interrupt flag, sets when active edge detected. bit 30

enumerator kLPUART_RxActiveFlag

Receiver Active Flag (RAF), sets at beginning of valid start. bit 24

enumerator kLPUART_DataMatch1Flag

The next character to be read from LPUART_DATA matches MA1. bit 15

enumerator kLPUART_DataMatch2Flag

The next character to be read from LPUART_DATA matches MA2. bit 14

enumerator kLPUART_TxFifoEmptyFlag

TXEMPT bit, sets if transmit buffer is empty. bit 7

enumerator kLPUART_RxFifoEmptyFlag

RXEMPT bit, sets if receive buffer is empty. bit 6

enumerator kLPUART_TxFifoOverflowFlag

TXOF bit, sets if transmit buffer overflow occurred. bit 1

enumerator kLPUART_RxFifoUnderflowFlag

RXUF bit, sets if receive buffer underflow occurred. bit 0

enumerator kLPUART_AllClearFlags

enumerator kLPUART_AllFlags

typedef enum *_lpuart_parity_mode* lpuart_parity_mode_t

LPUART parity mode.

typedef enum *_lpuart_data_bits* lpuart_data_bits_t

LPUART data bits count.

typedef enum *_lpuart_stop_bit_count* lpuart_stop_bit_count_t

LPUART stop bit count.

```
typedef enum _lpuart_transmit_cts_source lpuart_transmit_cts_source_t
    LPUART transmit CTS source.

typedef enum _lpuart_transmit_cts_config lpuart_transmit_cts_config_t
    LPUART transmit CTS configure.

typedef enum _lpuart_idle_type_select lpuart_idle_type_select_t
    LPUART idle flag type defines when the receiver starts counting.

typedef enum _lpuart_idle_config lpuart_idle_config_t
    LPUART idle detected configuration. This structure defines the number of idle characters
    that must be received before the IDLE flag is set.

typedef struct _lpuart_config lpuart_config_t
    LPUART configuration structure.

typedef struct _lpuart_transfer lpuart_transfer_t
    LPUART transfer structure.

typedef struct _lpuart_handle lpuart_handle_t

typedef void (*lpuart_transfer_callback_t)(LPUART_Type *base, lpuart_handle_t *handle,
status_t status, void *userData)
    LPUART transfer callback function.

typedef void (*lpuart_isr_t)(LPUART_Type *base, void *handle)

void *s_lpuartHandle[]

const IRQn_Type s_lpuartTxIRQ[]

lpuart_isr_t s_lpuartIsr[]

UART_RETRY_TIMES
    Retry times for waiting flag.

struct _lpuart_config
    #include <fsl_lpuart.h> LPUART configuration structure.
```

Public Members

```
uint32_t baudRate_Bps
    LPUART baud rate

lpuart_parity_mode_t parityMode
    Parity mode, disabled (default), even, odd

lpuart_data_bits_t dataBitsCount
    Data bits count, eight (default), seven

bool isMsb
    Data bits order, LSB (default), MSB

lpuart_stop_bit_count_t stopBitCount
    Number of stop bits, 1 stop bit (default) or 2 stop bits

uint8_t txFifoWatermark
    TX FIFO watermark

uint8_t rxFifoWatermark
    RX FIFO watermark
```

`bool enableRxRTS`
RX RTS enable

`bool enableTxCTS`
TX CTS enable

`lpuart_transmit_cts_source_t txCtsSource`
TX CTS source

`lpuart_transmit_cts_config_t txCtsConfig`
TX CTS configure

`lpuart_idle_type_select_t rxIdleType`
RX IDLE type.

`lpuart_idle_config_t rxIdleConfig`
RX IDLE configuration.

`bool enableTx`
Enable TX

`bool enableRx`
Enable RX

`struct __lpuart__transfer`
#include <fsl_lpuart.h> LPUART transfer structure.

Public Members

`size_t dataSize`
The byte count to be transfer.

`struct __lpuart__handle`
#include <fsl_lpuart.h> LPUART handle structure.

Public Members

`volatile size_t txDataSize`
Size of the remaining data to send.

`size_t txDataSizeAll`
Size of the data to send out.

`volatile size_t rxDataSize`
Size of the remaining data to receive.

`size_t rxDataSizeAll`
Size of the data to receive.

`size_t rxRingBufferSize`
Size of the ring buffer.

`volatile uint16_t rxRingBufferHead`
Index for the driver to store received data into ring buffer.

`volatile uint16_t rxRingBufferTail`
Index for the user to get data from the ring buffer.

`lpuart_transfer_callback_t callback`
Callback function.

`void *userData`
LPUART callback function parameter.

`volatile uint8_t txState`
TX transfer state.

`volatile uint8_t rxState`
RX transfer state.

`bool isSevenDataBits`
Seven data bits flag.

`bool is16bitData`
16bit data bits flag, only used for 9bit or 10bit data

`union __unnamed67__`

Public Members

`uint8_t *data`
The buffer of data to be transfer.

`uint8_t *rxData`
The buffer to receive data.

`uint16_t *rxData16`
The buffer to receive data.

`const uint8_t *txData`
The buffer of data to be sent.

`const uint16_t *txData16`
The buffer of data to be sent.

`union __unnamed69__`

Public Members

`const uint8_t *volatile txData`
Address of remaining data to send.

`const uint16_t *volatile txData16`
Address of remaining data to send.

`union __unnamed71__`

Public Members

`uint8_t *volatile rxData`
Address of remaining data to receive.

`uint16_t *volatile rxData16`
Address of remaining data to receive.

`union __unnamed73__`

Public Members

`uint8_t *rxRingBuffer`
Start address of the receiver ring buffer.

`uint16_t *rxRingBuffer16`
Start address of the receiver ring buffer.

2.50 LPUART eDMA Driver

```
void LPUART__TransferCreateHandleEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle,  
                                     lpuart_edma_transfer_callback_t callback, void  
                                     *userData, edma_handle_t *txEdmaHandle,  
                                     edma_handle_t *rxEdmaHandle)
```

Initializes the LPUART handle which is used in transactional functions.

Note: This function disables all LPUART interrupts.

Parameters

- `base` – LPUART peripheral base address.
- `handle` – Pointer to `lpuart_edma_handle_t` structure.
- `callback` – Callback function.
- `userData` – User data.
- `txEdmaHandle` – User requested DMA handle for TX DMA transfer.
- `rxEdmaHandle` – User requested DMA handle for RX DMA transfer.

```
status_t LPUART__SendEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle,  
                          lpuart_transfer_t *xfer)
```

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.
- `xfer` – LPUART eDMA transfer structure. See `lpuart_transfer_t`.

Return values

- `kStatus_Success` – if succeed, others failed.
- `kStatus_LPUART_TxBusy` – Previous transfer on going.
- `kStatus_InvalidArgument` – Invalid argument.

```
status_t LPUART__ReceiveEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle,  
                             lpuart_transfer_t *xfer)
```

Receives data using eDMA.

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to `lpuart_edma_handle_t` structure.
- xfer – LPUART eDMA transfer structure, see `lpuart_transfer_t`.

Return values

- `kStatus__Success` – if succeed, others fail.
- `kStatus__LPUART_RxBusy` – Previous transfer ongoing.
- `kStatus__InvalidArgument` – Invalid argument.

`void LPUART__TransferAbortSendEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle)`
Aborts the sent data using eDMA.

This function aborts the sent data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to `lpuart_edma_handle_t` structure.

`void LPUART__TransferAbortReceiveEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle)`
Aborts the received data using eDMA.

This function aborts the received data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to `lpuart_edma_handle_t` structure.

`status_t LPUART__TransferGetSendCountEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle, uint32_t *count)`

Gets the number of bytes written to the LPUART TX register.

This function gets the number of bytes written to the LPUART TX register by DMA.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Send bytes count.

Return values

- `kStatus__NoTransferInProgress` – No send in progress.
- `kStatus__InvalidArgument` – Parameter is invalid.
- `kStatus__Success` – Get successfully through the parameter count;

`status_t LPUART__TransferGetReceiveCountEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle, uint32_t *count)`

Gets the number of received bytes.

This function gets the number of received bytes.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- `kStatus_NoTransferInProgress` – No receive in progress.
- `kStatus_InvalidArgument` – Parameter is invalid.
- `kStatus_Success` – Get successfully through the parameter count;

`void LPUART_TransferEdmaHandleIRQ(LPUART_Type *base, void *lpuartEdmaHandle)`
LPUART eDMA IRQ handle function.

This function handles the LPUART tx complete IRQ request and invoke user callback. It is not set to static so that it can be used in user application.

Note: This function is used as default IRQ handler by double weak mechanism. If user's specific IRQ handler is implemented, make sure this function is invoked in the handler.

Parameters

- `base` – LPUART peripheral base address.
- `lpuartEdmaHandle` – LPUART handle pointer.

`FSL_LPUART_EDMA_DRIVER_VERSION`

LPUART EDMA driver version.

`typedef struct _lpuart_edma_handle lpuart_edma_handle_t`

`typedef void (*lpuart_edma_transfer_callback_t)(LPUART_Type *base, lpuart_edma_handle_t *handle, status_t status, void *userData)`

LPUART transfer callback function.

`struct _lpuart_edma_handle`

`#include <fsl_lpuart_edma.h>` LPUART eDMA handle.

Public Members

`lpuart_edma_transfer_callback_t` callback

Callback function.

`void *userData`

LPUART callback function parameter.

`size_t rxDataSizeAll`

Size of the data to receive.

`size_t txDataSizeAll`

Size of the data to send out.

`edma_handle_t *txEdmaHandle`

The eDMA TX channel used.

`edma_handle_t *rxEdmaHandle`

The eDMA RX channel used.

`uint8_t nbytes`

eDMA minor byte transfer count initially configured.

`volatile uint8_t txState`

TX transfer state.

`volatile uint8_t rxState`

RX transfer state

2.51 MCM: Miscellaneous Control Module

FSL_MCM_DRIVER_VERSION

MCM driver version.

Enum _mcm_interrupt_flag. Interrupt status flag mask. .

Values:

enumerator kMCM_CacheWriteBuffer

Cache Write Buffer Error Enable.

enumerator kMCM_ParityError

Cache Parity Error Enable.

enumerator kMCM_FPUInvalidOperation

FPU Invalid Operation Interrupt Enable.

enumerator kMCM_FPUDivideByZero

FPU Divide-by-zero Interrupt Enable.

enumerator kMCM_FPUOverflow

FPU Overflow Interrupt Enable.

enumerator kMCM_FPUUnderflow

FPU Underflow Interrupt Enable.

enumerator kMCM_FPUInexact

FPU Inexact Interrupt Enable.

enumerator kMCM_FPUInputDenormalInterrupt

FPU Input Denormal Interrupt Enable.

typedef union *_mcm_buffer_fault_attribute* mcm_buffer_fault_attribute_t

The union of buffer fault attribute.

typedef union *_mcm_lmem_fault_attribute* mcm_lmem_fault_attribute_t

The union of LMEM fault attribute.

static inline void MCM_EnableCrossbarRoundRobin(MCM_Type *base, bool enable)

Enables/Disables crossbar round robin.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable crossbar round robin.
 - **true** Enable crossbar round robin.
 - **false** disable crossbar round robin.

static inline void MCM_EnableInterruptStatus(MCM_Type *base, uint32_t mask)

Enables the interrupt.

Parameters

- base – MCM peripheral base address.
- mask – Interrupt status flags mask(_mcm_interrupt_flag).

static inline void MCM_DisableInterruptStatus(MCM_Type *base, uint32_t mask)

Disables the interrupt.

Parameters

- base – MCM peripheral base address.
- mask – Interrupt status flags mask(`_mcm_interrupt_flag`).

static inline uint16_t MCM_GetInterruptStatus(MCM_Type *base)

Gets the Interrupt status .

Parameters

- base – MCM peripheral base address.

static inline void MCM_ClearCacheWriteBufferErrorStatus(MCM_Type *base)

Clears the Interrupt status .

Parameters

- base – MCM peripheral base address.

static inline uint32_t MCM_GetBufferFaultAddress(MCM_Type *base)

Gets buffer fault address.

Parameters

- base – MCM peripheral base address.

static inline void MCM_GetBufferFaultAttribute(MCM_Type *base, *mcm_buffer_fault_attribute_t* *bufferfault)

Gets buffer fault attributes.

Parameters

- base – MCM peripheral base address.

static inline uint32_t MCM_GetBufferFaultData(MCM_Type *base)

Gets buffer fault data.

Parameters

- base – MCM peripheral base address.

static inline void MCM_LimitCodeCachePeripheralWriteBuffering(MCM_Type *base, bool enable)

Limit code cache peripheral write buffering.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable limit code cache peripheral write buffering.
 - **true** Enable limit code cache peripheral write buffering.
 - **false** disable limit code cache peripheral write buffering.

static inline void MCM_BypassFixedCodeCacheMap(MCM_Type *base, bool enable)

Bypass fixed code cache map.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable bypass fixed code cache map.
 - **true** Enable bypass fixed code cache map.
 - **false** disable bypass fixed code cache map.

static inline void MCM_EnableCodeBusCache(MCM_Type *base, bool enable)

Enables/Disables code bus cache.

Parameters

- base – MCM peripheral base address.
- enable – Used to disable/enable code bus cache.
 - **true** Enable code bus cache.
 - **false** disable code bus cache.

static inline void MCM_ForceCodeCacheToNoAllocation(MCM_Type *base, bool enable)

Force code cache to no allocation.

Parameters

- base – MCM peripheral base address.
- enable – Used to force code cache to allocation or no allocation.
 - **true** Force code cache to no allocation.
 - **false** Force code cache to allocation.

static inline void MCM_EnableCodeCacheWriteBuffer(MCM_Type *base, bool enable)

Enables/Disables code cache write buffer.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable code cache write buffer.
 - **true** Enable code cache write buffer.
 - **false** Disable code cache write buffer.

static inline void MCM_ClearCodeBusCache(MCM_Type *base)

Clear code bus cache.

Parameters

- base – MCM peripheral base address.

static inline void MCM_EnablePcParityFaultReport(MCM_Type *base, bool enable)

Enables/Disables PC Parity Fault Report.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable PC Parity Fault Report.
 - **true** Enable PC Parity Fault Report.
 - **false** disable PC Parity Fault Report.

static inline void MCM_EnablePcParity(MCM_Type *base, bool enable)

Enables/Disables PC Parity.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable PC Parity.
 - **true** Enable PC Parity.
 - **false** disable PC Parity.

```
static inline void MCM_LockConfigState(MCM_Type *base)
```

Lock the configuration state.

Parameters

- base – MCM peripheral base address.

```
static inline void MCM_EnableCacheParityReporting(MCM_Type *base, bool enable)
```

Enables/Disables cache parity reporting.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable cache parity reporting.
 - **true** Enable cache parity reporting.
 - **false** disable cache parity reporting.

```
static inline uint32_t MCM_GetLmemFaultAddress(MCM_Type *base)
```

Gets LMEM fault address.

Parameters

- base – MCM peripheral base address.

```
static inline void MCM_GetLmemFaultAttribute(MCM_Type *base, mcm_lmem_fault_attribute_t  
*lmemFault)
```

Get LMEM fault attributes.

Parameters

- base – MCM peripheral base address.

```
static inline uint64_t MCM_GetLmemFaultData(MCM_Type *base)
```

Gets LMEM fault data.

Parameters

- base – MCM peripheral base address.

MCM_LMFATR_TYPE_MASK

MCM_LMFATR_MODE_MASK

MCM_LMFATR_BUFF_MASK

MCM_LMFATR_CACH_MASK

MCM_ISCR_STAT_MASK

FSL_COMPONENT_ID

```
union _mcm_buffer_fault_attribute
```

#include <fsl_mcm.h> The union of buffer fault attribute.

Public Members

```
uint32_t attribute
```

Indicates the faulting attributes, when a properly-enabled cache write buffer error interrupt event is detected.

```
struct _mcm_buffer_fault_attribute._mcm_buffer_fault_attribut attribute_memory
```

```
struct _mcm_buffer_fault_attribut
```

#include <fsl_mcm.h>

Public Members

uint32_t busErrorDataAccessType

Indicates the type of cache write buffer access.

uint32_t busErrorPrivilegeLevel

Indicates the privilege level of the cache write buffer access.

uint32_t busErrorSize

Indicates the size of the cache write buffer access.

uint32_t busErrorAccess

Indicates the type of system bus access.

uint32_t busErrorMasterID

Indicates the crossbar switch bus master number of the captured cache write buffer bus error.

uint32_t busErrorOverrun

Indicates if another cache write buffer bus error is detected.

union _mcm_lmem_fault_attribute

#include <fsl_mcm.h> The union of LMEM fault attribute.

Public Members

uint32_t attribute

Indicates the attributes of the LMEM fault detected.

struct _mcm_lmem_fault_attribute._mcm_lmem_fault_attribut attribute_memory

struct _mcm_lmem_fault_attribut

#include <fsl_mcm.h>

Public Members

uint32_t parityFaultProtectionSignal

Indicates the features of parity fault protection signal.

uint32_t parityFaultMasterSize

Indicates the parity fault master size.

uint32_t parityFaultWrite

Indicates the parity fault is caused by read or write.

uint32_t backdoorAccess

Indicates the LMEM access fault is initiated by core access or backdoor access.

uint32_t parityFaultSyndrome

Indicates the parity fault syndrome.

uint32_t overrun

Indicates the number of faultss.

2.52 Mipi_dsi

`void DSI_Init(MIPI_DSI_Type *base, dsi_config_t *config)`

Initializes the MIPI DSI host with the user configuration.

This function initializes the MIPI DSI host with the configuration, it should be called before other MIPI DSI driver functions.

Parameters

- base – MIPI DSI host peripheral base address.
- config – Pointer to the user configuration structure.

`void DSI_Deinit(MIPI_DSI_Type *base)`

Deinitializes an MIPI DSI host.

This function should be called after all bother MIPI DSI driver functions.

Parameters

- base – MIPI DSI host peripheral base address.

`void DSI_GetDefaultConfig(dsi_config_t *config)`

Gets the default configuration to initialize the MIPI DSI host.

The default value is:

```
config->mode = kDSI_CommandMode;
config->packageFlags = kDSI_DpiEnableAll;
config->enableNoncontinuousClk = true;
config->HsRxDeviceReady_ByteClk = 0U;
config->lpRxDeviceReady_ByteClk = 0U;
config->HsTxDeviceReady_ByteClk = 0U;
config->lpTxDeviceReady_ByteClk = 0U;
```

Parameters

- config – Pointer to a user-defined configuration structure.

`uint32_t DSI_DphyGetPlldivider(uint32_t *m, uint32_t *n, uint32_t refClkFreq_Hz, uint32_t desiredOutFreq_Hz)`

Calculates the D-PHY PLL dividers to generate the desired output frequency.

The phy byte clock frequency(byte count per second) is generated by multiplying the refClkFreq_Hz, the formula is as follows, m & n is configured by mediamix control block.

$\text{desiredOutFreq\_Hz} = \text{refClkFreq\_Hz} * (M + 2) / (N + 1)$. M: 62 ~ 625 N: 0 ~ 15

Parameters

- m – Control of the feedback multiplication ratio.
- n – Control of the input frequency division ratio.
- refClkFreq_Hz – The D-PHY input reference clock frequency (REF_CLK).
- desiredOutFreq_Hz – Desired PLL output frequency.

Returns

The actually output frequency using the returned dividers. If can not find suitable dividers, return 0.

`status_t DSI_PowerUp(MIPI_DSI_Type *base)`

Power up the DSI.

Parameters

- base – MIPI DSI host peripheral base address.

Return values

- `kStatus_Success` – Data transfer finished with no error.
- `kStatus_Timeout` – Transfer failed because of timeout.

`static inline void DSI_PowerDown(MIPI_DSI_Type *base)`

Power down the DSI.

Parameters

- `base` – MIPI DSI host peripheral base address.

`static inline void DSI_EnableInterrupts(MIPI_DSI_Type *base, uint32_t intGroup1, uint32_t intGroup2)`

Enable the interrupts.

The interrupts to enable are passed in as OR'ed mask value of `_dsi_interrupt`.

Parameters

- `base` – MIPI DSI host peripheral base address.
- `intGroup1` – Interrupts to enable in group 1.
- `intGroup2` – Interrupts to enable in group 2.

`static inline void DSI_DisableInterrupts(MIPI_DSI_Type *base, uint32_t intGroup1, uint32_t intGroup2)`

Disable the interrupts.

The interrupts to disable are passed in as OR'ed mask value of `_dsi_interrupt`.

Parameters

- `base` – MIPI DSI host peripheral base address.
- `intGroup1` – Interrupts to disable in group 1.
- `intGroup2` – Interrupts to disable in group 2.

`static inline void DSI_GetAndClearInterruptStatus(MIPI_DSI_Type *base, uint32_t *intGroup1, uint32_t *intGroup2)`

Get and clear the interrupt status.

Parameters

- `base` – MIPI DSI host peripheral base address.
- `intGroup1` – Group 1 interrupt status.
- `intGroup2` – Group 2 interrupt status.

`void DSI_SetDpiConfig(MIPI_DSI_Type *base, const dsi_dpi_config_t *config, uint8_t laneNum)`

Configure the DPI interface.

This function sets the DPI interface configuration, it should be used in video mode.

Parameters

- `base` – MIPI DSI host peripheral base address.
- `config` – Pointer to the DPI interface configuration.
- `laneNum` – How many lanes in use.

`void DSI_SetCommandModeConfig(MIPI_DSI_Type *base, const dsi_command_config_t *config, uint32_t phyByteClkFreq_Hz)`

Configures the command mode configuration.

This function configures the timeout values for DSI command mode.

Parameters

- base – MIPI DSI host peripheral base address.
- config – Pointer to the command mode configuration structure.
- phyByteClkFreq_Hz – Byte clock frequency in each lane.

static inline void DSI_EnableCommandMode(MIPI_DSI_Type *base, bool enable)

Enables the command mode.

This function configures the timeout values for DSI command mode.

Parameters

- base – MIPI DSI host peripheral base address.
- enable – true to enable command mode and disable video mode, vise versa.

static inline void DSI_EnableVpgEnMode(MIPI_DSI_Type *base, bool enable)

Enables the VPG mode.

This function configures video mode pattern generator.

Parameters

- base – MIPI DSI host peripheral base address.
- enable – true to enable video mode pattern generator.

void DSI_GetDefaultDphyConfig(*dsi_dphy_config_t* *config, uint32_t phyByteClkFreq_Hz, uint8_t laneNum)

Gets the default D-PHY configuration.

Gets the default D-PHY configuration, the timing parameters are set according to D-PHY specification. User can use the configuration directly, or change the parameters according device specific requirements.

Parameters

- config – Pointer to the D-PHY configuration.
- phyByteClkFreq_Hz – Byte clock frequency in each lane.
- laneNum – How may lanes in use.

void DSI_InitDphy(MIPI_DSI_Type *base, const *dsi_dphy_config_t* *config)

Initializes the D-PHY.

This function configures the D-PHY timing and setups the D-PHY PLL based on user configuration. The default configuration can be obtained by calling the function DSI_GetDefaultDphyConfig.

Parameters

- base – MIPI DSI host peripheral base address.
- config – Pointer to the D-PHY configuration.

void DSI_SetPacketControl(MIPI_DSI_Type *base, uint8_t flags)

Configures the APB packet to send.

This function configures the next APB packet transfer feature. After configuration, user can write the payload by calling DSI_WriteTxPayload then call DSI_WriteTxHeader to start the tranasfer or just call DSI_WriteTxHeader alone if it is a short packet.

Parameters

- base – MIPI DSI host peripheral base address.
- flags – The transfer control flags, see ref `_dsi_transfer_flags`.


```
void DSI_WriteTxHeader(MIPI_DSI_Type *base, uint16_t wordCount, uint8_t virtualChannel,
                     dsi_tx_data_type_t dataType)
```

Writes tx header to command FIFO. This will trigger the packet transfer.

Parameters

- `base` – MIPI DSI host peripheral base address.
- `wordCount` – For long packet, this is the byte count of the payload. For short packet, this is $(data1 \ll 8) | data0$.
- `virtualChannel` – Virtual channel.
- `dataType` – The packet data type, (DI).

```
void DSI_WriteTxPayload(MIPI_DSI_Type *base, const uint8_t *payload, uint16_t payloadSize)
```

Fills the long APB packet payload.

Write the long packet payload to TX FIFO.

Parameters

- `base` – MIPI DSI host peripheral base address.
- `payload` – Pointer to the payload.
- `payloadSize` – Payload size in byte.

```
void DSI_WriteTxPayloadExt(MIPI_DSI_Type *base, const uint8_t *payload, uint16_t
                          payloadSize, bool sendDcsCmd, uint8_t dcsCmd)
```

Writes payload data to generic payload FIFO.

Write the long packet payload to TX FIFO. This function could be used in two ways

- Include the DCS command in the 1st byte of `payload`. In this case, the DCS command is the first byte of `payload`. The parameter `sendDcsCmd` is set to false, the `dcsCmd` is not used. This function is the same as `DSI_WriteTxPayload` when used in this way.
- The DCS command is not in `payload`, but specified by parameter `dcsCmd`. In this case, the parameter `sendDcsCmd` is set to true, the `dcsCmd` is the DCS command to send. The `payload` is sent after `dcsCmd`.

Parameters

- `base` – MIPI DSI host peripheral base address.
- `payload` – Pointer to the payload.
- `payloadSize` – Payload size in byte.
- `sendDcsCmd` – If set to true, the DCS command is specified by `dcsCmd`, otherwise the DCS command is included in the `payload`.
- `dcsCmd` – The DCS command to send, only used when `sendDCSCmd` is true.

```
void DSI_ReadRxData(MIPI_DSI_Type *base, uint8_t *payload, uint16_t payloadSize)
```

Reads the long APB packet payload.

Read the long packet payload from RX FIFO. This function reads directly from RX FIFO status. Upper layer should make sure the whole rx packet has been received.

Parameters

- `base` – MIPI DSI host peripheral base address.
- `payload` – Pointer to the payload buffer.
- `payloadSize` – Payload size in byte.

status_t DSI_TransferBlocking(MIPI_DSI_Type *base, *dsi_transfer_t* *xfer)

APB data transfer using blocking method.

Perform APB data transfer using blocking method. This function waits until all data send or received, or timeout happens.

Parameters

- base – MIPI DSI host peripheral base address.
- xfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Data transfer finished with no error.
- kStatus_Timeout – Transfer failed because of timeout.
- kStatus_DSI_RxDataError – RX data error, user could use ref DSI_GetRxErrorStatus to check the error details.
- kStatus_DSI_PhyError – PHY error detected during transfer.
- kStatus_DSI_ErrorReportReceived – Error Report packet received, user could use ref DSI_GetAndClearHostStatus to check the error report status.
- kStatus_DSI_NotSupported – Transfer format not supported.
- kStatus_DSI_Fail – Transfer failed for other reasons.

uint16_t Pll_Set_Hs_Freqrange(*uint32_t* bnd_width)

Lookup table method to obtain HS frequency range of operation selection override.

Parameters

- bnd_width – band width frequency in Hz

Returns

the *hsfreqrange_ovr*[6:0] value based on band width frequency in hz.

uint16_t Pll_Set_Pll_Prop_Param(*uint32_t* pll_freq_sel)

Lookup table method to obtain PLL Proportional Charge Pump control.

Parameters

- pll_freq_sel – PLL frequency in Mhz

Returns

the *pll_prop_cntrl_rw*[5:0] value based on video Pll frequency in Mhz.

uint16_t Pll_Set_Sr_Osc_Freq_Target(*uint32_t* pll_freq_sel)

Lookup table method to obtain DDL target oscillation frequency.

Parameters

- pll_freq_sel – PLL frequency in Mhz

Returns

the *sr_osc_freq_target*[11:0] value based on video Pll frequency in Mhz.

uint32_t Pll_Set_Pll_Vco_Freq(*uint32_t* pll_freq_sel)

calculate VCO frequency.

Parameters

- pll_freq_sel – PLL frequency in Hz

Returns

the *vco_freq_clk* value

uint16_t Pll_Set_Pll_Vco_Param(uint32_t pll_freq_sel)

Lookup table method to obtain VCO parameter.

Parameters

- pll_freq_sel – PLL frequency in Mhz

Returns

the pll_vco_cntrl_ovr_rw[5:0] value based on video Pll frequency in Mhz. If can not find suitable value, return default value 63.

void DSI_ConfigDphy(MIPI_DSI_Type *base, uint32_t phyRefClkFreq_Hz, uint32_t dataRateFreq_Hz)

config to set Dphy.

Parameters

- phyRefClkFreq_Hz – Dphy reference clock frequency in Hz
- dataRateFreq_Hz – line rate clock frequency.

FSL_MIPI_DSI_DRIVER_VERSION

Error codes for the MIPI DSI driver.

Values:

enumerator kStatus_DSI_Busy

DSI is busy.

enumerator kStatus_DSI_EccMultiBitError

Multibit ECC error detected in rx packet.

enumerator kStatus_DSI_CrcError

CRC error detected in rx packet.

enumerator kStatus_DSI_PacketSizeError

Rx packet size error.

enumerator kStatus_DSI_EotMissingError

Received transmission does not end with an EoT packet.

enumerator kStatus_DSI_ErrorReportReceived

Error report package received.

enumerator kStatus_DSI_NotSupported

The transfer type not supported.

enumerator kStatus_DSI_PhyError

Physical layer error.

Status and interrupt mask of acknowledge error sent by device caused by host, belongs to interrupt group1. INT_ST0 bit0-bit15.

Values:

enumerator kDSI_ErrorReportSot

SoT error detected in device's acknowledge error report.

enumerator kDSI_ErrorReportSotSync

SoT Sync error detected in device's acknowledge error report.

enumerator kDSI_ErrorReportEotSync

EoT Sync error detected in device's acknowledge error report.

enumerator kDSI_ErrorReportEscEntryCmd

Escape Mode Entry Command error detected in device's acknowledge error report.

enumerator kDSI_ErrorReportLpSync

Low-power Transmit Sync error detected in device's acknowledge error report.

enumerator kDSI_ErrorReportPeriphTo

Peripheral Timeout error detected in device's acknowledge error report.

enumerator kDSI_ErrorReportFalseControl

False Control error detected in device's acknowledge error report.

enumerator kDSI_ErrorReportDeviceSpecific1

The device specific error detected in device's acknowledge error report.

enumerator kDSI_ErrorReportEccOneBit

Single-bit ECC error detected in device's acknowledge error report.

enumerator kDSI_ErrorReportEccMultiBit

Multi-bit ECC error detected in device's acknowledge error report.

enumerator kDSI_ErrorReportChecksum

Checksum error detected in device's acknowledge error report.

enumerator kDSI_ErrorReportDataTypeUnrecognized

DSI data type not recognized error detected in device's acknowledge error report.

enumerator kDSI_ErrorReportVcIdInvalid

Virtual channel ID invalid error detected in device's acknowledge error report.

enumerator kDSI_ErrorReportTxLengthInvalid

Invalid transmission length error detected in device's acknowledge error report.

enumerator kDSI_ErrorReportDeviceSpecific2

The device specific error detected in device's acknowledge error report.

enumerator kDSI_ErrorReportProtocolViolation

Protocol violation error detected in device's acknowledge error report.

enumerator kDSI_ErrorReportAll

Status and interrupt mask of error in phy layer, belongs to interrupt group1. INT_ST0 bit16-bit20.

Values:

enumerator kDSI_PhyErrorEscEntry

Escape entry error from Lane 0.

enumerator kDSI_PhyErrorLpSync

Low-power data transmission synchronization error from Lane 0.

enumerator kDSI_PhyErrorControl

Control error from Lane 0.

enumerator kDSI_PhyErrorLp0Connection

LP0 connection error from Lane 0.

enumerator kDSI_PhyErrorLp1Connection

LP1 connection error from Lane 0.

enumerator kDSI_PhyErrorAll

Timeout error interrupt and status, belongs to interrupt group2. INT_ST1 bit0-bit6.

Values:

enumerator kDSI_TimeoutErrorHtx

High Speed forward TX timeout detected.

enumerator kDSI_TimeoutErrorLrx

Reverse Low power data receive timeout detected.

Host receive packet error status, belongs to interrupt group2. INT_ST1 bit0-bit6.

Values:

enumerator kDSI_RxErrorEccOneBit

ECC single bit error detected.

enumerator kDSI_RxErrorEccMultiBit

ECC multi bit error detected.

enumerator kDSI_RxErrorCrc

CRC error detected.

enumerator kDSI_RxErrorPacketSize

Packet size error detected.

enumerator kDSI_RxErrorEotMissing

Host receives a transmission that does not end with an EoT packet.

enumerator kDSI_RxErrorAll

Host receive error status, belongs to interrupt group2. INT_ST1 bit7-bit12 bit19?

Values:

enumerator kDSI_DpiPayloadFifoOverflow

During a DPI pixel line storage, the payload FIFO overflow occurs.

enumerator kDSI_GenericCommandFifoOverflow

System writes a command through the Generic interface while FIFO is full causing overflow.

enumerator kDSI_GenericPayloadFifoOverflow

System writes a payload data through the Generic interface while FIFO is full causing payload FIFO overflow.

enumerator kDSI_GenericPayloadFifoUnderflow

System writes the packet header before the packet payload is completed loaded into the payload FIFO during a packet build causing payload FIFO underflow.

enumerator kDSI_GenericReadFifoUnderflow

System requests data before it is fully received causing underflow.

enumerator kDSI_GenericReadFifoOverflow

The Read FIFO size is not correctly dimensioned for the max rx packet size causing generic read FIFO overflow.

`_dsi_dpi_package_flag` Flags for DPI package composition.

Values:

enumerator `kDSI_DpiEnableEotpTxHs`

Enables the EoTp transmission in high-speed.

enumerator `kDSI_DpiEnableEotpRx`

Enables the EoTp reception.

enumerator `kDSI_DpiEnableBta`

Enables the Bus Turn-Around (BTA) request.

enumerator `kDSI_DpiEnableEcc`

Enables the ECC reception, error correction, and reporting.

enumerator `kDSI_DpiEnableCrc`

Enables the CRC reception and error reporting.

enumerator `kDSI_DpiEnableEotpTxLp`

Enables the EoTp transmission in low-power.

enumerator `kDSI_DpiEnableAll`

enum `_dsi_operation_mode`

MIPI DSI operation mode.

Values:

enumerator `kDSI_VideoMode`

Video mode.

enumerator `kDSI_CommandMode`

Command mode.

enum `_dsi_dpi_color_coding`

MIPI DPI interface color coding.

Values:

enumerator `kDSI_DpiRGB16Bit`

16-bit configuration 1. RGB565: XXXXXXXX_RRRRRGGG_GGGBBBBB.

enumerator `kDSI_DpiRGB16BitLoose0`

16-bit configuration 2. RGB565: XXXRRRRR_XXGGGGGG_XXXBBBBB.

enumerator `kDSI_DpiRGB16BitLoose1`

16-bit configuration 3. RGB565: XXRRRRRX_XXGGGGGG_XXBBBBBX.

enumerator `kDSI_DpiRGB18Bit`

18-bit configuration 1. RGB666: XXXXXXRR_RRRRGGGG_GGBBBBBB.

enumerator `kDSI_DpiRGB18BitLoose`

18-bit configuration 2. RGB666: XXRRRRRR_XXGGGGGG_XXBBBBBB.

enumerator `kDSI_DpiRGB24Bit`

24-bit configuration. RGB888: RRRRRRRR_GGGGGGGG_BBBBBBBB.

enumerator `kDSI_DpiYCbCr20Bit`

20-bit configuration. YCbCr422 loosely packed.
Cle1: YYYYYYYY_YYXXCbCbCbCb_CbCbCbCbCbCbXX,
YYYYYYYY_YYXXCrCrCrCr_CrCrCrCrCrCrXX.

Cy-
Cycle2:

enumerator kDSI_DpiYCbCr24Bit

24-bit configuration. YCbCr422: CyCle1: YYYYYYYY_YYYYCbCbCbCb_CbCbCbCbCbCbCbCb,
Cycle2: YYYYYYYY_YYYYCrCrCrCr_CrCrCrCrCrCrCrCr.

enumerator kDSI_DpiYCbCr16Bit

16-bit configuration. YCbCr422 loosely packed. Cy-
Cle1: YYYYYYYY_YYXXCbCbCbCb_CbCbCbCbCbCbXX, Cycle2:
YYYYYYY_XXXXCrCrCrCr_CrCrCrCrXXXX.

enumerator kDSI_DpiRGB30Bit

30-bit configuration. RGB10.10.10: XXXRRRRR_RRRRGGGG_GGGGGGBB_BBBBBBBB.

enumerator kDSI_DpiRGB36Bit

36-bit configuration. RGB12.12.12. CyCle1: XXXXXXRR_RRRRRRRR_RRGGGGGG, Cy-
cle2: XXXXXXGG_GGGGBBBB_BBBBBBBB.

enumerator kDSI_DpiYCbCr12Bit

12-bit configuration. YCbCr420: CyCle1: Y1Y1Y1Y1Y1Y1Y1_Y0Y0Y0Y0Y0Y0Y0_CbCbCbCbCbCbCbCb
Cycle2: Y1Y1Y1Y1Y1Y1Y1Y1_Y0Y0Y0Y0Y0Y0Y0Y0_CrCrCrCrCrCrCrCr.

enumerator kDSI_DpiDcs24Bit

24-bit configuration with no specific coding.

_dsi_dpi_polarity_flag Flags for DPI signal polarity.

Values:

enumerator kDSI_DpiDataEnableActiveHigh

Data enable pin active high.

enumerator kDSI_DpiVsyncActiveHigh

VSYNC active high.

enumerator kDSI_DpiHsyncActiveHigh

HSYNC active high.

enumerator kDSI_DpiShutDownActiveHigh

Shutdown pin active high.

enumerator kDSI_DpiColorModeActiveHigh

Color mode pin active high.

enumerator kDSI_DpiDataEnableActiveLow

Data enable pin active low.

enumerator kDSI_DpiVsyncActiveLow

VSYNC active low.

enumerator kDSI_DpiHsyncActiveLow

HSYNC active low.

enumerator kDSI_DpiShutDownActiveLow

Shutdown pin active low.

enumerator kDSI_DpiColorModeActiveLow

Color mode pin active low.

enum _dsi_video_mode

DSI video mode.

Values:

enumerator kDSI_DpiNonBurstWithSyncPulse
Non-Burst mode with Sync Pulses.

enumerator kDSI_DpiNonBurstWithSyncEvent
Non-Burst mode with Sync Events.

enumerator kDSI_DpiBurst
Burst mode.

enum _dsi_video_pattern

Values:

enumerator kDSI_PatternDisable
Color bar pattern mode disabled.

enumerator kDSI_PatternVertical
Color bar pattern mode displayed vertically.

enumerator kDSI_PatternHorizontal
Color bar pattern mode displayed horizontally.

enum _dsi_tx_data_type

DSI TX data type.

Values:

enumerator kDSI_TxDataVsyncStart
V Sync start.

enumerator kDSI_TxDataVsyncEnd
V Sync end.

enumerator kDSI_TxDataHsyncStart
H Sync start.

enumerator kDSI_TxDataHsyncEnd
H Sync end.

enumerator kDSI_TxDataEoTp
End of transmission packet.

enumerator kDSI_TxDataCmOff
Color mode off.

enumerator kDSI_TxDataCmOn
Color mode on.

enumerator kDSI_TxDataShutDownPeriph
Shut down peripheral.

enumerator kDSI_TxDataTurnOnPeriph
Turn on peripheral.

enumerator kDSI_TxDataGenShortWrNoParam
Generic Short WRITE, no parameters.

enumerator kDSI_TxDataGenShortWrOneParam
Generic Short WRITE, one parameter.

enumerator kDSI_TxDataGenShortWrTwoParam
Generic Short WRITE, two parameter.

enumerator kDSI_TxDataGenShortRdNoParam
Generic Short READ, no parameters.

enumerator kDSI_TxDataGenShortRdOneParam
Generic Short READ, one parameter.

enumerator kDSI_TxDataGenShortRdTwoParam
Generic Short READ, two parameter.

enumerator kDSI_TxDataDcsShortWrNoParam
DCS Short WRITE, no parameters.

enumerator kDSI_TxDataDcsShortWrOneParam
DCS Short WRITE, one parameter.

enumerator kDSI_TxDataDcsShortRdNoParam
DCS Short READ, no parameters.

enumerator kDSI_TxDataSetMaxReturnPktSize
Set the Maximum Return Packet Size.

enumerator kDSI_TxDataNull
Null Packet, no data.

enumerator kDSI_TxDataBlanking
Blanking Packet, no data.

enumerator kDSI_TxDataGenLongWr
Generic long write.

enumerator kDSI_TxDataDcsLongWr
DCS Long Write/write_LUT Command Packet.

enumerator kDSI_TxDataLooselyPackedPixel20BitYCbCr
Loosely Packed Pixel Stream, 20-bit YCbCr, 4:2:2 Format.

enumerator kDSI_TxDataPackedPixel24BitYCbCr
Packed Pixel Stream, 24-bit YCbCr, 4:2:2 Format.

enumerator kDSI_TxDataPackedPixel16BitYCbCr
Packed Pixel Stream, 16-bit YCbCr, 4:2:2 Format.

enumerator kDSI_TxDataPackedPixel30BitRGB
Packed Pixel Stream, 30-bit RGB, 10-10-10 Format.

enumerator kDSI_TxDataPackedPixel36BitRGB
Packed Pixel Stream, 36-bit RGB, 12-12-12 Format.

enumerator kDSI_TxDataPackedPixel12BitYCrCb
Packed Pixel Stream, 12-bit YCbCr, 4:2:0 Format.

enumerator kDSI_TxDataPackedPixel16BitRGB
Packed Pixel Stream, 16-bit RGB, 5-6-5 Format.

enumerator kDSI_TxDataPackedPixel18BitRGB
Packed Pixel Stream, 18-bit RGB, 6-6-6 Format.

enumerator kDSI_TxDataLooselyPackedPixel18BitRGB
Loosely Packed Pixel Stream, 18-bit RGB, 6-6-6 Format.

enumerator kDSI_TxDataPackedPixel24BitRGB
Packed Pixel Stream, 24-bit RGB, 8-8-8 Format.

enum `_dsi_rx_data_type`

DSI RX data type.

Values:

enumerator `kDSI_RxDataAckAndErrorReport`

Acknowledge and Error Report

enumerator `kDSI_RxDataEoTp`

End of Transmission packet.

enumerator `kDSI_RxDataGenShortRdResponseOneByte`

Generic Short READ Response, 1 byte returned.

enumerator `kDSI_RxDataGenShortRdResponseTwoByte`

Generic Short READ Response, 2 byte returned.

enumerator `kDSI_RxDataGenLongRdResponse`

Generic Long READ Response.

enumerator `kDSI_RxDataDcsLongRdResponse`

DCS Long READ Response.

enumerator `kDSI_RxDataDcsShortRdResponseOneByte`

DCS Short READ Response, 1 byte returned.

enumerator `kDSI_RxDataDcsShortRdResponseTwoByte`

DCS Short READ Response, 2 byte returned.

`_dsi_transfer_flags` DSI transfer control flags.

Values:

enumerator `kDSI_TransferUseLowPower`

Use low power or not.

enumerator `kDSI_TransferPerformBTA`

Perform BTA or not at the end of a frame.

typedef enum `_dsi_operation_mode` `dsi_operation_mode_t`

MIPI DSI operation mode.

typedef struct `_dsi_config` `dsi_config_t`

MIPI DSI controller configuration.

typedef enum `_dsi_dpi_color_coding` `dsi_dpi_color_coding_t`

MIPI DPI interface color coding.

typedef enum `_dsi_video_mode` `dsi_video_mode_t`

DSI video mode.

typedef enum `_dsi_video_pattern` `dsi_video_pattern_t`

typedef struct `_dsi_dpi_config` `dsi_dpi_config_t`

MIPI DSI controller DPI interface configuration.

typedef struct `_dsi_command_config` `dsi_command_config_t`

MIPI DSI command mode configuration.

typedef struct `_dsi_dphy_config` `dsi_dphy_config_t`

MIPI DSI D-PHY configuration.

```
typedef enum _dsi_tx_data_type dsi_tx_data_type_t
    DSI TX data type.
```

```
typedef enum _dsi_rx_data_type dsi_rx_data_type_t
    DSI RX data type.
```

```
typedef struct _dsi_transfer dsi_transfer_t
    Structure for the data transfer.
```

```
uint32_t DSI_GetInstance(MIPI_DSI_Type *base)
    Gets the MIPI DSI host controller instance from peripheral base address.
```

Parameters

- base – MIPI DSI peripheral base address.

Returns

MIPI DSI instance.

TX_DPHY_TX_PLL_1

TX_DPHY_TX_PLL_5

TX_DPHY_TX_PLL_9

TX_DPHY_TX_PLL_13

TX_DPHY_TX_PLL_17

TX_DPHY_TX_PLL_22

TX_DPHY_TX_PLL_23

TX_DPHY_TX_PLL_24

TX_DPHY_TX_PLL_25

TX_DPHY_TX_PLL_27

TX_DPHY_TX_PLL_28

TX_DPHY_TX_PLL_29

TX_DPHY_TX_PLL_30

TX_DPHY_TX_PLL_31

TX_DPHY_TX_CB_0

TX_DPHY_TX_CB_1

TX_DPHY_TX_CB_2

TX_DPHY_TX_SLEW_5

TX_DPHY_TX_SLEW_6

TX_DPHY_TX_SLEW_7

TX_DPHY_TX_CLK_TERMLOWCAP

```
struct _dsi_config
    #include <fsl_mipi_dsi.h> MIPI DSI controller configuration.
```

Public Members

dsi_operation_mode_t mode

DSI operation mode. MODE_CFG[cmd_video_mode]

uint8_t packageFlags

OR'ed value of _dsi_dpi_package_flag that controls DPI package composition. PCK-HDL_CFG

bool enableNoncontinuousClk

Enables the automatic mechanism to stop providing clock in the clock lane when time allows. LPCLK_CTRL[auto_clklane_ctrl]

uint16_t HsRxDeviceReady_ByteClk

The min time the display device takes to process high-speed read from master before it can continue doing other stuff. The timer starts when D-PHY enters stop state and measured in lane byte clock. HS_RD_TO_CNT[hs_rd_to_cnt]

uint16_t lpRxDeviceReady_ByteClk

The min time the display device takes to process low-power read from master before it can continue doing other stuff. The timer starts when D-PHY enters stop state and measured in lane byte clock. LP_RD_TO_CNT[lp_rd_to_cnt]

uint16_t HsTxDeviceReady_ByteClk

The min time the display device takes to process high-speed write from master before it can continue doing other stuff. The timer starts when D-PHY enters stop state and measured in lane byte clock. HS_WR_TO_CNT[hs_wr_to_cnt]

uint16_t lpTxDeviceReady_ByteClk

The min time the display device takes to process low-power write from master before it can continue doing other stuff. The timer starts when D-PHY enters stop state and measured in lane byte clock. LP_WR_TO_CNT[lp_wr_to_cnt]

struct _dsi_dpi_config

#include <fsl_mipi_dsi.h> MIPI DSI controller DPI interface configuration.

Public Members

uint8_t virtualChannel

Virtual channel. DPI_VCID[dpi_vcid]

dsi_dpi_color_coding_t colorCoding

DPI color coding. DPI_COLOR_CODING

uint8_t polarityFlags

OR'ed value of _dsi_dpi_polarity_flag that controls signal polarity. DPI_CFG_POL

bool enablelpSwitch

Enable return to low-power inside the VSA/VBP/VFP/VACT/HBP/HFP period when timing allows. VID_MODE_CFG[bit8-13]

bool enableAck

Enable the request for an acknowledge response at the end of a frame. VID_MODE_CFG[frame_bta_ack_en]

dsi_video_mode_t videoMode

Video mode. VID_MODE_CFG[vid_mode_type]

uint16_t pixelPayloadSize

Color bar pattern. VID_MODE_CFG[vpg_orientation][vpg_en][vpg_mode=0] The number of pixels in a single video packet. For 18-bit not loosely packed data types, this number must be a multiple of 4, for YCbCr data types, it must be a multiple of 2. Recommended to set to the line size (in pixels). VID_PKT_SIZE

uint16_t vsw

Number of lines in vertical sync width. VID_VSA_LINES

uint16_t vbp

Number of lines in vertical back porch. VID_VBP_LINES

uint16_t vfp

Number of lines in vertical front porch. VID_VFP_LINES

uint16_t panelHeight

Number of lines in vertical active area. VID_VACTIVE_LINES

uint16_t hsw

Horizontal sync width, in dpi pixel clock. VID_HSA_TIME

uint16_t hbp

Horizontal back porch, in dpi pixel clock. VID_HBP_TIME

uint16_t hfp

Horizontal front porch, in dpi pixel clock. VID_HLINE_TIME = (hsw+hbp+hfp+width)

struct __dsi_command_config

#include <fsl_mipi_dsi.h> MIPI DSI command mode configuration.

Public Members

uint32_t escClkFreq_Hz

Escape clock frequency in Hz.

uint16_t lpRxTo_Ns

Timeout value that triggers a low-power reception timeout contention detection. TO_CNT_CFG[lprx_to_cnt]

uint16_t hsTxTo_Ns

Timeout value that triggers a high-speed transmission timeout contention detection. In non-burst mode, the time should be larger than 1.1 times of one frame data transmission time, in burst mode it should be one line. TO_CNT_CFG[hstx_to_cnt]

uint16_t btaTo_Ns

The time period for which MIPI DSI host keeps the link still after completing a Bus Turnaround. BTA_TO_CNT[bta_to_cnt]

struct __dsi_dphy_config

#include <fsl_mipi_dsi.h> MIPI DSI D-PHY configuration.

Public Members

uint8_t numLanes

Number of lanes. The value range is from 1-4, lane 0-3. PHY_IF_CFG[n_lanes]

uint8_t tStopState_ByteClk

Minimum time that the PHY controller stays in stop state before a HS transmission. TODO in what unit? PHY_IF_CFG[phy_stop_wait_time]

uint16_t tClkHs2Lp_ByteClk

Maximum time that the D-PHY clock lane takes to go from high-speed to low-power in lane byte clock. PHY_TMR_LPCLK_CFG[phy_clkhs2lp_time]

uint16_t tClkLp2Hs_ByteClk

Maximum time that the D-PHY clock lane takes to go from low-power to high-speed in lane byte clock. PHY_TMR_LPCLK_CFG[phy_clklp2hs_time]

uint16_t tDataHs2Lp_ByteClk

Maximum time that the D-PHY data lane takes to go from high-speed to low-power in lane byte clock. PHY_TMR_CFG[phy_hs2lp_time]

uint16_t tDataLp2Hs_ByteClk

Maximum time that the D-PHY data lane takes to go from low-power to high-speed in lane byte clock. PHY_TMR_CFG[phy_lp2hs_time]

uint16_t maxRead_ByteClk

Maximum time required to perform a read command in lane byte clock. PHY_TMR_RD_CFG[max_rd_time]

struct _dsi_transfer

#include <fsl_mipi_dsi.h> Structure for the data transfer.

Public Members

uint8_t virtualChannel

Virtual channel.

dsi_tx_data_type_t txDataType

TX data type.

uint8_t flags

Flags to control the transfer, see *_dsi_transfer_flags*.

const uint8_t *txData

The TX data buffer.

uint8_t *rxData

The TX data buffer.

uint16_t txDataSize

Size of the TX data.

uint16_t rxDataSize

Size of the RX data.

bool sendDcsCmd

If set to true, the DCS command is specified by *dcsCmd*, otherwise the DCS command is included in the *txData*.

uint8_t dcsCmd

The DCS command to send, only valid when *sendDcsCmd* is true.

2.53 MIPI_DSI: MIPI DSI Host Controller

2.54 MSGINTR: Message Unit

2.55 MSGINTR Driver

2.56 MU: Messaging Unit

`uint32_t MU_GetInstance(MU_Type *base)`

Get the MU instance index.

Parameters

- `base` – MU peripheral base address.

Returns

MU instance index.

`void MU_Init(MU_Type *base)`

Initializes the MU module.

This function enables the MU clock only.

Parameters

- `base` – MU peripheral base address.

`void MU_Deinit(MU_Type *base)`

De-initializes the MU module.

This function disables the MU clock only.

Parameters

- `base` – MU peripheral base address.

`static inline void MU_SendMsgNonBlocking(MU_Type *base, uint32_t regIndex, uint32_t msg)`

Writes a message to the TX register.

This function writes a message to the specific TX register. It does not check whether the TX register is empty or not. The upper layer should make sure the TX register is empty before calling this function. This function can be used in ISR for better performance.

```
while (!(kMU_Tx0EmptyFlag & MU_GetStatusFlags(base))) { } Wait for TX0 register empty.
MU_SendMsgNonBlocking(base, kMU_MsgReg0, MSG_VAL); Write message to the TX0 register.
```

Parameters

- `base` – MU peripheral base address.
- `regIndex` – TX register index, see `mu_msg_reg_index_t`.
- `msg` – Message to send.

`status_t MU_SendMsg(MU_Type *base, uint32_t regIndex, uint32_t msg)`

Blocks to send a message.

This function waits until the TX register is empty and sends the message. If `MU1_BUSY_POLL_COUNT` is defined and non-zero, the function will timeout after the specified number of polling iterations and returns `kStatus_Timeout`.

Parameters

- `base` – MU peripheral base address.
- `regIndex` – MU message register, see `mu_msg_reg_index_t`.
- `msg` – Message to send.

Return values

- `kStatus_Success` – Message sent successfully.
- `kStatus_Timeout` – Timeout occurred while waiting for TX register to be empty.

Returns

`status_t`

`static inline uint32_t MU_ReceiveMsgNonBlocking(MU_Type *base, uint32_t regIndex)`

Reads a message from the RX register.

This function reads a message from the specific RX register. It does not check whether the RX register is full or not. The upper layer should make sure the RX register is full before calling this function. This function can be used in ISR for better performance.

```
uint32_t msg;
while (!(kMU_Rx0FullFlag & MU_GetStatusFlags(base)))
{
    } Wait for the RX0 register full.

msg = MU_ReceiveMsgNonBlocking(base, kMU_MsgReg0); Read message from RX0 register.
```

Parameters

- `base` – MU peripheral base address.
- `regIndex` – RX register index, see `mu_msg_reg_index_t`.

Returns

The received message.

`status_t MU_ReceiveMsgTimeout(MU_Type *base, uint32_t regIndex, uint32_t *readValue)`

Blocks to receive a message with timeout protection.

This function waits until the RX register is full and receives the message. If `MU1_BUSY_POLL_COUNT` is defined and non-zero, the function will timeout after the specified number of polling iterations and return `kStatus_Timeout`.

This function provides the same blocking behavior as `MU_ReceiveMsg()` but with additional timeout protection to prevent system hangs if the other core becomes unresponsive or if hardware issues occur.

Note: Both `MU_ReceiveMsg()` and `MU_ReceiveMsgTimeout()` are blocking functions. The difference is that this function includes timeout protection while `MU_ReceiveMsg()` waits indefinitely.

Parameters

- `base` – MU peripheral base address.
- `regIndex` – RX register index, see `mu_msg_reg_index_t`.
- `readValue` – Pointer to store the received message.

Return values

- `kStatus_Success` – Message received successfully.
- `kStatus_InvalidArgument` – Invalid `readValue` pointer.
- `kStatus_Timeout` – Timeout occurred while waiting for RX register to be full.

Returns

`status_t`

uint32_t MU_ReceiveMsg(MU_Type *base, uint32_t regIndex)

Blocks to receive a message (infinite wait, no timeout protection).

This function waits until the RX register is full and receives the message. This function will wait indefinitely until a message is received.

Note: Both MU_ReceiveMsg() and MU_ReceiveMsgTimeout() are blocking functions. The difference is that MU_ReceiveMsgTimeout() includes timeout protection while this function waits indefinitely.

Warning: This function does not include timeout protection and may cause system hangs if the other core becomes unresponsive. For applications requiring timeout protection, use MU_ReceiveMsgTimeout() instead.

Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see mu_msg_reg_index_t.

Returns

The received message.

static inline void MU_SetFlagsNonBlocking(MU_Type *base, uint32_t flags)

Sets the 3-bit MU flags reflect on the other MU side.

This function sets the 3-bit MU flags directly. Every time the 3-bit MU flags are changed, the status flag kMU_FlagsUpdatingFlag asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag kMU_FlagsUpdatingFlag is cleared by hardware. During the flags updating period, the flags cannot be changed. The upper layer should make sure the status flag kMU_FlagsUpdatingFlag is cleared before calling this function.

```
while (kMU_FlagsUpdatingFlag & MU_GetStatusFlags(base))
{
    } Wait for previous MU flags updating.
```

```
MU_SetFlagsNonBlocking(base, 0U); Set the mU flags.
```

Parameters

- base – MU peripheral base address.
- flags – The 3-bit MU flags to set.

status_t MU_SetFlags(MU_Type *base, uint32_t flags)

brief Blocks setting the 3-bit MU flags reflect on the other MU side.

This function blocks setting the 3-bit MU flags. Every time the 3-bit MU flags are changed, the status flag kMU_FlagsUpdatingFlag asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag kMU_FlagsUpdatingFlag is cleared by hardware. During the flags updating period, the flags cannot be changed. This function waits for the MU status flag kMU_FlagsUpdatingFlag cleared and sets the 3-bit MU flags.

If MU1_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

return status_t retval kStatus_Success Flags were set successfully. retval kStatus_Timeout Timeout occurred while waiting for flags to update.

Parameters

- base – MU peripheral base address.
- flags – The 3-bit MU flags to set.

static inline uint32_t MU_GetFlags(MU_Type *base)

Gets the current value of the 3-bit MU flags set by the other side.

This function gets the current 3-bit MU flags on the current side.

Parameters

- base – MU peripheral base address.

Returns

flags Current value of the 3-bit flags.

uint32_t MU_GetStatusFlags(MU_Type *base)

Gets the MU status flags.

This function returns the bit mask of the MU status flags. See `_mu_status_flags`.

```
uint32_t flags;
flags = MU_GetStatusFlags(base); Get all status flags.
if (kMU_Tx0EmptyFlag & flags)
{
    The TX0 register is empty. Message can be sent.
    MU_SendMsgNonBlocking(base, kMU_MsgReg0, MSG0_VAL);
}
if (kMU_Tx1EmptyFlag & flags)
{
    The TX1 register is empty. Message can be sent.
    MU_SendMsgNonBlocking(base, kMU_MsgReg1, MSG1_VAL);
}
```

If there are more than 4 general purpose interrupts, use `MU_GetGeneralPurposeStatusFlags`.

Parameters

- base – MU peripheral base address.

Returns

Bit mask of the MU status flags, see `_mu_status_flags`.

static inline uint32_t MU_GetInterruptsPending(MU_Type *base)

Gets the MU IRQ pending status of enabled interrupts.

This function returns the bit mask of the pending MU IRQs of enabled interrupts. Only these flags are checked. kMU_Tx0EmptyFlag kMU_Tx1EmptyFlag kMU_Tx2EmptyFlag kMU_Tx3EmptyFlag kMU_Rx0FullFlag kMU_Rx1FullFlag kMU_Rx2FullFlag kMU_Rx3FullFlag kMU_GenInt0Flag kMU_GenInt1Flag kMU_GenInt2Flag kMU_GenInt3Flag

Parameters

- base – MU peripheral base address.

Returns

Bit mask of the MU IRQs pending.

```
static inline void MU_ClearStatusFlags(MU_Type *base, uint32_t flags)
```

Clears the specific MU status flags.

This function clears the specific MU status flags. The flags to clear should be passed in as bit mask. See `_mu_status_flags`.

Clear general interrupt 0 and general interrupt 1 pending flags.
`MU_ClearStatusFlags(base, kMU_GenInt0Flag | kMU_GenInt1Flag);`

If there are more than 4 general purpose interrupts, use `MU_ClearGeneralPurposeStatusFlags`.

Parameters

- base – MU peripheral base address.
- flags – Bit mask of the MU status flags. See `_mu_status_flags`. Only the following flags can be cleared by software (if applicable for particular device), other flags are cleared by hardware:
 - kMU_GenInt0Flag
 - kMU_GenInt1Flag
 - kMU_GenInt2Flag
 - kMU_GenInt3Flag
 - kMU_MuResetInterruptFlag
 - kMU_OtherSideEnterRunInterruptFlag
 - kMU_OtherSideEnterHaltInterruptFlag
 - kMU_OtherSideEnterWaitInterruptFlag
 - kMU_OtherSideEnterStopInterruptFlag
 - kMU_OtherSideEnterPowerDownInterruptFlag
 - kMU_ResetAssertInterruptFlag
 - kMU_HardwareResetInterruptFlag

```
static inline void MU_EnableInterrupts(MU_Type *base, uint32_t interrupts)
```

Enables the specific MU interrupts.

This function enables the specific MU interrupts. The interrupts to enable should be passed in as bit mask. See `_mu_interrupt_enable`.

Enable general interrupt 0 and TX0 empty interrupt.
`MU_EnableInterrupts(base, kMU_GenInt0InterruptEnable | kMU_Tx0EmptyInterruptEnable);`

If there are more than 4 general purpose interrupts, use `MU_EnableGeneralPurposeInterrupts`.

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the MU interrupts. See `_mu_interrupt_enable`.

```
static inline void MU_DisableInterrupts(MU_Type *base, uint32_t interrupts)
```

Disables the specific MU interrupts.

This function disables the specific MU interrupts. The interrupts to disable should be passed in as bit mask. See `_mu_interrupt_enable`.

Disable general interrupt 0 and TX0 empty interrupt.
`MU_DisableInterrupts(base, kMU_GenInt0InterruptEnable | kMU_Tx0EmptyInterruptEnable);`

If there are more than 4 general purpose interrupts, use `MU_DisableGeneralPurposeInterrupts`.

Parameters

- `base` – MU peripheral base address.
- `interrupts` – Bit mask of the MU interrupts. See `_mu_interrupt_enable`.

`status_t MU_TriggerInterrupts(MU_Type *base, uint32_t interrupts)`

Triggers interrupts to the other core.

This function triggers the specific interrupts to the other core. The interrupts to trigger are passed in as bit mask. See `_mu_interrupt_trigger`. The MU should not trigger an interrupt to the other core when the previous interrupt has not been processed by the other core. This function checks whether the previous interrupts have been processed. If not, it returns an error.

```
if (kStatus_Success != MU_TriggerInterrupts(base, kMU_GenInt0InterruptTrigger | kMU_
↪ GenInt2InterruptTrigger))
{
    Previous general purpose interrupt 0 or general purpose interrupt 2
    has not been processed by the other core.
}
```

If there are more than 4 general purpose interrupts, use `MU_TriggerGeneralPurposeInterrupts`.

Parameters

- `base` – MU peripheral base address.
- `interrupts` – Bit mask of the interrupts to trigger. See `_mu_interrupt_trigger`.

Return values

- `kStatus_Success` – Interrupts have been triggered successfully.
- `kStatus_Fail` – Previous interrupts have not been accepted.

`static inline void MU_EnableGeneralPurposeInterrupts(MU_Type *base, uint32_t interrupts)`

Enables the MU general purpose interrupts.

This function enables the MU general purpose interrupts. The interrupts to enable should be passed in as bit mask of `mu_general_purpose_interrupt_t`. The function `MU_EnableInterrupts` only support general interrupt 0~3, this function supports all general interrupts.

For example, to enable general purpose interrupt 0 and 3, use like this:

```
MU_EnableGeneralPurposeInterrupts(MU, kMU_GeneralPurposeInterrupt0 | kMU_
↪ GeneralPurposeInterrupt3);
```

Parameters

- `base` – MU peripheral base address.
- `interrupts` – Bit mask of the MU general purpose interrupts, see `mu_general_purpose_interrupt_t`.

`static inline void MU_DisableGeneralPurposeInterrupts(MU_Type *base, uint32_t interrupts)`

Disables the MU general purpose interrupts.

This function disables the MU general purpose interrupts. The interrupts to disable should be passed in as bit mask of `mu_general_purpose_interrupt_t`. The function `MU_DisableInterrupts` only support general interrupt 0~3, this function supports all general interrupts.

For example, to disable general purpose interrupt 0 and 3, use like this:

```
MU_EnableGeneralPurposeInterrupts(MU, kMU_GeneralPurposeInterrupt0 | kMU_
↳GeneralPurposeInterrupt3);
```

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the MU general purpose interrupts. see `mu_general_purpose_interrupt_t`.

static inline uint32_t MU_GetGeneralPurposeStatusFlags(MU_Type *base)

Gets the MU general purpose interrupt status flags.

This function returns the bit mask of the MU general purpose interrupt status flags. MU_GetStatusFlags can only get general purpose interrupt status 0~3, this function can get all general purpose interrupts status.

This example shows to check whether general purpose interrupt 0 and 3 happened.

```
uint32_t flags;
flags = MU_GetGeneralPurposeStatusFlags(base);
if (kMU_GeneralPurposeInterrupt0 & flags)
{
}
if (kMU_GeneralPurposeInterrupt3 & flags)
{
}
```

Parameters

- base – MU peripheral base address.

Returns

Bit mask of the MU general purpose interrupt status flags.

static inline void MU_ClearGeneralPurposeStatusFlags(MU_Type *base, uint32_t flags)

Clear the MU general purpose interrupt status flags.

This function clears the specific MU general purpose interrupt status flags. The flags to clear should be passed in as bit mask. `mu_general_purpose_interrupt_t_mu_status_flags`.

Example to clear general purpose interrupt 0 and general interrupt 1 pending flags.

```
MU_ClearGeneralPurposeStatusFlags(base, kMU_GeneralPurposeInterrupt0 | kMU_
↳GeneralPurposeInterrupt1);
```

Parameters

- base – MU peripheral base address.
- flags – Bit mask of the MU general purpose interrupt status flags. See `mu_general_purpose_interrupt_t`.

static inline uint32_t MU_GetRxStatusFlags(MU_Type *base)

Return the RX status flags in reverse numerical order.

This function return the RX status flags in reverse order. Note: RFn bits of SR[3-0](mu status register) are mapped in ascending numerical order: RF0 -> SR[0] RF1 -> SR[1] RF2 -> SR[2] RF3 -> SR[3] This function will return these bits in reverse numerical order(RF3->RF1) to comply with MU_GetRxStatusFlags() of mu driver. See MU_GetRxStatusFlags() from drivers/mu/fsl_mu.h

```
status_reg = MU_GetRxStatusFlags(base);
```

Parameters

- base – MU peripheral base address.

Returns

MU RX status flags in reverse order

status_t MU_TriggerGeneralPurposeInterrupts(MU_Type *base, uint32_t interrupts)

Triggers general purpose interrupts to the other core.

This function triggers the specific general purpose interrupts to the other core. The interrupts to trigger are passed in as bit mask. See `mu_general_purpose_interrupt_t`. The MU should not trigger an interrupt to the other core when the previous interrupt has not been processed by the other core. This function checks whether the previous interrupts have been processed. If not, it returns an error.

```
status_t status;
status = MU_TriggerGeneralPurposeInterrupts(base, kMU_GeneralPurposeInterrupt0 | kMU_
↪GeneralPurposeInterrupt2);

if (kStatus_Success != status)
{
    Previous general purpose interrupt 0 or general purpose interrupt 2
    has not been processed by the other core.
}
```

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the interrupts to trigger. See `mu_general_purpose_interrupt_t`.

Return values

- kStatus_Success – Interrupts have been triggered successfully.
- kStatus_Fail – Previous interrupts have not been accepted.

void MU_BootOtherCore(MU_Type *base, mu_core_boot_mode_t mode)

Boots the other core.

This function boots the other core with a boot configuration.

Parameters

- base – MU peripheral base address.
- mode – The other core boot mode.

void MU_HoldOtherCoreReset(MU_Type *base)

Holds the other core reset.

This function causes the other core to be held in reset following any reset event.

Parameters

- base – MU peripheral base address.

static inline status_t MU_ResetBothSides(MU_Type *base)

Resets the MU for both A side and B side.

This function resets the MU for both A side and B side. Before reset, it is recommended to interrupt processor B, because this function may affect the ongoing processor B programs.

If MU1_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations if waiting for the other side to come out of reset takes too long.

Note: For some platforms, only MU side A could use this function, check reference manual for details.

Parameters

- base – MU peripheral base address.

Return values

- kStatus_Success – The MU was reset successfully.
- kStatus_Timeout – Timeout occurred while waiting for the other side to come out of reset.

Returns

status_t

status_t MU_HardwareResetOtherCore(MU_Type *base, bool waitReset, bool holdReset, mu_core_boot_mode_t bootMode)

Hardware reset the other core.

This function resets the other core, the other core could mask the hardware reset by calling MU_MaskHardwareReset. The hardware reset mask feature is only available for some platforms. This function could be used together with MU_BootOtherCore to control the other core reset workflow.

If MU1_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout if waiting for the other core to enter or exit reset takes too long.

Example 1: Reset the other core, and no hold reset

```
MU_HardwareResetOtherCore(MU_A, true, false, bootMode);
```

In this example, the core at MU side B will reset with the specified boot mode.

Example 2: Reset the other core and hold it, then boot the other core later. Here the other core enters reset, and the reset is hold

```
MU_HardwareResetOtherCore(MU_A, true, true, modeDontCare);
```

Current core boot the other core when necessary.

```
MU_BootOtherCore(MU_A, bootMode);
```

Note: The feature waitReset, holdReset, and bootMode might be not supported for some platforms. waitReset is only available for platforms that FSL_FEATURE_MU_NO_CORE_STATUS not defined as 1 and FSL_FEATURE_MU_HAS_RESET_ASSERT_INT not defined as 0. holdReset is only available for platforms that FSL_FEATURE_MU_HAS_RSTH not defined as 0. bootMode is only available for platforms that FSL_FEATURE_MU_HAS_BOOT not defined as 0.

Parameters

- base – MU peripheral base address.

- `waitReset` – Wait the other core enters reset. Only work when there is `CSSR0[RAIP]`.
 - `true`: Wait until the other core enters reset, if the other core has masked the hardware reset, then this function will be blocked.
 - `false`: Don't wait the reset.
- `holdReset` – Hold the other core reset or not. Only work when there is `CCR0[RSTH]`.
 - `true`: Hold the other core in reset, this function returns directly when the other core enters reset.
 - `false`: Don't hold the other core in reset, this function waits until the other core out of reset.
- `bootMode` – Boot mode of the other core, if `holdReset` is true, this parameter is useless.

Return values

- `kStatus_Success` – The other core was reset successfully.
- `kStatus_Timeout` – Timeout occurred while waiting for the other core to enter or exit reset.

Returns

`status_t`

`FSL_MU_DRIVER_VERSION`

MU driver version.

`enum _mu_status_flags`

MU status flags.

Values:

enumerator `kMU_Tx0EmptyFlag`

TX0 empty.

enumerator `kMU_Tx1EmptyFlag`

TX1 empty.

enumerator `kMU_Tx2EmptyFlag`

TX2 empty.

enumerator `kMU_Tx3EmptyFlag`

TX3 empty.

enumerator `kMU_Rx0FullFlag`

RX0 full.

enumerator `kMU_Rx1FullFlag`

RX1 full.

enumerator `kMU_Rx2FullFlag`

RX2 full.

enumerator `kMU_Rx3FullFlag`

RX3 full.

enumerator `kMU_GenInt0Flag`

General purpose interrupt 0 pending.

enumerator kMU_GenInt1Flag

General purpose interrupt 1 pending.

enumerator kMU_GenInt2Flag

General purpose interrupt 2 pending.

enumerator kMU_GenInt3Flag

General purpose interrupt 3 pending.

enumerator kMU_RxFullPendingFlag

Any RX full flag is pending.

enumerator kMU_TxEmptyPendingFlag

Any TX empty flag is pending.

enumerator kMU_GenIntPendingFlag

Any general interrupt flag is pending.

enumerator kMU_EventPendingFlag

MU event pending.

enumerator kMU_FlagsUpdatingFlag

MU flags update is on-going.

enumerator kMU_MuInResetFlag

MU of any side is in reset.

enumerator kMU_MuResetInterruptFlag

The other side initializes MU reset.

enum _mu_interrupt_enable

MU interrupt source to enable.

Values:

enumerator kMU_Tx0EmptyInterruptEnable

TX0 empty.

enumerator kMU_Tx1EmptyInterruptEnable

TX1 empty.

enumerator kMU_Tx2EmptyInterruptEnable

TX2 empty.

enumerator kMU_Tx3EmptyInterruptEnable

TX3 empty.

enumerator kMU_Rx0FullInterruptEnable

RX0 full.

enumerator kMU_Rx1FullInterruptEnable

RX1 full.

enumerator kMU_Rx2FullInterruptEnable

RX2 full.

enumerator kMU_Rx3FullInterruptEnable

RX3 full.

enumerator kMU_GenInt0InterruptEnable

General purpose interrupt 0.

enumerator kMU_GenInt1InterruptEnable

General purpose interrupt 1.

enumerator kMU_GenInt2InterruptEnable

General purpose interrupt 2.

enumerator kMU_GenInt3InterruptEnable

General purpose interrupt 3.

enumerator kMU_MuResetInterruptEnable

The other side initializes MU reset.

enum _mu_interrupt_trigger

MU interrupt that could be triggered to the other core.

Values:

enumerator kMU_GenInt0InterruptTrigger

General purpose interrupt 0.

enumerator kMU_GenInt1InterruptTrigger

General purpose interrupt 1.

enumerator kMU_GenInt2InterruptTrigger

General purpose interrupt 2.

enumerator kMU_GenInt3InterruptTrigger

General purpose interrupt 3.

enum _mu_msg_reg_index

MU message register index.

Values:

enumerator kMU_MsgReg0

Message register 0.

enumerator kMU_MsgReg1

Message register 1.

enumerator kMU_MsgReg2

Message register 2.

enumerator kMU_MsgReg3

Message register 3.

enum _mu_general_purpose_interrupt

MU general purpose interrupts.

Values:

enumerator kMU_GeneralPurposeInterrupt0

General purpose interrupt 0

enumerator kMU_GeneralPurposeInterrupt1

General purpose interrupt 1

enumerator kMU_GeneralPurposeInterrupt2

General purpose interrupt 2

enumerator kMU_GeneralPurposeInterrupt3

General purpose interrupt 3

```
typedef enum _mu_msg_reg_index mu_msg_reg_index_t
```

MU message register index.

```
typedef enum _mu_general_purpose_interrupt mu_general_purpose_interrupt_t
```

MU general purpose interrupts.

```
MU_CORE_INTR(intr)
```

```
MU_MISC_INTR(intr)
```

```
MU_TX_INTR(intr)
```

```
MU_RX_INTR(intr)
```

```
MU_GI_INTR(intr)
```

```
MU_GET_CORE_INTR(intrs)
```

```
MU_GET_TX_INTR(intrs)
```

```
MU_GET_RX_INTR(intrs)
```

```
MU_GET_GI_INTR(intrs)
```

```
MU_CORE_FLAG(flag)
```

```
MU_STAT_FLAG(flag)
```

```
MU_TX_FLAG(flag)
```

```
MU_RX_FLAG(flag)
```

```
MU_GI_FLAG(flag)
```

```
MU_GET_CORE_FLAG(flags)
```

```
MU_GET_STAT_FLAG(flags)
```

```
MU_GET_TX_FLAG(flags)
```

```
MU_GET_RX_FLAG(flags)
```

```
MU_GET_GI_FLAG(flags)
```

```
MU1_BUSY_POLL_COUNT
```

Maximum polling iterations for MU waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the MU code before timing out and returning an error.

It applies to all waiting loops in MU driver, such as waiting for TX register to be empty or waiting for RX register to be full.

This is a count of loop iterations, not a time-based value.

If defined as 0, polling loops will continue indefinitely until their exit condition is met, which could potentially cause the system to hang if a core becomes unresponsive.

2.57 NETC driver

Status code for the NETC module.

Values:

enumerator kStatus_NETC_RxFrameEmpty

Rx BD ring empty.

enumerator kStatus_NETC_RxTsrResp

Rx timestamp reference response

enumerator kStatus_NETC_RxFrameError

Rx frame error.

enumerator kStatus_NETC_TxFrameOverLen

Tx frame over length.

enumerator kStatus_NETC_LackOfResource

Lack of resources to configure certain features.

enumerator kStatus_NETC_Unsupported

Unsupported operation/feature.

enumerator kStatus_NETC_RxHRZeroFrame

Rx frame host reason is zero

enumerator kStatus_NETC_RxHRNotZeroFrame

Rx frame host reason is not zero

enumerator kStatus_NETC_NotFound

No entry found in hardware tables

enumerator kStatus_NETC_EntryExists

An entry already exists in hardware tables

enum _netc_ep_event

Defines the common interrupt event for callback use.

Values:

enumerator kNETC_EPRxEvent

EP Rx interrupt event.

enumerator kNETC_EPTxEvent

EP Tx interrupt event.

enum _netc_ep_tx_status

Status for the transmit buffer descriptor.

Values:

enumerator kNETC_EPTxSuccess

Success transmission.

enumerator kNETC_EPTxProgramErr

Error exists in either the Tx BD, the Tx ring registers, or both.

enumerator kNETC_EPTxTsdDrop

The time defined in TX_START expired before frame could be transmitted.

enumerator kNETC_EPTxFrameSizeErr

Frame size error.

enumerator kNETC_EPTxNullAddr

Null address.

enumerator kNETC_EPTxInvalidLength

Invalid frame/buffer/chain length.

enumerator kNETC_EPTxSrcMacSpoofingDetect

Source MAC address spoofing detected.

enumerator kNETC_EPTxPortRestDrop

Frame dropped due to port reset.

enumerator kNETC_EPTxPortDisableDrop

Frame dropped due to port disable.

enumerator kNETC_EPTxVlanTpidDrop

VLAN TPID not allowed.

enumerator kNETC_EPTxSmsiParamErr

Programming error in buffer descriptor used for direct switch enqueue.

enumerator kNETC_EPTxFrameGateErr

Frame too large for time gating window.

enumerator kNETC_EPTxAxiReadErr

AXI read error.

enumerator kNETC_EPTxAxiWriteErr

AXI write error.

enumerator kNETC_EPTxMultiBitECCErr

Frame not transmitted(dropped) due to a multi-bit ECC error detected.

enumerator kNETC_EPTxParityErr

Parity error.

enumerator kNETC_EPTxSwCongestion

Frame dropped due to switch congestion.

enum __netc_vlan_tpid_select

Ethernet VLAN Tag protocol identifier.

Values:

enumerator kNETC_StanCvlan

0x8100.

enumerator kNETC_StanSvlan

0x88A8.

enumerator kNETC_CustomVlan1

CVLANR1[ETYPE]

enumerator kNETC_CustomVlan2

CVLANR2[ETYPE]

enum __netc_packet_type

Ethernet packet type enumerator.

Values:

enumerator kNETC_PacketUnicast

enumerator kNETC_PacketMulticast

enumerator kNETC_PacketBroadcast

enum __netc_host_reason

Host reason.

Values:

enumerator kNETC_RegularFrame

enumerator kNETC_IngressMirror

enumerator kNETC_MACLearning

enumerator kNETC_TimestampResp

enumerator kNETC_SoftwareDefHR0

enumerator kNETC_SoftwareDefHR1

enumerator kNETC_SoftwareDefHR2

enumerator kNETC_SoftwareDefHR3

enumerator kNETC_SoftwareDefHR4

enumerator kNETC_SoftwareDefHR5

enumerator kNETC_SoftwareDefHR6

enumerator kNETC_SoftwareDefHR7

enum _netc_msix_vector_ctrl

MSIX vector control field.

Values:

enumerator kNETC_MsixIntrMaskBit

MSIX vector control interrupt mask bit.

enum _netc_tx_ext_flags

METC Extension Transmit Buffer Descriptor Extension flags field.

Values:

enumerator kNETC_TxExtVlanInsert

Enable VLAN insert.

enumerator kNETC_TxExtTwoStepTs

Enable two-step timestamp offload.

typedef enum _netc_ep_event netc_ep_event_t

Defines the common interrupt event for callback use.

typedef enum _netc_ep_tx_status netc_ep_tx_status_t

Status for the transmit buffer descriptor.

typedef struct _netc_vlan netc_vlan_t

VLAN tag struct.

typedef enum _netc_vlan_tpid_select netc_vlan_tpid_select_t

Ethernet VLAN Tag protocol identifier.

typedef enum _netc_packet_type netc_packet_type_t

Ethernet packet type enumerator.

typedef enum _netc_host_reason netc_host_reason_t

Host reason.

typedef struct _ep_buffer_struct netc_buffer_struct_t

Buffer structure. Driver can send/receive one frame spread across multiple buffers.

```
typedef struct _ep_frame_struct netc_frame_struct_t
```

Frame structure for single Tx/Rx frame.

```
typedef struct _netc_frame_attr_struct netc_frame_attr_t
```

Frame attribute struct.

```
typedef struct _netc_tx_frame_info_struct netc_tx_frame_info_t
```

Frame attribute structure.

```
typedef enum _netc_msix_vector_ctrl netc_msix_vector_ctrl_t
```

MSIX vector control field.

```
typedef struct _netc_msix_entry netc_msix_entry_t
```

NETC MSIX entry structure.

```
typedef enum _netc_tx_ext_flags netc_tx_ext_flags_t
```

NETC Extension Transmit Buffer Descriptor Extension flags field.

```
FSL_NETC_DRIVER_VERSION
```

Driver Version.

```
NETC_ADDR_LOW_32BIT(x)
```

Macro to divides an address into a low 32 bits and a possible high 32 bits.

```
NETC_ADDR_HIGH_32BIT(x)
```

```
struct _netc_vlan
```

#include <fsl_netc.h> VLAN tag struct.

Public Members

```
uint32_t vid
```

Vlan Identifier.

```
uint32_t dei
```

Drop Eligible indicator.

```
uint32_t pcp
```

Priority.

```
uint32_t tpid
```

Tag protocol identifier.

```
struct _ep_buffer_struct
```

#include <fsl_netc.h> Buffer structure. Driver can send/receive one frame spread across multiple buffers.

Public Members

```
void *buffer
```

Buffer address.

```
uint16_t length
```

Buffer data length.

```
struct _ep_frame_struct
```

#include <fsl_netc.h> Frame structure for single Tx/Rx frame.

Public Members

netc_buffer_struct_t *buffArray

Buffer array. Tx: [in]App sets, Rx: [in/out]App sets prepared array, driver sets back received buffers array.

uint16_t length

Buffer array length. Tx: [in]App sets, Rx: [in/out]App sets prepared array length, driver sets back received buffers array length.

struct __netc_frame_attr_struct

#include <fsl_netc.h> Frame attribute struct.

Public Members

bool isTsAvail

Rx frame timestamp is available or not.

bool isVlanExtracted

Rx frame VLAN header is available or not.

uint32_t timestamp

The timestamp of this Rx frame.

struct __netc_tx_frame_info_struct

#include <fsl_netc.h> Frame attribute structure.

Public Members

bool isTsAvail

Tx frame timestamp is available or not.

uint32_t timestamp

The timestamp of this Tx frame, valid when isTsAvail is true.

bool isTxTsIdAvail

Switch port Tx frame timestamp Identifier is available or not.

uint16_t txtsid

The Transmit Timestamp Identifier, valid when isTsIdAvail is true, use for Switch management ENETC direct frame which has specified a timestamp request.

void *context

Private context provided by the user.

netc_ep_tx_status_t status

Transmit status.

struct __netc_msix_entry

#include <fsl_netc.h> NETC MSIX entry structure.

Public Members

uint64_t msgAddr

Message address.

uint32_t msgData

Message data.

uint32_t control

Vector control, netc_msix_vector_ctrl_t.

2.58 Abbreviation in NETC driver

2.59 API layer

2.60 NETC Endpoint (EP) Driver

2.61 Endpoint (EP) Generic Configuration

```
typedef struct _ep_handle ep_handle_t
```

Endpoint handle.

```
typedef status_t (*ep_reclaim_cb_t)(ep_handle_t *handle, uint8_t ring, netc_tx_frame_info_t *frameInfo, void *userData)
```

Callback for reclaimed tx frames.

```
typedef void (*ep_rx_alloc_cb_t)(ep_handle_t *handle, uint8_t ring, uint32_t length, void *userData)
```

Defines the EP Rx memory buffer alloc function pointer.

```
typedef void (*ep_rx_free_cb_t)(ep_handle_t *handle, uint8_t ring, void *address, void *userData)
```

Defines the EP Rx memory buffer free function pointer.

```
typedef status_t (*ep_get_link_status_cb_t)(ep_handle_t *handle, uint8_t *link)
```

Callback for getting link status.

```
typedef status_t (*ep_get_link_speed_cb_t)(ep_handle_t *handle, netc_hw_mii_speed_t *speed, netc_hw_mii_duplex_t *duplex)
```

Callback for getting link speed.

```
typedef status_t (*ep_preinit_vsi_cb_t)(netc_enetc_hw_t *hw, netc_hw_si_idx_t si)
```

Callback for vsi pre-init.

```
typedef struct _ep_config ep_config_t
```

Configuration for the endpoint handle.

```
typedef struct _ep_config_const ep_config_const_t
```

Configuration constant in handle.

```
status_t EP_Init(ep_handle_t *handle, uint8_t *macAddr, const ep_config_t *config, const netc_bdr_config_t *bdrConfig)
```

Initialize the endpoint with specified station interface.

Each station interface needs to call this API. In the case of a virtual station interface it's necessary that the physical station interface has been initialized beforehand.

Parameters

- handle –
- macAddr – Primary MAC address
- config – The user configuration

- bdrConfig – Array of buffer configurations (for each queue/ring)

Returns

status_t

status_t EP_Deinit(*ep_handle_t* *handle)

De-initialize the endpoint.

Parameters

- handle –

Returns

status_t

status_t EP_GetDefaultConfig(*ep_config_t* *config)

Get the default configuration.

Parameters

- config –

Returns

status_t

status_t EP_Up(*ep_handle_t* *handle, *netc_hw_mii_speed_t* speed, *netc_hw_mii_duplex_t* duplex)

Enable MAC transmission/reception To be called when the PHY link is up.

Parameters

- handle –
- speed –
- duplex –

Returns

status_t

status_t EP_Down(*ep_handle_t* *handle)

Disable MAC transmission/reception To be called when the PHY link is down.

Note: Must ensure all active Tx rings finish current transmission before call this API.

Parameters

- handle –

Returns

status_t

status_t EP_SetPrimaryMacAddr(*ep_handle_t* *handle, uint8_t *macAddr)

Set the Primary MAC address.

Parameters

- handle –
- macAddr –

Returns

status_t

static inline void EP_SetPortSpeed(*ep_handle_t* *handle, uint16_t pSpeed)

Set EP port speed.

Parameters

- handle –
- pSpeed –

struct __ep_config

#include <fsl_netc_endpoint.h> Configuration for the endpoint handle.

Public Members

netc_hw_si_idx_t si

Station interface index.

netc_hw_enetc_si_config_t siConfig

Station interface configuration.

uint8_t txPrioToTC[8]

Tx BD ring priority to Tx traffic class queue index mapping, range in TC0 ~ TC7.

netc_port_tx_tc_config_t txTcCfg[8]

Tx traffic class related configuration, valid only on ENETC 0.

netc_ep_psfpcfg_t psfpCfg

PSFP configuration, cover the ISI key construction profile and port ingress stream identification configuration.

bool enOuterAsInner

Enable use outer VLAN tag as the inner tag if only one tag is found.

netc_enetc_native_vlan_config_t rxOuterVLANCfg

Port outer native VLAN config.

netc_enetc_native_vlan_config_t rxInnerVLANCfg

Port inner native VLAN config.

netc_enetc_parser_config_t parserCfg

ENETC parser configuration.

uint32_t pauseOnThr

ENETC Port pause ON threshold value, value 0 means disables pause generation.

uint32_t pauseOffThr

ENETC Port pause OFF threshold value, value 0 means disables pause generation.

netc_msix_entry_t *msixEntry

MSIX table entry array.

uint8_t entryNum

MSIX entry number.

uint8_t cmdBdEntryIdx

MSIX entry index of command BD ring interrupt.

uint8_t siComEntryIdx

MSIX entry index of PSI-VSI communication interrupt.

uint8_t timerSyncEntryIdx

MSIX entry index of timer synchronous state change interrupt.

ep_reclaim_cb_t reclaimCallback

Callback for reclaimed Tx frames.

`void *userData`
User data, return in callback.

`bool rxCacheMaintain`
Enable/Disable Rx buffer cache maintain in driver.

`bool txCacheMaintain`
Enable/Disable Tx buffer cache maintain in driver.

`bool rxZeroCopy`
Enable/Disable zero-copy receive mode.

`ep_rx_alloc_cb_t rxBuffAlloc`
Callback function to alloc memory, must be provided for zero-copy Rx.

`ep_rx_free_cb_t rxBuffFree`
Callback function to free memory, must be provided for zero-copy Rx.

`ep_preinit_vsi_cb_t preinitVsi`
Callback function to pre-init VSI

`netc_cmd_bdr_config_t cmdBdrConfig`
Command BD ring configuration.

`struct __ep_config_const`
`#include <fsl_netc_endpoint.h>` Configuration constant in handle.

Public Members

`netc_hw_si_idx_t si`
Station interface index.

`uint32_t rxRingUse`
Number of Rx Rings to be used, when enable Rx ring group, this equal to the sum of all Rx group rings.

`uint32_t txRingUse`
Number of Tx Rings to be used, note that when SI is Switch management ENETC SI, the number not include Tx ring 0.

`uint32_t rxBdrGroupNum`
Rx BD ring group number, range in 0 ~ 2.

`uint32_t ringPerBdrGroup`
The ring number in every Rx BD ring group, range in 1 ~ 8, active when rxBdrGroupNum not equal zero.

`bool rxCacheMaintain`
Enable/Disable Rx buffer cache maintain in driver.

`bool txCacheMaintain`
Enable/Disable Tx buffer cache maintain in driver.

`bool rxZeroCopy`
Enable/Disable zero-copy receive mode.

`uint8_t entryNum`
MSIX entry number.

`ep_reclaim_cb_t reclaimCallback`
Callback for reclaimed Tx frames.

```

void *userData
    User data, return in callback.

ep_rx_alloc_cb_t rxBuffAlloc
    Callback function to alloc memory, must be provided for zero-copy Rx.

ep_rx_free_cb_t rxBuffFree
    Callback function to free memory, must be provided for zero-copy Rx.

struct _ep_handle
    #include <fsl_netc_endpoint.h> Handle for the endpoint Private internal data.

```

Public Members

```

netc_enetc_hw_t hw
    Hardware register map resource.

netc_enetc_cap_t capability
    ENETC capability.

ep_config_const_t cfg
    Endpoint configuration constant.

uint8_t ringShift
    Endpoint Tx ring shift.

netc_rx_bdr_t rxBdRing[1]
    Receive buffer descriptor ring.

netc_tx_bdr_t txBdRing[1]
    Transmit buffer descriptor ring.

netc_cmd_bdr_t cmdBdRing
    Command BD ring handle for endpoint.

uint8_t unicastHashCount[64]
    Unicast hash index collisions counter.

uint8_t multicastHashCount[64]
    Multicast hash index collisions counter.

uint8_t vlanHashCount[64]
    VLAN hash index collisions counter.

uint8_t macFilterCount[64]
    mac address filter index collisions counter.

uint8_t vlanFilterCount[64]
    vlan address filter index collisions counter.

ep_get_link_status_cb_t getLinkStatus
    Callback to get link status

ep_get_link_speed_cb_t getLinkSpeed
    Callback to get link speed

uint16_t vsiBitMapNotifyLinkStatus
    VSI bit map for link status notify

uint16_t vsiBitMapNotifyLinkSpeed
    VSI bit map for link speed notify

struct port

```

Public Members

netc_port_ethmac_t ethMac

Ethernet MAC configuration.

netc_port_common_t common

Port common configuration.

bool enableTg

Enable port time gate scheduling.

bool enPseudoMacTxPad

Enable pseudo MAC Port Transmit Padding, will pad the frame to a minimum of 60 bytes and append 4 octets of FCS.

2.62 Endpoint (EP) data path

typedef struct *netc_ep_ipf_config* netc_ep_ipf_config_t

Port Ingress Filter config.

typedef struct *netc_ep_psf_config* netc_ep_psf_config_t

PSFP config.

struct *_netc_ep_ipf_config*

#include <fsl_netc_endpoint.h> Port Ingress Filter config.

Public Members

netc_ipf_config_t dosCfg

Configuration for L2/3 DOS.

netc_port_ipf_config_t portConfig

Configuration for port connected to enetc peripheral.

struct *_netc_ep_psf_config*

#include <fsl_netc_endpoint.h> PSFP config.

2.63 Endpoint (EP) Interrupt Module

enum *_ep_interrupt_flag*

Interrupt enable/disable flags.

The value of the enumerator is not necessary match the bit in register. All interrupts in Endpoint are merged into this enum except the BDR specific interrupt. TODO SITMRIER

Values:

enumerator kNETC_EPPSIResetInterruptEnable

enumerator kNETC_EPPSIMsgRxInterruptEnable

typedef enum *_ep_interrupt_flag* ep_interrupt_flag_t

Interrupt enable/disable flags.

The value of the enumerator is not necessary match the bit in register. All interrupts in Endpoint are merged into this enum except the BDR specific interrupt. TODO SITMRIER

```
static inline void EP_CleanTxIntrFlags(ep_handle_t *handle, uint16_t txFrameIntrMask, uint16_t txThresIntrMask)
```

Clean the SI transmit interrupt flags.

Parameters

- handle – The EP handle.
- txFrameIntrMask – IPV value to be mapped, bit x represents ring x.
- txThresIntrMask – The Rx BD ring index to be mapped, bit x represents ring x.

```
static inline void EP_CleanRxIntrFlags(ep_handle_t *handle, uint32_t rxIntrMask)
```

Clean the SI receive interrupt flags.

Parameters

- handle – The EP handle.
- rxIntrMask – Rx interrupt bit mask, bit x represents ring x.

```
status_t EP_MsixSetGlobalMask(ep_handle_t *handle, bool mask)
```

Set the global MSIX mask status.

This function masks/unmasks global MSIX message. Mask - All of the vectors are masked, regardless of their per-entry mask bit states. Unmask - Each entry's mask status determines whether the vector is masked or not.

Parameters

- handle – The EP handle
- mask – The mask state. True: Mask, False: Unmask.

Returns

status_t

```
status_t EP_MsixSetEntryMask(ep_handle_t *handle, uint8_t entryIdx, bool mask)
```

Set the MSIX entry mask status for specified entry.

This function masks/unmasks MSIX message for specified entry.

Parameters

- handle – The EP handle
- entryIdx – The entry index in the table.
- mask – The mask state. True: Mask, False: Unmask.

Returns

status_t

```
status_t EP_MsixGetPendingStatus(ep_handle_t *handle, uint8_t pbaIdx, uint64_t *status)
```

Get the MSIX pending status in MSIX PBA table.

This function is to get the entry pending status from MSIX PBA table. If interrupt occurs but masked by vector control of entry, pending bit in PBA will be set.

Parameters

- handle – The EP handle
- pbaIdx – The index of PBA array with 64-bit unit.
- status – Pending status bit mask, bit n for entry n.

Returns

status_t

2.64 Endpoint (EP) Table Management Module

status_t EP_CmdBDRInit(*ep_handle_t* *handle, const *netc_cmd_bdr_config_t* *config)

Initialize endpoint command BD ring.

Parameters

- handle –
- config – The command BD ring configuration

Returns

status_t

status_t EP_CmdBDRDeinit(*ep_handle_t* *handle)

Deinit endpoint command BD ring.

Parameters

- handle –

Returns

status_t

2.65 Endpoint (EP) PSI/VSI

void EP_PsiEnableInterrupt(*ep_handle_t* *handle, uint32_t mask, bool enable)

PSI enables/disables specified interrupt.

Parameters

- handle – The EP handle.
- mask – The interrupt mask, refer to *netc_psi_msg_flags_t* which should be OR'd together.
- enable – Enable/Disable the interrupt.

uint32_t EP_PsiGetStatus(*ep_handle_t* *handle)

PSI gets interrupt event flag status.

Parameters

- handle – The EP handle.

Returns

The interrupt mask, refer to *netc_psi_msg_flags_t* which should be OR'd together.

void EP_PsiClearStatus(*ep_handle_t* *handle, uint32_t mask)

PSI clears interrupt event flag.

Parameters

- handle – The EP handle.
- mask – The interrupt mask, refer to *netc_psi_msg_flags_t* which should be OR'd together.

status_t EP_PsiSendMsg(*ep_handle_t* *handle, uint16_t msg, *netc_vsi_number_t* vsi)

PSI sends message to specified VSI(s)

Parameters

- handle – The EP handle.

- msg – The message to be sent.
- vsi – The VSI number.

Returns

status_t

bool EP_PsiCheckTxBusy(*ep_handle_t* *handle, *netc_vsi_number_t* vsi)

PSI checks Tx busy flag which should be cleaned when VSI receive the message data.

Parameters

- handle – The EP handle.
- vsi – The VSI number.

Returns

The busy status of specified VSI.

status_t EP_PsiSetRxBuffer(*ep_handle_t* *handle, *netc_vsi_number_t* vsi, uint64_t buffAddr)

PSI sets Rx buffer to receive message from specified VSI.

Note: The buffer memory size should be big enough for the message data from VSI

Parameters

- handle – The EP handle.
- vsi – The VSI number.
- buffAddr – The buffer address to store message data from VSI.

status_t EP_PsiGetRxMsg(*ep_handle_t* *handle, *netc_vsi_number_t* vsi, *netc_psi_rx_msg_t* *msgInfo)

PSI gets Rx message from specified VSI.

Parameters

- handle – The EP handle.
- vsi – The VSI number.
- msgInfo – The Rx message information.

void EP_VsiEnableInterrupt(*ep_handle_t* *handle, uint32_t mask, bool enable)

Enable VSI interrupt.

Parameters

- handle – The EP handle.
- mask – The interrupt mask, see *netc_vsi_msg_flags_t* which should be OR'd together.
- enable – Enable/Disable interrupt.

uint32_t EP_VsiGetStatus(*ep_handle_t* *handle)

Get VSI interrupt status.

Parameters

- handle – The EP handle.

Returns

A bitmask composed of *netc_vsi_msg_flags_t* enumerators OR'd together.

void EP_VsiClearStatus(*ep_handle_t* *handle, uint32_t mask)

Clear VSI interrupt status.

Parameters

- handle – The EP handle.
- mask – The interrupt mask, see *netc_vsi_msg_flags_t* which should be OR'd together.

status_t EP_VsiSendMsg(*ep_handle_t* *handle, uint64_t msgAddr, uint32_t msgLen)

VSI sends message to PSI.

Parameters

- handle – The EP handle.
- msgAddr – Address to store message ready to be sent, must be 64 bytes aligned.
- msgLen – The message length, must be 32 bytes aligned.

Returns

status_t

void EP_VsiCheckTxStatus(*ep_handle_t* *handle, *netc_vsi_msg_tx_status_t* *status)

Check VSI Tx status.

Parameters

- handle – The EP handle.
- status – The VSI Tx status structure.

status_t EP_VsiReceiveMsg(*ep_handle_t* *handle, uint16_t *msg)

VSI receives message from PSI.

Parameters

- handle – The EP handle.
- msg – The message from PSI.

Returns

status_t

2.66 Endpoint (EP) Ingress data path configuration

static inline *status_t* EP_RxParserConfig(*ep_handle_t* *handle, *netc_port_parser_config_t* *config)

Configure Parser in Receive Data Path.

Parameters

- handle –
- config –

Returns

status_t

static inline *status_t* EP_RxVlanCInit(*ep_handle_t* *handle, const *netc_vlan_classify_config_t* *config)

Configure the customer vlan type.

Parameters

- handle –

- config –

Returns

status_t

```
static inline status_t EP_RxVlanCConfigPort(ep_handle_t *handle,  
                                             netc_port_vlan_classify_config_t *config)
```

Configure the Accepted Vlan.

Parameters

- handle –
- config –

Returns

status_t

```
status_t EP_RxIPFInit(ep_handle_t *handle, netc_ep_ipf_config_t *config)
```

Enable / Disable Ingress Port Filtering.

Applied for both Switch and ENETC

Parameters

- handle –
- config – IPF general features

Returns

status_t

```
static inline uint32_t EP_RxIPFGetTableRemainWordNum(ep_handle_t *handle)
```

Get remaining available word number (words size is 6 bytes) of the ingress Port Filter Table.

Note: This is a ternary match table, and the entries can vary in size, from 2 to 14 words.

Parameters

- handle –

Returns

uint32_t

```
status_t EP_RxIPFAddTableEntry(ep_handle_t *handle, netc_tb_ipf_config_t *config, uint32_t  
                               *entryID)
```

Add an entry for the ingress Port Filter Table.

This function do an add & query with return hardware id which can be used as future query / delete / update.

Parameters

- handle –
- config – IPF instance configuration
- entryID – The table entry ID read out

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxIPFUpdateTableEntry(*ep_handle_t* *handle, uint32_t entryID, *netc_tb_ipf_cfge_t* *cfg)

Update entry in the ingress Port Filter Table.

Parameters

- handle –
- entryID –
- cfg –

Returns

status_t

Returns

See *netc_cmd_error_t*

status_t EP_RxIPFDeleteTableEntry(*ep_handle_t* *handle, uint32_t entryID)

Delete an entry for the ingress Port Filter Table.

Parameters

- handle –
- entryID – The table entry ID

Returns

status_t

status_t EP_RxIPFResetMatchCounter(*ep_handle_t* *handle, uint32_t entryID)

Reset the counter of an ingress port filter entry.

Parameters

- handle –
- entryID – The table entry ID

Returns

status_t

status_t EP_RxIPFGetMatchedCount(*ep_handle_t* *handle, uint32_t entryID, uint64_t *count)

Get the matched count for entry in IPF.

Parameters

- handle –
- entryID – The table entry ID
- count – A count of how many times this entry has been matched.

Returns

status_t

static inline *status_t* EP_RxPSFPInit(*ep_handle_t* *handle, const *netc_ep_psf_config_t* *config)

Init the ENETC PSFP, include.

Parameters

- handle –
- config –

Returns

status_t

static inline uint32_t EP_RxPSFPGetISITableRemainEntryNum(*ep_handle_t* *handle)

Get remaining available entry number (entry size is 24 bytes) of stream identification table.

Note: This is a Exact Match hash table, and it shares the remaining available entries with Ingress Stream Filter, table.

Parameters

- handle –

Returns

uint32_t

status_t EP_RxPSFPAddISITableEntry(*ep_handle_t* *handle, *netc_tb_isi_config_t* *config, uint32_t *entryID)

Add an entry into the stream identification table.

Parameters

- handle –
- config –
- entryID –

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxPSFPDelISITableEntry(*ep_handle_t* *handle, uint32_t entryID)

Delete an entry in the stream identification table.

Parameters

- handle –
- entryID –

Returns

status_t

Returns

See netc_cmd_error_t

static inline uint32_t EP_RxPSFPGetISTableRemainEntryNum(*ep_handle_t* *handle)

Get remaining available entry number of ingress stream table.

Note: This is a dynamic bounded index table, the remaining entry can't be zero before add entry into it

Parameters

- handle –

Returns

uint32_t

status_t EP_RxPSFPAddISTableEntry(*ep_handle_t* *handle, *netc_tb_is_config_t* *config)

Add an entry into the ingress stream table.

Parameters

- handle –
- config –

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxPSFPUUpdateISTableEntry(*ep_handle_t* *handle, *netc_tb_is_config_t* *config)

Update an entry in the ingress stream table.

Parameters

- handle –
- config –

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxPSFPDeISTableEntry(*ep_handle_t* *handle, uint32_t entryID)

Delete an entry in the stream identification table.

Parameters

- handle –
- entryID –

Returns

status_t

static inline uint32_t EP_RxPSFPGetISFTableRemainEntryNum(*ep_handle_t* *handle)

Get remaining available entry number (entry size is 24 bytes) of ingress stream filter table.

Note: This is a Exact Match hash table, and it shares the remaining available entries with Ingress Stream Identification table.

Parameters

- handle –

Returns

uint32_t

status_t EP_RxPSFPAddISFTableEntry(*ep_handle_t* *handle, *netc_tb_isf_config_t* *config, uint32_t *entryID)

Add an entry into the ingress stream filter table.

Parameters

- handle –
- config –
- entryID –

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxPSFPUdateISFTableEntry(*ep_handle_t* *handle, uint32_t entryID,
netc_tb_isf_cfge_t *cfg)

Update an entry into the ingress stream filter table.

Parameters

- handle –
- entryID –
- cfg –

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxPSFPDelISFTableEntry(*ep_handle_t* *handle, uint32_t entryID)

Del an entry into the stream filter table.

Parameters

- handle –
- entryID –

Returns

status_t

static inline uint32_t EP_RxPSFPGetRPTableRemainEntryNum(*ep_handle_t* *handle)

Get remaining available entry number of Rate Policer table.

Note: This is a dynamic bounded index table, the remaining entry can't be zero before add entry into it

Parameters

- handle –

Returns

uint32_t

status_t EP_RxPSFPAddRPTableEntry(*ep_handle_t* *handle, netc_tb_rp_config_t *config)

Add entry to Rate Policer table.

Parameters

- handle –
- config –

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxPSFPUdateRPTableEntry(*ep_handle_t* *handle, netc_tb_rp_config_t *config)

Update entry in Rate Policer table.

Parameters

- handle –
- config –

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxPSFPAddOrUpdateRPTableEntry(*ep_handle_t* *handle, *netc_tb_rp_config_t* *config)

Add or update entry in Rate Policer table.

Parameters

- handle –
- config –

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxPSFPDelRPTableEntry(*ep_handle_t* *handle, uint32_t entryID)

Delete entry in the Rate Policer table.

Parameters

- handle –
- entryID –

Returns

status_t

status_t EP_RxPSFPGetRPStatistic(*ep_handle_t* *handle, uint32_t entryID, *netc_tb_rp_stse_t* *statis)

Get statistic of specified Rate Policer entry.

Parameters

- handle –
- entryID –
- statis –

Returns

status_t

Returns

See netc_cmd_error_t

static inline uint32_t EP_RxPSFPGetISCTableRemainEntryNum(*ep_handle_t* *handle)

Get remaining available entry number of ingress stream count table.

Note: This is a dynamic bounded index table, the remaining entry can't be zero before add entry into it

Parameters

- handle –

Returns

uint32_t

status_t EP_RxPSFPAddISCTableEntry(*ep_handle_t* *handle, uint32_t entryID)

Add entry in ingress stream count table.

Parameters

- handle –
- entryID –

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxPSFPGetISCStatistic(*ep_handle_t* *handle, uint32_t entryID, *netc_tb_isc_stse_t* *statistic)

Get ingress stream count statistic.

Parameters

- handle –
- entryID –
- statistic –

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxPSFPResetISCStatistic(*ep_handle_t* *handle, uint32_t entryID)

Reset the count of the ingress stream count.

Parameters

- handle –
- entryID –

Returns

status_t

Returns

See netc_cmd_error_t

static inline uint32_t EP_RxPSFPGetSGITableRemainEntryNum(*ep_handle_t* *handle)

Get remaining available entry number of stream gate instance table.

Note: This is a dynamic bounded index table, the remaining entry can't be zero before add entry into it

Parameters

- handle –

Returns

uint32_t

status_t EP_RxPSFPAddSGITableEntry(*ep_handle_t* *handle, *netc_tb_sgi_config_t* *config)

Add entry in stream gate instance table.

Parameters

- handle –

- config –

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxPSFPUUpdateSGITableEntry(*ep_handle_t* *handle, *netc_tb_sgi_config_t* *config)

Update entry in stream gate instance table.

Parameters

- handle –
- config –

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxPSFPDelSGITableEntry(*ep_handle_t* *handle, uint32_t entryID)

Delete entry in stream gate instance table.

Parameters

- handle –
- entryID –

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxPSFPGetSGIState(*ep_handle_t* *handle, uint32_t entryID, *netc_tb_sgi_sgise_t* *state)

Get state of the stream gate instance for specified entry.

Parameters

- handle –
- entryID –
- state –

Returns

status_t

Returns

See netc_cmd_error_t

static inline uint32_t EP_RxPSFPGetSGCLTableRemainWordNum(*ep_handle_t* *handle)

Get remaining available words number of Stream Gate Control List table.

Note: This is a dynamic bounded index table, and number of words required for a stream gate control list is $1+N/2$ where N is number of gate time slots in the stream gate control list. The remaining word should be greater than the want added entry size

Parameters

- handle –

Returns

uint32_t

status_t EP_RxPSFPAddSGCLTableEntry(*ep_handle_t* *handle, *netc_th_sgcl_gcl_t* *config)

Add entry into Stream Gate Control List Table.

Parameters

- handle –
- config –

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxPSFPDelSGCLTableEntry(*ep_handle_t* *handle, uint32_t entryID)

Delete entry of Stream Gate Control List Table.

Parameters

- handle –
- entryID –

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxPSFPGetSGCLGateList(*ep_handle_t* *handle, *netc_th_sgcl_gcl_t* *gcl, uint32_t length)

Get Stream Gate Control List Table entry gate control list.

Parameters

- handle –
- gcl –
- length –

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxPSFPGetSGCLState(*ep_handle_t* *handle, uint32_t entryID, *netc_th_sgcl_sgclse_t* *state)

Get state (ref count) for Stream Gate Control List table entry.

Parameters

- handle –
- entryID –
- state –

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxL2MFInit(*ep_handle_t* *handle, *netc_si_l2mf_config_t* *config)

Init the L2 MAC Filter for a specified SI.

Parameters

- handle – EP handle
- config – The L2 MAC Filter configuration

Returns

status_t

status_t EP_RxL2MFAddHashEntry(*ep_handle_t* *handle, *netc_packet_type_t* type, *uint8_t* *macAddr)

Add entry into the MAC address hash filter with given MAC address Hardware layer will not maintain the counter of the hash filter. API layer shall cover this requirement.

Parameters

- handle – EP handle
- type – Unicast or multicast MAC address
- macAddr – MAC address to be added in filter table

Returns

status_t

status_t EP_RxL2MFDelHashEntry(*ep_handle_t* *handle, *netc_packet_type_t* type, *uint8_t* *macAddr)

Delete entry into the MAC address hash filter with given MAC address Hardware layer will not maintain the counter of the hash filter. API layer shall cover this requirement.

Parameters

- handle – EP handle
- type – Unicast or multicast MAC address
- macAddr – MAC address to be deleted from filter table

Returns

status_t

status_t EP_RxL2MFAddEMTableEntry(*ep_handle_t* *handle, *uint32_t* idx, *uint8_t* *macAddr)

Add entry into the MAC filter exact match table.

The entry is associated to the current Station Interface

Parameters

- handle –
- idx – Index in the entry table
- macAddr – MAC address for filter

Returns

status_t

Returns

See *netc_cmd_error_t*

status_t EP_RxL2MFDelEMTableEntry(*ep_handle_t* *handle, *uint32_t* idx)

Delete entry into the MAC filter exact match table.

Parameters

- handle – EP handle
- idx – Index in the entry table

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxL2VFInit(*ep_handle_t* *handle, *netc_si_l2vf_config_t* *config)

For VLAN filter, use inner vlan tag or outer vlan tag.

Parameters

- handle –
- config –

Returns

status_t

status_t EP_RxL2VFAddHashEntry(*ep_handle_t* *handle, uint16_t vlanId)

Add entry into the VLAN hash filter with given MAC address Hardware layer will not maintain the counter of the hash filter. API layer shall cover this requirement.

Parameters

- handle –
- vlanId – VLAN identifier for filter

Returns

status_t

status_t EP_RxL2VFDelHashEntry(*ep_handle_t* *handle, uint16_t vlanId)

Delete entry into the VLAN hash filter with given MAC address Hardware layer will not maintain the counter of the hash filter. API layer shall cover this requirement.

Parameters

- handle –
- vlanId – VLAN identifier for filter

Returns

status_t

status_t EP_RxL2VFAddEMTableEntry(*ep_handle_t* *handle, uint32_t idx, uint16_t vlanId, *netc_vlan_tpid_select_t* tpid)

Add entry into the MAC filter exact match table.

The entry is associated to the current Station Interface

Parameters

- handle –
- idx – Index in the entry table
- vlanId – VLAN identifier
- tpid – VLAN TPID

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_RxL2VFDelEMTableEntry(*ep_handle_t* *handle, uint32_t idx)

Delete entry into the VLAN filter exact match table.

Parameters

- handle –
- idx – Index in the entry table

Returns

status_t

Returns

See netc_cmd_error_t

static inline status_t EP_RxMapVlanToIpv(*ep_handle_t* *handle, *netc_vlan_t* vlan, uint8_t ipv)
Set the received Frame vlan to IPV mapping.

Parameters

- handle –
- vlan – Frame VLAN tag.
- ipv – The IPV value to be mapped.

Returns

status_t

static inline status_t EP_RxMapIpvToRing(*ep_handle_t* *handle, uint8_t ipv, uint8_t ring)
Set the IPV to Rx ring mapping.

Parameters

- handle –
- ipv – IPV value to be mapped.
- ring – The Rx BD ring index to be mapped.

Returns

status_t

static inline status_t EP_RxSetDefaultBDRGroup(*ep_handle_t* *handle,
netc_hw_enetc_si_rxr_group groupId)

Set the default used receive Rx BD ring group.

Note: The IPV mapped ring index is the relative index inside the default used group.

Parameters

- handle –
- groupId – The default Rx group index.

Returns

status_t

2.67 Endpoint (EP) Statistic Module

enum _ep_flags

Status/interrupt detect flags merged to same set of enum. TODO SITMRIDR.

Values:

enumerator kNETC_EPTimerSyncedFlag

enumerator kNETC_EPICMBlockedFlag

enumerator kNETC_EPWakeOnLANActiveFlag

typedef enum *ep_flags* ep_flags_t

Status/interrupt detect flags merged to same set of enum. TODO SITMRIDR.

static inline *status_t* EP_GetPortDiscardStatistic(*ep_handle_t* *handle, bool useTx,
netc_port_discard_statistic_t *statistic)

Get the ENETC port discard statistic and reason.

Get the discarded count of frames and its reasons.

Parameters

- handle –
- useTx – true - Tx port. false - Rx port.
- statistic – pointer to the statistic data

Returns

status_t

static inline *status_t* EP_ClearPortDiscardReason(*ep_handle_t* *handle, bool useTx, uint32_t
reason0, uint32_t reason1)

Clean the EP Port Rx discard reason. Set the related bits to 1 to clear the specific reasons.

Parameters

- handle –
- useTx – true - Tx port. false - Rx port.
- reason0 –
- reason1 –

Returns

status_t

static inline uint32_t EP_GetPortTGSLStatus(*ep_handle_t* *handle)

Get EP port time gate scheduling gate list status.

Parameters

- handle –

Returns

Port status flags which are ORed by the enumerators in the
netc_port_tgsl_status_t

2.68 Endpoint (EP) Egress data path configuration

status_t EP_TxTGSCfgAdminGcl(*ep_handle_t* *handle, *netc_tb_tgs_gcl_t* *config)

Config the Time Gate Scheduling entry admin gate control list.

This function is used to program the Enhanced Scheduled Transmisson. (IEEE802.1Qbv)

Parameters

- handle –
- config –

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_TxPortTGSEnable(*ep_handle_t* *handle, bool enable, uint8_t gateState)

Enable the EP port time gate scheduling.

Parameters

- handle –
- enable –
- gateState –

Returns

status_t

status_t EP_TxtTGSGetOperGcl(*ep_handle_t* *handle, *netc_th_tgs_gcl_t* *gcl, uint32_t length)

Get Time Gate Scheduling entry operation gate control list.

This function is used to read the Enhanced Scheduled Transmisson. (IEEE802.1Qbv)

Parameters

- handle –
- gcl –
- length –

Returns

status_t

Returns

See netc_cmd_error_t

status_t EP_TxTrafficClassConfig(*ep_handle_t* *handle, *netc_hw_tc_idx_t* tcIdx, const *netc_port_tx_tc_config_t* *config)

Config the TC (traffic class) property.

Parameters

- handle –
- tcIdx –
- config –

Returns

status_t

static inline void EP_TxTcConfigPreemption(*ep_handle_t* *handle, *netc_hw_tc_idx_t* tcIdx, const bool enable)

Config Preemption for each port TC (traffic class)

Parameters

- handle –
- tcIdx –
- enable –

static inline void EP_TxGetTcPreemption(*ep_handle_t* *handle, *netc_hw_tc_idx_t* tcIdx, bool *enabled)

Get Preemption configuration for each port TC (traffic class)

Parameters

- handle –

- tcIdx –
- enabled –

```
static inline void EP_TxPortEthMacConfigPreemption(ep_handle_t *handle, const
                                                    netc_port_preemption_config *config)
```

Configure Preemption control configuration for an ethernet MAC.

Parameters

- handle –
- config –

```
static inline void EP_TxPortGetEthMacPreemption(ep_handle_t *handle,
                                                netc_port_preemption_config *config,
                                                netc_port_phy_mac_preemption_status_t
                                                *status)
```

Get Preemption configuration from ethernet MAC port.

Parameters

- handle –
- config –
- status –

2.69 Endpoint (EP) Transmit/Receive

```
enum _ep_rx_flags
```

Values:

```
enumerator kEP_RX_RSS_VALID
```

Request timestamp.

```
enumerator kEP_RX_VLAN_VALID
```

Specify frame departure time.

```
enumerator kEP_RX_TIMESTAMP_VALID
```

Enable port masquerading.

```
enum _ep_tx_opt_flags
```

Values:

```
enumerator kEP_TX_OPT_REQ_TS
```

Request timestamp (IEEE 1588 PTP two-step timestamp).

```
enumerator kEP_TX_OPT_VLAN_INSERT
```

Enable VLAN insert.

```
enumerator kEP_TX_OPT_START_TIME
```

Specify frame departure time.

```
typedef enum _ep_rx_flags ep_rx_flags_t
```

```
typedef enum _ep_tx_opt_flags ep_tx_opt_flags
```

```
typedef struct _ep_tx_opt ep_tx_opt
```

```
status_t EP_SendFrameCommon(ep_handle_t *handle, netc_tx_bdr_t *txBdRing, uint8_t hwRing,
                           netc_frame_struct_t *frame, void *context, netc_tx_bd_t *txDesc,
                           bool txCacheMaintain)
```

Common part for transfer regular frame or Switch management frame.

Note: This function is internal used. Please use EP_SendFrame() or SWT_SendFrame() API to send frames.

Parameters

- handle –
- txBdRing – The Transmit buffer descriptor ring handle
- hwRing – The hardware Tx ring index
- frame – The frame descriptor pointer
- context – Private context provided back by ep_reclaim_cb_t
- txDesc – Point to the Transmits BD Description array.
- txCacheMaintain – Enable/Disable Tx buffer Cache Maintain.

Return values

status_t –

```
status_t EP_SendFrame(ep_handle_t *handle, uint8_t ring, netc_frame_struct_t *frame, void
                    *context, ep_tx_opt *opt)
```

Transmits a frame for specified ring. This API is zero-copy and requires the ep_reclaim_cb_t to be called to free the transmitted frame.

Parameters

- handle –
- ring – The ring index
- frame – The frame descriptor pointer
- context – Private context provided back by ep_reclaim_cb_t
- opt – Additional tx options. If NULL, default is tx timestamping enabled, no start time and no masquerading.

Return values

status_t –

```
static inline void EP_WaitUnitilTxComplete(ep_handle_t *handle, uint8_t ring)
```

Wait until the EP Tx ring has completed the transfer.

Note: Only call after EP_SendFrame() to do a no-interrupt transfer

Parameters

- handle –
- ring – The ring index

```
netc_tx_frame_info_t *EP_ReclaimTxDescCommon(ep_handle_t *handle, netc_tx_bdr_t *txBdRing,
                                             uint8_t hwRing, bool enCallback)
```

Common part of Reclaim tx descriptors for regular frame or Switch management frame.

Note: This function is internal used. Please use EP_ReclaimTxDescriptor() or SWT_ReclaimTxDescriptor() API to Reclaim tx descriptors.

Parameters

- handle –
- txBdRing – The Transmit buffer descriptor ring handle
- hwRing – The hardware Tx ring index
- enCallback – Enable/Disable call the Tx Reclaim callback functions.

void EP_ReclaimTxDescriptor(*ep_handle_t* *handle, uint8_t ring)

Reclaim tx descriptors. This function is used to update the tx descriptor status and get the tx timestamp. For each reclaimed transmit frame the ep_reclaim_cb_t is called.

This is called after being notified of a transmit completion from ISR. It runs until there are no more frames to be reclaimed in the BD ring.

Parameters

- handle –
- ring – The ring index

status_t EP_ReceiveFrameCommon(*ep_handle_t* *handle, *netc_rx_bdr_t* *rxBdRing, uint8_t ring, *netc_frame_struct_t* *frame, *netc_frame_attr_t* *attr, bool rxCacheMaintain)

Common part of receives one frame with zero copy from specified ring.

Note: This function is internal used. Please use EP_ReceiveFrame() or SWT_ReceiveFrame() API.

Parameters

- handle –
- rxBdRing – Rx BD ring handle
- ring – Ring index
- frame – Frame buffer point
- attr – Frame attribute pointer
- rxCacheMaintain – Enable/Disable Rx buffer Cache maintain

Returns

status_t

status_t EP_ReceiveFrame(*ep_handle_t* *handle, uint8_t ring, *netc_frame_struct_t* *frame, *netc_frame_attr_t* *attr)

Receives one frame with zero copy from specified ring.

Note: The sufficient rx frame data structure MUST be provided by application.

Parameters

- handle –
- ring – The ring index

- frame – The frame descriptor pointer
- attr – Frame attribute pointer

Returns

kStatus_Success Successfully receive a regular frame

Returns

kStatus_NETC_RxHRNotZeroFrame Frame in Rx BD ring is management frame, need call SWT_ReceiveFrame()

Returns

kStatus_NETC_RxTsrResp Frame in Rx BD ring is Transmit Timestamp Reference Response messages, need call SWT_GetTimestampRefResp() to get Transmit Timestamp Reference Response

Returns

kStatus_NETC_RxFrameEmpty Rx BD ring is empty

Returns

kStatus_NETC_RxFrameError Frame in Rx BD ring has error, need be dropped

Returns

kStatus_InvalidArgument Rx BD ring index is out of range

Returns

kStatus_NETC_LackOfResource Application provided buffer is not enough

void EP_DropFrame(*ep_handle_t* *handle, *netc_rx_bdr_t* *rxBdRing, uint8_t ring)

Drop one frame.

Note: This function is internal used.

Parameters

- handle –
- rxBdRing – Rx BD ring handle
- ring – Ring index

status_t EP_ReceiveFrameCopyCommon(*ep_handle_t* *handle, *netc_rx_bdr_t* *rxBdRing, uint8_t ring, void *buffer, uint32_t length, *netc_frame_attr_t* *attr, bool rxCacheMaintain)

Common part of receive regular frame or Switch management frame which will be copied in the provided buffer.

Note: This function is internal used. Please use EP_ReceiveFrameCopy() or SWT_ReceiveFrameCopy() API.

Parameters

- handle –
- rxBdRing – Rx BD ring handle
- ring – Ring index
- buffer – Buffer address
- length – Buffer length
- attr – Frame attribute pointer
- rxCacheMaintain – Enable/Disable Rx buffer Cache maintain

Returns

status_t

status_t EP_ReceiveFrameCopy(*ep_handle_t* *handle, uint8_t ring, void *buffer, uint32_t length, *netc_frame_attr_t* *attr)

Receives one frame which will be copied in the provided buffer from specified ring.

Note: The buffer size MUST be queried using EP_GetRxFrameSize() beforehand.

Parameters

- handle –
- ring – Ring index
- buffer – Buffer address
- length – Buffer length
- attr – Frame attribute pointer

Returns

kStatus_Success Successfully receive a regular frame

Returns

kStatus_InvalidArgument Rx BD ring index is out of range

status_t EP_GetRxFrameSizeCommon(*ep_handle_t* *handle, *netc_rx_bdr_t* *rxBdRing, uint32_t *length)

Common part of get pending frame size API for regular frame or Switch management frame.

Note: This function is internal used. Please use EP_GetRxFrameSize() or SWT_GetRxFrameSize() API.

Parameters

- handle –
- rxBdRing – Rx BD ring handle
- length – The length of the valid frame received.

Returns

status_t

status_t EP_GetRxFrameSize(*ep_handle_t* *handle, uint8_t ring, uint32_t *length)

Gets the size of the pending frame in the specified receive ring buffer.

Note: Frame size without FCS

Parameters

- handle – The ENET handler structure. This is the same handler pointer used in the ENET_Init.
- ring – The ring index
- length – The length of the valid frame received.

Returns

kStatus_Success Successfully get the length of a regular frame

Returns

kStatus_NETC_RxHRNotZeroFrame Frame in Rx BD ring is management frame, need call SWT_GetRxFramSize() to get frame size

Returns

kStatus_NETC_RxTsrResp Frame in Rx BD ring is Transmit Timestamp Reference Response messages, need call SWT_GetTimestampRefResp() to get Transmit Timestamp Reference Response

Returns

kStatus_NETC_RxFrameEmpty Rx BD ring is empty

Returns

kStatus_NETC_RxFrameError Frame in Rx BD ring has error, need be dropped

Returns

kStatus_InvalidArgument Rx BD ring index is out of range

```
struct __ep_tx_opt
```

```
    #include <fsl_netc_endpoint.h>
```

Public Members

```
uint32_t flags
```

A bitmask of ep_tx_opt_flags

```
uint32_t timestamp
```

Departure timestamp, used if kEP_TX_OPT_START_TIME is set

```
netc_enetc_vlan_tag_t vlan
```

VLAN tag which will be inserted, used if kEP_TX_OPT_VLAN_INSERT is set

2.70 Hardware layer

```
enum __netc_hw_enetc_idx
```

ENETC index enumerator.

Values:

```
enumerator kNETC_ENETC0
```

ENETC hardware 0

```
enumerator kNETC_ENETC1
```

ENETC hardware 0

```
enum __netc_hw_switch_idx
```

SWITCH index enumerator.

Values:

```
enumerator kNETC_SWITCH0
```

SWITCH hardware 0

```
enum __netc_hw_port_idx
```

Port Resource for the NETC module.

Values:

```
enumerator kNETC_ENETC0Port
```

MAC port for ENETC0

enumerator kNETC_ENETC1Port
Pseudo MAC port for ENETC1

enumerator kNETC_SWITCH0Port0
MAC port0 for SWITCH

enumerator kNETC_SWITCH0Port1
MAC port1 for SWITCH

enumerator kNETC_SWITCH0Port2
MAC port2 for SWITCH

enumerator kNETC_SWITCH0Port3
MAC port3 for SWITCH

enumerator kNETC_SWITCH0Port4
Pseudo port4 for SWITCH

enum __netc_hw_tc_idx
Traffic class enumerator.

Values:

enumerator kNETC_TxTC0
Traffic class 0

enumerator kNETC_TxTC1
Traffic class 1

enumerator kNETC_TxTC2
Traffic class 2

enumerator kNETC_TxTC3
Traffic class 3

enumerator kNETC_TxTC4
Traffic class 4

enumerator kNETC_TxTC5
Traffic class 5

enumerator kNETC_TxTC6
Traffic class 6

enumerator kNETC_TxTC7
Traffic class 7

enum __netc_hw_bdr_idx
Enumeration for the ENETC SI BDR identifier.

Values:

enumerator kNETC_BDR0

enumerator kNETC_BDR1

enumerator kNETC_BDR2

enumerator kNETC_BDR3

enumerator kNETC_BDR4

enumerator kNETC_BDR5

enumerator kNETC_BDR6
enumerator kNETC_BDR7
enumerator kNETC_BDR8
enumerator kNETC_BDR9
enumerator kNETC_BDR10
enumerator kNETC_BDR11
enumerator kNETC_BDR12
enumerator kNETC_BDR13

enum _netc_hw_swt_cbdr_idx
Switch command BD ring index enumerator.

Values:

enumerator kNETC_SWTCBDR0
Switch command BD ring 0
enumerator kNETC_SWTCBDR1
Switch command BD ring 1

enum _netc_hw_classs_queue_idx
Enumerator for ETM class queue identifier.

Values:

enumerator kNETC_ClassQueue0
ETM Class Queue 0
enumerator kNETC_ClassQueue1
ETM Class Queue 1
enumerator kNETC_ClassQueue2
ETM Class Queue 2
enumerator kNETC_ClassQueue3
ETM Class Queue 3
enumerator kNETC_ClassQueue4
ETM Class Queue 4
enumerator kNETC_ClassQueue5
ETM Class Queue 5
enumerator kNETC_ClassQueue6
ETM Class Queue 6
enumerator kNETC_ClassQueue7
ETM Class Queue 7

enum _netc_hw_congestion_group_idx
Enumerator for the ETM congestion group.

Values:

enumerator kNETC_CongGroup0
enumerator kNETC_CongGroup1

enum _netc_hw_mii_mode

Defines the MII/RGMII mode for data interface between the MAC and the PHY.

Values:

enumerator kNETC_XgmiiMode
XGMII mode for data interface.

enumerator kNETC_MiiMode
MII mode for data interface.

enumerator kNETC_GmiiMode
GMII mode for data interface.

enumerator kNETC_RmiiMode
RMII mode for data interface.

enumerator kNETC_RgmiiMode
RGMII mode for data interface.

enumerator kNETC_SgmiiMode
SGMII mode for data interface.

enum _netc_hw_mii_speed

Defines the speed for the *MII data interface.

Values:

enumerator kNETC_MiiSpeed10M
Speed 10 Mbps.

enumerator kNETC_MiiSpeed100M
Speed 100 Mbps.

enumerator kNETC_MiiSpeed1000M
Speed 1000 Mbps.

enumerator kNETC_MiiSpeed2500M
Speed 2500 Mbps.

enumerator kNETC_MiiSpeed5G
Speed 5Gbps.

enumerator kNETC_MiiSpeed10G
Speed 10Gbps Mbps.

enum _netc_hw_mii_duplex

Defines the half or full duplex for the MII data interface.

Values:

enumerator kNETC_MiiHalfDuplex
Half duplex mode.

enumerator kNETC_MiiFullDuplex
Full duplex mode.

typedef enum _netc_hw_enetc_idx netc_hw_enetc_idx_t
ENETC index enumerator.

typedef enum _netc_hw_switch_idx netc_hw_switch_idx_t
SWITCH index enumerator.

typedef enum *_netc_hw_port_idx* netc_hw_port_idx_t

Port Resource for the NETC module.

typedef enum *_netc_hw_tc_idx* netc_hw_tc_idx_t

Traffic class enumerator.

typedef enum *_netc_hw_bdr_idx* netc_hw_bdr_idx_t

Enumeration for the ENETC SI BDR identifier.

typedef enum *_netc_hw_swt_cbdr_idx* netc_hw_swt_cbdr_idx_t

Switch command BD ring index enumerator.

typedef enum *_netc_hw_classs_queue_idx* netc_hw_etm_class_queue_idx_t

Enumerator for ETM class queue identifier.

typedef enum *_netc_hw_congestion_group_idx* netc_hw_congestion_group_idx_t

Enumerator for the ETM congestion group.

typedef enum *_netc_hw_mii_mode* netc_hw_mii_mode_t

Defines the MII/RGMII mode for data interface between the MAC and the PHY.

typedef enum *_netc_hw_mii_speed* netc_hw_mii_speed_t

Defines the speed for the *MII data interface.

typedef enum *_netc_hw_mii_duplex* netc_hw_mii_duplex_t

Defines the half or full duplex for the MII data interface.

typedef struct *_netc_psfpc_profile* netc_isi_kc_rule_t

NETC PSFP kc profile configuration, the key size (not include the spmp and portp) is up to 16 bytes.

typedef struct *_netc_vlan_classify_config* netc_vlan_classify_config_t

NETC Vlan classification config.

typedef struct *_netc_qos_classify_profile* netc_qos_classify_profile_t

NETC Qos Classification profile file (vlan PCP/DEI to IPV/DR map)

typedef struct *_netc_ipf_config* netc_ipf_config_t

NETC Ingress Filter config.

typedef struct *_netc_func* netc_func_t

Register groups for the PCIe function.

typedef struct *_netc_port_hw* netc_port_hw_t

Register groups for the Port/Link hardware.

typedef struct *_netc_enetc_hw* netc_enetc_hw_t

Register group for the ENETC peripheral hardware.

typedef struct *_netc_timer_hw* netc_timer_hw_t

Register group for the Timer peripheral hardware.

typedef struct *_netc_mdio_hw* netc_mdio_hw_t

Register group for both EMDIO and port external MDIO.

getSiInstance(*si*)

Get SI information from netc_hw_si_idx_t.

The ENETC instance of this SI.

getSiNum(*si*)

The SI number in the ENETC.

getSiIdx(si)

The actual index in the netc_hw_si_idx_t.

NETC_MSIX_TABLE_OFFSET

MSIX table address offset.

NETC_MSIX_TABLE_PBA_OFFSET

MSIX PBA address offset.

NETC_NANOSECOND_ONE_SECOND

Nanosecond in one second.

struct __netc_psfp_kc_profile

#include <fsl_netc.h> NETC PSFP kc profile configuration, the key size (not include the spmp and portp) is up to 16 bytes.

Public Members

bool etp

2 Byte Ethertype field present in the key

bool sqtp

1 Byte Sequence Tag present in the key

bool ipcnp

inner VLAN header's PCP field present in the key

bool ividp

inner VLAN ID present in the key

bool opcpnp

outer VLAN header's PCP field present in the key

bool ovidp

outer VLAN ID present in the key

bool smacp

6 bytes of source MAC address present in the key

bool dmacp

6 bytes of destination MAC address present in the key

bool spmp

switch port masquerading flag present in the key

bool portp

source port present in the key

bool valid

Key Construction is valid

struct __netc_vlan_classify_config

#include <fsl_netc.h> NETC Vlan classification config.

Public Members

bool enableCustom1

Enable/Disable custom0 ether type

```
uint16_t custom1EtherType
    Ethertype
bool enableCustom2
    Enable/Disable custom0 ether type
uint16_t custom2EtherType
    Ethertype
uint16_t preStandRTAGType
    802.1CB draft 2.0 R-TAG Ethertype value. PSRTAGETR. Only applicable for switch
struct __netc_qos_classify_profile
    #include <fsl_netc.h> NETC Qos Classification profile file (vlan PCP/DEI to IPV/DR map)
```

Public Members

```
uint8_t ipv[16]
    Index is created from PCP (3 bits) + DEI (1 bit) field. Value is the mapped IPV for Qos.
uint8_t dr[16]
    Index is created from PCP (3 bits) + DEI (1 bit) field. Value is the mapped DR for QoS.
struct __netc_ipf_config
    #include <fsl_netc.h> NETC Ingress Filter config.
```

Public Members

```
bool l2DiscardMCSmac
    DOSL2CR. Discard received frames with Multicast SMAC address
bool l2DiscardSmacEquDmac
    DOSL2CR. Discard received frames with SMAC = DMAC
bool l3DiscardSipEquDip
    DOSL3CR. Discard IPV3/IPV6 source address == destination address
struct __netc_func
    #include <fsl_netc_hw.h> Register groups for the PCIe function.
struct __netc_port_hw
    #include <fsl_netc_hw.h> Register groups for the Port/Link hardware.
```

Public Members

```
NETC_PORT_Type *port
    Port Address
struct __netc_enetc_hw
    #include <fsl_netc_hw.h> Register group for the ENETC peripheral hardware.
```

Public Members

```
netc_func_t func
    PCIE function register
```

NETC_ENETC_Type *base
Base register of ENETC module

NETC_SW_ENETC_Type *common
Common register of ENETC module

netc_port_hw_t portGroup
Port register group

ENETC_GLOBAL_Type *global
Global NETC address

ENETC_SI_Type *si
Station Interface for the P/V SI

netc_msix_entry_t *msixTable
MSIX table address

struct _netc_timer_hw
#include <fsl_net_hw.h> Register group for the Timer peripheral hardware.

Public Members

ENETC_PCI_TYPE0_Type *func
PCIE function register

ENETC_PF_TMR_Type *base
Base register address for timer module

ENETC_GLOBAL_Type *global
Global NETC register address

netc_msix_entry_t *msixTable
MSIX table address

struct _netc_mdio_hw
#include <fsl_net_hw.h> Register group for both EMDIO and port external MDIO.

Public Members

__IO uint32_t EMDIO_CFG
External MDIO configuration register, offset: 0x1C00

__IO uint32_t EMDIO_CTL
External MDIO interface control register, offset: 0x1C04

__IO uint32_t EMDIO_DATA
External MDIO interface data register, offset: 0x1C08

__IO uint32_t EMDIO_ADDR
External MDIO register address register, offset: 0x1C0C

__IO uint32_t EMDIO_STAT
External MDIO status register, offset: 0x1C10

__IO uint32_t PHY_STATUS_CFG
PHY status configuration register, offset: 0x1C20

__IO uint32_t PHY_STATUS_CTL
PHY status control register, offset: 0x1C24

```
__IO uint32_t PHY_STATUS_DATA
    PHY status data register, offset: 0x1C28
__IO uint32_t PHY_STATUS_ADDR
    PHY status register address register, offset: 0x1C2C
__IO uint32_t PHY_STATUS_EVENT
    PHY status event register, offset: 0x1C30
__IO uint32_t PHY_STATUS_MASK
    PHY status mask register, offset: 0x1C34
```

struct payload

Public Members

```
uint8_t lbMask
    Payload Last Byte Mask
uint8_t fbMask
    Payload First Byte Mask
uint8_t byteOffset
    Payload Byte Offset where field extraction begins
uint8_t numBytes
    Specify the size (numBytes + 1) of the payload key field
uint8_t pfp
    Payload field Present
union __unnamed200__
```

Public Members

```
ENETC_PCI_TYPE0_Type *pf
    PSI function
ENETC_VF_PCI_TYPE0_Type *vf
    VSI function
union __unnamed202__
```

Public Members

```
NETC_ETH_LINK_Type *eth
    MAC Port Address
```

2.71 Hardware Common Functions

```
static inline uint16_t EP_IncreaseIndex(uint16_t index, uint32_t max)
```

```
uint16_t NETC_SIGetVsiIndex(netc_vsi_number_t vsi)
    Get the VSI index.
```

Parameters

- vsi – The VSI number.

static inline void NETC_IPFInit(NETC_SW_ENETC_Type *base, const *netc_ipf_config_t* *config)
Set layer2/3 Dos configuration.

Parameters

- base –
- config –

void NETC_PSFpKcProfileInit(NETC_SW_ENETC_Type *base, const *netc_isi_kc_rule_t* *rule, bool enKcPair1)

Initialize the Ingress Stream Identification Key construction rule profiles.

Parameters

- base –
- rule –
- enKcPair1 –

Returns

void

void NETC_RxVlanCInit(NETC_SW_ENETC_Type *base, const *netc_vlan_classify_config_t* *config, bool enRtag)

Initialize the customer vlan type.

Parameters

- base –
- config –
- enRtag –

Returns

void

void NETC_RxQoSInit(NETC_SW_ENETC_Type *base, const *netc_qos_classify_profile_t* *profile, bool enProfile1)

Initialize the ingress QoS classification.

Parameters

- base –
- profile –
- enProfile1 –

2.72 Hardware ENETC

typedef struct *netc_enetc_vlan_tag_t* netc_enetc_vlan_tag_t
ENETC Port outer/inner VLAN tag.

typedef struct *netc_enetc_discard_statistic* netc_enetc_port_discard_statistic_t
PORT discard count statistic.

typedef struct *netc_enetc_native_vlan_config_t* netc_enetc_native_vlan_config_t
ENETC Port outer/inner native VLAN config.

typedef struct *netc_enetc_parser_config_t* netc_enetc_parser_config_t
ENETC parser configuration.

```
typedef struct _netc_enetc_cap netc_enetc_cap_t
```

ENETC capability.

```
static inline bool NETC_EnetcHasManagement(NETC_ENETC_Type *base)
```

Check whether ENETC has switch management capability.

Parameters

- base – ENETC peripheral base address.

Returns

true or false

```
void NETC_EnetcGetCapability(NETC_ENETC_Type *base, netc_enetc_cap_t *capability)
```

Get ENETC capability.

Parameters

- base – ENETC peripheral base address.
- capability – Pointer to capability structure.

```
void NETC_EnetcSetSIMacAddr(NETC_ENETC_Type *base, uint8_t si, uint8_t *macAddr)
```

Set MAC address for specified VSI of ENETC.

Parameters

- base –
- macAddr –

```
status_t NETC_EnetcConfigureSI(NETC_ENETC_Type *base, uint8_t si, const  
                               netc_hw_enetc_si_config_t *psConfig)
```

Configure SI.

Parameters

- base – ENETC peripheral base address.
- si – The SI number
- psConfig – The SI configuration

Returns

status_t

```
status_t NETC_EnetcSetMsixEntryNum(NETC_ENETC_Type *base, uint8_t si, uint32_t msixNum)
```

Set SI MSIX table entry number.

Parameters

- base – ENETC peripheral base address.
- si – The SI number.
- msixNum – The MSIX table entry number.

Returns

status_t

```
static inline void NETC_EnetcEnableSI(NETC_ENETC_Type *base, uint8_t si, bool enable)
```

Enable/Disable specified SI.

Parameters

- base – ENETC peripheral base address.
- si – SI index.
- enable – Enable/Disable SI from ENETC layer.


```
void NETC__EnetcGetPortDiscardStatistic(NETC_ENETC_Type *base,  
                                       netc_enetc_port_discard_statistic_t *statistic)
```

Get ENETC discard statistic data.

Parameters

- base – ENETC peripheral base address.
- statistic – Statistic data.

```
void NETC__EnetcEnablePromiscuous(NETC_ENETC_Type *base, uint8_t si, bool enableUCPromis,  
                                  bool enableMCPromis)
```

Enable MAC promiscuous mode.

Parameters

- base – ENETC peripheral base address.
- si – SI index.
- enableUCPromis – Enable unicast frame promiscuous.
- enableMCPromis – Enable multicast frame promiscuous.

```
void NETC__EnetcConfigureVlanFilter(NETC_ENETC_Type *base, uint8_t si, netc_si_l2vf_config_t  
                                   *config)
```

Configure VLAN filter.

Parameters

- base – ENETC peripheral base address.
- si – SI index.
- config – Enable untagged VLAN frame promiscuous.

```
void NETC__EnetcAddMacAddrHash(NETC_ENETC_Type *base, uint8_t si, netc_packet_type_t  
                               type, uint8_t hashIndex)
```

Add the hash filter for the MAC address.

Hardware layer will not maintain the counter of the hash filter. API layer shall cover this requirement.

Parameters

- base – ENETC peripheral base address.
- si – SI index.
- type – Unicast or multicast frame type.
- hashIndex – The calculated hash index of MAC address.

```
void NETC__EnetcDelMacAddrHash(NETC_ENETC_Type *base, uint8_t si, netc_packet_type_t type,  
                               uint8_t hashIndex)
```

Remove the hash filter for the MAC address.

Parameters

- base – ENETC peripheral base address.
- si – SI index.
- type – Unicast or multicast frame type.
- hashIndex – The calculated hash index of MAC address.

```
void NETC__EnetcAddVlanHash(NETC_ENETC_Type *base, uint8_t si, uint8_t hashIndex)
```

Add the hash filter for the VLAN.

Parameters

- base – ENETC peripheral base address.
- si – SI index.
- hashIndex – The calculated hash index of MAC address.

```
void NETC__EnetcDelVlanHash(NETC_ENETC_Type *base, uint8_t si, uint8_t hashIndex)
```

Remove the hash filter for the VLAN.

Parameters

- base – ENETC peripheral base address.
- si – SI index.
- hashIndex – The calculated hash index of MAC address.

```
static inline status_t NETC__EnetcPortEnableTSD(NETC_ENETC_Type *base, netc_hw_tc_idx_t  
tcIdx, bool isEnabled)
```

Enable / Disable ENETC Port Time Specific Departure (TSD) feature.

It can't work with QBV CBS

Parameters

- base –
- tcIdx –
- isEnabled –

Returns

status_t

```
static inline void NETC__EnetcPortSetNativeVLAN(NETC_ENETC_Type *base, const  
netc_enetc_native_vlan_config_t *config, bool  
isOuter)
```

Set ENETC Rx native outer/inner VLAN.

It is used for classification when untagged frames are received by the port.

Parameters

- base –
- config –
- isOuter –

```
static inline void NETC__EnetcSetParser(NETC_ENETC_Type *base, const  
netc_enetc_parser_config_t *config)
```

Set ENETC Parser configuration.

PARCSCR and PARCE0CR - PARCE3CR.

Parameters

- base –
- config –

```
static inline void NETC__EnetcEnableWakeOnLan(NETC_ENETC_Type *base, bool isEnabled)
```

Enable / Disable ENETC Wake-on-LAN mode.

Only available on ENETC 0

Parameters

- base –
- isEnable –

Returns

status_t

struct __netc_enetc_vlan_tag_t

#include <fsl_enetc.h> ENETC Port outer/inner VLAN tag.

Public Members

uint16_t pcp

Priority code point

uint16_t dei

Drop eligible indicator

uint16_t vid

VLAN identifier

netc_vlan_tpid_select_t tpid

Tag protocol identifier

struct __netc_enetc_discard_statistic

#include <fsl_enetc.h> PORT discard count statistic.

Public Members

uint32_t ingressDR[4]

Discard count for port ingress congestion different DR

uint32_t broadcastReject

Broadcast frame drops count due to all SI enable broadcast reject

uint32_t smacPruning

Frames discard count due to port MAC source address pruning

uint32_t unicastMacFilt

Unicast frame discard count due to port MAC filtering

uint32_t multicastMacFilt

Multicast frame discard count due to MAC filtering

uint32_t unicastVlanFilt

Unicast frame discard count due to VLAN filtering

uint32_t multicastVlanFilt

Multicast frame discard count due to VLAN filtering

uint32_t broadcastVlanFilt

Broadcast frame discard count due to VLAN filtering

struct __netc_enetc_native_vlan_config_t

#include <fsl_enetc.h> ENETC Port outer/inner native VLAN config.

Public Members

bool enUnderZeroVid

Enable use the port default VLAN VID when the VID in the packet's is zero

bool enUnderNoVlan

Enable use the port default VLAN VID when the VLAN tag is not present

netc_enetc_vlan_tag_t vlanTag

Port native outer/inner VLAN tag, valid when enUnderZeroVid or enUnderNoVlan is true

struct _netc_enetc_parser_config_t

#include <fsl_netc.h> ENETC parser configuration.

Public Members

bool disL3Checksum

Disable Layer 3 IPv4 Header checksum validation.

bool disL4Checksum

Disable Layer 4 TCP and UDP checksum validation.

struct _netc_enetc_cap

#include <fsl_netc_hw_enetc.h> ENETC capability.

Public Members

bool funcSafety

Support for safety capability.

bool wol

Support for Wake-on-LAN in low-power mode.

bool rss

Support for RSS.

bool tsd

Support for time specific departure.

bool rfs

Support for RFS.

uint32_t ipvNum

IPV number.

uint32_t vsiNum

VSI number.

uint32_t msixNum

MSIX table vector/entry number.

uint32_t tcsNum

Traffic class number.

uint16_t uchNum

Unicast hash entry number.

uint16_t mchNum

Multicast hash entry number.

uint16_t rxBdrNum
Rx BD ring number.

uint16_t txBdrNum
Tx BD ring number.

struct custEtype

Public Members

uint16_t etype
Custom Ethertype value. Upon detecting this ether type the associated code point will be mapped to the parse summary as a Non IP code point.

bool en
Enables the detection and mapping.

uint8_t cp
This value is mapped to the parse summary as a Non IP code point.

2.73 Hardware Port

enum _netc_port_tgsl_status
Port time gate scheduling gate list status.
Values:

enumerator kNETC_OperListActive
Port operational gate control list is active.

enumerator kNETC_AdminListPending
Administrative gate control list is pending (configured but not installed yet).

enum _netc_port_discard_tpye
Port Tx/Rx discard counter in the datapath processing pipeline or bridge forwarding processing function.
Values:

enumerator kNETC_RxDiscard
Discarded frames in the receive port datapath processing pipeline.

enumerator kNETC_TxDiscard
Discarded frames in the egress datapath processing pipeline, only for switch.

enumerator kNETC_BridgeDiscard
Discarded frames in the bridge forwarding processing function, only for switch.

enum _netc_port_tpidlist
Defines Port TPID acceptance.
Values:

enumerator kNETC_OuterStanCvlan
Accept outer Standard C-VLAN 0x8100.

enumerator kNETC_OuterStanSvlan
Accept outer Standard S-VLAN 0x88A8.

enumerator kNETC_OuterCustomVlan1

Accept outer Custom VLAN as defined by CVLANR1[ETYPE].

enumerator kNETC_OuterCustomVlan2

Accept outer Custom VLAN as defined by CVLANR2[ETYPE].

enumerator kNETC_InnerStanCvlan

Accept inner Standard C-VLAN 0x8100.

enumerator kNETC_InnerStanSvlan

Accept inner Standard S-VLAN 0x88A8.

enumerator kNETC_InnerCustomVlan1

Accept inner Custom VLAN as defined by CVLANR1[ETYPE].

enumerator kNETC_InnerCustomVlan2

Accept inner Custom VLAN as defined by CVLANR2[ETYPE].

enum __netc_port_ts_select

Defines port timestamp selection.

Values:

enumerator kNETC_SyncTime

Synchronized time.

enumerator kNETC_FreeRunningTime

Free running time.

enum __netc_hw_preemption_mode

Port MAC preemption mode.

Values:

enumerator kNETC_PreemptDisable

Frame preemption is not enabled

enumerator kNETC_PreemptOn64B

Frame preemption is enabled, but transmit only preempts frames on 64B boundaries

enumerator kNETC_PreemptOn4B

Frame preemption is enabled, but transmit only preempts frames on 4B boundaries

enum __netc_hw_raf_size

Port MAC Remote Additional Fragment Size.

Values:

enumerator kNETC_RafSize64

Additional Fragment Size of 64 octets

enumerator kNETC_RafSize128

Additional Fragment Size of 128 octets

enumerator kNETC_RafSize256

Additional Fragment Size of 256 octets

enumerator kNETC_RafSize512

Additional Fragment Size of 512 octets

enum _netc_tc_sdu_type

Type of PDU/SDU (Protocol/Service Data Unit).

Note: Overhead values which adding to the transmitted frame of Length are specified by Port SDU config as follows:

- PPDU = add rxPpduBco/txPpduBco + rxMacsecBco/txMacsecBco bytes
 - MPDU = add rxMacsecBco/txMacsecBco bytes
 - MSDU = minus 16B (12B MAC Header + 4B FCS)
-

Values:

enumerator kNETC_PDU

Physical Layer PDU, Preamble, IFG, SFD along with MPDU. Not supported if cut-through frames are expected

enumerator kNETC_MPDU

MAC PDU, MAC Header, MSDU and FCS

enumerator kNETC_MSDU

MAC SDU, MPDU minus 12B MAC Header and 4B FCS. Not supported if cut-through frames are expected

enum _netc_port_sg_ogc_mode

Defines the Port's Stream Gate Open Gate Check mode.

Values:

enumerator kNETC_SGCheckSFD

Check whether frame SFD is within the open gate interval.

enumerator kNETC_SGCheckEntire

Check whether the entire frame is within the open gate interval.

enum _netc_port_intr_flags

Values:

enumerator kNETC_TxEmptyFlag

Tx FIFO empty flag.

enumerator kNETC_RxEmptyFlag

Rx FIFO empty flag.

enumerator kNETC_TxOverflowFlag

Tx overflow flag.

enumerator kNETC_TxUnderflowFlag

Tx underflow flag.

enumerator kNETC_RxOverflowFlag

Rx overflow flag.

enumerator kNETC_MagicPacketFlag

Magic packet detection indication flag.

enumerator kNETC_TxClockStopFlag

Tx clock stop detection flag.

enumerator kNETC_RxClockStopFlag

Rx clock stop detection flag.

enumerator kNETC_SpeedDuplexChangeFlag
Speed/Duplex Change flag

enumerator kNETC_MacMergeSMDErrFlag
MAC merge frame SMD error received event flag

enumerator kNETC_MacMergeAssemblyErrFlag
MAC merge frame assembly error event flag

enum _netc_port_loopback_mode_t
Defines the port MAC frame loopback mode.
Values:

enumerator kNETC_PortLpbWithExtTxClk
Port MAC frame loopback with external Tx clock.

enumerator kNETC_PortLpbWithIntTxClk
Port MAC frame loopback with internal Tx clock.

typedef enum _netc_port_tgsl_status netc_port_tgsl_status_t
Port time gate scheduling gate list status.

typedef enum _netc_port_discard_tpye netc_port_discard_tpye_t
Port Tx/Rx discard counter in the datapath processing pipeline or bridge forwarding processing function.

typedef enum _netc_port_tpidlist netc_port_tpidlist_t
Defines Port TPID acceptance.

typedef enum _netc_port_ts_select netc_port_ts_select_t
Defines port timestamp selection.

typedef struct _netc_port_qos_mode netc_port_qos_mode_t
Port Qos mode.

typedef struct _netc_port_parser_config netc_port_parser_config_t
Port Parser config.

typedef struct _netc_port_tg_config netc_port_tg_config_t
Port time gate config.

typedef struct _netc_port_tg_preemption_config netc_port_tg_preemption_config
Port time gate config when used with Frame Preemption.

typedef enum _netc_hw_preemption_mode netc_hw_preemption_mode_t
Port MAC preemption mode.

typedef enum _netc_hw_raf_size netc_hw_raf_size_t
Port MAC Remote Additional Fragment Size.

typedef struct _netc_port_preemption_config netc_port_preemption_config
Frame Preemption Portconfig.

typedef struct _netc_port_tc_cbs_config netc_port_tc_cbs_config_t
Configuration for the Credit Based Shaped for port TC.

Note: The 802.1Qav bandwidth availability parameters is is calculated as follows:

- $\text{idleSlope (bits)} = \text{portTxRate} * \text{bwWeight} / 100$
- $\text{sendSlope (bits)} = \text{portTxRate} * (100 - \text{bwWeight}) / 100$
- $\text{lowCredit (bits)} = \text{tcMaxFrameSize} * (100 - \text{bwWeight}) / 100$

- hiCredit (bits) calculation formula depends on the traffic class, Please refer to the Reference manual.
 - $\text{hiCredit (credits)} = (\text{enetClockFrequency} / \text{portTxRate}) * 100 * \text{hiCredit (bits)}$
-

typedef enum *_netc_tc_sdu_type* netc_tc_sdu_type_t

Type of PDU/SDU (Protocol/Service Data Unit).

Note: Overhead values which adding to the transmitted frame of Length are specified by Port SDU config as follows:

- PPDU = add rxPpduBco/txPpduBco + rxMacsecBco/txMacsecBco bytes
 - MPDU = add rxMacsecBco/txMacsecBco bytes
 - MSDU = minus 16B (12B MAC Header + 4B FCS)
-

typedef struct *_netc_port_tc_sdu_config* netc_port_tc_sdu_config_t

typedef struct *_netc_port_tx_tc_config* netc_port_tx_tc_config_t

Configuration for the port Tx Traffic Class.

typedef struct *_netc_port_discard_statistic* netc_port_discard_statistic_t

Switch or ENETC port Tx/Rx/Bridge discard statistic / reason.

typedef struct *_netc_port_vlan_classify_config* netc_port_vlan_classify_config_t

Port accepted Vlan classification config.

typedef struct *_netc_port_qos_classify_configs* netc_port_qos_classify_config_t

Port Qos Classification Config.

typedef struct *_netc_port_ipf_config_t* netc_port_ipf_config_t

Port Ingress Filter Config.

typedef struct *_netc_port_psf_p_isi_config* netc_port_psf_p_isi_config

PSFP port config.

Port ingress stream identification config

Note: The first stream identification find IS_EID has higher precedence value than the second, and the priority of the IS_EID found by the IPF is specified by the IPF entry RRR bit. The possible orderings are as follows

- RRR = 00b : IPF > enKC0 > enKC1 > defaultISEID
 - RRR = 01b : enKC0 > IPF > enKC1 > defaultISEID
 - RRR = 10b : enKC0 > enKC1 > IPF > defaultISEID
-

typedef struct *_netc_port_ethmac* netc_port_ethmac_t

typedef enum *_netc_port_sg_ogc_mode* netc_port_sg_ogc_mode_t

Defines the Port's Stream Gate Open Gate Check mode.

typedef struct *_netc_port_common* netc_port_common_t

Port common configuration.

typedef enum *_netc_port_intr_flags* netc_port_intr_flags_t

typedef enum *_netc_port_loopback_mode_t* netc_port_loopback_mode_t

Defines the port MAC frame loopback mode.

status_t NETC_PortConfig(NETC_PORT_Type *base, const *netc_port_common_t* *config)

Configure specified PORT.

Parameters

- base – NETC port module base address.
- config – Port configuration structure.

Returns

status_t

void NETC_PortSetMacAddr(NETC_PORT_Type *base, const uint8_t *macAddr)

Set the MAC address.

Parameters

- handle –
- macAddr –

bool NETC_PortIsPseudo(NETC_PORT_Type *base)

Check whether this port a pseudo MAC port.

Parameters

- base – PORT peripheral base address.

void NETC_PortGetDiscardStatistic(NETC_PORT_Type *base, *netc_port_discard_tpye_t* discardType, *netc_port_discard_statistic_t* *statistic)

Get specified PORT discard counter.

Parameters

- base – NETC port module base address.
- discardType – Port discard type.
- statistic – pointer to the statistic data

void NETC_PortClearDiscardReason(NETC_PORT_Type *base, *netc_port_discard_tpye_t* discardType, uint32_t reason0, uint32_t reason1)

Clean the Port Rx discard reason. Set the related bits to 1 to clear the specific reasons.

Parameters

- base – NETC port module base address.
- discardType – Port discard type.
- reason0 –
- reason1 –

static inline uint32_t NETC_PortGetTGSLStatus(NETC_PORT_Type *base)

Get port time gate scheduling gate list status.

Parameters

- base – NETC port module base address.

Returns

Port status flags which are ORed by the enumerators in the *netc_port_tgsl_status_t*

void NETC_PortEthMacGracefulStop(NETC_PORT_Type *base)

Do graceful stop for Port Ethernet MAC receive/transmit.

Parameters

- base – NETC port module base address.

```
static inline void NETC_PortSetSpeed(NETC_PORT_Type *base, uint16_t pSpeed)
```

Set port speed.

Parameters

- base – NETC port module base address.
- pSpeed – Transmit Port Speed = 10Mbps * (pSpeed+1), Used by ETS, Qbu and to determine if cut-through is permissible.

```
NETC_PORT_MIN_FRAME_SIZE
```

The port supported minimum/maximum frame size.

```
NETC_PORT_MAX_FRAME_SIZE
```

```
struct _netc_port_qos_mode
```

#include <fsl_netc.h> Port Qos mode.

Public Members

```
uint8_t qosVlanMap
```

Transmit QoS to VLAN PCP Mapping Profile index, only active on switch port

```
uint8_t vlanQosMap
```

Receive VLAN PCP/DE to QoS Mapping Profile index, only active on switch port

```
uint8_t defaultIpv
```

Port default IPV

```
uint8_t defaultDr
```

Port default DR

```
bool enVlanInfo
```

Enable use VLAN info to determine IPV and DR (base on VLANIPVMPaR0/1 and VLAN-DRMPaR)

```
bool vlanTagSelect
```

True: Outer VLAN, False: Inner VLAN. Active when enVlanInfo is true

```
struct _netc_port_parser_config
```

#include <fsl_netc.h> Port Parser config.

Public Members

```
uint8_t l2PloadCount
```

L2 payload fields size in bytes

```
bool enableL3Parser
```

Enable/Disable parser for L3

```
uint8_t l3PayloadCount
```

L3 payload fields size in bytes

```
bool enableL4Parser
```

Enable/Disable parser for L4

```
uint8_t l4PayloadCount
```

L4 payload fields size in bytes

```
struct _netc_port_tg_config
```

#include <fsl_netc.h> Port time gate config.

Public Members

uint16_t advOffset

Advance time offset in ns.

uint32_t holdSkew

Hold-Skew in ns, not effective on ports connected to a pseudo-MAC

struct __netc_port_tg_preemption_config

#include <fsl_netc.h> Port time gate config when used with Frame Preemption.

Public Members

uint16_t holdAdvance

the amount of time in ns prior to the Set-And-Hold-MAC time slot for asserting a Hold request. Used with frame Preemption.

uint16_t releaseAdvance

the amount of time in ns prior to the Set-And-Release-MAC time slot for asserting a Release request. Used with Frame Preemption.

struct __netc_port_preemption_config

#include <fsl_netc.h> Frame Preemption Portconfig.

Public Members

bool enMergeVerify

Enable verify the merged preemption frame, need to enable when preemptMode is not zero

uint8_t mergeVerifyTime

The nominal wait time between verification attempts in milliseconds, range in 1 ~ 128

netc_hw_preemption_mode_t preemptMode

When set to not zero, PMAC frames may be preempted by EMAC frames

netc_hw_raf_size_t raf_size

Additional Fragment Size. Indicates the smallest sized fragments that can be sent on Tx

bool PreemptionActive

Local preemption active. Indicates whether preemption is active for this port. This bit will be set if preemption is both enabled and has completed the verification process

struct __netc_port_tc_cbs_config

#include <fsl_netc.h> Configuration for the Credit Based Shaped for port TC.

Note: The 802.1Qav bandwidth availability parameters is is calculated as follows:

- idleSlope (bits) = portTxRate * bwWeight / 100
 - sendSlope (bits) = portTxRate * (100 - bwWeight) / 100
 - lowCredit (bits) = tcMaxFrameSize * (100 - bwWeight) / 100
 - hiCredit (bits) calculation formula depends on the traffic class, Please refer to the Reference manual.
 - hiCredit (credits) = (enetClockFrequency / portTxRate) * 100 * hiCredit (bits)
-

Public Members

uint8_t bwWeight

Percentage units of the port transmit rate and the credit-based shaper (range from 0 ~ 100), the sum of all traffic class credit-based shaper's bandwidth cannot exceed 100

uint32_t hiCredit

The maximum allowed accumulation of credits when conflicting transfers occur, in credit units ((enetClockFrequency / portTxRate) * 100)

struct __netc_port_tc_sdu_config

#include <fsl_netc.h>

Public Members

bool enTxMaxSduCheck

Enable Tx Max SDU check for Store and Forward frames, the frame which greater than maxSduSized will be discarded, Cut-Through frames will always perform Max SDU check

netc_tc_sdu_type_t sduType

Specifies the type of PDU/SDU whose length is being validated as seen on the link

uint16_t maxSduSized

Transmit Maximum SDU size in bytes, the dequeued frame will be discarded when it SDU size exceeds this value

struct __netc_port_tx_tc_config

#include <fsl_netc.h> Configuration for the port Tx Traffic Class.

Public Members

bool enPreemption

Frames from traffic class are transmitted on the preemptable MAC, not supported on internal port (ENETC 1 port and Switch port 4)

bool enTcGate

Enable the traffic class gate when no gate control list is operational, or when time gate scheduling is disabled.

bool enableTsd

Enable Time Specific Departure traffic class, only applicable to ENETC

bool enableCbs

Enable Credit based shaper for traffic class

netc_port_tc_cbs_config_t cbsCfg

Configure transmit traffic class credit based shaper (PTC0CBSR0/PTC0CBSR1) if enableCbs set to true

struct __netc_port_discard_statistic

#include <fsl_netc.h> Switch or ENETC port Tx/Rx/Bridge discard statistic / reason.

Public Members

uint32_t count

Count of discarded frames. PRXDCR, PTXDCR or BPD CR.

uint32_t reason0

Discard Reason. Find bit detail from PT/RXDCRR0 or BPDCRR0.

uint32_t reason1

Discard Reason. Find bit detail from PT/RXDCRR1 or BPDCRR1.

struct __netc_port_vlan_classify_config

#include <fsl_netc.h> Port accepted Vlan classification config.

Public Members

uint8_t innerMask

Bitmap identifying which TPIDs are acceptable as Inner VLAN tag. See PTAR

uint8_t outerMask

Bitmap identifying which TPIDs are acceptable as Outter VLAN tag. See PTAR

struct __netc_port_qos_classify_configs

#include <fsl_netc.h> Port Qos Classification Config.

Public Members

uint8_t vlanQosMap

Receive VLAN PCP/DE to QoS Mapping Profile index

uint8_t defaultIpv

Port default IPV

uint8_t defaultDr

Port default DR

bool enVlanInfo

Enable use VLAN info to determine IPV and DR ,base on VLAN to IPV map (VLANIPVM-PaR0/1) and VLAN to DR map (VLANDRMPaR)

bool vlanTagSelect

True: Use received Outer VLAN, False: Use received Innner VLAN. Active when en-VlanInfo is true

struct __netc_port_ipf_config_t

#include <fsl_netc.h> Port Ingress Filter Config.

Public Members

bool enL2Dos

Enable port L2 Ethernet DoS Protection

bool enL3Dos

Enable port L3 IP DoS Protection

bool enIPFTable

Enable port IPF lookup

struct __netc_port_psfpi_config

#include <fsl_netc.h> PSFP port config.

Port ingress stream identification config

Note: The first stream identification find IS_EID has higher precedence value than the second, and the priority of the IS_EID found by the IPF is specified by the IPF entry RRR bit. The possible orderings are as follows

- RRR = 00b : IPF > enKC0 > enKC1 > defaultISEID
 - RRR = 01b : enKC0 > IPF > enKC1 > defaultISEID
 - RRR = 10b : enKC0 > enKC1 > IPF > defaultISEID
-

Public Members

uint16_t defaultISEID

Default Ingress Stream Entry ID, has lower precedence value than ISI entry and IPF entry defined IS_EID. 0xFFFF means NULL

bool enKC1

Enable do the second stream identification with key construction rule 1 or rule 3

bool enKC0

Enable do the first stream identification with key construction rule 0 or rule 2

bool kcPair

Indicates which Key Construction pair to use for this port, false - user pair0. true - use pair1 only applicable for Switch

struct __netc_port_ethmac

#include <fsl_netc.h>

Public Members

bool enableRevMii

Enable RevMII mode.

netc_port_ts_select_t txTsSelect

Tx timestamp clock source.

bool isTsPointPhy

True: Timestamp is captured based on PHY SFD detect pulse on Rx and Tx for 2-step timestamping. False: Based on SFD detect at boundary of MAC merge layer and pins/protocol gaskets.

netc_hw_mii_mode_t miiMode

MII mode.

netc_hw_mii_speed_t miiSpeed

MII Speed.

netc_hw_mii_duplex_t miiDuplex

MII duplex.

bool enTxPad

Enable ETH MAC Tx Padding, which will pad the frame to a minimum of 60 bytes and append 4 octets of FCS.

uint8_t rxMinFrameSize

Receive Minimum Frame Length size in bytes, range in 18 ~ 64, received frames shorter than 18B are discarded silently. Both for express MAC and preemptable MAC.

uint16_t rxMaxFrameSize

Receive Maximum Frame Length size in bytes, up to 2000, received frames that exceed this stated maximum are truncated. Both for express MAC and preemptable MAC.

netc_port_preemption_config PreemptionConfig

Frame Preemption configuration

bool rgmiiClkStop

True: RGMII transmit clock is stoppable during low power idle. False: It's not stoppable.

bool enableHalfDuplexFlowCtrl

Enable/Disable half-duplex flow control.

uint16_t maxBackPressOn

Maximum amount of time backpressure can stay asserted before stopping to prevent excess defer on link partner, in byte times.

uint16_t minBackPressOff

Minimum amount of time backpressure will stay off after reaching the ON max, before backpressure can reassert after checking if icm_pause_notification is still or again asserted, in byte times.

uint32_t txWakeupTimeCycleEEE

Energy Efficient Ethernet feature. Defines the number of NETC cycles (which represents time) required by the PHY to wait before transmitting a new frame after the application has indicated it wants to end the low power state.

uint32_t txSleepTimeCycleEEE

Energy Efficient Ethernet feature. Defines the number of NETC cycles (which represents time) where Tx is idle before mac transmits low power EEE. A value of 0 does not activate low power EEE transmission.

struct __netc_port_common

#include <fsl_netc.h> Port common configuration.

Public Members

netc_port_vlan_classify_config_t acceptTpid

Port acceptable VLAN tpid configure.

netc_port_ts_select_t rxTsSelect

Eth MAC Rx or pseudo MAC Tx timestamp clock source

uint16_t pSpeed

Transmit Port Speed = 10Mbps * (pSpeed+1), Used by ETS, Qbu and to determine if cut-through is permissible

uint8_t rxMacsecBco

Port receive MACSec byte count overhead which due to MACSec encapsulation

uint8_t rxPpduBco

Port receive PPDU Byte count overhead which includes IPG, SFD and Preamble

uint8_t txMacsecBco

Port transmit MACSec byte count overhead which due to MACSec encapsulation

uint8_t txPpduBco

Port transmit PPDU Byte count overhead which includes IPG, SFD and Preamble

netc_port_sg_ogc_mode_t ogcMode

Stream Gate Open Gate Check mode, 0b is check whether SFD is within the open gate interval, 1b is check whether the entire frame is within the open gate interval

uint32_t pDelay

Link propagation delay in ns

uint8_t macAddr[6]

Port MAC address, used for Switch egress frame modification action or ENETC SIO primary MAC address

netc_port_qos_classify_config_t qosMode

Port Rx Qos Classification config

netc_port_ipf_config_t ipfCfg

Port ingress port filter configuration

netc_port_tg_config_t timeGate

Port Tx time gate config

netc_port_parser_config_t parser

Port Rx Parser config

2.74 Hardware Port MAC

enum *_netc_port_phy_mac_type*

Defines the Ethernet MAC physical port type.

Values:

enumerator *kNETC_ExpressMAC*

The MAC which handles express traffic when frame preemption is enabled or handles all traffic when frame preemption is disabled.

enumerator *kNETC_PreemptableMAC*

The MAC which handles preemptive traffic when frame preemption is enabled.

enum *_netc_port_preemption_verify_status*

Defines the state of the mac merge sublayer with respect to verification as defined in IEEE Std 802.3br-2016.

Values:

enumerator *kNETC_VerifyDisable*

Verification is disabled

enumerator *kNETC_VerifyInProgress*

Verification is in progress

enumerator *kNETC_VerifySuccess*

Verification was successful

enumerator *kNETC_VerifyFaile*

Verification failed

enumerator *kNETC_VerifyUndefined*

Verification is in an undefined state

typedef enum *_netc_port_phy_mac_type* *netc_port_phy_mac_type_t*

Defines the Ethernet MAC physical port type.

typedef enum *_netc_port_preemption_verify_status* netc_port_preemption_verify_status_t
Defines the state of the mac merge sublayer with respect to verification as defined in IEEE Std 802.3br-2016.

typedef struct *_netc_port_phy_mac_preemption_status* netc_port_phy_mac_preemption_status_t
Port MAC preemption Status.

typedef struct *_netc_port_phy_mac_traffic_statistic* netc_port_phy_mac_traffic_statistic_t
Ethernet MAC physical port traffic (Tx/Rx) statistics counters, when enable frame preemption, one physical MAC will be divided into a pMAC and a eMAC and statistics counters will also have two groups.

typedef struct *_netc_port_phy_mac_discard_statistic* netc_port_phy_mac_discard_statistic_t
Ethernet MAC physical port frame discard/errors status statistics counters, when enable frame preemption, one physical MAC will be divided into a pMAC and a eMAC and statistics counters will also have two groups.

typedef struct *_netc_port_phy_mac_preemption_statistic*
netc_port_phy_mac_preemption_statistic_t
Ethernet physical MAC port preemption (Tx/Rx) related statistics counters.

typedef struct *_netc_port_pseudo_mac_traffic_statistic* netc_port_pseudo_mac_traffic_statistic_t
Ethernet pseudo MAC port traffic (Tx/Rx) statistics counters.

uint32_t NETC_GetPortMacInterruptFlags(NETC_ETH_LINK_Type *base,
netc_port_phy_mac_type_t macType)

Get port MAC interrupt flags.

Parameters

- base – NETC ETH link base register.
- mac – MAC type.

void NETC_ClearPortMacInterruptFlags(NETC_ETH_LINK_Type *base, netc_port_phy_mac_type_t
macType, uint32_t mask)

Clear port MAC interrupt flags.

Parameters

- base – NETC ETH link base register.
- mac – MAC type.
- mask – Bit mask of interrupts to enable. See netc_port_intr_flags_t for the set of constants that should be OR'd together to form the bit mask.

void NETC_EnablePortMacInterrupts(NETC_ETH_LINK_Type *base, netc_port_phy_mac_type_t
macType, uint32_t mask, bool enable)

Enable/Disable port MAC interrupts.

Parameters

- base – NETC ETH link base register.
- mac – MAC type.
- mask – Bit mask of interrupts to enable. See netc_port_intr_flags_t for the set of constants that should be OR'd together to form the bit mask.
- enable – Enable/Disable interrupts.

status_t NETC_PortEnableLoopback(NETC_ETH_LINK_Type *base, netc_port_loopback_mode_t
loopMode, bool enable)

Enable/Disable Loopback for specified MAC.

Parameters

- base –
- loopMode –
- enable –

Returns

status_t

static inline *netc_hw_mii_mode_t* NETC_PortGetMIIMode(NETC_ETH_LINK_Type *base)

Get ethernet MAC port MII mode.

Parameters

- base – Ethernet MAC port peripheral base address.

Returns

netc_hw_mii_mode_t

status_t NETC_PortSetMII(NETC_ETH_LINK_Type *base, *netc_hw_mii_mode_t* miiMode, *netc_hw_mii_speed_t* speed, *netc_hw_mii_duplex_t* duplex)

Configure ethernet MAC interface mode, speed and duplex for specified PORT.

Parameters

- base – Ethernet MAC port peripheral base address.
- miiMode – The Ethernet MAC MII mode.
- speed – The Ethernet MAC speed.
- duplex – The Ethernet MAC duplex.

Returns

status_t

status_t NETC_PortSetMaxFrameSize(NETC_ETH_LINK_Type *base, uint16_t size)

Set the maximum supported received frame size.

Parameters

- base – Ethernet MAC port peripheral base address.
- size – Maximum frame size to set.

Returns

status_t

status_t NETC_PortConfigEthMac(NETC_ETH_LINK_Type *base, const *netc_port_ethmac_t* *config)

Configure ethernet MAC for specified PORT. Set the MII mode, speed/duplex, reverse mode, etc.

Parameters

- base – Ethernet MAC port peripheral base address.
- config – The Ethernet MAC configuration.

Returns

status_t

static inline void NETC_PortConfigEthMacPreemption(NETC_ETH_LINK_Type *base, const *netc_port_preemption_config* *config)

Configure ethernet MAC for Frame preemption on specified PORT.

Parameters

- base – Ethernet MAC port peripheral base address.

- config – The Ethernet MAC configuration.

Returns

status_t

static inline void NETC_PortSoftwareResetEthMac(NETC_ETH_LINK_Type *base)

Do software reset for Ethernet MAC.

Note: This can reset all statistic counters.

Parameters

- base – PORT MAC peripheral base address.

static inline void NETC_PortGetPhyMacPreemptionStatus(NETC_ETH_LINK_Type *base,
netc_port_phy_mac_preemption_status_t
*status)

Get Ethernet MAC preemption status.

Parameters

- base – PORT MAC peripheral base address.
- status – Point to the buffer which store status.

static inline void NETC_PortGetPhyMacPreemptionControl(NETC_ETH_LINK_Type *base,
netc_port_preemption_config *config)

Get Ethernet MAC preemption control parameters.

Parameters

- base – PORT MAC peripheral base address.
- config – Pointer to the NETC port preemption configuration.

void NETC_PortGetPhyMacTxStatistic(NETC_ETH_LINK_Type *base, netc_port_phy_mac_type_t
macType, netc_port_phy_mac_traffic_statistic_t *statistic)

Get Ethernet MAC Tx Traffic Statistics .

Parameters

- base – PORT MAC peripheral base address.
- macType – Express MAC or Preemptable MAC.
- status – Point to the buffer which store statistics.

void NETC_PortGetPhyMacRxStatistic(NETC_ETH_LINK_Type *base, netc_port_phy_mac_type_t
macType, netc_port_phy_mac_traffic_statistic_t *statistic)

Get Ethernet MAC Rx Traffic Statistics .

Parameters

- base – PORT MAC peripheral base address.
- macType – Express MAC or Preemptable MAC.
- status – Point to the buffer which store statistics.

void NETC_PortGetPhyMacDiscardStatistic(NETC_ETH_LINK_Type *base,
netc_port_phy_mac_type_t macType,
netc_port_phy_mac_discard_statistic_t *statistic)

Get Ethernet MAC Rx/Tx Drops/Errors Statistics .

Parameters

- base – PORT MAC peripheral base address.

- macType – Express MAC or Preemptable MAC.
- status – Point to the buffer which store statistics.

```
void NETC_PortGetPhyMacPreemptionStatistic(NETC_ETH_LINK_Type *base,
                                           netc_port_phy_mac_preemption_statistic_t
                                           *statistic)
```

Get Ethernet Preemptable MAC preemption Statistics .

Parameters

- base – PORT MAC peripheral base address.
- status – Point to the buffer which store statistics.

```
status_t NETC_PortConfigTxIpGpreamble(NETC_ETH_LINK_Type *base, uint8_t preambleCnt,
                                       uint8_t ipgLen)
```

Configure the port MAC flexible preamble and IPG length.

Parameters

- base – PORT MAC peripheral base address.
- preambleCnt – Flexible Preamble Count. Valid values are 1 to 7(default).
- ipgLen – Transmit inter-packet gap value. Valid values are 4 to 24(default) with 12 being default.

Returns

status_t

```
struct _netc_port_phy_mac_preemption_status
#include <fsl_netc.h> Port MAC preemption Status.
```

Public Members

bool mergeActive

Transmit preemption is active or not

netc_port_preemption_verify_status_t verifyStatus

Transmit preemption is active or not

```
struct _netc_port_phy_mac_traffic_statistic
```

#include <fsl_netc.h> Ethernet MAC physical port traffic (Tx/Rx) statistics counters, when enable frame preemption, one physical MAC will be divided into a pMAC and a eMAC and statistics counters will also have two groups.

Public Members

uint64_t totalOctet

Count of MAC received/transmitted good/error Ethernet octets.

uint64_t validOctet

Count of MAC received/transmitted good Ethernet octets.

uint64_t pauseFrame

Count of MAC received/transmitted valid PAUSE frames.

uint64_t validFrame

Count of MAC received/transmitted valid frames.

uint64_t vlanFrame

Count of MAC received/transmitted valid VLAN tagged frames.

uint64_t unicastFrame

Count of MAC received/transmitted valid unicast frames.

uint64_t multicastFrame

Count of MAC received/transmitted valid multicast frames.

uint64_t broadcastFrame

Count of MAC received/transmitted valid broadcast frames.

uint64_t totalPacket

Count of MAC received/transmitted good/error packets.

uint64_t rxMinPacket

Count of MAC received min to 63-octet packets.

uint64_t total64BPacket

Count of MAC received/transmitted 64 octet packets.

uint64_t total65To127BPacket

Count of MAC received/transmitted 65 to 127 octet packets.

uint64_t total128To255BPacket

Count of MAC received/transmitted 128 to 255 octet packets.

uint64_t total256To511BPacket

Count of MAC received/transmitted 256 to 511 octet packets.

uint64_t total511To1023BPacket

Count of MAC received/transmitted 512 to 1023 octet packets.

uint64_t total1024To1522BPacket

Count of MAC received/transmitted 1024 to 1522 octet packets.

uint64_t total1523ToMaxBPacket

Count of MAC received/transmitted 1523 to Max octet packets.

uint64_t controlPacket

Count of MAC received/transmitted control packets.

struct _netc_port_phy_mac_discard_statistic

#include <fsl_netc.h> Ethernet MAC physical port frame discard/errors status statistics counters, when enable frame preemption, one physical MAC will be divided into a pMAC and a eMAC and statistics counters will also have two groups.

Public Members

uint64_t rxError

Count of MAC received error frames.

uint64_t rxUndersized

Count of MAC received undersized frames.

uint64_t rxOversized

Count of MAC received oversized frames.

uint64_t rxErrorFCS

Count of MAC received check sequence (FCS) error frames.

uint64_t rxFragment

Count of MAC frames which is shorter than the MIN length and received with a wrong FCS/CRC.

uint64_t rxJabber

Count of MAC frames which is larger than the MAX length and received with a wrong FCS/CRC.

uint64_t rxDiscard

Count of MAC drops frame.

uint64_t rxDiscardNoTruncated

Count of MAC non-truncated drops frame.

uint64_t txErrorFCS

Count of MAC transmitted bad FCS frames.

uint64_t txUndersized

Count of MAC transmitted less than 64B with good FCS frames.

struct _netc_port_phy_mac_preemption_statistic

#include <fsl_netc.h> Ethernet physical MAC port preemption (Tx/Rx) related statistics counters.

Public Members

uint32_t rxReassembledFrame

Count of MAC frames that were successfully reassembled and delivered to the MAC.

uint32_t rxReassembledError

Count of MAC frames with reassembly errors.

uint32_t rxMPacket

Count of the number of additional mPackets received due to preemption.

uint32_t rxSMDError

Count of received MAC frames / MAC frame fragments rejected due to unknown SMD.

uint32_t txPreemptionReq

Count of the number of tx preemption HOLD requests.

uint32_t txMPacket

Count of the number of additional mPackets transmitted due to preemption.

struct _netc_port_pseudo_mac_traffic_statistic

#include <fsl_netc.h> Ethernet pseudo MAC port traffic (Tx/Rx) statistics counters.

Public Members

uint64_t totalOctet

Count of MAC received/transmitted octets.

uint64_t unicastFrame

Count of MAC received/transmitted unicast frames.

uint64_t multicastFrame

Count of MAC received/transmitted multicast frames.

uint64_t broadcastFrame

Count of MAC received/transmitted broadcast frames .

2.75 Hardware Port Rx

```
static inline void NETC_PortSetParser(NETC_PORT_Type *base, const netc_port_parser_config_t
                                     *config)
```

Set port Parser.

Parameters

- base – PORT peripheral base address.
- config – The port Parser configuration.

```
static inline void NETC_PortSetVlanClassify(NETC_PORT_Type *base, const
                                             netc_port_vlan_classify_config_t *config)
```

Set port acceptable VLAN.

Parameters

- base – PORT peripheral base address.
- config – The port acceptable vlan classification configuration.

```
static inline status_t NETC_PortSetQosClassify(NETC_PORT_Type *base, const
                                                netc_port_qos_classify_config_t *config)
```

Set port Qos Classification.

Parameters

- base – PORT peripheral base address.
- config – The port QoS classification configuration.

Returns

status_t

```
static inline void NETC_PortSetIPF(NETC_PORT_Type *base, const netc_port_ipf_config_t
                                   *config)
```

Set port ingress filter.

Parameters

- base – PORT peripheral base address.
- config – The port ingress filter configuration.

```
static inline void NETC_PortSetISI(NETC_PORT_Type *base, const netc_port_psfpi_isi_config
                                   *config)
```

Set port Ingress stream identification.

Parameters

- base – PORT peripheral base address.
- config – The port Ingress stream identification configuration.

2.76 Hardware Port Tx

```
static inline status_t NETC_PortConfigTGS(NETC_PORT_Type *base, const netc_port_tg_config_t
                                           *config)
```

Configure the port time gating Scheduling.

Parameters

- base –

- config –

Returns

status_t

status_t NETC_PortConfigTcCBS(NETC_PORT_Type *base, netc_hw_tc_idx_t tcIdx, const netc_port_tc_cbs_config_t *config)

Config the Credit-Based Shaper (CBS) for specified Port Traffic Class.

Parameters

- base –
- tcIdx –
- config –

Returns

status_t

static inline status_t NETC_PortConfigTcMaxSDU(NETC_PORT_Type *base, netc_hw_tc_idx_t tcIdx, const netc_port_tc_sdu_config_t *config)

Config the max Transmit max SDU for specified Port Traffic Class.

Parameters

- base –
- tcIdx –
- config –

Returns

status_t

static inline status_t NETC_PortGetTcMaxSDU(NETC_PORT_Type *base, netc_hw_tc_idx_t tcIdx, netc_port_tc_sdu_config_t *config)

Read the max Transmit max SDU for specified Port Traffic Class.

Parameters

- base –
- tcIdx –
- config –

Returns

status_t

static inline void NETC_PortConfigTcPreemption(NETC_PORT_Type *base, netc_hw_tc_idx_t tcIdx, const bool enable)

Config Frame Preemption for specified Port Traffic Class.

Parameters

- base – NETC PORT base peripheral address
- tcIdx – traffic class index
- enable – enable/disable feature on traffic class

static inline void NETC_PortGetTcPreemption(NETC_PORT_Type *base, netc_hw_tc_idx_t tcIdx, bool *enabled)

Get Frame Preemption configuration for specified Port Traffic Class.

Parameters

- base – NETC PORT base peripheral address

- tcIdx – traffic class index
- enabled – port tx traffic class enabled flag

```
static inline void NETC_PortGetTGSFPConfig(NETC_PORT_Type *base,  
                                           netc_port_tg_preemption_config *config)
```

Get the port time gating Scheduling configuration specific for when used with Frame Pre-emption.

Parameters

- base – NETC PORT base peripheral address
- config –

2.77 Hardware Station Interface(SI)

```
NETC_SI_TXDESCRIP_RD_FL(n)
```

Defines for read format.

```
NETC_SI_TXDESCRIP_RD_TSE_MASK
```

```
NETC_SI_TXDESCRIP_RD_TXSTART(n)
```

```
NETC_SI_TXDESCRIP_RD_DR(n)
```

```
NETC_SI_TXDESCRIP_RD_IPV(n)
```

```
NETC_SI_TXDESCRIP_RD_PORT(n)
```

```
NETC_SI_TXDESCRIP_RD_TSR_MASK
```

```
NETC_SI_TXDESCRIP_RD_SMSO_MASK
```

```
NETC_SI_TXDESCRIP_RD_FLQ(n)
```

```
enum _netc_hw_enetc_si_vlan_type
```

VLAN Ethertypes.

Values:

```
enumerator kNETC_ENETC_StanCVlan  
Standard C-VLAN 0x8100.
```

```
enumerator kNETC_ENETC_StanSVlan  
Standard S-VLAN 0x88A8.
```

```
enumerator kNETC_ENETC_CustomVlan1  
Custom VLAN as defined by CVLANR1[ETYPE].
```

```
enumerator kNETC_ENETC_CustomVlan2  
Custom VLAN as defined by CVLANR2[ETYPE].
```

```
enum _netc_hw_enetc_si_rxr_group
```

SI receive BD ring group index.

Values:

```
enumerator kNETC_SiBDRGroupOne  
SI Rx BD ring group index one.
```

```
enumerator kNETC_SiBDRGroupTwo  
SI Rx BD ring group index two.
```

enum `_netc_tx_bdr_flags`

Status/Interrupts flags for the TX BDR. Each flag get its own bit thus it support bit AND/OR operation.

Values:

enumerator `kNETC_TxBDRSystemBusErrorFlag`

enumerator `kNETC_TxBDRBusyFlag`

enumerator `kNETC_TxBDRStatusFlagsMask`

enum `_netc_rx_bdr_flags`

Status/Interrupts flags for the RX BDR. Each flag get its own bit thus it support bit AND/OR operation.

Values:

enumerator `kNETC_RxBDRSystemBusErrorFlag`

enumerator `kNETC_RxBDREmptyFlag`

enum `_netc_psi_msg_flags_t`

PSI message interrupt type.

Values:

enumerator `kNETC_PsiRxMsgFromVsi1Flag`

Message receive interrupt enable, initiated by VSI1.

enumerator `kNETC_PsiRxMsgFromVsi2Flag`

Message receive interrupt enable, initiated by VSI2.

enumerator `kNETC_PsiRxMsgFromVsi3Flag`

Message receive interrupt enable, initiated by VSI3.

enumerator `kNETC_PsiFLRFromVsi1Flag`

Function level reset interrupt enable, initiated by VSI1.

enum `_netc_vsi_msg_flags`

VSI message interrupt flags.

Values:

enumerator `kNETC_VsiMsgTxFlag`

Message sent to PSI has completed and response received.

enumerator `kNETC_VsiMsgRxFlag`

Message received from PSI.

enum `_netc_vsi_number`

VSI number bit map, VSI1 starts from bit1.

Values:

enumerator `kNETC_Vsi1`

enumerator `kNETC_Vsi2`

enumerator `kNETC_Vsi3`

enum `_enetc_si_bdr_priority`

ENETC Station Interface BD Ring priority enumeration.

Values:

```
enumerator kNETC_SIBdrPriorityLowest
enumerator kNETC_SIBdrPriority0
enumerator kNETC_SIBdrPriority1
enumerator kNETC_SIBdrPriority2
enumerator kNETC_SIBdrPriority3
enumerator kNETC_SIBdrPriority4
enumerator kNETC_SIBdrPriority5
enumerator kNETC_SIBdrPriority6
enumerator kNETC_SIBdrPriority7
enumerator kNETC_SIBdrPriorityHighest

typedef enum _netc_hw_enetc_si_vlan_type netc_hw_enetc_si_vlan_type
    VLAN Ethertypes.

typedef enum _netc_hw_enetc_si_rxr_group netc_hw_enetc_si_rxr_group
    SI receive BD ring group index.

typedef struct _netc_hw_enetc_si_config netc_hw_enetc_si_config_t
    Station Interface configuration.

typedef struct _netc_si_l2mf_config netc_si_l2mf_config_t
    L2 Mac Filter Configuration for SI.

typedef struct _netc_si_l2vf_config netc_si_l2vf_config_t
    L2 VLAN Filter Configuration for SI.

typedef struct _netc_si_discard_statistic netc_si_discard_statistic_t
    SI frame drop statistic struct.

typedef struct _netc_si_traffic_statistic netc_si_traffic_statistic_t
    SI traffic statistic struct.

typedef struct _netc_si_config netc_si_config_t
    SI Configuration.

typedef union _netc_tx_bd netc_tx_bd_t
    Transmit Buffer Descriptor format.
    A union type cover the BD used as Standard/Extended/WriteBack format.

typedef union _netc_rx_bd netc_rx_bd_t
    Receive Buffer Descriptor format.

typedef struct _netc_tx_bdr_config netc_tx_bdr_config_t
    Configuration for the SI Tx Buffer Descriptor Ring Configuration.

typedef struct _netc_tx_bdr netc_tx_bdr_t
    Transmit BD ring handler data structure.

typedef enum _netc_tx_bdr_flags netc_tx_bdr_flags_t
    Status/Interrupts flags for the TX BDR. Each flag get its own bit thus it support bit AND/OR
    operation.

typedef struct _netc_rx_bdr_config netc_rx_bdr_config_t
    Configuration for the SI Rx Buffer Descriptor Ring Configuration.
```

```
typedef struct _netc_rx_bdr netc_rx_bdr_t
```

Receive BD ring handler data structure.

```
typedef enum _netc_rx_bdr_flags_t netc_rx_bdr_flags_t
```

Status/Interrupts flags for the RX BDR. Each flag get its own bit thus it support bit AND/OR operation.

```
typedef struct _netc_bdr_config netc_bdr_config_t
```

Configuration for the buffer descriptors ring.

```
typedef enum _netc_psi_msg_flags_t netc_psi_msg_flags_t
```

PSI message interrupt type.

```
typedef enum _netc_vsi_msg_flags netc_vsi_msg_flags_t
```

VSI message interrupt flags.

```
typedef enum _netc_vsi_number netc_vsi_number_t
```

VSI number bit map, VSI1 starts from bit1.

```
typedef struct _netc_psi_rx_msg netc_psi_rx_msg_t
```

PSI receive message information.

```
typedef struct _netc_vsi_msg_tx_status netc_vsi_msg_tx_status_t
```

VSI message transmit status.

```
typedef enum _enetc_si_bdr_priority enetc_si_bdr_priority_t
```

ENETC Station Interface BD Ring priority enumeration.

```
static inline void NETC_ClearTxDescriptor(netc_tx_bd_t *txDesc)
```

```
static inline void NETC_CopyTxDescriptor(netc_tx_bd_t *txDescDst, netc_tx_bd_t *txDescSrc)
```

```
static inline void NETC_SIEnable(ENETC_SI_Type *base, bool enable)
```

Enable the Station Interface(SI)

Parameters

- base –

```
static inline void NETC_SIRxRingEnable(ENETC_SI_Type *base, uint8_t ring, bool enable)
```

Enable the specified Rx BD ring.

Parameters

- base –
- ring – The ring index.
- enable – Enable/Disable the ring.

```
static inline void NETC_SIEnablePromisc(ENETC_SI_Type *base, netc_packet_type_t type, bool enable)
```

Enable/Disable unicast/multicast/boardcast promisc mode for specified SI.

Parameters

- base –
- type –
- enable –

```
static inline void NETC_SISetTxProducer(ENETC_SI_Type *base, uint8_t ring, uint16_t producer)
```

Set producer index of specified Tx BD ring.

Parameters

- base – SI base address.
- ring – BD ring index.
- producer – The producer index of specified ring.

static inline uint16_t NETC_SIGetTxConsumer(ENETC_SI_Type *base, uint8_t ring)

Get consumer index of specified Tx BD ring.

Parameters

- base – SI base address.
- ring – BD ring index.

Returns

consumer The consumer index of specified ring.

static inline void NETC_SISetRxConsumer(ENETC_SI_Type *base, uint8_t ring, uint16_t consumer)

Set consumer index of specified Rx BD ring.

Parameters

- base – SI base address.
- ring – BD ring index.
- consumer – The consumer index of specified ring.

static inline uint16_t NETC_SIGetRxProducer(ENETC_SI_Type *base, uint8_t ring)

Get producer index of specified Rx BD ring.

Parameters

- base – SI base address.
- ring – BD ring index.

Returns

producer The producer index of specified ring.

status_t NETC_SISetTxBDR(ENETC_SI_Type *base, uint8_t ring, const netc_tx_bdr_config_t *bdrConfig)

Configure the Transmit Buffer Descriptor Ring for specified SI.

Parameters

- base – SI base address.
- ring – BD ring index.
- bdrConfig – The BD ring configuration.

Returns

status_t

status_t NETC_SISetRxBDR(ENETC_SI_Type *base, uint8_t ring, const netc_rx_bdr_config_t *bdrConfig)

Configure the Rx Buffer Descriptor Ring for specified SI.

Parameters

- base – SI base address.
- ring – BD ring index.
- bdrConfig – The BD ring configuration.

Returns

status_t

```
static inline void NETC_SIMapVlanToIpv(ENETC_SI_Type *base, uint8_t pcpDei, uint8_t ipv)
```

Enable the mapping of VLAN to IPV.

Parameters

- base – SI base address.
- pcpDei – The VLAN tag pcp dei value (use NETC_VLAN_PCP_DEI_VALUE macro).
- ipv – The IPV value for this VLAN mapping.

```
static inline void NETC_SIEnableVlanToIpv(ENETC_SI_Type *base, bool enable)
```

Enable the mapping of VLAN to IPV.

Parameters

- base – SI base address.
- enable – Whether enable mapping.

Returns

status_t

```
static inline void NETC_SIMapIpvToRing(ENETC_SI_Type *base, uint8_t ipv, uint8_t ring)
```

Set the IPV to ring mapping.

Parameters

- base – SI base address.
- ipv – IPV value to be mapped.
- ring – The Rx BD ring index to be mapped.

Returns

status_t

```
static inline void NETC_SISetRxBDRGroup(ENETC_SI_Type *base, uint8_t groupNum, uint8_t ringPerGroup)
```

Set the group of Rx BD ring.

Parameters

- base – SI base address.
- groupNum – Total group number.
- ringPerGroup – Rings per group.

Returns

status_t

```
static inline void NETC_SISetDefaultRxBDRGroup(ENETC_SI_Type *base,  
                                              netc_hw_enetc_si_rxr_group groupIdx)
```

Set the default used receive Rx BD ring group.

Note: The IPV mapped ring index is the relative index inside the default used group.

Parameters

- base – SI base address.
- groupNum – The default Rx group index.

Returns

status_t

```
static inline void NETC_SICleanTxIntrFlags(ENETC_SI_Type *base, uint16_t txFrameIntrMask,
                                           uint16_t txThresIntrMask)
```

Clean the SI transmit interrupt flags.

Parameters

- base – SI base address.
- txFrameIntrMask – IPV value to be mapped, bit x represents ring x.
- txThresIntrMask – The Rx BD ring index to be mapped, bit x represents ring x.

```
static inline void NETC_SICleanRxIntrFlags(ENETC_SI_Type *base, uint32_t rxIntrMask)
```

Clean the SI receive interrupt flags.

Parameters

- base – SI base address.
- rxIntrMask – Rx interrupt bit mask, bit x represents ring x.

```
void NETC_SIPsiEnableInterrupt(ENETC_SI_Type *base, uint32_t mask, bool enable)
```

PSI enables/disables specified interrupt.

Parameters

- base – SI base address.
- mask – The interrupt mask, refer to `netc_psi_msg_flags_t` which should be OR'd together.
- enable – Enable/Disable the interrupt.

```
static inline uint32_t NETC_SIPsiGetStatus(ENETC_SI_Type *base)
```

PSI gets interrupt event flag status.

Parameters

- base – SI base address.

Returns

The interrupt mask, refer to `netc_psi_msg_flags_t` which should be OR'd together.

```
static inline void NETC_SIPsiClearStatus(ENETC_SI_Type *base, uint32_t mask)
```

PSI clears interrupt event flag.

Parameters

- base – SI base address.
- mask – The interrupt mask, refer to `netc_psi_msg_flags_t` which should be OR'd together.

```
status_t NETC_SIPsiSendMsg(ENETC_SI_Type *base, uint16_t msg, netc_vsi_number_t vsi)
```

PSI sends message to specified VSI(s)

Parameters

- base – SI base address.
- msg – The message to be sent.
- vsi – The VSI number.

Returns

status_t

static inline bool NETC_SIPsiCheckTxBusy(ENETC_SI_Type *base, *netc_vsi_number_t* vsi)
PSI checks Tx busy flag which should be cleaned when VSI receive the message data.

Parameters

- base – SI base address.
- vsi – The VSI number.

Returns

The busy status of specified VSI.

status_t NETC_SIPsiSetRxBuffer(ENETC_SI_Type *base, *netc_vsi_number_t* vsi, uint64_t buffAddr)

PSI sets Rx buffer to receive message from specified VSI.

Note: The buffer memory size should be big enough for the message data from VSI

Parameters

- base – SI base address.
- vsi – The VSI number.
- buffAddr – The buffer address to store message data from VSI, must be 64 bytes aligned.

Returns

status_t

status_t NETC_SIPsiGetRxMsg(ENETC_SI_Type *base, *netc_vsi_number_t* vsi, *netc_psi_rx_msg_t* *msgInfo)

PSI gets Rx message from specified VSI.

Parameters

- base – SI base address.
- vsi – The VSI number.
- msgInfo – The Rx message information.

void NETC_SIVsiEnableInterrupt(ENETC_SI_Type *base, uint32_t mask, bool enable)

Enable VSI interrupt.

Parameters

- base – SI base address.
- mask – The interrupt mask, see *netc_vsi_msg_flags_t* which should be OR'd together.
- enable – Enable/Disable interrupt.

static inline uint32_t NETC_SIVsiGetStatus(ENETC_SI_Type *base)

Get VSI interrupt status.

Parameters

- base – SI base address.

Returns

A bitmask composed of *netc_vsi_msg_flags_t* enumerators OR'd together.

static inline void NETC_SIVsiClearStatus(ENETC_SI_Type *base, uint32_t mask)

Clear VSI interrupt status.

Parameters

- base – SI base address.
- mask – The interrupt mask, see `netc_vsi_msg_flags_t` which should be OR'd together.

status_t NETC_SIVsiSendMsg(ENETC_SI_Type *base, uint64_t msgAddr, uint32_t msgLen)

VSI sends message to PSI.

Parameters

- base – SI base address.
- msgAddr – Address to store message ready to be sent, must be 64 bytes aligned.
- msgLen – The message length, must be 32 bytes aligned.

Returns

status_t

void NETC_SIVsiCheckTxStatus(ENETC_SI_Type *base, *netc_vsi_msg_tx_status_t* *status)

Check VSI Tx status.

Parameters

- base – SI base address.
- status – The VSI Tx status structure.

status_t NETC_SIVsiReceiveMsg(ENETC_SI_Type *base, uint16_t *msg)

VSI receives message from PSI.

Parameters

- base – SI base address.
- msg – The message from PSI.

Returns

status_t

void NETC_SIGetDiscardStatistic(ENETC_SI_Type *base, *netc_si_discard_statistic_t* *statistic)

Get the discard statistic from SI layer.

Parameters

- base – SI base address.
- statistic – The statistic data.

void NETC_SIGetTrafficStatistic(ENETC_SI_Type *base, *netc_si_traffic_statistic_t* *statistic)

Get the traffic statistic from SI layer.

Parameters

- base – SI base address.
- statistic – The statistic data.

NETC_VLAN_PCP_DEI_VALUE(*pcp*, *dei*)

Macro to cover VLAN PCP, DEI value to internal used `pcpDei` value.

struct `_netc_hw_enetc_si_config`

`#include <fsl_enetc.h>` Station Interface configuration.

Public Members**uint32_t** bandWeight

Station interface traffic class bandwidth weight

uint32_t vlanCtrl

VLAN Ethertypes can be inserted by the SI driver, set with OR of netc_hw_enetc_si_vlan_type.

uint32_t antiSpoofEnable

Anti-spoofing enable

uint32_t vlanInsertEnable

Software SI-based VLAN Insertion enable, avtive when enSIBaseVlan is true

uint32_t vlanExtractEnable

SI-based VLAN removed from frame enable, avtive when enSIBaseVlan is true

uint32_t sourcePruneEnable

Source pruning enable

uint32_t rxRingUse

Number of Rx Rings to be used, when enable Rx ring group, this equal to the sum of all Rx group rings.

uint32_t txRingUse

Number of Tx Rings to be used, note that when SI is Switch management ENETC SI, the number not include Tx ring 0.

uint32_t valnToIpvEnable

Enable the VLAN PCP/DEI value (use NETC_VLAN_PCP_DEI_VALUE marco) to internal priority value mapping.

uint32_t rxBdrGroupNum

Rx BD ring group number, range in 0 ~ 2.

uint32_t ringPerBdrGroup

The ring number in every Rx BD ring group, range in 1 ~ 8, active when rxBdrGroupNum not equal zero.

netc_hw_enetc_si_rxr_group defaultRxBdrGroup

The selected Rx BD ring group, active when rxBdrGroupNum not equal zero.

uint8_t vlanToIpvMap[16]

Frame VLAN pcp|dei to IPV mapping, active when valnToIpvEnable is true.

uint8_t ipvToRingMap[8]

BD ring used within the default Rx BD ring group for IPV n, active when rxBdrGroupNum not equal zero.

uint8_t vsiTcToTC[8]

Maps the VSI traffic class to transmit traffic class, done after the ENETC txPrio to TC mapping, only available for VSI.

bool enSIBaseVlan

Enable use SI-based VLAN information.

netc_enetc_vlan_tag_t siBaseVlan

SI-based VLAN information, active when enSIBaseVlan is true.

struct _netc_si_l2mf_config

#include <fsl_netc.h> L2 Mac Filter Configuration for SI.

Public Members

bool macUCPromis
Enable/Disable MAC unicast promiscuous.

bool macMCPromis
Enable/Disable MAC multicast promiscuous.

bool rejectUC
Reject Unicast.

bool rejectMC
Reject Multicast.

bool rejectBC
Reject Broadcast.

struct __netc_si_l2vf_config
#include <fsl_netc.h> L2 VLAN Filter Configuration for SI.

Public Members

bool acceptUntagged
Accept/Reject untagged frame.

bool enPromis
Enable/Disable VLAN promiscuous.

bool useOuterVlanTag
Use outer/inner VLAN tag for filtering.

struct __netc_si_discard_statistic
#include <fsl_netc.h> SI frame drop statistic struct.

Public Members

uint32_t programError
Due to programming error (non-existing BD ring or non-existing group, or SI disabled or BD ring disabled).

uint32_t busError
Due to system bus error.

uint32_t lackBD[14]
Due to lack of Rx BDs available.

struct __netc_si_traffic_statistic
#include <fsl_netc.h> SI traffic statistic struct.

struct __netc_si_config
#include <fsl_netc.h> SI Configuration.

Public Members

uint32_t tcBWWeight
SI traffic class bandwidth weight.

union __netc_tx_bd
#include <fsl_netc.h> Transmit Buffer Descriptor format.
A union type cover the BD used as Standard/Extended/WriteBack format.

Public Members

struct *_netc_tx_bd* standard

struct *_netc_tx_bd* ext

struct *_netc_tx_bd* writeback

uint64_t dword[2]

union *_netc_rx_bd*

#include <fsl_netc.h> Receive Buffer Descriptor format.

Public Members

struct *_netc_rx_bd* standard

struct *_netc_rx_bd* writeback

struct *_netc_rx_bd* ext

struct *_netc_rx_bd* resp

struct *_netc_tx_bdr_config*

#include <fsl_netc.h> Configuration for the SI Tx Buffer Descriptor Ring Configuration.

Public Members

uint32_t len

Size of BD ring which shall be multiple of 8 BD.

netc_tx_bd_t *bdArray

BDR base address which shall be 128 bytes aligned.

netc_tx_frame_info_t *dirtyArray

Tx cleanup ring.

bool enIntr

Enable/Disable completion interrupt.

bool enThresIntr

Enable/Disable threshold interrupt.

bool enCoalIntr

Enable/Disable interrupt coalescing.

uint32_t intrThreshold

Interrupt coalescing packet threshold.

uint32_t intrTimerThres

Interrupt coalescing timer threshold, specified in NETC clock cycles.

uint8_t msixEntryIdx

MSIX entry index of Tx ring interrupt.

bool isVlanInsert

Enable/Disable VLAN insert offload.

bool isUserCRC

Enable/Disable user provided the CRC32 - FCS at end of frame.

uint8_t wrWeight

Weight used for arbitration when rings have same priority.

uint8_t priority

Priority of the Tx BDR.

struct __netc_tx_bdr

#include <fsl_netc.h> Transmit BD ring handler data structure.

Public Members

netc_tx_bd_t *bdBase

Tx BDR base address.

netc_tx_frame_info_t *dirtyBase

Tx cleanup ring base address.

uint16_t producerIndex

Current index for transmit.

uint16_t cleanIndex

Current index for tx cleaning.

uint32_t len

Length of this BD ring.

struct __netc_rx_bdr_config

#include <fsl_netc.h> Configuration for the SI Rx Buffer Descriptor Ring Configuration.

Public Members

bool extendDescEn

False - Use 16Bytes standard BD. True - Use 32Bytes extended BD.

netc_rx_bd_t *bdArray

BD ring base address which shall be 128 bytes aligned.

uint32_t len

BD ring length in the unit of 16Bytes standard BD. Shall be multiple of 8/16 for standard/extended BD.

uint64_t *buffAddrArray

Rx buffers array with BD length(half of BD length if use extended BD).

uint16_t buffSize

Size of all Rx buffers in this BD ring.

bool enThresIntr

Enable/Disable threshold interrupt.

bool enCoalIntr

Enable/Disable interrupt coalescing.

uint32_t intrThreshold

Interrupt coalescing packet threshold.

uint32_t intrTimerThres

Interrupt coalescing timer threshold, specified in NETC clock cycles.

`uint8_t msixEntryIdx`
MSIX entry index of Rx ring interrupt.

`bool disVlanPresent`
Disable/Enable VLAN in BD.

`bool enVlanExtract`
Enable/Disable VLAN extract.

`bool isKeepCRC`
Whether user provided the CRC32 - FCS at end of frame.

`bool congestionMode`
False - lossy. True - lossless.

`bool enHeaderAlign`
Enable/disable +2B alignment to frame.

`struct __netc_rx_bdr`
#include <fsl_netc.h> Receive BD ring handler data structure.

Public Members

`netc_rx_bd_t *bdBase`
Rx BDR base address.

`bool extendDesc`
Use extended buffer descriptor.

`uint16_t index`
Current index for read.

`uint32_t len`
Length of this BD ring, unit of 16Bytes standard BD.

`uint64_t *buffArray`
Rx buffers array of this ring.

`uint32_t buffSize`
Rx buffers size for all BDs in this ring.

`struct __netc_bdr_config`
#include <fsl_netc.h> Configuration for the buffer descriptors ring.

Public Members

`netc_rx_bdr_config_t *rxBdrConfig`
Receive buffer ring configuration array.

`netc_tx_bdr_config_t *txBdrConfig`
Transmit buffer ring configuration array.

`struct __netc_psi_rx_msg`
#include <fsl_netc.h> PSI receive message information.

Public Members

uint8_t *msgBuff

The buffer address application set before receiving message.

uint32_t msgLen

Received message length.

struct __netc_vsi_msg_tx_status

#include <fsl_netc.h> VSI message transmit status.

Public Members

bool txBusy

The VSI Tx busy flag, become idle when the PSI receive and clear the related status.

bool isTxErr

Tx error flag.

uint16_t msgCode

The error code or user-defined content.

struct standard

Public Members

uint64_t addr

Address of the buffer. Little Endian.

uint16_t bufLen

Length of buffer specifying effective number of bytes.

uint16_t frameLen

Length of Frame.

uint32_t flags

Flags qualified setting.

uint32_t enableInterrupt

Whether enable interrupt on complete of BD.

uint32_t isExtended

Extended BD format flag.

uint32_t isFinal

Final BD flag.

struct ext

Public Members

uint32_t timestamp

IEEE1588 PTP one-step timestamp.

uint32_t __pad0__

Ignore 2-bit MSB.

uint16_t tpid

VLAN TPID type, see netc_vlan_tpid_select_t.

uint16_t vid
VLAN ID.

uint16_t dei
VLAN DEI.

uint16_t pcp
VLAN PCP.

uint8_t eFlags
Tx extension flags.

uint8_t isFinal
Final BD flag.

struct writeback

Public Members

uint32_t timestamp
Timestamp write back.

uint32_t __pad0__
/ Transmit timestamp identifier, only active on Switch management ENETC.

uint32_t status
Status.

uint32_t written
Write-back flag.

struct standard

Public Members

uint64_t addr
Software write address.

struct writeback

Public Members

uint16_t internetChecksum
Internet Checksum.

uint16_t parserSummary
Parser Summary.

uint16_t bufLen
Length of received buffer.

uint16_t vid
VLAN ID.

uint16_t dei
VLAN DEI.

uint16_t pcp
VLAN PCP.

uint8_t tpid

VLAN TPID.

uint8_t hr

Host Reason.

uint8_t flags

Rx information flags.

uint8_t error

Rx error code.

uint8_t isReady

Received data ready flag.

uint8_t isFinal

Final BD flag.

union ____unnamed194____

Public Members

struct __netc_rx_bd

uint32_t rssHashSwt

RSS hash while not used as switch management port.

struct ____unnamed196____

Public Members

uint32_t srcPort

Source port received from switch management port.

uint32_t rssHash

RSS Hash high field value.

struct ext

Public Members

uint32_t timestamp

Rx Timestamp.

struct resp

Public Members

uint32_t timestamp

Switch response timestamp.

uint16_t txtsid

Transmit timestamp identifier.

uint32_t hr

Host Reason.

uint32_t error

Error status code.

uint32_t isReady

Received data ready flag.

uint32_t isFinal

Final BD flag.

2.78 Hardware Switch

enum __netc_swt_port_bitmap

The switch port bitmap.

Values:

enumerator kNETC_SWTPort0Bit

Switch port0 bitmap

enumerator kNETC_SWTPort1Bit

Switch port1 bitmap

enumerator kNETC_SWTPort2Bit

Switch port2 bitmap

enumerator kNETC_SWTPort3Bit

Switch port3 bitmap

enumerator kNETC_SWTPort4Bit

Switch port4 (internal port) bitmap

enum __netc_swt_imr_dest_port

The switch ingress mirror destination port.

Values:

enumerator kNETC_SWTPort0

Switch port0

enumerator kNETC_SWTPort1

Switch port1

enumerator kNETC_SWTPort2

Switch port2

enumerator kNETC_SWTPort3

Switch port3

enumerator kNETC_SWTPort4

Switch port4

enumerator kNETC_SWTMPort

Switch management port

enum __netc_swt_mac_forward_mode

The switch MAC forwarding options.

Values:

enumerator kNETC_NoFDBLookUp

No FDB lookup is performed, the frame is flooded.

enumerator kNETC_FDBLookUpWithFlood

FDB lookup is performed, and if there is no match, the frame is flooded to the port bitmap in VLAN filter entry.

enumerator kNETC_FDBLookUpWithDiscard

FDB lookup is performed, and if there is no match, the frame is discarded.

enum __netc_swt_mac_learn_mode

The switch MAC learning options.

Values:

enumerator kNETC_DisableMACLearn

Disable MAC learning. SMAC FDB lookup is by-passed.

enumerator kNETC_HardwareMACLearn

Hardware MAC learning is enabled.

enumerator kNETC_SeSoftwareMACLearn

Software MAC learning secure. FDB lookup based on FID and SMAC is performed and if an entry is not found, the frame is redirected to the switch management port.

enumerator kNETC_UnseSoftwareMACLearn

Software MAC learning unsecure. FDB lookup based on FID and SMAC is performed and if an entry is not found, the frame is copied to the switch management port.

enumerator kNETC_DisableMACLearnWithSMAC

Disable MAC learning with SMAC validation. FDB lookup based on FID and SMAC is performed and if an entry is not found, the frame is discarded.

enum __netc_swt_port_tx_vlan_act

Switch transmit Bridge Port VLAN Tag Action.

Values:

enumerator kNETC_NoTxVlanModify

No egress VLAN modification performed

enumerator kNETC_TxDelOuterVlan

Delete outer VLAN tag

enumerator kNETC_TxReplOuterVlanVid

Replace outer VLAN tag's VID with 0; frame to be transmitted as a priority tag frame

enum __netc_swt_port_stg_mode

Switch port spanning tree group work mode.

Values:

enumerator kNETC_DiscardFrame

Tx or RX Frames on this port with current spanning tree group ID will be discarded

enumerator kNETC_LearnWithoutFowrad

RX Frames on this port with current spanning tree group ID will do Learn SMAC, but do not forward, Tx Frame will be discarded

enumerator kNETC_ForwardFrame

RX Frames on this port with current spanning tree group ID will do both MAC learning and forwarding, Tx Frame will be forwarded.

typedef enum __netc_swt_port_bitmap netc_swt_port_bitmap_t

The switch port bitmap.

```
typedef enum _netc_swt_imr_dest_port netc_swt_imr_dest_port_t
```

The switch ingress mirror destination port.

```
typedef enum _netc_swt_mac_forward_mode netc_swt_mac_forward_mode_t
```

The switch MAC forwarding options.

```
typedef enum _netc_swt_mac_learn_mode netc_swt_mac_learn_mode_t
```

The switch MAC learning options.

```
typedef enum _netc_swt_port_tx_vlan_act netc_swt_port_tx_vlan_act_t
```

Switch transmit Bridge Port VLAN Tag Action.

```
typedef enum _netc_swt_port_stg_mode netc_swt_port_stg_mode_t
```

Switch port spanning tree group work mode.

```
typedef struct _etc_swt_imr_config netc_swt_imr_config_t
```

Switch Ingress mirror destination config.

```
typedef struct _netc_swt_port_config netc_swt_port_bridge_config_t
```

Switch port bridge configuration.

```
typedef struct _netc_swt_port_fm_config netc_swt_port_fm_config_t
```

Switch Port level Frame Modification configuration (PPCPDEIMR and PQOSMR[QVMP])

```
typedef struct _netc_swt_default_vlan_filter netc_swt_default_vlan_filter_t
```

Switch VLAN filter hash table default entry configuration, which determines the default entry when not found in VLAN filter lookup.

```
typedef struct _netc_swt_bridge_config netc_swt_bridge_config_t
```

Bridge config.

```
typedef struct _netc_swt_psf_config netc_swt_psf_config_t
```

Switch PSFP configuration.

```
typedef struct _netc_qos_classify_config netc_swt_qos_classify_config_t
```

Switch Qos Classification configuration (include two profiles)

```
typedef struct _netc_swt_qos_to_vlan_config netc_swt_qos_to_vlan_config_t
```

Switch QoS to PCP mapping and PCP to PCP mapping configuration when egress packet modification the VLAN tag.

```
typedef struct _netc_switch_inuse_fdb_statistic netc_switch_inuse_fdb_statistic_t
```

Switch static/dynamic FDB entries in-use statistic.

```
typedef struct _netc_swt_port_sr_config netc_swt_port_sr_config_t
```

Port seamless redundancy configuration.

```
struct _etc_swt_imr_config
```

#include <fsl_net.h> Switch Ingress mirror destination config.

Public Members

```
bool enMirror
```

Enable ingress mirroring

```
netc_swt_imr_dest_port_t destPort
```

Port where ingress mirrored frames are sent

```
uint8_t dr
```

Mirrored packet's DR (drop resilience)

uint8_t ipv

Mirrored packet's IPV (internal priority value)

uint8_t efmLengthChange

Egress Frame Modification Frame Length change in 2s complement notation, Valid if efmEntryID is not null

uint16_t efmEntryID

Egress Frame Modification Entry Id, note 0xFFFF is a Null Frame Modification Entry, Only applicable if destPort != kNETC_SWTMPort

struct _netc_swt_port_config

#include <fsl_netc.h> Switch port bridge configuration.

Public Members

netc_swt_port_tx_vlan_act_t txVlanAction

Only applies for the frame outer VLAN tag's VID is equal to the port default VID

bool isRxVlanAware

Receive VLAN Aware Mode

bool acceptUntag

Accept untagged frame

bool acceptPriorityTag

Accept priority tagged frame (VID = 0)

bool acceptSingleTag

Accept single tagged frame

bool acceptDoubleTag

Accept double tagged frame (outer and inner)

bool enSrcPortPrun

Enable/Disable received frame be transmitted to same port it was received

bool enMacStationMove

Enable/Disable received frame which ingress port not match the FDB entry Destination Port Bitmap

bool enBcastStormCtrl

Enable/Disable Storm control for broadcast frames

bool enMcastStormCtrl

Enable/Disable Storm control for multicast frames

bool enUnMcastStormCtrl

Enable/Disable Storm control for unknown multicast frames

bool enUnUcastStormCtrl

Enable/Disable Storm control for unknown unicast frames

uint32_t bcastRpEntryID

Broadcast rate policer entry ID. Valid if enBroadStormCtrl = true

uint32_t mcastEntryID

Known multicast rate policer entry ID. Valid if enBroadStormCtrl = true

uint32_t unMcastRpEntryID

Unknown multicast rate policer entry ID. Valid if enUnMultiStormCtrl = true

uint32_t unUcastRpEntryID

Unknown unicast rate policer entry ID. Valid if enUnUniStormCtrl = true

uint16_t maxDynaFDBEntry

The maximum number of dynamic entries in the FDB table, 0 means no limit

struct __netc_swt_port_fm_config

#include <fsl_net.c.h> Switch Port level Frame Modification configuration (PPCPDEIMR and PQOSMR[QVMP])

Public Members

bool ignoreFMMiscfg

Enable/Disable ignore the Frame Modification Misconfiguration Action

bool enEgressPcpMap

Enable egress frame modification of outer VLAN tag's PCP value is mapped to a new value based on egressPcpMap, used for Frame Modification VLAN Outer PCP action

bool enIngressPcpMap

Enable ingress frame modification of outer VLAN tag's PCP value is mapped to a new value based on egressPcpMap, used for Frame Modification VLAN Outer PCP action

bool enUpdateVlanDei

Enable update DR value in the outer VLAN based on DEnDEI field, used for egress Frame Modification Outer DEI action

uint8_t drToDeiMap

Mapping of internal QoS's DR value n to VLAN DEI, The 4 bits correspond to the DR3 ~ DR0, and 1 means DRn mapping to DEI 1, 0 means DRn mapping to DEI 0

uint8_t egressPcpMap

Egress PCP to PCP Mapping Profile instance, active when enEgressPcpMap is true

uint8_t ingressPcpMap

Ingress PCP to PCP Mapping Profile instance, active when enIngressPcpMap is true

uint8_t qosVlanMap

Transmit QoS to VLAN PCP Mapping Profile index, used for egress Frame Modification VLAN Add/Replace Action

struct __netc_swt_default_vlan_filter

#include <fsl_net.c.h> Switch VLAN filter hash table default entry configuration, which determines the default entry when not found in VLAN filter lookup.

Public Members

bool enIPMFlood

Enable IP Multicast Flooding

bool enIPMFilter

Enable IP Multicast Filtering

uint8_t stgID

Spanning Tree Group Member ID, range in 0 ~ 15

uint8_t portMembership

The bit 0 ~ 4 correspond to the 5 ports, When bit set (0b1), means the port is a member of this VLAN. Port membership is used for source/destination pruning

bool enUseFilterID

Enable use the specified filterID as FID, otherwise will use the frame VID

uint16_t filterID

Used as a key value to do FDB table and the L2 IPV4 Multicast Filter table lookup. Valid if enUseFilterID is true

netc_swt_mac_forward_mode_t mfo

MAC forwarding options

netc_swt_mac_learn_mode_t mlo

MAC learning options

uint16_t baseETEID

Base Egress Treatment Entry ID

uint8_t etaPortBitmap

Egress Treatment Applicability Port. Valid if baseETEID is not null.

struct __netc_swt_bridge_config

#include <fsl_net.h> Bridge config.

Public Members

netc_swt_default_vlan_filter_t dVFCfg

Default VLAN filter entry configuration when not found in VLAN filter lookup

struct __netc_swt_psfpc_config

#include <fsl_net.h> Switch PSFP configuration.

Public Members

netc_isi_kc_rule_t kcRule[4]

Key construction rules

struct __netc_qos_classify_config

#include <fsl_net.h> Switch Qos Classification configuration (include two profiles)

struct __netc_swt_qos_to_vlan_config

#include <fsl_net.h> Switch QoS to PCP mapping and PCP to PCP mapping configuration when egress packet modification the VLAN tag.

struct fsl_net

#include <fsl_net.h>

Public Members

uint8_t qos[32]

Index is created from IPV (3 bits) + DR (2 bits) field. Value is the mapped PCP for VLAN tag.

uint8_t pcp[8]

Index is created from outer PCP (3 bits) field. Value is the mapped PCP for VLAN tag.

struct __netc_switch_inuse_fdb_statistic

#include <fsl_net.h> Switch static/dynamic FDB entries in-use statistic.

Public Members

uint16_t camEntries

Number of FDB entries in-use in the CAM.

uint16_t staticEntries

Number of static FDB entries in-use (both hash-based and CAM-based entries).

uint16_t dynamicEntries

Number of dynamic FDB entries in-use (hash-based and CAM-based entries).

uint16_t dynamicEntriesHWM

High water mark of dynamic entries in-use in the FDB table.

struct __netc_swt_port_sr_config

#include <fsl_netc.h> Port seamless redundancy configuration.

struct defaultVlan

Public Members

uint32_t vid

Vlan Identifier.

uint32_t dei

Drop eligible indicator

uint32_t pcp

Priority code point.

uint32_t tpid

Tag protocol identifier, 0 = Standard C-VLAN 0x8100, 1 = Standard S-VLAN 0x88A8.

2.79 Hardware Table Access Functions

enum __netc_tb_index

Table index.

Values:

enumerator kNETC_TGSTable

Time Gate Scheduling table index

enumerator kNETC_RPTable

Rate Policer table index

enumerator kNETC_IPFTable

Ingress Port filter table index

enumerator kNETC_FDBTable

FDB table index

enumerator kNETC_L2MCFTable

L2 IPV4 Multicast Filter table index

enumerator kNETC_VFTable

VLAN Filter table index

enumerator kNETC_ECQTable
ETM Class Queue table index

enumerator kNETC_ECSTable
ETM Class Scheduler table index

enumerator kNETC_ISITable
Ingress Stream Identification table index

enumerator kNETC_ISTable
Ingress Stream table index

enumerator kNETC_ISFTable
Ingress Stream Filter table index

enumerator kNETC_ETTable
Egress Treatment table index

enumerator kNETC_ISGTable
Ingress Sequence Generation table index

enumerator kNETC_ESRTable
Egress Sequence Recovery table index

enumerator kNETC_SGITable
Stream Gate Instance table index

enumerator kNETC_SGCLTable
Stream Gate Control List table index

enumerator kNETC_ISCTable
Ingress Stream Count table index

enumerator kNETC_ECTable
Egress Count table index

enumerator kNETC_FMTTable
Frame Modification table index

enumerator kNETC_BPTable
Buffer Pool table index

enumerator kNETC_SBPTable
Shared Buffer Pool table index

enumerator kNETC_ECGTable
ETM Class Group table index

enumerator kNETC_FMDTable
Frame Modification Data table index

enum __netc_tb_cmd
Table management command operations.

Values:

enumerator kNETC_DeleteEntry
Delete operation

enumerator kNETC_UpdateEntry
Update operation

enumerator kNETC_QueryEntry

Query operation

enumerator kNETC_QueryAndDeleteEntry

Query operation followed by a delete operation

enumerator kNETC_QueryAndUpdateEntry

Query operation followed by a update operation

enumerator kNETC_AddEntry

Add operation

enumerator kNETC_AddOrUpdateEntry

If the entry exists, is update operation, if not exist, is the Add operation

enumerator kNETC_AddAndQueryEntry

Add operation followed by a query operation

enumerator kNETC_AddQueryAndUpdateEntry

Add operation followed by a query operation, Then, if the entry existed prior to the Add operation of this command, the Update operation will be performed.

enum __netc_tb_access_mode

Table Access Method.

Values:

enumerator kNETC_EntryIDMatch

Entry ID Match

enumerator kNETC_ExactKeyMatch

Exact Match Key Element Match

enumerator kNETC_Search

Search with search criteria

enumerator kNETC_TernaryKeyMatch

Ternary Match Key Element Match

enum __netc_cbd_version

NTMP version.

Values:

enumerator kNETC_NtmpV1_0

NTMP Version 1.0

enumerator kNETC_NtmpV2_0

NTMP Version 2.0

enum __netc_cmd_error

Table command response error status.

Values:

enumerator kNETC_FormatError

Format error : 1. Illegal class or command. 2. Invalid SF bit setting. 3. LENGTH is zero for long format. 4. LENGTH is too small for buffer size.

enumerator kNETC_SizeError

Size error : 1. Invalid table index, out of range. 2. Table overflow, no additional entries available.

enumerator `kNETC_AccessError`
Access violation error, the entity is not allowed to perform the task requested

enumerator `kNETC_ClassError`
Class specific error

enumerator `kNETC_IntegrityError`
Integrity error, the command did not execute due to a data integrity error (ECC on internal memory or AXI read/write error)

enumerator `kNETC_InvTableID`
Invalid table ID

enumerator `kNETC_InvAccMethod`
Invalid Access method

enumerator `kNETC_TableIdxOutOfRange`
Table index out of range

enumerator `kNETC_DBNotEnough`
Request data buffer size or response data buffer size is not sufficient

enumerator `kNETC_InvCmd`
Invalid command

enumerator `kNETC_ReqDBError`
Request Data buffer error

enumerator `kNETC_MultiBitError`
Multi-bit ECC or parity error observed during command processing

enumerator `kNETC_HashEntryLimit`
Exceeded hash entry limit

enumerator `kNETC_HashChainLimit`
Exceeded maximum hash collision chain limit and the CAM if present is full

enumerator `kNETC_InvHWGenEntryID`
Invalid ENTRY_ID for ENTRY_ID generated by hardware

enumerator `kNETC_SrchResDBNotEnough`
Search command filled the response data buffer before completing the command

enumerator `kNETC_CmdIdxTableWithITM`
Command for index table before OSR[ITM_STATE]=0

enumerator `kNETC_InvQueryAction`
Invalid Query action

enumerator `kNETC_InvTableAccPrivilege`
Invalid table access privilege

enumerator `kNETC_ReadSysBusErr`
System Bus Read Error

enumerator `kNETC_WriteSysBusErr`
System Bus Write Error

enumerator `kNETC_ClientErr`
Client encountered a fault

enumerator `kNETC_TGSCmdIssue`
Command issued when time gating function is disabled for the port.

enumerator kNETC_TGSUpdateExistList

Update action attempted on an existing admin gate control list. (should delete admin gate control list first before creating a new admin list)

enumerator kNETC_TGSUpdateOverLength

Update action attempted exceeds TGSTCAPR[MAX_GCL_LEN]

enumerator kNETC_TGSUpdateOverSize

Update action attempted exceeds TGSTCAPR[NUM_WORDS].

enumerator kNETC_TGSEntryNotEnough

Insufficient resources to perform the requested operation (not enough free time gate list entries)

enumerator kNETC_TGSUpdateNSList

Update action attempted with ADMIN_CYCLE_TIME, ADMIN_TIME_INTERVAL_GE_i or truncated ADMIN_TIME_INTERVAL_GE_n due ADMIN_CYCLE_TIME specified is not sufficient to transmit 64 byte of frame data + header overhead.

enumerator kNETC_TGSUpdateEarlierStartTime

Update action attempted with ADMIN_BASE_TIME specified s more than one second in the past from tcs advance time.

enumerator kNETC_TGSUpdateOverflowCycle

Update action attempted with ADMIN_CYCLE_TIME + ADMIN_CYCLE_TIME_EXT is greater than $2^{32}-1$.

enumerator kNETC_TGSQueryBeforeListActive

Query action issued when config change occurred. Retry query.

enumerator kNETC_TGSUpdateInvGateValue

Update action attempted with ADMIN_HR_CB_GE_i set to an invalid value

enumerator kNETC_RPSDUTypeOutOfRange

SDU_TYPE specified in entry CFGE_DATA is out of range

enumerator kNETC_IPFInvHR

HR value not valid. Only checked if command issued from the Switch and FLTFA=0x2 or FLTFA=0x3

enumerator kNETC_IPFEntryNotFit

Entry being added does not fit in table

enumerator kNETC_IPFWithoutSTSE

CFGE_DATA update without STSE_DATA update

enumerator kNETC_IPFInvRPP

RPR set to a reserved value. Only checked if FLTA=0x2.

enumerator kNETC_IPFFLTATGTOutOfRange

FLTA_TGT is outside valid range and not NULL. Only checked if FLTA>0x0

enumerator kNETC_IPFInvSwtFLTA

FLTA=0x3 when command issued from the Switch.

enumerator kNETC_IPFInvEnetcFLTA

FLTFA>0x1 when command issued from an ENETC PF.

enumerator kNETC_FDBReachPortLimit

Failed to add or update and entry because the Port BPCR[DYN_LIMIT] has been reached

enumerator kNETC_FDBReachSwtLimit

Failed to add entry because the Switch FDBHTMCR[DYN_LIMIT] has been reached.

enumerator kNETC_FDBInvEPORT

EPORT value not valid. Only checked if (OETEID=0x1 OR CTD=0x1)

enumerator kNETC_FDBETEIDOutRange

ET_EID is out of range and not NULL. Only checked if OETEID>0x0

enumerator kNETC_FDBParityErr

Parity error encountered when adding guaranteed entry

enumerator kNETC_L2MCFInvEPORT

EPORT value not valid. Only checked if (OETEID=0x1 OR CTD=0x1)

enumerator kNETC_L2MCFETEIDOutRange

ET_EID is not NULL or within the valid range. Only checked if OETEID>0x0.

enumerator kNETC_L2MCFInvKEYTYPE

KEY_TYPE value not valid

enumerator kNETC_VFBASEETEIDOutRange

BASE_ET_EID is out of range or MLO is not valid.

enumerator kNETC_ECQCQ2CGMAPOutRange

CQ2CG_MAP value out-of-range in update command.

enumerator kNETC_ISIPortIDOutRange

Port ID specified in KEYE_DATA is out of range.

enumerator kNETC_ISIInvISEID

IS_EID in invalid.

enumerator kNETC_ISInvOpt

Option specified in one or more of the following fields is not valid – FA, CTD or ISQA, SDU_TYPE.

enumerator kNETC_ISInvID

One or more of following : 1. Entry IDs are not in valid range or Entry ID is not Null. 2. Check valid ranges specified for these Entry IDs in Ingress Stream table entry – RP_EID, SGI_EID, ISQ_EID, ET_EID or EPORT. 3. ET_EID is checked if (FA =010b .. 101b) & (OETEID!=0). 4. EPORT is checked if (FA = 010b .. 101b) & (OETEID= 0x1 OR CTD= 0x1). 5. HR is chkd if FA = 001b, 100b, or 101b. HR specified cannot be 0x0

enumerator kNETC_ISInvFMEID

FM_EID format or index is out of range : 1. FM_EID format option type is invalid. 2. FM_EID format is option 1 and the Index is out of range and not Null, or FM_EID format is option 2 and VUDA or SQTA is out of range.

enumerator kNETC_ISFInvISEID

IS_EID in KEYE_DATA is invalid.

enumerator kNETC_ISFInvCFGE

Any of the following in CFGE_DATA is invalid : 1. One or more of following Entry IDs are not in valid range or Entry ID specified is not Null. Checks are performed for following Entry IDs CFGE DATA – RP_EID, SGI_EID, ISC_EID. 2. SDU_TYPE is invalid

enumerator kNETC_ETInvOpt

Command option specified is invalid or not supported. ESQA is not 00 or 10 (others are reserved), or ECA > 1 (reserved).

enumerator kNETC_ETInvFMEID

FM_EID format or index is out of range. Check performed is as follows : 1. EFM_EID format option type is invalid, or EFM_EID format is option 1 and the Index is out of range and not Null, or EFM_EID format is option 2 and VUDA or SQTA is out of range . 2. the Egress Counter Table index EC_EID is out of range. 3. The Egress Sequence Actions Target Entry ID ESQA_TGT_EID is out of range

enumerator kNETC_ISGInvQSTAG

SQ_TAG specified is not valid

enumerator kNETC_SGISGCLEIDOutOfRange

SGCL_EID specified in out of range for Add or Update operation.

enumerator kNETC_SGIIInvSDUTYPE

SDU_TYPE is specified is invalid for Add or Update operation.

enumerator kNETC_SGISGCLEIDNotAlloc

Either the SGCL_EID specified as admin gate control list in Add or Update operation has not been allocated or SGCL_EID is not the first entry in gate control list or the reference count in SGCL entry is not 0.

enumerator kNETC_SGIIInvSGCLEID

SGCL_EID specified for Update operation is in invalid.

enumerator kNETC_SGIIInvADMINBASETIME

ADMIN_BASE_TIME specified for Add or Update operation is more than 2^{30} ns in the past.

enumerator kNETC_SGIIInvCYCLETIME

Cumulated time value of CYCLE_TIME in Stream gate Control list plus CYCLE_TIME_EXT specified in Add or Update operation is $\geq 2^{30}$ ns or CYCLE_TIME specified is 0.

enumerator kNETC_SGCLOverLength

Number words required for the LIST_LENGTH specified for the Add operation exceeds the number of words allocated for SGCL table

enumerator kNETC_SGCLTimeIntervalZero

TIME_INTERVAL_GE_N specified in Add operation is 0. Note that upper 2 bits of TIME_INTERVAL_GE_N are ignored, TIME_INTERVAL_GE_N[29:0] must not be 0.

enumerator kNETC_SGCLTimeIntervalOverflow

Cumulated time value of TIME_INTERVAL_GE_N[29:0] for the gate list specified in Add operation is $\geq 2^{30}$ ns.

enumerator kNETC_FMInvEMEID

FM_EID format is invalid

enumerator kNETC_FMOptOutOfRange

Following fields specified are out of range - MAC_HDR_ACT, VLAN_HDR_ACT, SQT_ACT, OUTER_PCP_DEI_ACT, PLD_ACT.

enumerator kNETC_FMFMDOutOfRange

FMD_EID, FMD_BYTES specified is out of range. When FMD_EID is not set to Null, valid range is $FMD\_EID[15:0] * 24 + FMD\_BYTES \leq (FMDITCAPR[NUM\_WORDS] * 24)$.

enumerator kNETC_BPSBPEIDOutOfRange

SBP_EN is 1 and SBP_EID value is out-of-range in update command

enum _netc_fm_vlan_ud_act

Frame Modification VLAN Update/Delete Action.

Note: Misconfiguration error if replace or delete action is specified and if VLAN tag is not present in frame.

Values:

enumerator kNETC_NoUDVlanAction

No Update/Delete VLAN action

enumerator kNETC_ReplVlanPcpAndDei

Replace outer VLAN's PCP/DEI based on the port's PPCPDEIMR. The tag's original VID and TPID are preserved

enumerator kNETC_DelVlan

Delete outer VLAN Tag

enum _netc_fm_sq_t_act

Frame Modification Sequence Tag Action.

Note: Must be set to 000b for Ingress frame modification, otherwise misconfiguration error..

Values:

enumerator kNETC_NoSqtAction

No SQT action

enumerator kNETC_ReomveRTag

Remove R-TAG/draft 2.0 R-TAG/HSR tag, If R-TAG/HSR tag not present, misconfiguration error.

enum _netc_fm_vlan_ar_act

Frame Modification VLAN Add/Replace Action.

Note: For ingress frame modificaion with 00b or 01b, use the ingress port to select PCP and DEI from the Bridge port default VLAN register (BPDVR). For egress frame modification with 00b or 01b, use the internal QoS associated with the frame (IPV, DR) to access the QoS to PCP mapping profile (PQOSMR[QVMP] , QOSVLANMPaR0/1/2/3) to set the new PCP value. Use internal DR associated with frame to access the DR to DEI mapping profile (PPCPDEIMR[DRnDEI]) to set the new DEI value.

Values:

enumerator kNETC_AddCVlanPcpAndDei

Add outer VLAN with VID and PCP/DEI updated as described above. TPID=0x8100

enumerator kNETC_AddSVlanPcpAndDei

Add outer VLAN with VID and PCP/DEI updated as described above. TPID=0x88A8

enumerator kNETC_ReplVidOnly

Replace VLAN with VID. The tag's original PCP, DEI and TPID are preserved

enumerator kNETC_ReplVidPcpAndDei

Replace VLAN with VID and PCP/DEI updated by port's PPCPDEIMR. The tag's original TPID is preserved

enum _netc_tb_eteid_access_mode

Define FDB/L2MCF/IS table entry access the primary Egress Treatment table entry group mode.

Note: The FDB/L2 IPv4 Multicast filter table has precedence over any assignment made via the Ingress Stream table. For Multit port mode, the index to access the Egress Treatment table is computed by adding an offset to the base index of the Egress Treatment group. That offset is derived from the applicability bitmap as follows: starting from the lowest significant bit of the bitmap, the first encountered bit set to 1, corresponds to offset 0, and so on. This continues till the destination port location in the bitmap is reached

Values:

enumerator kNETC_NoETAccess

No Egress Treatment table access

enumerator kNETC_SinglePortETAccess

Only frame sent to a special port (define in ePort) can access a single Egress Treatment table entry, the applicability bitmap specified by FDB/L2MCF/IS ePort field

enumerator kNETC_MultPortPackedETAccess

Only frames sent to a special set of ports (ports set to 1 in ePortBitmap) can access the Egress Treatment table, the applicability bitmap = IS ePortBitmap field or FDB/L2MCF portBitmap field

enumerator kNETC_MultPortAbsETAccess

Frames sent to all of ports can access the Egress Treatment table, means the applicability bitmap is set with 1 for all ports

enum _netc_tb_ipf_update_action

Ingress Port Filter Table Update Actions.

Values:

enumerator kNETC_IPFCfgEUpdate

Configuration Element Update

enumerator kNETC_IPFStsEUpdate

Statistics Element Update

enum _netc_tb_ipf_attr_mask

Ingress Port Filter frame attribute mask.

Values:

enumerator kNETC_IPFSwtPortMasMask

Switch port masquerading Mask

enumerator kNETC_IPFEthernetMask

Ethernet type Mask

enumerator kNETC_IPFOuterVlanMask

Outer VLAN Mask

enumerator kNETC_IPFInnerVlanMask

Inner VLAN Mask

enumerator kNETC_IPFSeqTagMask

Sequence Tag Code Mask

enumerator kNETC_IPFIpHeaderMask
IP Header Mask

enumerator kNETC_IPFIpVersionMask
IP Version Mask

enumerator kNETC_IPFIpExtMask
IPv4 option / IPv6 extension Mask

enumerator kNETC_IPFL4HeaderMask
L4 Code Mask

enumerator kNETC_IPFWakeOnLanMask
Wake-on-LAN Magic Packet Mask

enum _netc_tb_ipf_seq_tag
Ingress Port Filter frame attribute Sequence Tag Code.

Values:

enumerator kNETC_IPFNoRtag
R-TAG/HSR tag is not present

enumerator kNETC_IPFDraftRtag
802.1CB draft 2.0 R-TAG is present

enumerator kNETC_IPFRtag
802.1CB R-TAG is present

enumerator kNETC_IPFHsrTag
HSR Tag is present

enum _netc_tb_ipf_l4_header
Ingress Port Filter frame attribute L4 Header Code.

Values:

enumerator kNETC_IPFOtherL4
The L4 Header is considered as other L4 if it is not one of the following L4 Headers

enumerator kNETC_IPFTcp
TCP header is present

enumerator kNETC_IPFUdp
UDP header is present

enumerator kNETC_IPFSctp
SCTP header is present

enum _netc_tb_ipf_forward_action
Ingress port filter forwarding Action.

Values:

enumerator kNETC_IPFForwardDiscard
Frame be discard

enumerator kNETC_IPFForwardPermit
Frame be permit

enumerator kNETC_IPFRedirectToMgmtPort
Redirect frame to switch management port without any frame modification, Switch only

enumerator kNETC_IPFCopyToMgmtPort

Copy frame to switch management port without any frame modification, Switch only

enum __netc_tb_ipf_filter_action

Ingress port filter Filter Action.

Values:

enumerator kNETC_IPFNoAction

No action

enumerator kNETC_IPFWithRatePolicer

Rate action with the Rate Policer Entry ID (RP_EID) set to the value configured in the fltaTgt field

enumerator kNETC_IPFWithIngressStream

Ingress stream identification action where the Ingress Stream Entry ID (IS_EID) is set to the value configured in the fltaTgt field

enumerator kNETC_IPFWithL2Filtering

Setting a pre L2 filtering SI bitmap (set to the value configured in the fltaTgt) that will be used by the L2 filtering function to determine the final SI bitmap, ENETC only

enum __netc_tb_isi_key_type

Stream identification table key type.

Values:

enumerator kNETC_KCRule0

Use key construction rule 0 (ISIDKC0CR0)

enumerator kNETC_KCRule1

Use key construction rule 1 (ISIDKC1CR0)

enumerator kNETC_KCRule2

Use key construction rule 2 (ISIDKC2CR0). Only for SWITCH

enumerator kNETC_KCRule3

Use key construction rule 3 (ISIDKC3CR0). Only for SWITCH

enum __netc_tb_is_isq_action

Ingress Stream table Ingress Sequence Action.

Values:

enumerator kNETC_ISNotPerformFRER

Not perform Frame FRER sequence generation function

enumerator kNETC_ISPerformFRER

Perform Frame FRER sequence generation function

enum __netc_tb_is_forward_action

Ingress Stream table forwarding Action.

Values:

enumerator kNETC_ISDiscard

Frame be discard

enumerator kNETC_ISRedirectToMgmtPort

Frame be Re-direct frame to switch management port, Switch only

enumerator kNETC_ISAllow

Frame is allow without setting the pre L2 filtering SI bitmap, ENETC only

enumerator kNETC_ISAllowWithSIMap

Frame is allow with setting the pre L2 filtering SI bitmap to the value configured in the SI_MAP field, ENETC only

enumerator kNETC_ISSStreamForward

Frame is forwarded to the port(s) specified in the EGRESS_PORT_BITMAP field, Switch only

enumerator kNETC_ISBridgeForward

Frame is do 802.1Q Bridge forwarding (VLAN processing and L2 forwarding), Switch only

enumerator kNETC_ISCopyToMgmtPortAndStream

Copy frame to switch management port with specified HR and stream forwarding, Switch only

enumerator kNETC_ISCopyToMgmtPortAndBridge

Copy frame to switch management port with specified HR and Bridge forwarding, Switch only

enum __netc_tb_is_ctd_mode

Ingress Stream table Cut-Through Disable mode.

Values:

enumerator kNETC_ISNoCTD

Do not override cut-through state

enumerator kNETC_ISSinglePortCTD

Disable cut-through for the outgoing port specified in the cfge ePort field

enumerator kNETC_ISAllPortCTD

Disable cut-through for all ports specified in cfge portBitmap field

enum __netc_tb_rp_update_action

Rate Policer Table Update Actions.

Values:

enumerator kNETC_RPCfgEUpdate

Configuration Element Update

enumerator kNETC_RPFfeEUpdate

Functional Enable Element Update

enumerator kNETC_RPPsEUpdate

Policer State Element Update Element Update

enumerator kNETC_RPStsEUpdate

Statistics Element Update

enum __netc_tb_sgi_update_action

Stream Gate Instance table Update Actions.

Values:

enumerator kNETC_SGIAcfEUpdate

Admin Configuration Element

enumerator kNETC_SGICfgEUpdate

Configuration Element Update

enumerator kNETC_SGISgisEUpdate

Stream Gate Instance State Element Update

enum _netc_tb_sgi_state

Stream Gate Instance State.

Values:

enumerator kNETC_GSNotOper

Gate instance is not operational or Gate instance and lists are not valid

enumerator kNETC_GSUseDefaultParam

Gate instance is operational but no stream gate control list specified, use default Gate Instance parameters

enumerator kNETC_GSUseDefUntilAdminAct

Use default Gate Instance parameters until administrative stream gate control list takes effect

enumerator kNETC_GSUseOperUntilAdminAct

Use Operational stream gate control list until new administrative stream gate control list takes effect

enumerator kNETC_GSUseOperList

Operational stream gate control list is in effect

enum _netc_tb_fm_layer2_act

Frame Modification table Layer 2 Actions.

Note: This field must be set to 0 for traffic destined to a pseudo link. This field must be set to 0 for any device with ASIL-B safety requirements.

Values:

enumerator kNETC_UseL2HeaderAct

L2 actions are specified in L2 header action fields macHdrAct, vlanHdrAct and sqtAct

enumerator kNETC_UseSpecPayload

The entire L2 PDU is replaced with fmdBytes of data specified in fmdEID, not applicable for ingress frame modifications

enum _netc_tb_fm_mac_header_act

Frame Modification table Layer 2 Header MAC Actions.

Note: Ingress frame modifications only support kNETC_NoAction or kNETC_ReplDmac.

Values:

enumerator kNETC_NoMacAction

No Mac header action

enumerator kNETC_ReplSmac

Replace SMAC with the contents of the port's PMAR0/1 register, The port is specified by smacPort field

enumerator kNETC_ReplSmacAndDmacAct1

Replace SMAC and DMAC, The content of SMAC is the same as kNETC_ReplaceSMAC, the DMAC is specified by dmac[6] field

enumerator kNETC_ReplSmacAndDmacAct2

Replace SMAC and DMAC, The content of SMAC is the same as kNETC_ReplaceSMAC, the DMAC is specified by frame's SMAC

enumerator kNETC_ReplDmac

Replace DMAC with specified dmac[6] field value

enumerator kNETC_SwapDmacAndSmac

Swap DMAC and SMAC

enum __netc_tb_fm_vlan_header_act

Frame Modification table Layer 2 VLAN Actions.

Note: For use Delete or Replace action, if no outer VLAN header is present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

Values:

enumerator kNETC_NoVlanAction

No VLAN header action

enumerator kNETC_DelOuterVlan

Delete outer VLAN header

enumerator kNETC_AddOuterVlan

Add outer VLAN header (new VLAN data will be inserted in the outer position), the VID, PCP, DEI and TPID values are specified by outerVidAct, outerPcpAct, outerDeiAct and outerTpidAct field

enumerator kNETC_ReplOuterVlan

Replace outer VLAN header, the VID, PCP, DEI and TPID values are specified by outerVidAct, outerPcpAct, outerDeiAct and outerTpidAct field

enum __netc_tb_fm_outer_vid_act

Frame Modification table Layer 2 outer VLAN VID Actions.

Note: For use kNETC_UseFrameVID action, if no outer VLAN header is present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

Values:

enumerator kNETC_UseFrameVid

Use the VID from the valid outer VLAN header of the received frame

enumerator kNETC_UseSpecVid

Use the VID specified in the outerVlanID field

enum __netc_tb_fm_outer_tpid_act

Frame Modification table Outer TPID action.

Note: For use kNETC_UseFrameTpid action, If outer VLAN header not present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

Values:

enumerator kNETC_UseFrameTpid

Use TPID from outer VLAN header

enumerator kNETC_UseStdCVlan

Set TPID to Standard C-VLAN 0x8100

enumerator kNETC_UseStdSVlan
Set TPID to Standard S-VLAN 0x88A8

enumerator kNETC_UseCustomCVlan
Set TPID to Custom C-VLAN as defined by CVLANR1[ETYPE]

enumerator kNETC_UseCustomSVlan
Set TPID to Custom S-VLAN as defined by CVLANR2[ETYPE]

enum _netc_tb_fm_outer_pcp_act
Frame Modification table Outer PCP action.

Note: For use kNETC_UseFramePcp/kNETC_UseFramePcpMap action, If outer VLAN header not present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

Values:

enumerator kNETC_UseFramePcp
Use PCP from frame outer VLAN header

enumerator kNETC_UseSpecPcp
Use the PCP specified in the outerVlanPcp field

enumerator kNETC_UseFramePcpMap
The PCP is mapping from frame outer VLAN PCP (do mapping according to the PCP to PCP mapping profile which specified in PPCPDEIMR[IPCPMP/EPCMP])

enumerator kNETC_UseQosMap
The PCP is mapping from internal QoS (IPV, DR) (do mapping according to the QoS to PCP mapping profile which specified in QOSVLANMPaR0/1/2/3), not applicable for ingress frame modifications

enum _netc_tb_fm_outer_dei_act
Frame Modification table Outer DEI action.

Note: For use kNETC_UseFrameDei action, If outer VLAN header not present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

Values:

enumerator kNETC_UseFrameDei
Use DEI from frame outer VLAN header

enumerator kNETC_UseSpecDei
Use the DEI specified in the outerVlanDei field

enumerator kNETC_UseDrMap
The DEI is mapping from internal DR (do mapping according to the DR to DEI mapping profile which specified in PPCPDEIMR[DRnDEI], not applicable for ingress frame modifications

enum _netc_tb_fm_payload_act
Frame Modification table Payload Actions.

Note: This field must be set to 0 for traffic destined to a pseudo link. This field must be set to 0 for any device with ASIL-B safety requirements.

Values:

enumerator kNETC_NoAction

No Action

enumerator kNETC_ReplAllEthPld

Remove entire Ethernet payload and insert with fmdBytes of data specified in fmdEID

enumerator kNETC_ReplPldWithOffset

Replace fmdBytes of raw data in the Ethernet payload starting at pldOffset, data specified in fmdEID

enum __netc_tb_fdb_update_action

FDB table Update Actions.

Values:

enumerator kNETC_FDBCfgEUpdate

Configuration Element Update

enumerator kNETC_FDBActEUpdate

Activity Element Update

enum __netc_tb_fdb_ctd_mode

FDB table Cut-Through Disable mode.

Values:

enumerator kNETC_FDBNoCTD

Do not override cut-through state

enumerator kNETC_FDBSinglePortCTD

Disable cut-through for the outgoing port specified in the cfge ePort field

enumerator kNETC_FDBAllPortCTD

Disable cut-through for all ports specified in cfge portBitmap field

enum __netc_tb_fdb_sc_keye_mc

FDB table search criteria Key Element Match Criteria.

Values:

enumerator kNETC_FDBKeyeMacthAny

Match any Key Element Criteria

enumerator kNETC_FDBKeyeMacthFID

Match Key Element FID

enumerator kNETC_FDBKeyeMacthMacMulticast

Match Key Element MAC Multicast bit (MAC_ADDR most significant byte's least significant bit)

enumerator kNETC_FDBKeyeMacthBoth

Match both FID field and MAC Multicast bit

enum __netc_tb_fdb_sc_cfge_mc

FDB table search criteria Configuration Element Match Criteria.

Values:

enumerator kNETC_FDBCfgeMacthAny

Match any Configuration Element Criteria

enumerator kNETC_FDBCfgeMacthDynamic

Match Configuration Element dynamic field

enumerator kNETC_FDBCfgeMacthPortBitmap
Match Configuration Element portBitmap field

enumerator kNETC_FDBCfgeMacthBoth
Match both dynamic field and portBitmap

enum _netc_tb_fdb_sc_acte_mc
FDB table search criteria Activity Element Match Criteria.
Values:

enumerator kNETC_FDBActeMacthAny
Match any Activity Element Criteria

enumerator kNETC_FDBActeMatchExact
Exact match with Activity Element

enum _netc_tb_l2mcf_key_type
L2 IPV4 Multicast Filter table key type.
Values:

enumerator kNETC_IPv4ASMKey
Key consists of a filtering ID (FID) and destination multicast IPv4 address

enumerator kNETC_IPv4SSMKey
Key consists of a filtering ID (FID), IPv4 source address and multicast IPv4 destination address

enum _etc_tb_l2mcf_sc_keye_mc
L2 IPV4 Multicast Filter table search criteria Key Element Match Criteria.
Values:

enumerator kNETC_L2MCFKeyeMacthAny
Match any Key Element Criteria

enumerator kNETC_L2MCFKeyeMacthFID
Match Key Element FID

enum _etc_tb_l2mcf_sc_cfge_mc
L2 IPV4 Multicast Filter table search criteria Configuration Element Match Criteria.
Values:

enumerator kNETC_L2MCFCfgeMacthAny
Match any Configuration Element Criteria

enumerator kNETC_L2MCFCfgeMacthDynamic
Match Configuration Element dynamic field

enumerator kNETC_L2MCFCfgeMacthPortBitmap
Match Configuration Element portBitmap field

enumerator kNETC_L2MCFCfgeMacthBoth
Match both dynamic field and portBitmap

enum _etc_tb_l2mcf_sc_acte_mc
FDB table search criteria Activity Element Match Criteria.
Values:

enumerator kNETC_L2MCFActeMacthAny
Match any Activity Element Criteria

enumerator kNETC_L2MCFActeMatchExact
Exact match with Activity Element

enum _netc_tb_iseqg_sqtg
Sequence Tag Type.

Values:

enumerator kNETC_SqDraftRTag
802.1CB draft 2.0 R-TAG.

enumerator kNETC_SqRTag
802.1CB R-TAG.

enumerator kNETC_SqHsrTag
HSR Tag.

enum _netc_tb_iseqg_update_action
Ingress Sequence Generation Table Update Actions.

Values:

enumerator kNETC_ISEQGCfgEUpdate
Configuration Element Update

enumerator kNETC_ISEQGSgsEUpdate
Sequence Generation Element Update

enum _netc_tb_eseqr_sqtg
Egress Sequence Recovery table Sequence Tag Type.

Values:

enumerator kNETC_AcceptAnyTag
Accept any incoming tag type (802.1CB draft 2.0 R-TAG, 802.1CB R-TAG or HSR Tag)

enumerator kNETC_AcceptDraftRTag
802.1CB draft 2.0 R-TAG.

enumerator kNETC_AcceptRTag
802.1CB R-TAG.

enumerator kNETC_AcceptHsrTag
HSR Tag.

enum _netc_tb_tgs_entry_id
Time Gate Scheduling table entry ID for switch and ENETC.

Values:

enumerator kNETC_TGSSwtPort0
Switch PORT 0 entry ID

enumerator kNETC_TGSSwtPort1
Switch PORT 1 entry ID

enumerator kNETC_TGSSwtPort2
Switch PORT 2 entry ID

enumerator kNETC_TGSSwtPort3
Switch PORT 3 entry ID

enumerator kNETC_TGSSwtPort4
Switch PORT 4 entry ID

enumerator kNETC_TGSEnetc0Port
ENETC 0 port entry ID

enumerator kNETC_TGSEnetc1Port
ENETC 1 port entry ID

enum __netc_tb_tgs_gate_type
Administrative gate operation type.
Values:

enumerator kNETC_SetGateStates
HoldRequest is unchanged

enumerator kNETC_SetAndHoldMac
HoldRequest is set to value hold, only active when enable preemption

enumerator kNETC_SetAndReleaseMac
HoldRequest is set to value release, only active when enable preemption

enum __netc_tb_et_efm_mode
Egress Frame Modification entry mode.
Values:

enumerator kNETC_NormalMode
Egress Frame Modification entry use normal/Default mode

enumerator kNETC_L2Act1
Egress Frame Modification entry l2Act = kNETC_UseSpecPayload

enumerator kNETC_PldAct1
Egress Frame Modification entry pldAct = kNETC_ReplAlLEthPld

enum __netc_tb_et_esq_act
Egress Sequence Actions.
Values:

enumerator kNETC_NoEsqAction
No Egress Sequence Action required

enumerator kNETC_HasEsqAction
Has Egress Sequence Recovery action

enum __netc_tb_et_ec_act
Egress Counter Action.
Values:

enumerator kNETC_NoEcCounter
Do not increment egress frame counter

enumerator kNETC_HasEcCounter
Increment egress frame counter

enum __netc_tb_etmcq_update_action
ETM Class Queue table Update Actions.
Values:

enumerator kNETC_CQCfgEUpdate
Configuration Element Update

enumerator kNETC_CQStsEUpdate

Statistics Element Update, all counters (except FRM_CNT) within the Statistics Element are reset

enum __netc_tb_etmcs_entry_id

ETM Class Scheduler table entry ID.

Values:

enumerator kNETC_CSSwtPort0

CS Switch PORT 0 entry ID

enumerator kNETC_CSSwtPort1

CS Switch PORT 1 entry ID

enumerator kNETC_CSSwtPort2

CS Switch PORT 2 entry ID

enumerator kNETC_CSSwtPort3

CS Switch PORT 3 entry ID

enumerator kNETC_CSSwtPort4

CS Switch PORT 4 entry ID

enum __netc_tb_etmcs_ca_assg

ETM Class Scheduler table Class queue assignment to scheduler inputs mode.

Values:

enumerator kNETC_CQ7AssignToSchedIn15

CQ 7 assignment to scheduler input 15, means all CQ use strict priority

enumerator kNETC_CQ7AssignToSchedIn14

CQ 7 assignment to scheduler input 14

enumerator kNETC_CQ7AssignToSchedIn13

CQ 7 assignment to scheduler input 13

enumerator kNETC_CQ7AssignToSchedIn12

CQ 7 assignment to scheduler input 12

enumerator kNETC_CQ7AssignToSchedIn11

CQ 7 assignment to scheduler input 11

enumerator kNETC_CQ7AssignToSchedIn10

CQ 7 assignment to scheduler input 10

enumerator kNETC_CQ7AssignToSchedIn9

CQ 7 assignment to scheduler input 9

enumerator kNETC_CQ7AssignToSchedIn8

CQ 7 assignment to scheduler input 8

enumerator kNETC_CQ7AssignToSchedIn7

CQ 7 assignment to scheduler input 7, means all CQ use weighted fair

enum __netc_tb_bp_fc_cfg

Buffer Pool Flow Control (FC) Configuration.

Values:

enumerator kNETC_FlowCtrlDisable

Flow Control disabled

enumerator `kNETC_FlowCtrlWithBP`

Flow Control enabled using only buffer pool FC state.

enumerator `kNETC_FlowCtrlWithSBP`

Flow Control enabled using only shared buffer pool FC state.

enumerator `kNETC_FlowCtrlWithBPAndSBP`

Flow Control enabled using both buffer pool and shared buffer pool FC state, only both 1 trigger the Flow Control ON

typedef enum `_netc_tb_index` `netc_tb_index_t`

Table index.

typedef enum `_netc_tb_cmd` `netc_tb_cmd_t`

Table management command operations.

typedef enum `_netc_tb_access_mode` `netc_tb_access_mode_t`

Table Access Method.

typedef enum `_netc_cbd_version` `netc_cbd_version_t`

NTMP version.

typedef enum `_netc_cmd_error` `netc_cmd_error_t`

Table command response error status.

typedef union `_netc_cmd_bd` `netc_cmd_bd_t`

The Switch/SI command BD data structure.

typedef struct `_netc_cmd_bdr_config` `netc_cmd_bdr_config_t`

Configuration for the Switch/SI command BD Ring Configuration.

typedef struct `_netc_cmd_bdr` `netc_cmd_bdr_t`

The Switch/SI command BD ring handle data structure.

typedef struct `_netc_tb_common_header` `netc_tb_common_header_t`

Table request data buffer common header.

typedef enum `_netc_fm_vlan_ud_act` `netc_fm_vlan_ud_act_t`

Frame Modification VLAN Update/Delete Action.

Note: Misconfiguration error if replace or delete action is specified and if VLAN tag is not present in frame.

typedef enum `_netc_fm_sqt_act` `netc_fm_sqt_act_t`

Frame Modification Sequence Tag Action.

Note: Must be set to 000b for Ingress frame modification, otherwise misconfiguration error..

typedef enum `_netc_fm_vlan_ar_act` `netc_fm_vlan_ar_act_t`

Frame Modification VLAN Add/Replace Action.

Note: For ingress frame modification with 00b or 01b, use the ingress port to select PCP and DEI from the Bridge port default VLAN register (BPDVR). For egress frame modification with 00b or 01b, use the internal QoS associated with the frame (IPV, DR) to access the QoS to PCP mapping profile (PQOSMR[QVMP] , QOSVLANMPaR0/1/2/3) to set the new PCP value. Use internal DR associated with frame to access the DR to DEI mapping profile (PPCPDEIMR[DRnDEI]) to set the new DEI value.

typedef enum *_netc_tb_eteid_access_mode* netc_tb_eteid_access_mode_t

Define FDB/L2MCF/IS table entry access the primary Egress Treatment table entry group mode.

Note: The FDB/L2 IPv4 Multicast filter table has precedence over any assignment made via the Ingress Stream table. For Multicast port mode, the index to access the Egress Treatment table is computed by adding an offset to the base index of the Egress Treatment group. That offset is derived from the applicability bitmap as follows: starting from the lowest significant bit of the bitmap, the first encountered bit set to 1, corresponds to offset 0, and so on. This continues till the destination port location in the bitmap is reached

typedef enum *_netc_tb_ipf_update_action* netc_tb_ipf_update_action_t

Ingress Port Filter Table Update Actions.

typedef enum *_netc_tb_ipf_attr_mask* netc_tb_ipf_attr_mask_t

Ingress Port Filter frame attribute mask.

typedef enum *_netc_tb_ipf_seq_tag* netc_tb_ipf_seq_tag_t

Ingress Port Filter frame attribute Sequence Tag Code.

typedef enum *_netc_tb_ipf_l4_header* netc_tb_ipf_l4_header_t

Ingress Port Filter frame attribute L4 Header Code.

typedef struct *_netc_tb_ipf_keye* netc_tb_ipf_keye_t

Ingress Port Filter key element.

typedef enum *_netc_tb_ipf_forward_action* netc_tb_ipf_forward_action_t

Ingress port filter forwarding Action.

typedef enum *_netc_tb_ipf_filter_action* netc_tb_ipf_filter_action_t

Ingress port filter Filter Action.

typedef struct *_netc_tb_ipf_cfge* netc_tb_ipf_cfge_t

Ingress port filter config element.

typedef struct *_netc_tb_ipf_stse* netc_tb_ipf_stse_t

Ingress port filter statistic element.

typedef struct *_netc_tb_ipf_req_data* netc_tb_ipf_req_data_t

Ingress port filter table entry config.

typedef struct *_netc_tb_ipf_rsp_data* netc_tb_ipf_rsp_data_t

Ingress port filter table response data.

typedef struct *_netc_tb_ipf_data* netc_tb_ipf_data_t

Ingress Port filter table data buffer.

typedef struct *_netc_tb_ipf_config* netc_tb_ipf_config_t

Ingress Port filter entry config.

typedef enum *_netc_tb_isi_key_type* netc_tb_isi_key_type

Stream identification table key type.

typedef struct *_netc_tb_isi_keye* netc_tb_isi_keye_t

Stream identification table key element.

typedef struct *_netc_tb_isi_cfge* netc_tb_isi_cfge_t

Stream identification table config element.

typedef struct *_netc_tb_isi_req_data* netc_tb_isi_req_data_t

Stream identification table request data buffer.

```
typedef struct _netc_tb_isi_rsp_data netc_tb_isi_rsp_data_t
    Stream identification table request response data buffer.

typedef struct _netc_tb_isi_data netc_tb_isi_data_t
    Stream identification table data buffer.

typedef struct _netc_tb_isi_config netc_tb_isi_config_t
    Stream identification table entry config.

typedef enum _netc_tb_is_isq_action netc_tb_is_isq_action_t
    Ingress Stream table Ingress Sequence Action.

typedef enum _netc_tb_is_forward_action netc_tb_is_forward_action_t
    Ingress Stream table forwarding Action.

typedef enum _netc_tb_is_ctd_mode netc_tb_is_ctd_mode_t
    Ingress Stream table Cut-Through Disable mode.

typedef netc_tb_eteid_access_mode_t netc_tb_is_eteid_mode_t
    Ingress Stream table Override ET_EID mode.

typedef struct _netc_tb_is_cfge netc_tb_is_cfge_t
    Ingress Stream table config element.

typedef struct _netc_tb_is_req_data netc_tb_is_req_data_t
    Ingress Stream table request data buffer.

typedef struct _netc_tb_is_rsp_data netc_tb_is_rsp_data_t
    Ingress Stream table request response data buffer.

typedef struct _netc_tb_is_data netc_tb_is_data_t
    Ingress Stream table data buffer.

typedef struct _netc_tb_is_config netc_tb_is_config_t
    Ingress Stream table entry config.

typedef struct _netc_tb_isf_keye netc_tb_isf_keye_t
    Ingress Stream Filter table key element.

typedef struct _netc_tb_isf_cfge netc_tb_isf_cfge_t
    Ingress Stream Filter table config element.

typedef struct _netc_tb_isf_req_data netc_tb_isf_req_data_t
    Ingress Stream Filter table request data buffer.

typedef struct _netc_tb_isf_rsp_data netc_tb_isf_rsp_data_t
    Ingress Stream Filter table request response data buffer.

typedef struct _netc_tb_isf_data netc_tb_isf_data_t
    Ingress Stream Filter table data buffer.

typedef struct _netc_tb_isf_config netc_tb_isf_config_t
    Ingress Stream Filter table entry config.

typedef enum _netc_tb_rp_update_action netc_tb_rp_update_action_t
    Rate Policer Table Update Actions.

typedef netc_tc_sdu_type_t netc_tb_rp_sdu_type_t
    Rate Policer Table Protocol/Service Data Unit Type.

typedef struct _netc_tb_rp_cfge netc_tb_rp_cfge_t
    Rate Policer table config element.
```

```
typedef struct _netc_tb_rp_fee netc_tb_rp_fee_t
    Rate Policer table Function Enable element.
typedef struct _netc_tb_rp_pse netc_tb_rp_pse_t
    Rate Policer table Policer State element.
typedef struct _netc_tb_rp_stse netc_tb_rp_stse_t
    Rate Policer table statistic element.
typedef struct _netc_tb_rp_req_data netc_tb_rp_req_data_t
    Rate Policer table request data buffer.
typedef struct _netc_tb_rp_rsp_data netc_tb_rp_rsp_data_t
    Rate Policer table request response data buffer.
typedef struct _netc_tb_rp_data netc_tb_rp_data_t
    Rate Policer table data buffer.
typedef struct _netc_tb_rp_config netc_tb_rp_config_t
    Rate Policer table entry config.
typedef struct _netc_tb_isc_stse netc_tb_isc_stse_t
    Ingress Stream Count table statistic element.
typedef struct _netc_tb_isc_req_data netc_tb_isc_req_data_t
    Ingress Stream Count table request data buffer.
typedef struct _netc_tb_isc_rsp_data netc_tb_isc_rsp_data_t
    Ingress Stream Count table request response data buffer.
typedef struct _netc_tb_isc_data netc_tb_isc_data_t
    Ingress Stream Count table data buffer.
typedef enum _netc_tb_sgi_update_action netc_tb_sgi_update_action_t
    Stream Gate Instance table Update Actions.
typedef netc_tc_sdu_type_t netc_tb_sgi_sdu_type_t
    Stream Gate Instance table Protocol/Service Data Unit Type.
typedef enum _netc_tb_sgi_state netc_tb_sgi_state_t
    Stream Gate Instance State.
typedef struct _netc_tb_sgi_cfge netc_tb_sgi_cfge_t
    Stream Gate Instance table config element.
typedef struct _netc_tb_sgi_acfge netc_tb_sgi_acfge_t
    Stream Gate Instance table Admin Configuration element.
typedef struct _netc_tb_sgi_icfge netc_tb_sgi_icfge_t
    Stream Gate Instance table Initial Configuration element.
typedef struct _netc_tb_sgi_sgise netc_tb_sgi_sgise_t
    Stream Gate Instance table stream gate instance state element.
typedef struct _netc_tb_sgi_req_data netc_tb_sgi_req_data_t
    Stream Gate Instance table request data buffer.
typedef struct _netc_tb_sgi_rsp_data netc_tb_sgi_rsp_data_t
    Stream Gate Instance table request response data buffer.
typedef struct _netc_tb_sgi_data netc_tb_sgi_data_t
    Stream Gate Instance table data buffer.
```



```
typedef struct _netc_tb_sgi_config netc_tb_sgi_config_t
```

Stream Gate Instance table entry config.

```
typedef struct _netc_sgcl_gate_entry netc_sgcl_gate_entry_t
```

Defines the Stream Gate Control entry structure.

```
typedef struct _netc_tb_sgcl_cfge netc_tb_sgcl_cfge_t
```

Stream Gate Control List table config element.

```
typedef struct _netc_tb_sgcl_sgclse netc_tb_sgcl_sgclse_t
```

Stream Gate Control List table Stream Gate Control List State element.

```
typedef struct _netc_tb_sgcl_req_data netc_tb_sgcl_req_data_t
```

Stream Gate Control List table request data buffer.

```
typedef struct _netc_tb_sgcl_rsp_data netc_tb_sgcl_rsp_data_t
```

Stream Gate Control List table request response data buffer.

```
typedef struct _netc_tb_sgcl_data netc_tb_sgcl_data_t
```

Stream Gate Control List table data buffer.

```
typedef struct _netc_tb_sgcl_gcl netc_tb_sgcl_gcl_t
```

Stream Gate Control List table entry gate control list structure.

```
typedef enum _netc_tb_fm_layer2_act netc_tb_fm_layer2_act_t
```

Frame Modification table Layer 2 Actions.

Note: This field must be set to 0 for traffic destined to a pseudo link. This field must be set to 0 for any device with ASIL-B safety requirements.

```
typedef enum _netc_tb_fm_mac_header_act netc_tb_fm_mac_header_act_t
```

Frame Modification table Layer 2 Header MAC Actions.

Note: Ingress frame modifications only support kNETC_NoAction or kNETC_ReplDmac.

```
typedef enum _netc_tb_fm_vlan_header_act netc_tb_fm_vlan_header_act_t
```

Frame Modification table Layer 2 VLAN Actions.

Note: For use Delete or Replace action, if no outer VLAN header is present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

```
typedef enum _netc_tb_fm_outer_vid_act netc_tb_fm_outer_vid_act_t
```

Frame Modification table Layer 2 outer VLAN VID Actions.

Note: For use kNETC_UseFrameVID action, if no outer VLAN header is present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

```
typedef netc_fm_sqt_act_t netc_tb_fm_sqt_act_t
```

Frame Modification table Sequence Tag Action.

Note: For use kNETC_ReomveTag action, If R-TAG/draft 2.0 R-TAG/HSR tag not present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

typedef enum *_netc_tb_fm_outer_tpid_act* netc_tb_fm_outer_tpid_act_t
Frame Modification table Outer TPID action.

Note: For use kNETC_UseFrameTpid action, If outer VLAN header not present, then a mis-configuration event will be generated and handled according to the port's PFMCR register.

typedef enum *_netc_tb_fm_outer_pcp_act* netc_tb_fm_outer_pcp_act_t
Frame Modification table Outer PCP action.

Note: For use kNETC_UseFramePcp/kNETC_UseFramePcpMap action, If outer VLAN header not present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

typedef enum *_netc_tb_fm_outer_dei_act* netc_tb_fm_outer_dei_act_t
Frame Modification table Outer DEI action.

Note: For use kNETC_UseFrameDei action, If outer VLAN header not present, then a mis-configuration event will be generated and handled according to the port's PFMCR register.

typedef enum *_netc_tb_fm_payload_act* netc_tb_fm_payload_act_t
Frame Modification table Payload Actions.

Note: This field must be set to 0 for traffic destined to a pseudo link. This field must be set to 0 for any device with ASIL-B safety requirements.

typedef struct *_netc_tb_fm_cfge* netc_tb_fm_cfge_t
Frame Modification table config element.

typedef struct *_netc_tb_fm_req_data* netc_tb_fm_req_data_t
Frame Modification table request data buffer.

typedef struct *_netc_tb_fm_rsp_data* netc_tb_fm_rsp_data_t
Frame Modification table request response data buffer.

typedef struct *_netc_tb_fm_data* netc_tb_fm_data_t
Frame Modification table data buffer.

typedef struct *_netc_tb_fm_config* netc_tb_fm_config_t
Frame Modification table entry config.

typedef struct *_netc_tb_fmd_req_data* netc_tb_fmd_req_data_t
Frame Modification Data table request data buffer.

typedef struct *_netc_tb_fmd_rsp_data* netc_tb_fmd_rsp_data_t
Frame Modification Data table request response data buffer.

typedef struct *_netc_tb_fmd_data* netc_tb_fmd_data_t
Frame Modification Data table data buffer.

typedef struct *_netc_tb_fmd_update_config* netc_tb_fmd_update_config_t
Frame Modification data table entry update config.

typedef struct *_netc_tb_fmd_query_buffer* netc_tb_fmd_query_buffer_t
Frame Modification data table entry query data buffer.

typedef struct *_netc_tb_vf_keye* netc_tb_vf_keye_t

Vlan Filter table key element.

typedef struct *_netc_tb_vf_cfge* netc_tb_vf_cfge_t

Vlan Filter table config element.

typedef struct *_netc_tb_vf_search_criteria* netc_tb_vf_search_criteria_t

Vlan Filter table search criteria format.

typedef struct *_netc_tb_vf_req_data* netc_tb_vf_req_data_t

Vlan Filter table request data buffer.

typedef struct *_netc_tb_vf_rsp_data* netc_tb_vf_rsp_data_t

Vlan Filter table request response data buffer.

typedef struct *_netc_tb_vf_data* netc_tb_vf_data_t

Vlan Filter table data buffer.

typedef struct *_netc_tb_vf_config* netc_tb_vf_config_t

Vlan Filter table entry config.

typedef enum *_netc_tb_fdb_update_action* netc_tb_fdb_update_action_t

FDB table Update Actions.

typedef *netc_tb_eteid_access_mode_t* netc_tb_fdb_oetoid_mode_t

FDB table entry defined the egress packet processing actions (will cover the actions which specified in the Egress Treatment table)

typedef enum *_netc_tb_fdb_ctd_mode* netc_tb_fdb_ctd_mode_t

FDB table Cut-Through Disable mode.

typedef struct *_netc_tb_fdb_keye* netc_tb_fdb_keye_t

typedef struct *_netc_tb_fdb_cfge* netc_tb_fdb_cfge_t

FDB table configuration element.

typedef struct *_netc_tb_fdb_acte* netc_tb_fdb_acte_t

FDB table Activity element.

typedef enum *_netc_tb_fdb_sc_keye_mc* netc_tb_fdb_sc_keye_mc_t

FDB table search criteria Key Element Match Criteria.

typedef enum *_netc_tb_fdb_sc_cfge_mc* netc_tb_fdb_sc_cfge_mc_t

FDB table search criteria Configuration Element Match Criteria.

typedef enum *_netc_tb_fdb_sc_acte_mc* netc_tb_fdb_sc_acte_mc_t

FDB table search criteria Activity Element Match Criteria.

typedef struct *_netc_tb_fdb_search_criteria* netc_tb_fdb_search_criteria_t

FDB table search criteria format.

typedef struct *_netc_tb_fdb_req_data* netc_tb_fdb_req_data_t

FDB table request data buffer.

typedef struct *_netc_tb_fdb_rsp_data* netc_tb_fdb_rsp_data_t

FDB table request response data buffer.

typedef struct *_netc_tb_fdb_data* netc_tb_fdb_data_t

FDB table data buffer.

typedef struct *_netc_tb_fdb_config* netc_tb_fdb_config_t

FDB table entry config.

```
typedef enum _netc_tb_l2mcf_key_type netc_tb_l2mcf_key_type_t
    L2 IPV4 Multicast Filter table key type.
typedef struct _netc_tb_l2mcf_keye netc_tb_l2mcf_keye_t
    L2 IPV4 Multicast Filter table key element.
typedef enum _etc_tb_l2mcf_sc_keye_mc etc_tb_l2mcf_sc_keye_mc_t
    L2 IPV4 Multicast Filter table search criteria Key Element Match Criteria.
typedef enum _etc_tb_l2mcf_sc_cfge_mc etc_tb_l2mcf_sc_cfge_mc_t
    L2 IPV4 Multicast Filter table search criteria Configuration Element Match Criteria.
typedef enum _etc_tb_l2mcf_sc_acte_mc etc_tb_l2mcf_sc_acte_mc_t
    FDB table search criteria Activity Element Match Criteria.
typedef netc_tb_fdb_cfge_t netc_tb_l2mcf_cfge_t
    L2 IPV4 Multicast Filter table config element.
typedef netc_tb_fdb_acte_t netc_tb_l2mcf_acte_t
    L2 IPV4 Multicast Filter table activity lement.
typedef struct _etc_tb_l2mcf_search_criteria netc_tb_l2mcf_search_criteria_t
    L2 IPV4 Multicast Filter table search criteria format.
typedef struct _netc_tb_l2mcf_req_data netc_tb_l2mcf_req_data_t
    L2 IPV4 Multicast Filter table request data buffer.
typedef struct _netc_tb_l2mcf_rsp_data netc_tb_l2mcf_rsp_data_t
    L2 IPV4 Multicast Filter table request response data buffer.
typedef struct _netc_tb_l2mcf_data netc_tb_l2mcf_data_t
    L2 IPV4 Multicast Filter table data buffer.
typedef struct _netc_tb_l2mcf_config netc_tb_l2mcf_config_t
    L2 IPV4 Multicast Filter table entry config.
typedef enum _netc_tb_iseqg_sqtg netc_tb_iseqg_sqtg_t
    Sequence Tag Type.
typedef struct _netc_tb_iseqg_cfge netc_tb_iseqg_cfge_t
    Ingress Sequence Generation table config element.
typedef struct _netc_tb_iseqg_sgse netc_tb_iseqg_sgse_t
    Ingress Sequence Generation table Sequence generation state element.
typedef struct _netc_tb_iseqg_req_data netc_tb_iseqg_req_data_t
    Ingress Sequence Generation table request data buffer.
typedef struct _netc_tb_iseqg_rsp_data netc_tb_iseqg_rsp_data_t
    Ingress Sequence Generation table request response data buffer.
typedef struct _netc_tb_iseqg_data netc_tb_iseqg_data_t
    Ingress Sequence Generation table data buffer.
typedef struct _netc_tb_iseqg_config netc_tb_iseqg_config_t
    Ingress Sequence Generation table entry config.
typedef enum _netc_tb_iseqg_update_action netc_tb_iseqg_update_action_t
    Ingress Sequence Generation Table Update Actions.
typedef enum _netc_tb_eseqr_sqtg netc_tb_eseqr_sqtg_t
    Egress Sequence Recovery table Sequence Tag Type.
```

```
typedef struct _netc_tb_eseqr_cfge netc_tb_eseqr_cfge_t
    Egress Sequence Recovery table config element.
typedef struct _netc_tb_eseqr_stse netc_tb_eseqr_stse_t
    Egress Sequence Recovery table statistic element.
typedef struct _netc_tb_eseqr_srse netc_tb_eseqr_srse_t
    Egress Sequence Recovery table sequence recovery state element.
typedef struct _netc_tb_eseqr_req_data netc_tb_eseqr_req_data_t
    Egress Sequence Recovery table request data buffer.
typedef struct _netc_tb_eseqr_rsp_data netc_tb_eseqr_rsp_data_t
    Egress Sequence Recovery table request response data buffer.
typedef struct _netc_tb_eseqr_data netc_tb_eseqr_data_t
    Egress Sequence Recovery table data buffer.
typedef struct _netc_tb_eseqr_config netc_tb_eseqr_config_t
    Egress Sequence Recovery table entry config.
typedef enum _netc_tb_tgs_entry_id netc_tb_tgs_entry_id_t
    Time Gate Scheduling table entry ID for switch and ENETC.
typedef enum _netc_tb_tgs_gate_type netc_tb_tgs_gate_type_t
    Administrative gate operation type.
typedef struct _netc_tgs_gate_entry netc_tgs_gate_entry_t
    Defines the Time Gate Scheduling gate control entry structure.
typedef struct _netc_tb_tgs_cfge netc_tb_tgs_cfge_t
    Time Gate Scheduling table config element.
typedef struct _netc_tb_tgs_olse netc_tb_tgs_olse_t
    Time Gate Scheduling table statistic element.
typedef struct _netc_tb_tgs_req_data netc_tb_tgs_req_data_t
    Time Gate Scheduling table request data buffer.
typedef struct _netc_tb_tgs_rsp_data netc_tb_tgs_rsp_data_t
    Time Gate Scheduling table request response data buffer.
typedef struct _netc_tb_tgs_data netc_tb_tgs_data_t
    Time Gate Scheduling table data buffer, set with max size.
typedef struct _netc_tb_tgs_gcl netc_tb_tgs_gcl_t
    Time Gate Scheduling table entry gate control list structure.
typedef enum _netc_tb_et_efm_mode netc_tb_et_efm_mode_t
    Egress Frame Modification entry mode.
typedef enum _netc_tb_et_esq_act netc_tb_et_esq_act_t
    Egress Sequence Actions.
typedef enum _netc_tb_et_ec_act netc_tb_et_ec_act_t
    Egress Counter Action.
typedef struct _netc_tb_et_cfge netc_tb_et_cfge_t
    Egress Treatment table config element.
typedef struct _netc_tb_et_req_data netc_tb_et_req_data_t
    Egress Treatment table request data buffer.
```

```
typedef struct _netc_tb_et_rsp_data netc_tb_et_rsp_data_t
    Egress Treatment table request response data buffer.
typedef struct _netc_tb_et_data netc_tb_et_data_t
    Egress Treatment table data buffer.
typedef struct _netc_tb_et_config netc_tb_et_config_t
    Egress Treatment table entry config.
typedef enum _netc_tb_etmcq_update_action netc_tb_etmcq_update_action_t
    ETM Class Queue table Update Actions.
typedef struct _netc_tb_etmcq_cfge netc_tb_etmcq_cfge_t
    ETM Class Queue table config element.
typedef struct _netc_tb_etmcq_stse netc_tb_etmcq_stse_t
    ETM Class Queue table statistic element.
typedef struct _netc_tb_etmcq_req_data netc_tb_etmcq_req_data_t
    ETM Class Queue table request data buffer.
typedef struct _netc_tb_etmcq_rsp_data netc_tb_etmcq_rsp_data_t
    ETM Class Queue table request response data buffer.
typedef struct _netc_tb_etmcq_data netc_tb_etmcq_data_t
    ETM Class Queue table data buffer.
typedef struct _netc_tb_etmcq_config netc_tb_etmcq_config_t
    ETM Class Queue table entry config.
typedef enum _netc_tb_etmcs_entry_id netc_tb_etmcs_entry_id_t
    ETM Class Scheduler table entry ID.
typedef enum _netc_tb_etmcs_ca_assg netc_tb_etmcs_ca_assg_t
    ETM Class Scheduler table Class queue assignment to scheduler inputs mode.
typedef struct _netc_tb_etmcs_cfge netc_tb_etmcs_cfge_t
    ETM Class Scheduler table config element.
typedef struct _netc_tb_etmcs_req_data netc_tb_etmcs_req_data_t
    ETM Class Scheduler table request data buffer.
typedef struct _netc_tb_etmcs_rsp_data netc_tb_etmcs_rsp_data_t
    ETM Class Scheduler table request response data buffer.
typedef struct _netc_tb_etmcs_data netc_tb_etmcs_data_t
    ETM Class Scheduler table data buffer.
typedef struct _netc_tb_etmcs_config netc_tb_etmcs_config_t
    ETM Class Scheduler table entry config.
typedef struct _netc_tb_etmcg_cfge netc_tb_etmcg_cfge_t
    ETM Congestion Group table config element.
typedef struct _netc_tb_etmcg_stse netc_tb_etmcg_stse_t
    ETM Congestion Group table statistic element.
typedef struct _netc_tb_etmcg_req_data netc_tb_etmcg_req_data_t
    ETM Congestion Group table request data buffer.
typedef struct _netc_tb_etmcg_rsp_data netc_tb_etmcg_rsp_data_t
    ETM Congestion Group table request response data buffer.
```

```
typedef struct _netc_tb_etmcg_data netc_tb_etmcg_data_t
    ETM Congestion Group table data buffer.
typedef struct _netc_tb_etmcg_config netc_tb_etmcg_config_t
    ETM Congestion Group table entry config.
typedef struct _netc_tb_ec_stse netc_tb_ec_stse_t
    Egress Count table statistic element.
typedef struct _netc_tb_ec_req_data netc_tb_ec_req_data_t
    Egress Count table request data buffer.
typedef struct _netc_tb_ec_rsp_data netc_tb_ec_rsp_data_t
    Egress Count table request response data buffer.
typedef struct _netc_tb_ec_data netc_tb_ec_data_t
    Egress Count table data buffer.
typedef enum _netc_tb_bp_fc_cfg netc_tb_bp_fc_cfg_t
    Buffer Pool Flow Control (FC) Configuration.
typedef struct _netc_tb_bp_cfge netc_tb_bp_cfge_t
    Buffer Pool table config element.
typedef struct _netc_tb_bp_bpse netc_tb_bp_bpse_t
    Buffer Pool table State Element Data.
typedef struct _netc_tb_bp_req_data netc_tb_bp_req_data_t
    Buffer Pool table request data buffer.
typedef struct _netc_tb_bp_rsp_data netc_tb_bp_rsp_data_t
    Buffer Pool table request response data buffer.
typedef struct _netc_tb_bp_data netc_tb_bp_data_t
    Buffer Pool table data buffer.
typedef struct _netc_tb_bp_config netc_tb_bp_config_t
    Buffer Pool table entry config.
typedef struct _netc_tb_sbp_cfge netc_tb_sbp_cfge_t
    Shared Buffer Pool table config element.
typedef struct _netc_tb_sbp_sbpse netc_tb_sbp_sbpse_t
    Shared Buffer Pool table State Element Data.
typedef struct _netc_tb_sbp_req_data netc_tb_sbp_req_data_t
    Shared Buffer Pool table request data buffer.
typedef struct _netc_tb_sbp_rsp_data netc_tb_sbp_rsp_data_t
    Shared Buffer Pool table request response data buffer.
typedef struct _netc_tb_sbp_data netc_tb_sbp_data_t
    Shared Buffer Pool table data buffer.
typedef struct _netc_tb_sbp_config netc_tb_sbp_config_t
    Shared Buffer Pool table entry config.
typedef union _netc_tb_data_buffer netc_tb_data_buffer_t
    Table common data buffer.
typedef struct _netc_cbdr_hw netc_cbdr_hw_t
    Register group for SI/Switch command bd ring.
```

```
typedef struct _netc_cbdr_handle netc_cbdr_handle_t
```

Handle for common part of EP/Switch NTMP.

```
status_t NETC_CmdBDRInit(netc_cbdr_hw_t *base, const netc_cmd_bdr_config_t *config)
```

Initialize the command BD ring.

Parameters

- base –
- config –

Returns

kStatus_Success

Returns

kStatus_Fail

```
status_t NETC_CmdBDRDeinit(netc_cbdr_hw_t *base)
```

Deinitialize the command BD ring.

Parameters

- base –

Returns

kStatus_Success

```
status_t NETC_CmdBDSendCommand(netc_cbdr_hw_t *base, netc_cmd_bdr_t *cbdr,  
                                netc_cmd_bd_t *cbd, netc_cbd_version_t version)
```

Send the Command Buffer Descriptor to operate on a NTMP table.

Parameters

- base –
- cbdr –
- cbd –
- version –

Returns

status_t

Returns

See netc_cmd_error_t

```
status_t NETC_AddIPFTableEntry(netc_cbdr_handle_t *handle, netc_tb_ipf_config_t *config,  
                               uint32_t *entryID)
```

Add entry into the ingress Port Filter Table.

Parameters

- handle –
- config –
- entryID –

Returns

status_t

Returns

See netc_cmd_error_t

```
status_t NETC_UpdateIPFTableEntry(netc_cbdr_handle_t *handle, uint32_t entryID,  
                                   netc_tb_ipf_cfge_t *cfg)
```

Update entry in the ingress Port Filter Table.

Parameters

- handle –
- entryID –
- cfg –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_QueryIPFTableEntry(*netc_cldr_handle_t* *handle, uint32_t entryID,
netc_tb_ipf_config_t *config)

Query entry in the ingress Port Filter Table.

Parameters

- handle –
- entryID –
- config –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_DelIPFTableEntry(*netc_cldr_handle_t* *handle, uint32_t entryID)

Delete an entry in the ingress Port Filter Table.

Parameters

- handle –
- entryID –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_ResetIPFMatchCounter(*netc_cldr_handle_t* *handle, uint32_t entryID)

Reset the counter of an ingress port filter Table entry.

Parameters

- handle –
- entryID –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_GetIPFMatchedCount(*netc_cldr_handle_t* *handle, uint32_t entryID, uint64_t
*count)

Get the matched count of an ingress port filter Table entry.

Parameters

- handle –
- entryID –

- count –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_AddISITableEntry(*netc_cldr_handle_t* *handle, *netc_tb_isi_config_t* *config, uint32_t *entryID)

Add entry into Ingress Stream Identification table.

Parameters

- handle –
- config –
- entryID –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_DeISITableEntry(*netc_cldr_handle_t* *handle, uint32_t entryID)

Delete an entry in Ingress stream identification table.

Parameters

- handle –
- entryID –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_QueryISITableEntry(*netc_cldr_handle_t* *handle, uint32_t entryID, *netc_tb_isi_config_t* *config)

Query Ingress Stream Identification table.

Parameters

- handle –
- entryID –
- config –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_QueryISITableEntryWithKey(*netc_cldr_handle_t* *handle, *netc_tb_isi_keye_t* *keye, *netc_tb_isi_rsp_data_t* *rsp)

Query Ingress Stream Identification table with key.

Parameters

- handle –
- keye –
- rsp –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC__AddOrUpdateISTableEntry(*netc_cldr_handle_t* *handle, *netc_tb_is_config_t* *config, bool isAdd)

Add or update entry in Ingress Stream table.

Parameters

- handle –
- config –
- isAdd –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC__QueryISTableEntry(*netc_cldr_handle_t* *handle, uint32_t entryID, *netc_tb_is_config_t* *config)

Query Ingress Stream table.

Parameters

- handle –
- entryID –
- config –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC__DelISTableEntry(*netc_cldr_handle_t* *handle, uint32_t entryID)

Delete an entry in Ingress stream table.

Parameters

- handle –
- entryID –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC__AddISFTableEntry(*netc_cldr_handle_t* *handle, *netc_tb_isf_config_t* *config, uint32_t *entryID)

Add entry into ingress stream filter table.

Parameters

- handle –
- config –
- entryID –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_UpdateISFTableEntry(*netc_cldr_handle_t* *handle, uint32_t entryID,
netc_tb_isf_cfg_t *cfg)

Update entry into ingress stream filter table.

Parameters

- handle –
- entryID –
- cfg –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_DelISFTableEntry(*netc_cldr_handle_t* *handle, uint32_t entryID)

Delete an entry in Ingress stream filter table.

Parameters

- handle –
- entryID –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_QueryISFTableEntry(*netc_cldr_handle_t* *handle, *netc_tb_isf_keye_t* *keye,
netc_tb_isf_rsp_data_t *rsp)

Query entry from the Ingress stream filter table.

Parameters

- handle –
- keye –
- rsp –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_AddISCTableEntry(*netc_cldr_handle_t* *handle, uint32_t entryID)

Add entry in ingress stream count table.

Parameters

- handle –
- entryID –

Returns

status_t

status_t NETC_GetISCStatistic(*netc_cbd_r_handle_t* *handle, uint32_t entryID, *netc_tb_isc_stse_t* *statistic)

Get ingress stream count statistic.

Parameters

- handle –
- entryID –
- statistic –

Returns

status_t

Returns

See *netc_cmd_error_t*

status_t NETC_ResetISCStatistic(*netc_cbd_r_handle_t* *handle, uint32_t entryID)

Reset the count of the ingress stream count.

Parameters

- handle –
- entryID –

Returns

status_t

Returns

See *netc_cmd_error_t*

status_t NETC_AddOrUpdateSGITableEntry(*netc_cbd_r_handle_t* *handle, *netc_tb_sgi_config_t* *config, bool isAdd)

Add or update entry in stream gate instance table.

Parameters

- handle –
- config –
- isAdd –

Returns

status_t

Returns

See *netc_cmd_error_t*

status_t NETC_ResetIRXOEXSGITableEntry(*netc_cbd_r_handle_t* *handle, uint32_t entryID)

Reset IRX and OEX flags in stream gate instance entry.

Parameters

- handle –
- entryID –

Returns

status_t

Returns

See *netc_cmd_error_t*

status_t NETC_DelSGITableEntry(*netc_cbd_r_handle_t* *handle, uint32_t entryID)

Delete entry in the stream gate instance table.

Parameters

- handle –
- entryID –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_GetSGIState(*netc_cldr_handle_t* *handle, uint32_t entryID, *netc_tb_sgi_sgise_t* *statis)

Get statistic of specified stream gate instance table entry.

Parameters

- handle –
- entryID –
- statis –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_QuerySGITableEntry(*netc_cldr_handle_t* *handle, uint32_t entryID, *netc_tb_sgi_config_t* *config)

Query entry from the stream gate instance table.

Parameters

- handle –
- entryID –
- config –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_AddSGCLTableEntry(*netc_cldr_handle_t* *handle, *netc_tb_sgcl_gcl_t* *config)

Add entry into Stream Gate Control List Table.

Parameters

- handle –
- config –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_DelSGCLTableEntry(*netc_cldr_handle_t* *handle, uint32_t entryID)

Delete entry of Stream Gate Control List Table.

Parameters

- handle –
- entryID –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_GetSGCLGateList(*netc_cbdr_handle_t* *handle, *netc_tb_sgcl_gcl_t* *gcl, uint32_t length)

Get Stream Gate Control List Table entry gate control list.

Parameters

- handle –
- gcl –
- length –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_GetSGCLState(*netc_cbdr_handle_t* *handle, uint32_t entryID, *netc_tb_sgcl_sgclse_t* *state)

Get state (ref count) for Stream Gate Control List table entry.

Parameters

- handle –
- entryID –
- state –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_QueryRPTableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID, *netc_tb_rp_rsp_data_t* *rsp)

Query entry from the Rate Policer table.

Parameters

- handle –
- entryID –
- rsp –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_AddOrUpdateRPTableEntry(*netc_cbdr_handle_t* *handle, *netc_tb_rp_config_t* *config, *netc_tb_cmd_t* cmd)

Add or update entry in Rate Policer table.

Parameters

- handle –
- config –

- cmd –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_DelRPTableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID)

Delete entry in the Rate Policer table.

Parameters

- handle –
- entryID –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_GetRPStatistic(*netc_cbdr_handle_t* *handle, uint32_t entryID, *netc_tb_rp_stse_t* *statis)

Get statistic of specified Rate Policer table entry.

Parameters

- handle –
- entryID –
- statis –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_ResetMRRPTableEntry(*netc_cbdr_handle_t* *handle, uint32_t entryID)

Reset mark red parameter of specified Rate Policer table entry.

Parameters

- handle –
- entryID –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_ConfigTGSAdminList(*netc_cbdr_handle_t* *handle, *netc_tb_tgs_gcl_t* *config)

Config the QBV (Time Gate Scheduling)

Parameters

- handle –
- config –

Returns

status_t

Returns

See netc_cmd_error_t

status_t NETC_GetTGSOperationList(*netc_cldr_handle_t* *handle, *netc_tb_tgs_gcl_t* *gcl, *uint32_t* length)

Get time gate table operation list.

Parameters

- handle –
- gcl –
- length –

Returns

status_t

Returns

See *netc_cmd_error_t*

NETC_FD_EID_ENCODE_OPTION_0(entryId)

Frame Modification Entry ID encode options.

Note: *sqta* should be *netc_fm_sqt_act_t* type, *vuda* should be *netc_fm_vlan_ud_act_t* type and *vara* should be *netc_fm_vlan_ar_act_t* type.

NETC_FD_EID_ENCODE_OPTION_1(sqta, vuda)

NETC_FD_EID_ENCODE_OPTION_2(vara, vid)

NETC_ISI_VLAN_FRAME_KEY(valid, pcp, vid)

2 Bytes VLAN field which may added to the frame Key

NETC_TB_SGCL_MAX_ENTRY

Stream Gate Control List Table maximum gate control list length.

NETC_TB_FMD_UPDATE_CONFIG_LENGTH(x)

TFrame Modification Data table update config Data Buffer length, x is the number of update data bytes.

NETC_TB_TGS_MAX_ENTRY

Time Gate Scheduling Table maximum gate control list length (TGSTCAPR[MAX_GCL_LEN])

NETC_TB_ETM_CQ_ENTRY_ID(portID, cqID)

ETM Class Queue table entry ID macro, cqID is represents the Class Queue ID ,rang in 0 ~ 7, portID is Switch ID, rang in 0 ~ 4.

NETC_TB_ETM_CG_ENTRY_ID(portID, cgID)

ETM Congestion Group table entry ID macro, cgID is represents the Congestion Group ID ,rang in 0 ~ 7, portID is Switch ID, rang in 0 ~ 4.

NETC_TB_BP_THRESH(mant, exp)

Buffer pool and shared buffer pool threshold macro, the threshold = MANT*2^EXP, uint is internal memory words (avergae of 20 bytes each)

union *_netc_cmd_bd*

#include <fsl_netc.h> The Switch/SI command BD data structure.

Public Members

struct *_netc_cmd_bd* req

struct *_netc_cmd_bd* resp

struct *_netc_cmd_bd* generic

struct *_netc_cmd_bdr_config*

#include <fsl_netc.h> Configuration for the Switch/SI command BD Ring Configuration.

Public Members

netc_cmd_bd_t *bdBase

BDR base address which shall be 128 bytes aligned

uint16_t bdLength

Size of BD ring which shall be multiple of 8 BD

bool enCompInt

Enable/Disable command BD completion interrupt

struct *_netc_cmd_bdr*

#include <fsl_netc.h> The Switch/SI command BD ring handle data structure.

Public Members

netc_cmd_bd_t *bdBase

BDR base address which shall be 128 bytes aligned

uint16_t bdLength

Size of BD ring

uint16_t producerIndex

Current index for execution.

uint16_t cleanIndex

Current index for cleaning.

bool bdrEnable

Current command BD ring is enable or not.

struct *_netc_tb_common_header*

#include <fsl_netc.h> Table request data buffer common header.

Public Members

uint32_t updateActions

Update Actions

uint32_t queryActions

Query Actions

struct *_netc_tb_ipf_keye*

#include <fsl_netc.h> Ingress Port Filter key element.

Public Members

uint16_t precedence

Precedence value of an entry

struct *_netc_tb_ipf_keye* frameAttr

Frame Attribute flags

`uint16_t frameAttrMask`
Frame attribute mask, set with OR of `netc_tb_ipf_attr_mask_t`

`uint16_t dscp`
Differentiated Services Code Point

`uint16_t dscpMask`
Differentiated Services Code Point Mask

`uint16_t srcPort`
Source Port ID

`uint16_t srcPortMask`
Source Port ID Mask

`uint16_t outerVlanTCI`
Outer VLAN Tag Control Information

`uint16_t outerVlanTCIMask`
Outer VLAN Tag Control Information Mask

`uint8_t dmac[6]`
Destination MAC Address

`uint8_t dmacMask[6]`
Destination MAC Address Mask

`uint8_t smac[6]`
Source MAC Address

`uint8_t smacMask[6]`
Source MAC Address Mask

`uint16_t innerVlanTCI`
Inner VLAN Tag Control Information

`uint16_t innerVlanTCIMask`
Inner VLAN Tag Control Information Mask

`uint16_t etherType`
2-byte EtherType

`uint16_t etherTypeMask`
EtherType Mask

`uint8_t IPProtocol`
IP Protocol

`uint8_t IPProtocolMask`
IP Protocol Mask

`uint8_t srcIPAddr[16]`
IP Source Address, Bits 127-0: IPv6 source address, Bits 127-96: IPv4 source address

`uint8_t srcIPAddrMask[16]`
IP Source Address Mask

`uint16_t l4SrcPort`
L4 Source Port

`uint16_t l4SrcPortMask`
L4 Source Port Mask

uint8_t destIPAddr[16]

IP Destination Address, Bits 127-0: IPv6 source address, Bits 127-96: IPv4 source address

uint8_t destIPAddrMask[16]

IP Destination Address Mask

uint16_t l4DestPort

L4 Destination Port

uint16_t l4DestPortMask

L4 Destination Port Mask

struct __netc__tb_ipf_cfge

#include <fsl_netc.h> Ingress port filter config element.

Public Members

uint32_t ipv

Internal Priority Value

uint32_t oipv

Overwrite IPV

uint32_t dr

Drop Resilience

uint32_t odr

Overwrite DR

netc__tb_ipf_forward_action_t fltfa

Filter Forwarding action.

uint32_t imire

Ingress Mirroring Enable

uint32_t wolte

Wake-onLAN trigger enable

netc__tb_ipf_filter_action_t flta

Filter Action.

uint32_t rpr

Relative Precedent Resolution

uint32_t ctd

Cut through disable.

netc__host_reason_t hr

Host Reason metadata when frame is redirected/copied to the switch management port

uint32_t timecape

Timestamp capture enable

uint32_t fltaTgt

Target for selected switch forwarding action or filter action

struct __netc__tb_ipf_stse

#include <fsl_netc.h> Ingress port filter statistic element.

Public Members

uint32_t matchCount[2]

A count of how many times this entry has been matched.

struct __netc_tb_ipf_req_data

#include <fsl_netc.h> Ingress port filter table entry config.

struct __netc_tb_ipf_rsp_data

#include <fsl_netc.h> Ingress port filter table response data.

Public Members

uint32_t entryID

Present only for commands which perform a query

netc_tb_ipf_keye_t keye

Present only for commands which perform a query

netc_tb_ipf_stse_t stse

Present only for commands which perform a query

netc_tb_ipf_cfge_t cfge

Present only for commands which perform a query

struct __netc_tb_ipf_data

#include <fsl_netc.h> Ingress Port filter table data buffer.

struct __netc_tb_ipf_config

#include <fsl_netc.h> Ingress Port filter entry config.

struct __netc_tb_isi_keye

#include <fsl_netc.h> Stream identification table key element.

Public Members

netc_tb_isi_key_type keyType

Define the key type used for the current isi entry

uint8_t srcPortID

Source Port ID, used when kc portp filed is 1. Only for SWITCH

uint8_t spm

Source Port Masquerading, used when kc spm filed is 1. Only for SWITCH

uint8_t framekey[16]

Frame portion of the key.

struct __netc_tb_isi_cfge

#include <fsl_netc.h> Stream identification table config element.

Public Members

uint32_t iSEID

Ingress stream entry ID, 0xFFFFFFFF means NULL

struct __netc_tb_isi_req_data

#include <fsl_netc.h> Stream identification table request data buffer.

struct __netc_tb_isi_rsp_data

#include <fsl_netc.h> Stream identification table request response data buffer.

Public Members

uint32_t entryID

Only present for query command

netc_tb_isi_keye_t keye

Only present for query command

netc_tb_isi_cfge_t cfge

Only present for query command

struct __netc_tb_isi_data

#include <fsl_netc.h> Stream identification table data buffer.

struct __netc_tb_isi_config

#include <fsl_netc.h> Stream identification table entry config.

struct __netc_tb_is_cfge

#include <fsl_netc.h> Ingress Stream table config element.

Public Members

uint32_t sfe

Stream Filtering Enable

uint32_t ipv

Internal Priority Value, active when opiv is set to 1

uint32_t oipv

Override internal priority value

uint32_t dr

Drop Resilience, active when odr is set to 1

uint32_t odr

Overwrite DR

uint32_t imire

Ingress Mirroring Enable, not applicable to ENETC

uint32_t timecape

Timestamp Capture Enable, not applicable to ENETC

uint32_t sppd

Source Port Pruning Disable, not applicable to ENETC

netc_tb_is_isq_action_t isqa

Ingress Sequence Action, not applicable to ENETC

uint32_t orp

Override Rate Policer ID

uint32_t osgi

Override stream gate instance entry id (default is NULL)

netc_host_reason_t hr

Host Reason when frame is redirected (fa = 01b) to the switch management port or copied to the switch management port (fa = 100b or 101b), value specified has to be a software defined Host Reason (8-15).

netc_tb_is_forward_action_t fa

Forward Option

netc_tc_sdu_type_t sduType

Service Data Unit Type to user for MSDU

uint32_t msdu

Maximum Service Data Unit

uint32_t ifmeLenChange

Ingress Frame Modification Entry Frame Length Change, specified in unit of bytes using a 2's complement notation

uint32_t eport

Egress Port which need do egress packet processing, active when oeteid is set to 1, not applicable to ENETC

netc_tb_is_oeteid_mode_t oETEID

Override ET_EID (Egress Treatment table entry, which specified egress packet processing actions)

netc_tb_is_ctd_mode_t ctd

Cut-Through Disable mode, valid if fa = 010b ~ 101b

uint32_t isqEID

Ingress Sequence Generation Entry ID, Valid when isqa is set to 1. 0xFFFF_FFFF is NULL. Not applicable to ENETC

uint32_t rpEID

Rate Policer Entry ID, Valid when orp =1. 0xFFFF_FFFF is NULL

uint32_t sgiEID

Stream Gate Instance Entry ID, Valid when osgi =1. 0xFFFF_FFFF is NULL

uint32_t ifmEID

Ingress Frame Modification Entry ID. 0xFFFF_FFFF is NULL

uint32_t etEID

Base Egress Treatment Entry ID for primary Egress Treatment group, Valid alid if fa = 010b ~ 101b. 0xFFFF_FFFF is NULL. Not applicable to ENETC

uint32_t iscEID

Ingress Stream counter Index. 0xFFFF_FFFF is NULL.

uint32_t ePortBitmap

Egress Port bitmap, identifies the ports to which the frame is to be forwarding or ET applicability port bitmap when oETEID = 10b. Not applicable to ENETC

uint32_t siMap

Station Interface Map, only valid for ENETC function when fa field is set to 10b

struct __netc_tb_is_req_data

#include <fsl_net.h> Ingress Stream table request data buffer.

Public Members

netc_tb_is_cfge_t cfge

Only perform for update or add command

struct __netc_tb_is_rsp_data

#include <fsl_net.h> Ingress Stream table request response data buffer.

Public Members

uint32_t entryID
Only perform for query command

netc_tb_is_cfge_t cfge
Only perform for query command

struct *_netc_tb_is_data*
#include <fsl_netc.h> Ingress Stream table data buffer.

struct *_netc_tb_is_config*
#include <fsl_netc.h> Ingress Stream table entry config.

struct *_netc_tb_isf_keye*
#include <fsl_netc.h> Ingress Stream Filter table key element.

Public Members

uint32_t isEID
Ingress Stream Entry ID

uint8_t pcp
Priority Code Point, Outer VLAN TAG PCP of the received frame

struct *_netc_tb_isf_cfge*
#include <fsl_netc.h> Ingress Stream Filter table config element.

Public Members

uint32_t ipv
Internal Priority Value, active when opiv is set to 1

uint32_t oipv
Override internal priority value

uint32_t dr
Drop Resilience, active when odr is set to 1

uint32_t odr
Overwrite DR

uint32_t imire
Ingress Mirroring Enable, not applicable to ENETC

uint32_t timecape
Timestamp Capture Enable, not applicable to ENETC

uint32_t osgi
Override stream gate instance entry id

uint32_t ctd
Cut-Through Disable, will disable cut-through for all destined ports when set 1, not applicable to ENETC

uint32_t orp
Override Rate Policer (instance) ID

netc_tc_sdu_type_t sduType
Service Data Unit Type to user for MSDU

uint32_t msdu

Maximum Service Data Unit

uint32_t rpEID

Rate Policer Entry ID, Valid when orp =1. 0xFFFF_FFFF is NULL

uint32_t sgiEID

Stream Gate Instance Entry ID, Valid when osgi =1. 0xFFFF_FFFF is NULL

uint32_t iscEID

Ingress Stream counter Index. 0xFFFF_FFFF is NULL.

struct __netc__tb_isf_req_data

#include <fsl_netc.h> Ingress Stream Filter table request data buffer.

Public Members

netc_tb_isf_cfge_t cfge

Only perform for update or add command

struct __netc__tb_isf_rsp_data

#include <fsl_netc.h> Ingress Stream Filter table request response data buffer.

Public Members

uint32_t entryID

Only perform for query command

netc_tb_isf_keye_t keye

Only perform for query command

netc_tb_isf_cfge_t cfge

Only perform for query command

struct __netc__tb_isf_data

#include <fsl_netc.h> Ingress Stream Filter table data buffer.

struct __netc__tb_isf_config

#include <fsl_netc.h> Ingress Stream Filter table entry config.

struct __netc__tb_rp_cfge

#include <fsl_netc.h> Rate Policer table config element.

Public Members

uint32_t cir

Committed information Rate

uint32_t cbs

Committed Burst Size

uint32_t eir

Excess information Rate

uint32_t ebs

Excess Burst Size

uint32_t mren

Mark All Frames Red Enable, Not valid when ndor=1

uint32_t doy

Drop on Yellow enable

uint32_t cm

Color mode, 0b = Color blind, 1b = Color aware

uint32_t cf

Coupling flag, enables coupling the Committed (C) bucket and Excess (E) bucket

uint32_t ndor

No drop on red

netc_tb_rp_sdu_type_t sduType

Service Data Unit Type

uint32_t ____pad0____

Reserved

struct __netc_tb_rp_fee

#include <fsl_net.h> Rate Policer table Function Enable element.

Public Members

uint8_t fen

Function Enable

uint8_t ____pad0____

Reserved

struct __netc_tb_rp_pse

#include <fsl_net.h> Rate Policer table Policer State element.

Public Members

uint8_t mr

Mark Red Flag

uint8_t res0

Reserved

struct __netc_tb_rp_stse

#include <fsl_net.h> Rate Policer table statistic element.

Public Members

uint32_t byteCount[2]

Number of bytes received by the rate policer instance

uint32_t dropFrames[2]

Number of frames dropped by the rate policer instance

uint32_t dr0GrnFrames[2]

Number of frames marked green with DR=0 by the rate policer instance

uint32_t dr1GrnFrames[2]

Number of frames marked green with DR=1 by the rate policer instance

uint32_t dr2GrnFrames[2]

Number of frames marked yellow with DR=2 by the rate policer instance

uint32_t remarkYlwFrames[2]

Number of frames re-marked from green to yellow by the rate policer instance

uint32_t dr3RedFrames[2]

Number of frames marked red with DR=3 by the rate policer instance

uint32_t remarkRedFrames[2]

Number of frames re-marked from green or yellow to red by the rate policer instance

uint32_t lts

Last timestamp

uint32_t bci

Committed token bucket contents, integer portion (31 bits)

uint32_t bcs

Committed token bucket sign bit (1 bit)

uint32_t bcf

Committed token bucket contents, fractional portion (31 bits)

uint32_t bei

Excess token bucket contents, integer portion (32 bits)

uint32_t bef

Excess token bucket contents, fractional portion (31 bits)

uint32_t bes

Committed token bucket sign bit

struct __netc__tb_rp_req_data

#include <fsl_netc.h> Rate Policer table request data buffer.

struct __netc__tb_rp_rsp_data

#include <fsl_netc.h> Rate Policer table request response data buffer.

Public Members

uint32_t entryID

Present only for commands which perform a query

netc__tb_rp_stse_t stse

Present only for commands which perform a query

struct __netc__tb_rp_data

#include <fsl_netc.h> Rate Policer table data buffer.

struct __netc__tb_rp_config

#include <fsl_netc.h> Rate Policer table entry config.

struct __netc__tb_isc_stse

#include <fsl_netc.h> Ingress Stream Count table statistic element.

Public Members

uint32_t rxCount

Receive Count

uint32_t msduDropCount

MSDU Drop Count

uint32_t policerDropCount

Policer Drop Count

uint32_t sgDropCount

Stream Gating Drop Count

struct __netc__tb__isc_req_data

#include <fsl_netc.h> Ingress Stream Count table request data buffer.

struct __netc__tb__isc_rsp_data

#include <fsl_netc.h> Ingress Stream Count table request response data buffer.

struct __netc__tb__isc_data

#include <fsl_netc.h> Ingress Stream Count table data buffer.

struct __netc__tb__sgi_cfge

#include <fsl_netc.h> Stream Gate Instance table config element.

Public Members

uint8_t oexen

Octets Exceeded (Gate Closed Due To Octets Exceeded function) Enable

uint8_t irxen

Invalid Receive (Gate Closed Due To Invalid Rx) Enable

netc__tb__sgi_sdu_type_t sduType

The type of PDU/SDU for Interval Octets Maximum check for Gate Entry

struct __netc__tb__sgi_acfge

#include <fsl_netc.h> Stream Gate Instance table Admin Configuration element.

Public Members

uint32_t adminSgclEID

Administrative Stream Gate Control List Entry ID, 0xFFFFFFFF is NULL

uint32_t adminBaseTime[2]

Admin Base Time

uint32_t adminCycleTimeExt

Admin Cycle Time Extension

struct __netc__tb__sgi_icfge

#include <fsl_netc.h> Stream Gate Instance table Initial Configuration element.

Public Members

uint8_t ipv

Internal Priority Value (IPV), Valid if oipv is 1

uint8_t oipv

Override frame IPV, otherwise the IPV value is determined by the stream gate control list entry

uint8_t gst

Specifies Gate State before the administrative stream gate control list takes affect, 0b = Closed; 1b = Open

uint8_t ctd

Specifies Cut Through disable status before the administrative stream gate control list takes affect , Not applicable to ENETC function

struct __netc__tb__sgi__sgise

#include <fsl_netc.h> Stream Gate Instance table stream gate instance state element.

Public Members

uint32_t operSgclEID

Operational Stream Gate Control List Entry ID

uint32_t configChangeTime[2]

Configuration Change Time

uint32_t operBaseTime[2]

Operational Base Time

uint32_t operCycleTimeExt

Oper Cycle Time Extension

uint32_t oex

Octets Exceeded Flag

uint32_t irx

Invalid Receive Flag

netc__tb__sgi__state_t state

Current Gate Instance State

struct __netc__tb__sgi__req__data

#include <fsl_netc.h> Stream Gate Instance table request data buffer.

struct __netc__tb__sgi__rsp__data

#include <fsl_netc.h> Stream Gate Instance table request response data buffer.

struct __netc__tb__sgi__data

#include <fsl_netc.h> Stream Gate Instance table data buffer.

struct __netc__tb__sgi__config

#include <fsl_netc.h> Stream Gate Instance table entry config.

struct __netc__sgcl__gate__entry

#include <fsl_netc.h> Defines the Stream Gate Control entry structure.

Public Members

uint32_t timeInterval

Time Interval for Gate Entry

uint32_t iom

Interval Octets Maximum for Gate Entry, specifies the maximum bytes (octets) allowed to pass (open), valid if iomen = 1

uint32_t ipv

Internal Priority Value for Gate Entry

uint32_t oipv

Override Internal Priority Value for Gate Entry

uint32_t ctd

Cut Through Disable for Gate Entry

uint32_t iomen

Interval Octet Maximum Enabled for Gate Entry, 0b = Don't track count, 1b = Track count

uint32_t gtst

Gate State for Gate Entry, 0b = Closed; 1b = Open

struct _netc_tb_sgcl_cfge

#include <fsl_netc.h> Stream Gate Control List table config element.

Public Members

uint32_t cycleTime

Cycle Time

uint8_t listLength

List Length

uint16_t extOipv

Extension (means the stream gate control list ends and before cycleTime restarts) Override Internal Priority Value

uint16_t extIpv

List Extension Internal Priority Value, valid if extOipv = 1

uint16_t extCtd

Extension Cut Through Disabled, 0b = No action, 1b = Disabled

uint16_t extGtst

Extension Gate State, 0b = closed, 1b = Open

struct _netc_tb_sgcl_sgclse

#include <fsl_netc.h> Stream Gate Control List table Stream Gate Control List State element.

Public Members

uint8_t refCount

Reference Count, 1 indicates that the gate control list is an administrative or an operational gate control list in a stream gate instance

struct _netc_tb_sgcl_req_data

#include <fsl_netc.h> Stream Gate Control List table request data buffer.

struct _netc_tb_sgcl_rsp_data

#include <fsl_netc.h> Stream Gate Control List table request response data buffer.

struct _netc_tb_sgcl_data

#include <fsl_netc.h> Stream Gate Control List table data buffer.

struct _netc_tb_sgcl_gcl

#include <fsl_netc.h> Stream Gate Control List table entry gate control list structure.

Public Members

uint16_t extOipv

Extension (means the stream gate control list ends and before cycleTime restarts) Override Internal Priority Value

```

uint16_t extIpv
    List Extension Internal Priority Value, valid if extOipv = 1
uint16_t extCtd
    Extension Cut Through Disabled, 0b = No action, 1b = Disabled
uint16_t extGtst
    Extension Gate State, 0b = closed, 1b = Open
uint32_t cycleTime
    Cycle Time
uint32_t numEntries
    Control List entry numbers
netc_sgcl_gate_entry_t *gcList
    Pointer to stream gate control list array
struct __netc_tb_fm_cfge
    #include <fsl_netc.h> Frame Modification table config element.

```

Public Members

```

netc_tb_fm_layer2_act_t l2Act
    Layer 2 Actions
netc_tb_fm_mac_header_act_t macHdrAct
    Layer 2 Header MAC Actions
netc_tb_fm_vlan_header_act_t vlanHdrAct
    Layer 2 VLAN Actions
netc_tb_fm_outer_vid_act_t outerVidAct
    Outer VID Actions
netc_tb_fm_sqt_act_t sqtAct
    Sequence Tag Action, Not applicable for ingress frame modifications
uint16_t smacPort
    Source MAC Address Register Port, valid if macHdrAct=010b,011b,100b
uint8_t dmac[6]
    Destination MAC Address, valid if macHdrAct = 011b,101b
uint32_t outerVlanID
    Outer VLAN VID, valid if outerVidAct = 01b
uint32_t outerVlanPcp
    Outer VLAN PCP, valid if outerPcpAct = 01b
uint32_t outerVlanDei
    Outer VLAN DEI, valid if outerDeiAct = 01b
netc_tb_fm_outer_tpid_act_t outerTpidAct
    Outer TPID action
netc_tb_fm_outer_pcp_act_t outerPcpAct
    Outer PCP action
netc_tb_fm_outer_dei_act_t outerDeiAct
    Outer DEI action

```

netc_tb_fm_payload_act_t pldAct

Payload Actions, Not applicable for ingress frame modifications

uint8_t pldOffset

Payload Offset, valid if outerPldAct = 010b

uint16_t fmdBytes

Frame Modification Bytes, valid if outerPldAct = 001b,010b or l2Act = 1b

uint32_t fmdEID

Frame Modification Data Entry ID, valid if outerPldAct = 001b,010b or l2Act = 1b.
0xFFFF is null pointer

struct __netc_tb_fm_req_data

#include <fsl_netc.h> Frame Modification table request data buffer.

struct __netc_tb_fm_rsp_data

#include <fsl_netc.h> Frame Modification table request response data buffer.

struct __netc_tb_fm_data

#include <fsl_netc.h> Frame Modification table data buffer.

struct __netc_tb_fm_config

#include <fsl_netc.h> Frame Modification table entry config.

struct __netc_tb_fmd_req_data

#include <fsl_netc.h> Frame Modification Data table request data buffer.

Public Members

uint8_t cfge[]

Configuration Element Data size is variable

struct __netc_tb_fmd_rsp_data

#include <fsl_netc.h> Frame Modification Data table request response data buffer.

Public Members

uint8_t cfge[]

Configuration Element Data size is variable

struct __netc_tb_fmd_data

#include <fsl_netc.h> Frame Modification Data table data buffer.

struct __netc_tb_fmd_update_config

#include <fsl_netc.h> Frame Modification data table entry update config.

Public Members

uint32_t res

Hold for request->commonHeader

uint8_t cfge[]

Configuration Element Data size is variable

struct __netc_tb_fmd_query_buffer

#include <fsl_netc.h> Frame Modification data table entry query data buffer.

Public Members

uint32_t entryID
EntryID of the queried entry

uint8_t cfge[]
Configuration Element Data size is variable

struct __netc__tb__vf__keye
#include <fsl_netc.h> Vlan Filter table key element.

struct __netc__tb__vf__cfge
#include <fsl_netc.h> Vlan Filter table config element.

Public Members

uint32_t portMembership
Port Membership Bitmap

uint32_t stgID
Spanning Tree Group Member ID

uint32_t fid
Filtering ID

uint32_t mlo
MAC Learning Options

uint32_t mfo
MAC Forwarding Options

uint32_t ipmfe
IP Multicast Filtering Enable

uint32_t ipmfle
IP Multicast Flooding Enable

uint32_t etaPortBitmap
Egress Treatment Applicability Port Bitmap for the secondary Egress Treatment group

uint32_t baseETEID
Base Egress Treatment Entry ID for the secondary Egress Treatment group

struct __netc__tb__vf__search__criteria
#include <fsl_netc.h> Vlan Filter table search criteria format.

Public Members

uint32_t resumeEntryId
Resume Entry ID, when starting a search, pass the NULL Entry ID.

struct __netc__tb__vf__req__data
#include <fsl_netc.h> Vlan Filter table request data buffer.

Public Members

netc__tb__vf__cfge__t cfge
Present only for update or add commands

struct __netc__tb__vf__rsp__data
#include <fsl_netc.h> Vlan Filter table request response data buffer.

Public Members

uint32_t status

Present only for query command with search access method

uint32_t entryID

Present only for query command

netc_tb_vf_keye_t keye

Present only for query command

netc_tb_vf_cfge_t cfge

Present only for query command

struct _netc_tb_vf_data

#include <fsl_netc.h> Vlan Filter table data buffer.

struct _netc_tb_vf_config

#include <fsl_netc.h> Vlan Filter table entry config.

struct _netc_tb_fdb_keye

#include <fsl_netc.h>

Public Members

uint8_t macAddr[6]

Destination MAC address of the frame for MAC forwarding lookups and the source MAC address of the frame for MAC learning lookups

uint32_t fid

Filtering ID, is obtained from an ingress lookup into the VLAN Filter table

struct _netc_tb_fdb_cfge

#include <fsl_netc.h> FDB table configuration element.

Public Members

uint32_t portBitmap

Forwarding destination Port Bitmap and ET applicability port bitmap when oETEID = 10b

netc_tb_fdb_oeteid_mode_t oETEID

Override ET_EID option

uint32_t ePort

Egress Ports, active when oETEid = 01b or ctd = 01b

uint32_t iMirE

Ingress Mirroring Enable

netc_tb_fdb_ctd_mode_t ctd

Cut-Through Disable

uint32_t dynamic

Static or Dynamic Entry, 0b = Static entry, 1b = Dynamic entry

uint32_t timeCapE

Timestamp Capture Enable when set

uint32_t etEID

Base egress treatment table entry id for primary Egress Treatment group, is valid if the oETEID field is set to value other than kNETC_FDBNoEPP. 0xFFFFFFFF is NULL.

struct __netc__tb_fdb_acte

#include <fsl_netc.h> FDB table Activity element.

Public Members

uint8_t actCnt

Activity Counter

uint8_t actFlag

Activity Flag

struct __netc__tb_fdb_search_criteria

#include <fsl_netc.h> FDB table search criteria format.

Public Members

uint32_t resumeEntryId

Resume Entry ID, pass the NULL Entry ID when starting a search

netc_tb_fdb_keye_t keye

Key Element data which used to match against the table entries

netc_tb_fdb_cfge_t cfge

Configuration Element data which used to match against the table entries

struct __netc__tb_fdb_req_data

#include <fsl_netc.h> FDB table request data buffer.

Public Members

netc_tb_common_header_t commonHeader

Define update actions (use *netc_tb_fdb_update_action_t*) and query actions

netc_tb_fdb_cfge_t cfge

Present only for commands which perform an update or add

struct __netc__tb_fdb_rsp_data

#include <fsl_netc.h> FDB table request response data buffer.

Public Members

uint32_t status

RESUME_ENTRY_ID, valid only in responses for commands which use the Search Access Method

uint32_t entryID

Present only for query command

netc_tb_fdb_keye_t keye

Present only for query command

netc_tb_fdb_cfge_t cfge

Present only for query command

netc_tb_fdb_acte_t acte

Present only for query command

struct __netc_tb_fdb_data

#include <fsl_netc.h> FDB table data buffer.

struct __netc_tb_fdb_config

#include <fsl_netc.h> FDB table entry config.

struct __netc_tb_l2mcf_keye

#include <fsl_netc.h> L2 IPV4 Multicast Filter table key element.

Public Members

netc_tb_l2mcf_key_type_t keyType

Key Type

uint32_t fid

Filtering ID

uint32_t ipv4DestAddr

IPv4 Destination Address

uint32_t ipv4SrcAddr

IPv4 Source Address

struct __etc_tb_l2mcf_search_criteria

#include <fsl_netc.h> L2 IPV4 Multicast Filter table search criteria format.

Public Members

uint32_t resumeEntryId

Resume Entry ID, pass the NULL Entry ID when starting a search

netc_tb_l2mcf_keye_t keye

Key Element data which used to match against the table entries

netc_tb_l2mcf_cfge_t cfge

Configuration Element data which used to match against the table entries

struct __netc_tb_l2mcf_req_data

#include <fsl_netc.h> L2 IPV4 Multicast Filter table request data buffer.

struct __netc_tb_l2mcf_rsp_data

#include <fsl_netc.h> L2 IPV4 Multicast Filter table request response data buffer.

struct __netc_tb_l2mcf_data

#include <fsl_netc.h> L2 IPV4 Multicast Filter table data buffer.

struct __netc_tb_l2mcf_config

#include <fsl_netc.h> L2 IPV4 Multicast Filter table entry config.

struct __netc_tb_iseqg_cfge

#include <fsl_netc.h> Ingress Sequence Generation table config element.

Public Members

netc_tb_iseqg_sqt *t* sqTag
Sequence Tag Type.

uint8_t __pad0__
Reserved.

struct __netc_tb_iseqg_sgse
#include <fsl_netc.h> Ingress Sequence Generation table Sequence generation state element.

Public Members

uint16_t sqgNum
Sequence Generation Number

struct __netc_tb_iseqg_req_data
#include <fsl_netc.h> Ingress Sequence Generation table request data buffer.

struct __netc_tb_iseqg_rsp_data
#include <fsl_netc.h> Ingress Sequence Generation table request response data buffer.

struct __netc_tb_iseqg_data
#include <fsl_netc.h> Ingress Sequence Generation table data buffer.

struct __netc_tb_iseqg_config
#include <fsl_netc.h> Ingress Sequence Generation table entry config.

struct __netc_tb_eseqr_cfge
#include <fsl_netc.h> Egress Sequence Recovery table config element.

Public Members

netc_tb_eseqr_sqt *t* sqTag
Sequence Tag, specify the expected sequence tag type in the frame

uint32_t sqrTnsq
Sequence Recovery Take No Sequence

uint32_t sqrAlg
Sequence Recovery Algorithm, 0b = Vector algorithm, 1b = Match algorithm

uint32_t sqrType
Sequence Recovery Function type, 0b = Sequence recovery function, 1b = Individual recovery function

uint32_t sqrHl
Sequence Recovery History Length, valid if *sqrAlg* = 0b

uint32_t sqrFwl
Sequence Recovery Future Window Length, valid if *sqrAlg* = 0b

uint32_t sqrTp
Sequence Timeout Period, the unit is 1.048576 milliseconds

struct __netc_tb_eseqr_stse
#include <fsl_netc.h> Egress Sequence Recovery table statistic element.

Public Members

uint32_t inOrderPackets[2]

In Order Packets

uint32_t outOfOrderPackets[2]

Out of Order Packets

uint32_t roguePackets[2]

Rogue Packets

uint32_t duplicatePackets[2]

Duplicate Packets

uint32_t lostPackets[2]

Lost Packets

uint32_t taglessPackets[2]

Tag-Less Packets

uint32_t esqrResetCounts

Sequence Recovery Resets

struct __netc__tb__eseqr__srse

#include <fsl_net.c.h> Egress Sequence Recovery table sequence recovery state element.

Public Members

uint32_t sqrNum

Sequence Recovery Number

uint32_t takeAny

Take Any

uint32_t lce

Lost Count Enable

uint32_t sqrTs

Sequence Recovery Timestamp

uint32_t sqrHistory[4]

Recovery History bit vector, each bit corresponding to sequence numbers, bit 1 means a packet with that sequence number has been previously received

struct __netc__tb__eseqr__req__data

#include <fsl_net.c.h> Egress Sequence Recovery table request data buffer.

struct __netc__tb__eseqr__rsp__data

#include <fsl_net.c.h> Egress Sequence Recovery table request response data buffer.

struct __netc__tb__eseqr__data

#include <fsl_net.c.h> Egress Sequence Recovery table data buffer.

struct __netc__tb__eseqr__config

#include <fsl_net.c.h> Egress Sequence Recovery table entry config.

struct __netc__tgs__gate__entry

#include <fsl_net.c.h> Defines the Time Gate Scheduling gate control entry structure.

Public Members

uint32_t interval
Entry Time Interval

struct _netc_tb_tgs_cfge
#include <fsl_netc.h> Time Gate Scheduling table config element.

Public Members

uint64_t adminBaseTime
Administrative Base Time

uint32_t adminCycleTime
Administrative Cycle Time

uint32_t adminCycleTimeExt
Administrative Cycle Time Extension

uint32_t adminControlListLength
Administrative Control List Length

netc_tgs_gate_entry_t adminGcl[]
Administrative Gate control list

struct _netc_tb_tgs_olse
#include <fsl_netc.h> Time Gate Scheduling table statistic element.

Public Members

uint64_t configChangeTime
The time at which this operational gate control list became active

uint64_t configChangeError
Count of error configuration changes

uint64_t operBaseTime
Operational Base Time

uint32_t operCycleTime
Operational Cycle Time

uint32_t operCycleTimeExt
Operational Cycle Time Extension

uint32_t operControlListLength
Operational Control List Length

netc_tgs_gate_entry_t operGcl[]
Operational Gate control list

struct _netc_tb_tgs_req_data
#include <fsl_netc.h> Time Gate Scheduling table request data buffer.

Public Members

netc_tb_tgs_cfge_t cfge
Present only for commands which perform a update

struct _netc_tb_tgs_rsp_data
#include <fsl_netc.h> Time Gate Scheduling table request response data buffer.

Public Members

uint32_t entryID

Present only for commands which perform a query

netc_tb_tgs_cfge_t cfge

Present only for commands which perform a query

netc_tb_tgs_olse_t olse

Present only for commands which perform a query

struct *_netc_tb_tgs_data*

#include <fsl_netc.h> Time Gate Scheduling table data buffer, set with max size.

struct *_netc_tb_tgs_gcl*

#include <fsl_netc.h> Time Gate Scheduling table entry gate control list structure.

Public Members

uint64_t baseTime

Base Time

uint32_t cycleTime

Cycle Time

uint32_t extTime

Cycle Time Extension

uint32_t numEntries

Control List entry numbers

netc_tgs_gate_entry_t *gcList

Pointer to time gate control list array

struct *_netc_tb_et_cfge*

#include <fsl_netc.h> Egress Treatment table config element.

Public Members

netc_tb_et_efm_mode_t efmMode

Egress Frame Modification mode

netc_tb_et_esq_act_t esqa

Egress Sequence Actions

netc_tb_et_ec_act_t eca

Egress Counter Action

uint8_t __pad1__

Reserve for data align

uint8_t efmLenChange

Egress Frame Modification Length Change, specified in units of bytes using a 2's complement notation

uint16_t efmDataLen

Egress Frame Modification Data Length

uint32_t efmEID

Egress Frame Modification Entry Id

uint32_t ecEID
Egress Count Table Entry ID

uint32_t esqaTgtEID
Egress Sequence Actions Target Entry ID, active when esqa = 10b

struct __netc__tb__et__req__data
#include <fsl_netc.h> Egress Treatment table request data buffer.

struct __netc__tb__et__rsp__data
#include <fsl_netc.h> Egress Treatment table request response data buffer.

struct __netc__tb__et__data
#include <fsl_netc.h> Egress Treatment table data buffer.

struct __netc__tb__et__config
#include <fsl_netc.h> Egress Treatment table entry config.

struct __netc__tb__etmcq__cfge
#include <fsl_netc.h> ETM Class Queue table config element.

Public Members

netc_hw_etm_class_queue_idx_t cq2cgMap
Class Queue to Congestion Group Mapping

struct __netc__tb__etmcq__stse
#include <fsl_netc.h> ETM Class Queue table statistic element.

Public Members

uint32_t rejByteCnt[2]
Reject Byte Count

uint32_t rejFrameCnt[2]
Reject Frame Count

uint32_t deqByteCnt[2]
Dequeue Byte Count

uint32_t deqFrameCnt[2]
Dequeue Frame Count

uint32_t dropByteCnt[2]
Dropped Frames, Memory Lost

uint32_t dropFrameCnt[2]
Dropped Frames, Memory Recovered

uint32_t frmCnt
Frame Count

struct __netc__tb__etmcq__req__data
#include <fsl_netc.h> ETM Class Queue table request data buffer.

struct __netc__tb__etmcq__rsp__data
#include <fsl_netc.h> ETM Class Queue table request response data buffer.

struct __netc__tb__etmcq__data
#include <fsl_netc.h> ETM Class Queue table data buffer.

struct __netc_tb_etmcq_config
#include <fsl_netc.h> ETM Class Queue table entry config.

Public Members

uint32_t entryID
Need use NETC_TB_ETM_CQ_ENTRY_ID macro to create entry ID

struct __netc_tb_etmcs_cfge
#include <fsl_netc.h> ETM Class Scheduler table config element.

Public Members

netc_tb_etmcs_ca_assg_t cqAssg
Class Queue Assignment, input 0 to 7 are weighted fair whereby input 8 to 15 are strict priority

uint32_t oal
Overread accounting length

struct __netc_tb_etmcs_cfge wbfsWeight[8]
Weight for scheduler input 0 ~ 7, effective weight is: $(2^x)/(1-(y/64))$

struct __netc_tb_etmcs_req_data
#include <fsl_netc.h> ETM Class Scheduler table request data buffer.

Public Members

netc_tb_etmcs_entry_id_t entryID
One class scheduler entry per port

struct __netc_tb_etmcs_rsp_data
#include <fsl_netc.h> ETM Class Scheduler table request response data buffer.

struct __netc_tb_etmcs_data
#include <fsl_netc.h> ETM Class Scheduler table data buffer.

struct __netc_tb_etmcs_config
#include <fsl_netc.h> ETM Class Scheduler table entry config.

Public Members

netc_tb_etmcs_entry_id_t entryID
One class scheduler entry per port

struct __netc_tb_etmcs_cfge
#include <fsl_netc.h> ETM Congestion Group table config element.

Public Members

uint16_t tdDr0En
Tail drop enable for DR0 Frame

uint16_t tdDr1En
Tail drop enable for DR1 Frame

uint16_t tdDr2En
Tail drop enable for DR2 Frame

uint16_t tdDr3En
Tail drop enable for DR3 Frame

uint16_t oal
Overhead accounting length, 2's complement value (range -2048 to +2047)

struct __netc__tb__etmcbg_cfge tdDRThresh[4]
Tail Drop Threshold ($TA * 2^{Tn}$) for DR0 ~ DR3 Frames, valid if tdDrnEn = 1b

struct __netc__tb__etmcbg_stse
#include <fsl_netcbg.h> ETM Congestion Group table statistic element.

Public Members

uint32_t byteCount[2]
Number of bytes currently in use in all class queues that are members of this group.

struct __netc__tb__etmcbg_req_data
#include <fsl_netcbg.h> ETM Congestion Group table request data buffer.

struct __netc__tb__etmcbg_rsp_data
#include <fsl_netcbg.h> ETM Congestion Group table request response data buffer.

struct __netc__tb__etmcbg_data
#include <fsl_netcbg.h> ETM Congestion Group table data buffer.

struct __netc__tb__etmcbg_config
#include <fsl_netcbg.h> ETM Congestion Group table entry config.

Public Members

uint32_t entryID
Need use NETC_TB_ETM_CG_ENTRY_ID macro to create entry ID

struct __netc__tb__ec_stse
#include <fsl_netcbg.h> Egress Count table statistic element.

Public Members

uint32_t enqFrmCnt[2]
Enqueued Frame Count

uint32_t rejFrmCnt[2]
Rejected Frame Count

struct __netc__tb__ec_req_data
#include <fsl_netcbg.h> Egress Count table request data buffer.

struct __netc__tb__ec_rsp_data
#include <fsl_netcbg.h> Egress Count table request response data buffer.

struct __netc__tb__ec_data
#include <fsl_netcbg.h> Egress Count table data buffer.

struct __netc__tb__bp_cfge
#include <fsl_netcbg.h> Buffer Pool table config element.

Public Members

bool sbpEn
Shared Buffer Pool Enable, set true means a shared buffer pool is associated with this buffer pool

netc_tb_bp_fc_cfg_t gcCfg
Flow Control (FC) Configuration

uint8_t pfcVector
Priority Flow Control (PFC) Vector; not support in NETC 3.0 and 3.1 version

uint16_t maxThresh
Maximum Threshold, value 0 means disable maximum threshold checking, use NETC_TB_BP_THRESH macro to set this value

uint16_t fcOnThresh
Flow Control On Threshold, If the buffer pool usage crosses this threshold, and if fcOnThresh is greater than fcOffThresh, the flow control state of the buffer pool is set to 1, use NETC_TB_BP_THRESH macro to set this value.

uint16_t fcOffThresh
Flow Control Off Threshold, If buffer pool usage drops to this threshold or below, the flow control state of the buffer pool is set to 0, , use NETC_TB_BP_THRESH macro to set this value

uint32_t sbpThresh
Shared Buffer Pool Threshold, use NETC_TB_BP_THRESH macro to set this value

uint32_t sbpEid
Shared Buffer Pool Entry ID, valid if sbpEn is true

uint32_t fcPorts
Flow Control Port bitmap, indicates which ports are to be flow controlled for this buffer pool

struct _netc_tb_bp_bpse
#include <fsl_netc.h> Buffer Pool table State Element Data.

Public Members

uint32_t amountUsed
Amount Used, number of internal memory words (average of 20 bytes each) currently in use in this buffer pool.

uint32_t amountUsedHWM
Amount Used High Watermark, value sticks at the highest AMOUNT_USED seen since the last watermark reset

uint32_t fcState
Flow Control (FC) State, ON (1) or OFF (0)

uint32_t bpd
Buffer Pool Disabled, 1 means the buffer pool has been disabled due to an uncorrectable ECC error

struct _netc_tb_bp_req_data
#include <fsl_netc.h> Buffer Pool table request data buffer.

struct _netc_tb_bp_rsp_data
#include <fsl_netc.h> Buffer Pool table request response data buffer.

struct __netc_tb_bp_data
 #include <fsl_netc.h> Buffer Pool table data buffer.

struct __netc_tb_bp_config
 #include <fsl_netc.h> Buffer Pool table entry config.

Public Members

uint32_t entryID
 Buffer pool ID, range in 0 ~ (SWT_GetBPTableEntryNum() - 1)
netc_tb_bp_cfge_t cfge
 Buffer Pool table config element

struct __netc_tb_sbp_cfge
 #include <fsl_netc.h> Shared Buffer Pool table config element.

Public Members

uint32_t maxThresh
 Maximum Threshold, If shared buffer pool usage is greater than or equal to this threshold, use NETC_TB_BP_THRESH macro to set this value

uint16_t fcOnThresh
 Flow Control On Threshold, If the shared buffer pool usage crosses this threshold, and if fcOnThresh is greater than fcOffThresh, the flow control state of the buffer pool is set to 1, use NETC_TB_BP_THRESH macro to set this value.

uint16_t fcOffThresh
 Flow Control Off Threshold, If shared buffer pool usage drops to this threshold or below, the flow control state of the buffer pool is set to 0, use NETC_TB_BP_THRESH macro to set this value

struct __netc_tb_sbp_sbpse
 #include <fsl_netc.h> Shared Buffer Pool table State Element Data.

Public Members

uint32_t amountUsed
 Amount Used, number of internal memory words (average of 20 bytes each) currently in use in this buffer pool.

uint32_t amountUsedHWM
 Amount Used High Watermark, value sticks at the highest AMOUNT_USED seen since the last watermark reset

uint32_t fcState
 Flow Control (FC) State, ON (1) or OFF (0)

struct __netc_tb_sbp_req_data
 #include <fsl_netc.h> Shared Buffer Pool table request data buffer.

struct __netc_tb_sbp_rsp_data
 #include <fsl_netc.h> Shared Buffer Pool table request response data buffer.

struct __netc_tb_sbp_data
 #include <fsl_netc.h> Shared Buffer Pool table data buffer.

struct __netc_tb_sbp_config
 #include <fsl_netc.h> Shared Buffer Pool table entry config.

Public Members

`uint32_t entryID`

Shared Buffer pool ID, range in 0 ~ (SWT_GetSBPTableEntryNum() - 1)

`netc_tb_sbp_cfge_t cfge`

Shared Buffer Pool table config element

`union __netc_tb_data_buffer`

`#include <fsl_netc.h>` Table common data buffer.

Public Members

`netc_tb_tgs_data_t tgs`

Time Gate Scheduling table data buffer

`netc_tb_rp_data_t rp`

Rate Policer table data buffer

`netc_tb_ipf_data_t ipf`

Ingress Port filter table data buffer

`netc_tb_fdb_data_t fdb`

FDB table data buffer

`netc_tb_l2mcf_data_t l2mcf`

L2 IPV4 Multicast Filter table data buffer

`netc_tb_vf_data_t vf`

VLAN Filter table data buffer

`netc_tb_isi_data_t isi`

Ingress Stream Identification table data buffer

`netc_tb_is_data_t is`

Ingress Stream table data buffer

`netc_tb_isf_data_t isf`

Ingress Stream Filter table data buffer

`netc_tb_isc_data_t isc`

Ingress Stream Count table data buffer

`netc_tb_sgi_data_t sgi`

Stream Gate Instance table data buffer

`netc_tb_sgcl_data_t sgcl`

Stream Gate Control List table data buffer

`netc_tb_fm_data_t fm`

Frame Modification table data buffer

`netc_tb_fmd_data_t fmd`

Frame Modification Data table data buffer

`netc_tb_et_data_t et`

Egress Treatment table data buffer

`netc_tb_ec_data_t ec`

Egress Count table data buffer

netc_tb_etmcq_data_t eq
ETM Class Queue table data buffer

netc_tb_etmcs_data_t cs
ETM Class Scheduler table data buffer

netc_tb_etmcg_data_t cg
ETM Class Group table data buffer

netc_tb_iseqg_data_t iseqg
Ingress Sequence Generation table data buffer

netc_tb_eseqr_data_t eseqr
Egress Sequence Recovery table data buffer

netc_tb_bp_data_t bp
Buffer Pool table data buffer

netc_tb_sbp_data_t sbp
Shared Buffer Pool table data buffer

struct *_netc_cbdr_hw*
#include <fsl_netc_hw.h> Register group for SI/Switch command bd ring.

Public Members

__IO uint32_t CBDRMR
Command BDR mode register.

__I uint32_t CBDRSR
Command BDR status register.

__IO uint32_t CBDRBAR0
Command BDR base address register 0

__IO uint32_t CBDRBAR1
Command BDR base address register 1

__IO uint32_t CBDRPIR
Command BDR producer index register

__IO uint32_t CBDRCIR
Command BDR consumer index register

__IO uint32_t CBDRLNR
Command BDR length register

struct *_netc_cbdr_handle*
#include <fsl_netc_hw.h> Handle for common part of EP/Switch NTMP.

Public Members

netc_cbdr_hw_t *base
Point to hardware command bd ring register group.

netc_cmd_bdr_t *cmdr
Point to command BD ring handle.

netc_tb_data_buffer_t *buffer
Point to table common data buffer.

struct req

Public Members

uint64_t addr
The request and response data buffers address

struct ____unnamed91____

Public Members

uint32_t resLength
The length of the Response Data Buffer

uint32_t reqLength
The length of the Request Data Buffer

struct ____unnamed93____

Public Members

netc_tb_cmd_t cmd
Access table entry command, see *netc_tb_cmd_t*.

netc_tb_access_mode_t accessType
Access table entry method, see *netc_tb_access_mode_t*.

uint32_t ____pad1____
RSS Hash high field value.

uint32_t version
Protocol Version.

uint32_t enCompInt
Command Completion Interrupt.

uint32_t resReady
Response Ready.

struct ____unnamed95____

Public Members

uint32_t npf
NTMP Protocol Format.

struct resp

struct ____unnamed97____

Public Members

uint32_t numMatched
Number of Entries Matched.

uint32_t error
Error status.

uint32_t resReady
Response Ready.

struct generic

Public Members

uint64_t addr

Data.

uint32_t en

Enable entry.

uint32_t siBitMap

Station interfaces 15-0 for which this filter applies.

uint32_t index

The index refers to an entry location within a table.

uint32_t length

NA

uint32_t cmd

Command.

uint32_t ____pad2____

< Class of command.

uint32_t status

Status.

uint32_t ci

Completion interrupt.

uint32_t sf

Short format.

struct frameAttr

Public Members

uint16_t swtPortMas

Switch port masquerading, applicable only if the incoming port is designated as a switch management port

uint16_t ethernet

Ethernet type Present

uint16_t outerVlan

Outer VLAN Present

uint16_t innerVlan

Inner VLAN Present

netc_tb_ipf_seq_tag_t seqTag

Sequence Tag Code

uint16_t ipHeader

IP Header Present

uint16_t ipVersion

0b = IPv4, 1b = IPv6

uint16_t ipExt

IPv4 option / IPv6 extension present

```
netc_tb_ipf_l4_header_t l4Header
    L4 Header code
uint16_t wakeOnLan
    Wake-on-LAN Magic Packet Present
struct payload
```

Public Members

```
uint8_t data
    Payload Byte n
uint8_t mask
    Payload Byte n Mask
union ____unnamed101____
```

Public Members

```
netc_tb_ipf_keye_t keye
uint32_t entryID
uint32_t sCriteria
union ____unnamed103____
```

Public Members

```
netc_tb_ipf_req_data_t request
netc_tb_ipf_rsp_data_t response
union ____unnamed105____
```

Public Members

```
uint32_t entryID
uint32_t sCriteria
netc_tb_isi_keye_t keye
union ____unnamed107____
```

Public Members

```
netc_tb_isi_req_data_t request
netc_tb_isi_rsp_data_t response
union ____unnamed109____
```

Public Members

netc_tb_is_req_data_t request

netc_tb_is_rsp_data_t response

union ____unnamed111____

Public Members

uint32_t entryID

uint32_t sCriteria

netc_tb_isf_keye_t keye

union ____unnamed113____

Public Members

netc_tb_isf_req_data_t request

netc_tb_isf_rsp_data_t response

union ____unnamed115____

Public Members

netc_tb_rp_cfge_t cfge

Present only for commands which perform an update or add

struct __netc_tb_rp_req_data

struct ____unnamed117____

Public Members

netc_tb_rp_fee_t fee

Present only for commands which perform an update or add

union ____unnamed119____

Public Members

netc_tb_rp_cfge_t cfge

Present only for commands which perform a query

struct __netc_tb_rp_rsp_data

struct __netc_tb_rp_rsp_data

struct ____unnamed121____

Public Members*netc_tb_rp_fee_t* fee

Present only for commands which perform a query

struct ____unnamed123____

Public Members*netc_tb_rp_pse_t* pse

Present only for commands which perform a query

union ____unnamed125____

Public Members*netc_tb_rp_req_data_t* request*netc_tb_rp_rsp_data_t* response

union ____unnamed127____

Public Members*netc_tb_isc_req_data_t* request*netc_tb_isc_rsp_data_t* response

union ____unnamed129____

Public Members*netc_tb_sgi_sgise_t* sgise

struct __netc__tb__sgi__rsp__data

struct ____unnamed131____

union ____unnamed133____

Public Members*netc_tb_sgi_req_data_t* request*netc_tb_sgi_rsp_data_t* response

union ____unnamed135____

Public Members*netc_tb_sgcl_req_data_t* request*netc_tb_sgcl_rsp_data_t* response

struct __netc__tb__sgcl__data

struct ____unnamed137____

union ____unnamed139____

Public Members

netc_tb_fm_req_data_t request

netc_tb_fm_rsp_data_t response

union __unnamed141__

Public Members

netc_tb_fmd_req_data_t request

netc_tb_fmd_rsp_data_t response

union __unnamed143__

Public Members

uint32_t entryID

netc_tb_vf_search_criteria_t sCriteria

Active when access method is kNETC_Search

netc_tb_vf_keye_t keye

union __unnamed145__

Public Members

netc_tb_vf_req_data_t request

netc_tb_vf_rsp_data_t response

struct __unnamed147__

Public Members

netc_tb_fdb_acte_t acte

Activity Element data which used to match against the table entries

netc_tb_fdb_sc_keye_mc_t keyeMc

Key Element data match criteria

netc_tb_fdb_sc_cfge_mc_t cfgeMc

Configuration Element data match criteria

netc_tb_fdb_sc_acte_mc_t acteMc

Activity Element data match criteria

union __unnamed149__

Public Members

uint32_t entryID

Active when access method is kNETC_EntryIDMatch

netc_tb_fdb_keye_t keye

Active when access method is kNETC_ExactKeyMatch

netc_tb_fdb_search_criteria_t sCriteria

Active when access method is kNETC_Search

union ____unnamed151____

Public Members

netc_tb_fdb_req_data_t request

netc_tb_fdb_rsp_data_t response

struct ____unnamed153____

Public Members

netc_tb_l2mcf_acte_t acte

Activity Element data which used to match against the table entries

etc_tb_l2mcf_sc_keye_mc_t keyeMc

Key Element data match criteria

etc_tb_l2mcf_sc_cfge_mc_t cfgeMc

Configuration Element data match criteria

etc_tb_l2mcf_sc_acte_mc_t acteMc

Activity Element data match criteria

union ____unnamed155____

Public Members

uint32_t entryID

netc_tb_l2mcf_search_criteria_t sCriteria

netc_tb_l2mcf_keye_t keye

union ____unnamed157____

Public Members

netc_tb_l2mcf_req_data_t request

netc_tb_l2mcf_rsp_data_t response

union ____unnamed159____

Public Members

netc_tb_iseqg_req_data_t request

netc_tb_iseqg_rsp_data_t response

union ____unnamed161____

Public Members

netc_tb_eseqr_req_data_t request

netc_tb_eseqr_rsp_data_t response

union ____unnamed163____

Public Members

struct __netc_tgs_gate_entry

uint32_t gate

Entry Gate Mask

struct ____unnamed165____

Public Members

uint32_t tcGateState

Traffic Class Gate States for Gate Entry, 8 bits for 8 Traffic Class , 0b means Gate closed, 1b means Gate open

netc_tb_tgs_gate_type_t operType

Gate operation type (IEEE 802.1Q-2018) field for gate control list entry i

union ____unnamed167____

Public Members

netc_tb_tgs_req_data_t request

netc_tb_tgs_rsp_data_t response

struct __netc_tb_tgs_data

struct ____unnamed169____

union ____unnamed171____

Public Members

netc_tb_et_req_data_t request

netc_tb_et_rsp_data_t response

union ____unnamed173____

Public Members

netc_tb_etmcq_req_data_t request

netc_tb_etmcq_rsp_data_t response

struct wbfWeight

Public Members

uint8_t xCode
Weight code x value

uint8_t yCode
Weight code y value

union __unnamed176__

Public Members

netc_tb_etmcs_req_data_t request

netc_tb_etmcs_rsp_data_t response

struct tdDRThresh

Public Members

uint16_t tn
TA

uint16_t ta
Tn

union __unnamed179__

Public Members

netc_tb_etmcg_req_data_t request

netc_tb_etmcg_rsp_data_t response

union __unnamed181__

Public Members

netc_tb_ec_req_data_t request

netc_tb_ec_rsp_data_t response

union __unnamed183__

Public Members

netc_tb_bp_req_data_t request

netc_tb_bp_rsp_data_t response

union __unnamed185__

Public Members

netc_tb_sbp_req_data_t request

netc_tb_sbp_rsp_data_t response

2.80 NETC MDIO Driver

2.81 MDIO initialization module

```
enum _netc_mdio_type
    Enumeration for the MAC port MDIO type.
    Values:
    enumerator kNETC_EMdio
        Bound handle to EMDIO access, submodule of NETC.
    enumerator kNETC_InternalMdio
        Bound handle to MAC port internal MDIO access, submodule of EP/Switch.
    enumerator kNETC_ExternalMdio
        Bound handle to MAC port external MDIO access, submodule of EP/Switch.
typedef enum _netc_mdio_type netc_mdio_type_t
    Enumeration for the MAC port MDIO type.
typedef struct _netc_mdio netc_mdio_t
    Structure to choose MDIO entity(Internal/external MDIO for specified EP/Switch port)
typedef struct _netc_mdio_handle netc_mdio_handle_t
    MDIO handle.
typedef struct _netc_mdio_config netc_mdio_config_t
    MDIO configuration structure.
status_t NETC_MDIOInit(netc_mdio_handle_t *handle, netc_mdio_config_t *config)
    Initialize the MDIO.
```

Note: The EMDIO can be used independently. The port internal/external MDIO is a part of EP/Switch, should be initialized and used after EP/Switch is enabled.

Parameters

- handle – MDIO handle.
- config – MDIO configuration.

Returns

status_t

```
struct _netc_mdio
    #include <fsl_netc_mdio.h> Structure to choose MDIO entity(Internal/external MDIO for
    specified EP/Switch port)
```

Public Members

```
netc_mdio_type_t type
    Internal or external MAC port MDIO.
netc_hw_eth_port_idx_t port
    MDIO port index, only meaningful when port MDIO type is used.
```

```
struct _netc_mdio_handle
    #include <fsl_netc_mdio.h> MDIO handle.
```

Public Members

netc_mdio_t mdio
MDIO identifier.

struct *_netc_mdio_config*
#include <fsl_netc_mdio.h> MDIO configuration structure.

Public Members

netc_mdio_t mdio
MDIO identifier.

uint32_t srcClockHz
MDIO reference clock for MDC frequency calculation.

bool isNegativeDriven
MDIO driven at positive(false)/negative(true) of MDC edge.

bool isPreambleDisable
Enable/Disable generation of MDIO preamble.

2.82 MDIO PHY status module

typedef struct *_netc_mdio_phy_status* *netc_mdio_phy_status_t*
PHY auto status check configuration structure.

status_t NETC_MDIOSetPhyStatusCheck(*netc_mdio_handle_t* *handle, *netc_mdio_phy_status_t* *config)

Setup the mechanism to check PHY status automatically This is a hardware mechanism to read specified PHY register in a configured time interval instead of polling the PHY in software.

Parameters

- handle – MDIO handle.
- config – The configuration of the PHY status automatical check.

Returns

status_t

void NETC_MDIOPhyStatusGetFlags(*netc_mdio_handle_t* *handle, uint16_t *low2HighMask, uint16_t *high2LowMask)

Get the PHY register bit status transition interrupt flag(s).

Parameters

- handle – MDIO handle.
- low2HighMask – The interrupt flag of a 0->1 transition on a corresponding bit of PHY register.
- high2LowMask – The interrupt flag of a 1->0 transition on a corresponding bit of PHY register.

void NETC_MDIOPhyStatusClearFlags(*netc_mdio_handle_t* *handle, uint16_t low2HighMask, uint16_t high2LowMask)

Clear the PHY register bit status transition interrupt flag(s).

Parameters

- handle – MDIO handle.
- low2HighMask – Clear the interrupt flag of a 0->1 transition on a corresponding bit of PHY register.
- high2LowMask – Clear the interrupt flag of a 1->0 transition on a corresponding bit of PHY register.

struct `_netc_mdio_phy_status`

#include <fsl_netcdm.h> PHY auto status check configuration structure.

Public Members

uint16_t interval

PHY status read interval in units of 1-2 ms. A value of 0 indicates disable.

bool isC45Used

PHY status read with Clause 22/45 MDIO access.

uint8_t phyOrPortAddr

MDIO PHY address(Clause 22) / port address(Clause 45).

uint8_t regiOrDevAddr

MDIO register address(Clause 22) / device address(Clause 45).

uint16_t c45RegiAddr

MDIO register address(Clause 45).

uint16_t enableIntrHigh2Low

Bit high-to-low event interrupt enable.

uint16_t enableIntrLow2High

Bit low-to-high event interrupt enable.

2.83 MDIO write/read module

status_t NETC_MDIOWrite(*netc_mdio_handle_t* *handle, uint8_t phyAddr, uint8_t regAddr, uint16_t data)

IEEE802.3 Clause 22 MDIO write data.

Parameters

- handle – MDIO handle.
- phyAddr – The PHY address.
- regAddr – The PHY register address.
- data – The data written to PHY.

Returns

status_t

status_t NETC_MDIORead(*netc_mdio_handle_t* *handle, uint8_t phyAddr, uint8_t regAddr, uint16_t *pData)

IEEE802.3 Clause 22 MDIO read data.

Parameters

- handle – MDIO handle.
- phyAddr – The PHY address.

- regAddr – The PHY register address.
- pData – The received data from PHY.

Returns

status_t

status_t NETC_MDIOC45Write(*netc_mdio_handle_t* *handle, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr, uint16_t data)

IEEE802.3 Clause 45 MDIO write data.

Parameters

- handle – MDIO handle.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.
- data – The data written to PHY.

Returns

status_t

status_t NETC_MDIOC45Read(*netc_mdio_handle_t* *handle, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr, uint16_t *pData)

IEEE802.3 Clause 45 MDIO read data.

Parameters

- handle – MDIO handle.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.
- pData – The received data from PHY.

Returns

status_t

2.84 NETC Timer Driver

2.85 Timer adjustment module

void NETC_TimerGetTime(ENETC_PF_TMR_Type *base, uint64_t *nanosecond)

Get the timer's current or default time.

Parameters

- base – NETC timer base address.
- nanosecond – Time in nanosecond.

void NETC_TimerGetCurrentTime(*netc_timer_handle_t* *handle, uint64_t *nanosecond)

Get the timer's current time.

Parameters

- handle – NETC timer handle.
- nanosecond – Time in nanosecond.

```
void NETC_TimerGetFreeRunningTime(netc_timer_handle_t *handle, uint64_t *nanosecond)
```

Get the timer's freerunning time.

Parameters

- handle – NETC timer handle.
- nanosecond – Time in nanosecond.

```
void NETC_TimerAddOffset(netc_timer_handle_t *handle, int64_t nanosecond)
```

Correct the current timer.

Note: Need stop all mechanisms based on 1588 timers during call this API. To take time gate scheduling as example, if the port is enabled with TGS and it contains an operational list, user need to call `SWT_TxPortTGSEnable()/SWT_TxPortTGSEnable()` to disable the port time gate before call this API, and re-enable and configure the port TGS after the execution of this API.

Parameters

- handle – NETC timer handle.
- nanosecond – Time in nanosecond.

```
void NETC_TimerAdjustFreq(netc_timer_handle_t *handle, int32_t ppb)
```

Adjust the timer frequency.

Parameters

- handle – NETC timer handle.
- ppb – Parts per billion.

```
void NETC_TimerGetFrtSrtTime(netc_timer_handle_t *handle, uint64_t *frt, uint64_t *srt)
```

Get the free running and synchronized current time with an atomic read.

Parameters

- handle – NETC timer handle.
- frt – The free-running time in nanosecond.
- srt – The synchronized current time in nanosecond.

2.86 Timer initialization module

```
enum _netc_timer_irq_flags
```

Timer interrupt flags.

Values:

```
enumerator kNETC_TimerFiper1IrqFlag
```

Periodic pulse 1 interrupt.

```
enumerator kNETC_TimerFiper2IrqFlag
```

Periodic pulse 2 interrupt.

```
enumerator kNETC_TimerFiper3IrqFlag
```

Periodic pulse 3 interrupt.

```
enumerator kNETC_TimerAlarm1IrqFlag
```

Alarm 1 interrupt.

enumerator kNETC_TimerAlarm2IrqFlag

Alarm 2 interrupt.

enumerator kNETC_TimerExtTrig1ThresholdIrqFlag

External trigger 1 timestamp FIFO threshold hit interrupt.

enumerator kNETC_TimerExtTrig2ThresholdIrqFlag

External trigger 2 timestamp FIFO threshold hit interrupt.

enumerator kNETC_TimerExtTrig1TsAvailIrqFlag

External trigger 1 new timestamp available interrupt.

enumerator kNETC_TimerExtTrig2TsAvailIrqFlag

External trigger 2 new timestamp available interrupt.

enumerator kNETC_TimerExtTrig1OverflowIrqFlag

External trigger 1 timestamp FIFO overflow interrupt.

enumerator kNETC_TimerExtTrig2OverflowIrqFlag

External trigger 2 timestamp FIFO overflow interrupt.

typedef enum *_netc_timer_irq_flags* netc_timer_irq_flags_t

Timer interrupt flags.

typedef struct *_netc_timer_handle* netc_timer_handle_t

typedef struct *_netc_timer_config* netc_timer_config_t

Structure to configure timer.

status_t NETC_TimerInit(*netc_timer_handle_t* *handle, const *netc_timer_config_t* *config)

Initialize the NETC PTP1588 timer.

Parameters

- handle – NETC timer handle.
- config – The configuration of the timer.

Returns

status_t

void NETC_TimerDeinit(*netc_timer_handle_t* *handle)

Deinitialize the NETC PTP1588 timer.

Parameters

- handle – NETC timer handle.

void NETC_TimerInitHandle(*netc_timer_handle_t* *handle)

Initialize a NETC PTP1588 timer handle.

Parameters

- handle – NETC timer handle.

void NETC_TimerEnable(*netc_timer_handle_t* *handle, bool enable)

Enable/Disable the NETC PTP1588 timer.

Parameters

- handle – NETC timer handle.
- enable – Whether enable the PTP1588 timer.

struct *_netc_timer_handle*

#include <fsl_netc_timer.h> Timer handler structure.

Public Members

netc_timer_hw_t hw

Hardware register map resource.

uint32_t timerFreq

Timer clock frequency(Hz).

uint8_t entryNum

MSIX entry number.

struct __netc_timer_config

#include <fsl_netc_timer.h> Structure to configure timer.

Public Members

bool clkOutputPhase

True: Inverted divided clock is output, False: Non-inverted divided clock is output.

bool clkInputPhase

True: Inverted frequency tuned timer input clock, False: Non-inverted frequency tuned timer input clock.

bool enableTimer

True: Enable 1588 timer, False: Disable 1588 timer, use default counter.

netc_timer_ref_clk_t clockSelect

Timer reference clock.

uint32_t refClkHz

Timer reference clock frequency in Hz.

int32_t defaultPpb

Default ppb.

netc_msix_entry_t *msixEntry

MSIX table entry array.

uint8_t entryNum

MSIX entry number.

2.87 Local time synchronization module

enum __netc_timer_alarm_index

Enumeration for NETC timer alarm index.

Values:

enumerator kNETC_TimerAlarm1

enumerator kNETC_TimerAlarm2

enum __netc_timer_fiper_index

Enumeration for NETC timer FIPER index.

Values:

enumerator kNETC_TimerFiper1

enumerator kNETC_TimerFiper2

enumerator kNETC_TimerFiper3

typedef enum *_netc_timer_alarm_index* netc_timer_alarm_index_t

Enumeration for NETC timer alarm index.

typedef struct *_netc_timer_alarm_t* netc_timer_alarm_t

Structure to configure timer alarm.

typedef enum *_netc_timer_fiper_index* netc_timer_fiper_index_t

Enumeration for NETC timer FIPER index.

typedef struct *_netc_timer_fiper_config* netc_timer_fiper_config_t

Structure to configure timer FIPER.

typedef struct *_netc_timer_fiper* netc_timer_fiper_t

Structure to set and start timer FIPER.

typedef struct *_netc_timer_ext_pulse_trig* netc_timer_ext_trig_t

Structure to configure external pulse trigger timestamp.

void NETC_TimerConfigureAlarm(*netc_timer_handle_t* *handle, *netc_timer_alarm_index_t* alarmId, const *netc_timer_alarm_t* *alarm)

Configure the timer alarm feature.

Parameters

- handle – NETC timer handle.
- alarmId – The alarm index.
- alarm – The timer alarm configuration structure.

void NETC_TimerStartAlarm(*netc_timer_handle_t* *handle, *netc_timer_alarm_index_t* alarmId, uint64_t nanosecond)

Start the alarm with specified time after alarm feature is configured This function can generate a pulse on a GPIO and/or an interrupt at specified future time. It also can trigger FIPER at specified time.

Parameters

- handle – NETC timer handle.
- alarmId – The alarm index.
- nanosecond – The time in nanosecond to generate alarm pulse.

void NETC_TimerStopAlarm(*netc_timer_handle_t* *handle, *netc_timer_alarm_index_t* alarmId)

Stop the alarm before/after it's fired This function can deactivate alarm.

Parameters

- handle – NETC timer handle.
- alarmId – The alarm index.

void NETC_TimerConfigureFIPER(*netc_timer_handle_t* *handle, const *netc_timer_fiper_config_t* *config)

Configure the timer FIPER feature.

Parameters

- handle – NETC timer handle.
- config – The timer FIPER configuration structure.


```
void NETC_TimerStartFIPER(netc_timer_handle_t *handle, netc_timer_fiper_index_t fiperId,  
                        const netc_timer_fiper_t *fiper)
```

Start the timer FIPER to generate pulse This function can generate a periodic(Fixed Period-FIPER) pulse on a GPIO pin and/or an interrupt to the host.

Parameters

- handle – NETC timer handle.
- fiperId – The timer FIPER index.
- fiper – The timer FIPER configuration structure.

```
void NETC_TimerStopFIPER(netc_timer_handle_t *handle, netc_timer_fiper_index_t fiperId)
```

Stop the timer FIPER to generate pulse.

Parameters

- handle – NETC timer handle.
- fiperId – The timer FIPER index.

```
void NETC_TimerConfigureExtPulseTrig(netc_timer_handle_t *handle,  
                                    netc_timer_exttrig_index_t extTrigId, const  
                                    netc_timer_ext_trig_t *extTrig)
```

Configure the external pulse trigger timestamp capture.

Parameters

- handle – NETC timer handle.
- extTrigId – The timer FIPER index.
- extTrig – The external pulse trigger configuration structure.

```
status_t NETC_TimerSetTsFifoThreshold(netc_timer_handle_t *handle, uint8_t threshold)
```

Set timestamp FIFO threshold of the external pulse trigger timestamp capture.

Parameters

- handle – NETC timer handle.
- threshold – Timestamp FIFO threshold.

Returns

status_t

```
status_t NETC_TimerReadExtPulseCaptureTime(netc_timer_handle_t *handle,  
                                           netc_timer_exttrig_index_t extTrigId, uint64_t  
                                           *nanosecond)
```

Read the timestamp captured by external pulse trigger in FIFO.

Parameters

- handle – NETC timer handle.
- extTrigId – The timer FIPER index.
- nanosecond – Timestamp in nanosecond.

Returns

status_t

```
static inline void NETC_TimerClearInterruptStatus(netc_timer_handle_t *handle, uint32_t flags)
```

Clear timer interrupt flags.

Parameters

- handle – NETC timer handle.

- flags – Timer interrupt flags. This is a logical OR of enumeration :: `netc_timer_irq_flags_t`.

`status_t` NETC_TimerMsixSetGlobalMask(*netc_timer_handle_t* *handle, bool mask)

Set the global MSIX mask status.

This function masks/unmasks global MSIX message. Mask - All of the vectors are masked, regardless of their per-entry mask bit states. Unmask - Each entry's mask status determines whether the vector is masked or not.

Parameters

- handle – The timer handle
- mask – The mask state. True: Mask, False: Unmask.

Returns

`status_t`

`status_t` NETC_TimerMsixSetEntryMask(*netc_timer_handle_t* *handle, uint8_t entryIdx, bool mask)

Set the MSIX entry mask status for specified entry.

This function masks/unmasks MSIX message for specified entry.

Parameters

- handle – NETC timer handle.
- entryIdx – The entry index in the table.
- mask – The mask state. True: Mask, False: Unmask.

Returns

`status_t`

`status_t` NETC_TimerMsixGetPendingStatus(*netc_timer_handle_t* *handle, uint8_t pbaIdx, uint64_t *status)

Get the MSIX pending status in MSIX PBA table.

This function is to get the entry pending status from MSIX PBA table. If interrupt occurs but masked by vector control of entry, pending bit in PBA will be set.

Parameters

- handle – NETC timer handle.
- pbaIdx – The index of PBA array with 64-bit unit.
- status – Pending status bit mask, bit n for entry n.

Returns

`status_t`

`struct __netc_timer_alarm_t`

`#include <fsl_netc_timer.h>` Structure to configure timer alarm.

Public Members

bool enableInterrupt

Enable/Disable ALARM interrupt enable.

bool polarity

True: Active low output, False: Active high output.

bool pulseGenSync

True: ALARM output asserted synchronous to timer generated clock, False: ALARM output asserted immediately.

uint8_t pulseWidth

Pulse width in number of timer generated clocks the alarm will be active for.

struct __netc_timer_fiper_config

#include <fsl_netc_timer.h> Structure to configure timer FIPER.

Public Members

bool startCondition

True: FIPER is enabled through timer enable and alarm getting set, False: FIPER is enabled through timer enable.

bool fiper1Loopback

True: FIPER1 pulse is looped back into Trigger1 input, False: Trigger1 input is based upon normal external trigger input.

bool fiper2Loopback

True: FIPER2 pulse is looped back into Trigger2 input, False: Trigger2 input is based upon normal external trigger input.

uint16_t prescale

Output FIPER pulse clock is generated by dividing the timer input clock by this number. Must be an even value.

struct __netc_timer_fiper

#include <fsl_netc_timer.h> Structure to set and start timer FIPER.

Public Members

bool enableInterrupt

Enable/Disable FIPER interrupt interrupt.

bool pulseGenSync

True: FIPER output asserted synchronous to timer generated clock, False: FIPER output asserted immediately.

uint8_t pulseWidth

FIPER pulse width.

uint32_t pulsePeriod

Interval of FIPER pulses.

struct __netc_timer_ext_pulse_trig

#include <fsl_netc_timer.h> Structure to configure external pulse trigger timestamp.

Public Members

bool polarity

Time stamp on the falling(true)/rising(false) edge of the external trigger.

bool enableFifoOverflowInterrupt

Enable/Disable FIFO Overflow interrupt.

`bool enableFifoThresholdHitInterrupt`
Enable/Disable FIFO Threshold Hit interrupt.

`bool enableTsAvailInterrupt`
Enable/Disable timestamp capture interrupt.

2.88 OTFAD: On The Fly AES-128 Decryption Driver

`void OTFAD_GetDefaultConfig(otfad_config_t *config)`
OTFAD module initialization function.

Parameters

- `config` – OTFAD configuration.

`void OTFAD_Init(OTFAD_Type *base, const otfad_config_t *config)`
OTFAD module initialization function.

Parameters

- `base` – OTFAD base address.
- `config` – OTFAD configuration.

`void OTFAD_Deinit(OTFAD_Type *base)`
Deinitializes the OTFAD.

`static inline uint32_t OTFAD_GetOperateMode(OTFAD_Type *base)`
OTFAD module get operate mode.

Parameters

- `base` – OTFAD base address.

`static inline uint32_t OTFAD_GetStatus(OTFAD_Type *base)`
OTFAD module get status.

Parameters

- `base` – OTFAD base address.

`status_t OTFAD_SetEncryptionConfig(OTFAD_Type *base, const otfad_encryption_config_t *config)`
OTFAD module set encryption configuration.

Note: if enable keyblob process, the first 256 bytes external memory is use for keyblob data, so this region shouldn't be in OTFAD region.

Parameters

- `base` – OTFAD base address.
- `config` – encryption configuration.

`status_t OTFAD_GetEncryptionConfig(OTFAD_Type *base, otfad_encryption_config_t *config)`
OTFAD module get encryption configuration.

Note: if enable keyblob process, the first 256 bytes external memory is use for keyblob data, so this region shouldn't be in OTFAD region.

Parameters

- base – OTFAD base address.
- config – encryption configuration.

status_t OTFAD_HitDetermination(OTFAD_Type *base, uint32_t address, uint8_t *contextIndex)
OTFAD module do hit determination.

Parameters

- base – OTFAD base address.
- address – the physical address space assigned to the QuadSPI(FlexSPI) module.
- contextIndex – hit context region index.

Returns

status, such as kStatus_Success or kStatus_OTFAD_ResRegAccessMode.

FSL_OTFAD_DRIVER_VERSION

Driver version.

Status codes for the OTFAD driver.

Values:

enumerator kStatus_OTFAD_ResRegAccessMode
Restricted register mode

enumerator kStatus_OTFAD_AddressError
End address less than start address

enumerator kStatus_OTFAD_RegionOverlap
the OTFAD does not support any form of memory region overlap, for system accesses that hit in multiple contexts or no contexts, the fetched data is simply bypassed

enumerator kStatus_OTFAD_RegionMiss
For accesses that hit in a single context, but not the selected one

OTFAD context type.

Values:

enumerator kOTFAD_Context_0
context 0

enumerator kOTFAD_Context_1
context 1

enumerator kOTFAD_Context_2
context 2

enumerator kOTFAD_Context_3
context 3

OTFAD operate mode.

Values:

enumerator kOTFAD_NRM
Normal Mode

```
enumerator kOTFAD_SVM
    Security Violation Mode
enumerator kOTFAD_LDM
    Logically Disabled Mode
typedef struct _otfad_encryption_config otfad_encryption_config_t
    OTFAD encryption configuration structure.
typedef struct _otfad_config otfad_config_t
    OTFAD configuration structure.
struct _otfad_encryption_config
    #include <fsl_otfad.h> OTFAD encryption configuration structure.
```

Public Members

```
bool valid
    The context is valid or not
bool AESdecryption
    AES decryption enable
uint8_t readOnly
    read write attribute for the entire set of context registers
uint8_t contextIndex
    OTFAD context index
uint32_t startAddr
    Start address
uint32_t endAddr
    End address
uint32_t key[4]
    Encryption key
uint32_t counter[2]
    Encryption counter
struct _otfad_config
    #include <fsl_otfad.h> OTFAD configuration structure.
```

Public Members

```
bool forceSVM
    Force entry into SVM after a write
bool forceLDM
    Force entry into LDM after a write
bool restrictedRegAccess
    Restricted register access enable
bool enableOTFAD
    OTFAD has decryption enabled
```

2.89 PDM: Microphone Interface

2.90 PDM Driver

`void PDM_Init(PDM_Type *base, const pdm_config_t *config)`

Initializes the PDM peripheral.

Ungates the PDM clock, resets the module, and configures PDM with a configuration structure. The configuration structure can be custom filled or set with default values by `PDM_GetDefaultConfig()`.

Note: This API should be called at the beginning of the application to use the PDM driver. Otherwise, accessing the PDM module can cause a hard fault because the clock is not enabled.

Parameters

- `base` – PDM base pointer
- `config` – PDM configuration structure.

`void PDM_Deinit(PDM_Type *base)`

De-initializes the PDM peripheral.

This API gates the PDM clock. The PDM module can't operate unless `PDM_Init` is called to enable the clock.

Parameters

- `base` – PDM base pointer

`static inline void PDM_Reset(PDM_Type *base)`

Resets the PDM module.

Parameters

- `base` – PDM base pointer

`static inline void PDM_Enable(PDM_Type *base, bool enable)`

Enables/disables PDM interface.

Parameters

- `base` – PDM base pointer
- `enable` – True means PDM interface is enabled, false means PDM interface is disabled.

`static inline void PDM_EnableDebugMode(PDM_Type *base, bool enable)`

Enables/disables debug mode for PDM. The PDM interface cannot enter debug mode once in Disable/Low Leakage or Low Power mode.

Parameters

- `base` – PDM base pointer
- `enable` – True means PDM interface enter debug mode, false means PDM interface in normal mode.

`static inline void PDM_EnableInDebugMode(PDM_Type *base, bool enable)`

Enables/disables PDM interface in debug mode.

Parameters

- base – PDM base pointer
- enable – True means PDM interface is enabled debug mode, false means PDM interface is disabled after after completing the current frame in debug mode.

static inline void PDM_EnterLowLeakageMode(PDM_Type *base, bool enable)

Enables/disables PDM interface disable/Low Leakage mode.

Parameters

- base – PDM base pointer
- enable – True means PDM interface is in disable/low leakage mode, False means PDM interface is in normal mode.

static inline void PDM_EnableChannel(PDM_Type *base, uint8_t channel, bool enable)

Enables/disables the PDM channel.

Parameters

- base – PDM base pointer
- channel – PDM channel number need to enable or disable.
- enable – True means enable PDM channel, false means disable.

void PDM_SetChannelConfig(PDM_Type *base, uint32_t channel, const *pdm_channel_config_t* *config)

PDM one channel configurations.

Parameters

- base – PDM base pointer
- config – PDM channel configurations.
- channel – channel number. after completing the current frame in debug mode.

status_t PDM_SetSampleRateConfig(PDM_Type *base, uint32_t sourceClock_HZ, uint32_t sampleRate_HZ)

PDM set sample rate.

Note: This function is depend on the configuration of the PDM and PDM channel, so the correct call sequence is

```
PDM_Init(base, pdmConfig)
PDM_SetChannelConfig(base, channel, &channelConfig)
PDM_SetSampleRateConfig(base, source, sampleRate)
```

Parameters

- base – PDM base pointer
- sourceClock_HZ – PDM source clock frequency.
- sampleRate_HZ – PDM sample rate.

status_t PDM_SetSampleRate(PDM_Type *base, uint32_t enableChannelMask, *pdm_df_quality_mode_t* qualityMode, uint8_t osr, uint32_t clkDiv)

PDM set sample rate.

Deprecated:

Do not use this function. It has been superceded by PDM_SetSampleRateConfig

Parameters

- base – PDM base pointer
- enableChannelMask – PDM channel enable mask.
- qualityMode – quality mode.
- osr – cic oversample rate
- clkDiv – clock divider

uint32_t PDM_GetInstance(PDM_Type *base)

Get the instance number for PDM.

Parameters

- base – PDM base pointer.

static inline uint32_t PDM_GetStatus(PDM_Type *base)

Gets the PDM internal status flag. Use the Status Mask in `_pdm_internal_status` to get the status value needed.

Parameters

- base – PDM base pointer

Returns

PDM status flag value.

static inline uint32_t PDM_GetFifoStatus(PDM_Type *base)

Gets the PDM FIFO status flag. Use the Status Mask in `_pdm_fifo_status` to get the status value needed.

Parameters

- base – PDM base pointer

Returns

FIFO status.

static inline uint32_t PDM_GetRangeStatus(PDM_Type *base)

Gets the PDM Range status flag. Use the Status Mask in `_pdm_range_status` to get the status value needed.

Parameters

- base – PDM base pointer

Returns

output status.

static inline void PDM_ClearStatus(PDM_Type *base, uint32_t mask)

Clears the PDM Tx status.

Parameters

- base – PDM base pointer
- mask – State mask. It can be a combination of the status between `kPDM_StatusFrequencyLow` and `kPDM_StatusCh7FifoDataAvaliable`.

static inline void PDM_ClearFIFOStatus(PDM_Type *base, uint32_t mask)

Clears the PDM Tx status.

Parameters

- base – PDM base pointer
- mask – State mask. It can be a combination of the status in `_pdm_fifo_status`.

static inline void PDM_ClearRangeStatus(PDM_Type *base, uint32_t mask)

Clears the PDM range status.

Parameters

- base – PDM base pointer
- mask – State mask. It can be a combination of the status in `_pdm_range_status`.

void PDM_EnableInterrupts(PDM_Type *base, uint32_t mask)

Enables the PDM interrupt requests.

Parameters

- base – PDM base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
 - kPDM_ErrorInterruptEnable
 - kPDM_FIFOInterruptEnable

static inline void PDM_DisableInterrupts(PDM_Type *base, uint32_t mask)

Disables the PDM interrupt requests.

Parameters

- base – PDM base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
 - kPDM_ErrorInterruptEnable
 - kPDM_FIFOInterruptEnable

static inline void PDM_EnableDMA(PDM_Type *base, bool enable)

Enables/disables the PDM DMA requests.

Parameters

- base – PDM base pointer
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t PDM_GetDataRegisterAddress(PDM_Type *base, uint32_t channel)

Gets the PDM data register address.

This API is used to provide a transfer address for the PDM DMA transfer configuration.

Parameters

- base – PDM base pointer.
- channel – Which data channel used.

Returns

data register address.

void PDM_ReadFifo(PDM_Type *base, uint32_t startChannel, uint32_t channelNums, void *buffer, size_t size, uint32_t dataWidth)

PDM read fifo.

Note: : This function support 16 bit only for IP version that only supports 16bit.

Parameters

- base – PDM base pointer.
- startChannel – start channel number.
- channelNums – total enabled channelnums.
- buffer – received buffer address.
- size – number of samples to read.
- dataWidth – sample width.

void PDM_SetChannelGain(PDM_Type *base, uint32_t channel, *pdm_df_output_gain_t* gain)

Set the PDM channel gain.

Please note for different quality mode, the valid gain value is different, reference RM for detail.

Parameters

- base – PDM base pointer.
- channel – PDM channel index.
- gain – channel gain, the register gain value range is 0 - 15.

void PDM_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 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 . Recommend 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 . Recommend 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 2.9.3

PDM return status.

Values:

enumerator kStatus_PDM_Busy

PDM is busy.

enumerator kStatus_PDM_CLK_LOW

PDM clock frequency low

enumerator kStatus_PDM_FIFO_ERROR

PDM FIFO underrun or overflow

enumerator kStatus_PDM_QueueFull

PDM FIFO underrun or overflow

enumerator kStatus_PDM_Idle

PDM is idle

enumerator kStatus_PDM_Output_ERROR

PDM is output error

enumerator kStatus_PDM_ChannelConfig_Failed

PDM channel config failed

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_StatusFrequencyLow
Mic app clock frequency not high enough

enumerator kPDM_StatusCh0FifoDataAvaliable
channel 0 fifo data reached watermark level

enumerator kPDM_StatusCh1FifoDataAvaliable
channel 1 fifo data reached watermark level

enumerator kPDM_StatusCh2FifoDataAvaliable
channel 2 fifo data reached watermark level

enumerator kPDM_StatusCh3FifoDataAvaliable
channel 3 fifo data reached watermark level

enum __pdm_channel_enable_mask
PDM channel enable mask.

Values:

enumerator kPDM_EnableChannel0
channgel 0 enable mask

enumerator kPDM_EnableChannel1
channgel 1 enable mask

enumerator kPDM_EnableChannel2
channgel 2 enable mask

enumerator kPDM_EnableChannel3
channgel 3 enable mask

enumerator kPDM_EnableChannelAll

enum __pdm_fifo_status
The PDM fifo status.

Values:

enumerator kPDM_FifoStatusUnderflowCh0
channel0 fifo status underflow

enumerator kPDM_FifoStatusUnderflowCh1
channel1 fifo status underflow

enumerator kPDM_FifoStatusUnderflowCh2
channel2 fifo status underflow

enumerator kPDM_FifoStatusUnderflowCh3
channel3 fifo status underflow

enumerator kPDM_FifoStatusOverflowCh0
channel0 fifo status overflow

enumerator kPDM_FifoStatusOverflowCh1
channel1 fifo status overflow

enumerator kPDM_FifoStatusOverflowCh2
channel2 fifo status overflow

enumerator kPDM_FifoStatusOverflowCh3
channel3 fifo status overflow

enum __pdm_range_status
The PDM output status.

Values:

enumerator kPDM_RangeStatusUnderFlowCh0
channel0 range status underflow

enumerator kPDM_RangeStatusUnderFlowCh1
channel1 range status underflow

enumerator kPDM_RangeStatusUnderFlowCh2
channel2 range status underflow

enumerator kPDM_RangeStatusUnderFlowCh3
channel3 range status underflow

enumerator kPDM_RangeStatusOverFlowCh0
channel0 range status overflow

enumerator kPDM_RangeStatusOverFlowCh1
channel1 range status overflow

enumerator kPDM_RangeStatusOverFlowCh2
channel2 range status overflow

enumerator kPDM_RangeStatusOverFlowCh3
channel3 range status overflow

enum __pdm_dc_removal
PDM DC removal configurations.

Values:

enumerator kPDM_DcRemovalCutOff21Hz
DC removal cut off 21HZ

enumerator kPDM_DcRemovalCutOff83Hz
DC removal cut off 83HZ

enumerator kPDM_DcRemovalCutOff152Hz
DC removal cut off 152HZ

enumerator kPDM_DcRemovalBypass
DC removal 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_DataWwidth24

PDM data width 24bit

enumerator kPDM_DataWwidth32

PDM data width 32bit

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_removal_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

bool enableFilterBypass

Switchable bypass path for the decimation filter

uint8_t fifoWatermark

Watermark value for FIFO

pdm_df_quality_mode_t qualityMode

Quality mode

uint8_t cicOverSampleRate

CIC filter over sampling rate

```
struct __pdm_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 filter 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

2.91 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 funcio 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],
        .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_EDMA_MultiChannelInterleavePerChannelSample

enumerator kPDM_EDMA_MultiChannelInterleavePerChannelBlock

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

2.92 RGPIO: Rapid General-Purpose Input/Output Driver

FSL_RGPIODRIVER_VERSION

RGPIO driver version 2.1.0.

enum _rgpio_pin_direction

RGPIO direction definition.

Values:

enumerator kRGPIO_DigitalInput

Set current pin as digital input

enumerator kRGPIO_DigitalOutput

Set current pin as digital output

enum _rgpio_checker_attribute

RGPIO checker attribute.

Values:

enumerator kRGPIO_UsernonsecureRWUsersecureRWPrivilegedsecureRW

User nonsecure:Read+Write; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kRGPIO_UsernonsecureRUsersecureRWPrivilegedsecureRW

User nonsecure:Read; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kRGPIO_UsernonsecureNUsersecureRWPrivilegedsecureRW

User nonsecure:None; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kRGPIO_UsernonsecureRUsersecureRPrivilegedsecureRW

User nonsecure:Read; User Secure:Read; Privileged Secure:Read+Write

enumerator kRGPIO_UsernonsecureNUsersecureRPrivilegedsecureRW

User nonsecure:None; User Secure:Read; Privileged Secure:Read+Write

enumerator kRGPIO_UsernonsecureNUsersecureNPrivilegedsecureRW

User nonsecure:None; User Secure:None; Privileged Secure:Read+Write

enumerator kRGPIO_UsernonsecureNUsersecureNPrivilegedsecureR

User nonsecure:None; User Secure:None; Privileged Secure:Read

enumerator kRGPIO_UsernonsecureNUsersecureNPrivilegedsecureN

User nonsecure:None; User Secure:None; Privileged Secure:None

enumerator kRGPIO_IgnoreAttributeCheck

Ignores the attribute check

enum _rgpio_interrupt_sel

Configures the interrupt generation condition.

Values:

enumerator kRGPIO_InterruptOutput0

Interrupt/DMA request/trigger output 0.

enumerator kRGPIO_InterruptOutput1

Interrupt/DMA request/trigger output 1.

enumerator kRGPIO_InterruptOutput2

Interrupt/DMA request/trigger output 2.

enumerator kRGPIO_InterruptOutput3
Interrupt/DMA request/trigger output 3.

enum _rgpio_interrupt_config
Configures the interrupt generation condition.

Values:

enumerator kRGPIO_InterruptOrDMADisabled
Interrupt/DMA request is disabled.

enumerator kRGPIO_DMARisingEdge
DMA request on rising edge.

enumerator kRGPIO_DMAFallingEdge
DMA request on falling edge.

enumerator kRGPIO_DMAEitherEdge
DMA request on either edge.

enumerator kRGPIO_FlagRisingEdge
Flag sets on rising edge.

enumerator kRGPIO_FlagFallingEdge
Flag sets on falling edge.

enumerator kRGPIO_FlagEitherEdge
Flag sets on either edge.

enumerator kRGPIO_InterruptLogicZero
Interrupt when logic zero.

enumerator kRGPIO_InterruptRisingEdge
Interrupt on rising edge.

enumerator kRGPIO_InterruptFallingEdge
Interrupt on falling edge.

enumerator kRGPIO_InterruptEitherEdge
Interrupt on either edge.

enumerator kRGPIO_InterruptLogicOne
Interrupt when logic one.

enumerator kRGPIO_ActiveHighTriggerOutputEnable
Enable active high-trigger output.

enumerator kRGPIO_ActiveLowTriggerOutputEnable
Enable active low-trigger output.

typedef enum _rgpio_pin_direction rgpio_pin_direction_t
RGPIO direction definition.

typedef enum _rgpio_checker_attribute rgpio_checker_attribute_t
RGPIO checker attribute.

typedef enum _rgpio_interrupt_sel rgpio_interrupt_sel_t
Configures the interrupt generation condition.

typedef enum _rgpio_interrupt_config rgpio_interrupt_config_t
Configures the interrupt generation condition.

```
typedef struct _rgpio_pin_config rgpio_pin_config_t
```

The RGPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT_SetPinConfig().

```
struct _rgpio_pin_config
```

#include <fsl_rgpio.h> The RGPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT_SetPinConfig().

Public Members

rgpio_pin_direction_t pinDirection

RGPIO direction, input or output

uint8_t outputLogic

Set a default output logic, which has no use in input

2.93 RGPIO Driver

```
void RGPIO_PinInit(RGPIO_Type *base, uint32_t pin, const rgpio_pin_config_t *config)
```

Initializes a RGPIO pin used by the board.

To initialize the RGPIO, define a pin configuration, as either input or output, in the user file. Then, call the RGPIO_PinInit() function.

This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
rgpio_pin_config_t config =
{
    kRGPIO_DigitalInput,
    0,
}
Define a digital output pin configuration,
rgpio_pin_config_t config =
{
    kRGPIO_DigitalOutput,
    0,
}
```

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- pin – RGPIO port pin number
- config – RGPIO pin configuration pointer

```
uint32_t RGPIO_GetInstance(RGPIO_Type *base)
```

Gets the RGPIO instance according to the RGPIO base.

Parameters

- base – RGPIO peripheral base pointer(PTA, PTB, PTC, etc.)

Return values

RGPIO – instance

```
static inline void RGPIO_PinWrite(RGPIO_Type *base, uint32_t pin, uint8_t output)
```

Sets the output level of the multiple RGPIO pins to the logic 1 or 0.

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- pin – RGPIO pin number
- output – RGPIO pin output logic level.
 - 0: corresponding pin output low-logic level.
 - 1: corresponding pin output high-logic level.

```
static inline void RGPIO_WritePinOutput(RGPIO_Type *base, uint32_t pin, uint8_t output)
```

Sets the output level of the multiple RGPIO pins to the logic 1 or 0.

Deprecated:

Do not use this function. It has been superceded by RGPIO_PinWrite.

```
static inline void RGPIO_PortSet(RGPIO_Type *base, uint32_t mask)
```

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

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number macro

```
static inline void RGPIO_SetPinsOutput(RGPIO_Type *base, uint32_t mask)
```

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

Deprecated:

Do not use this function. It has been superceded by RGPIO_PortSet.

```
static inline void RGPIO_PortClear(RGPIO_Type *base, uint32_t mask)
```

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

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number macro

```
static inline void RGPIO_ClearPinsOutput(RGPIO_Type *base, uint32_t mask)
```

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

Deprecated:

Do not use this function. It has been superceded by RGPIO_PortClear.

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number macro


```
static inline void RGPIO_PortToggle(RGPIO_Type *base, uint32_t mask)
```

Reverses the current output logic of the multiple RGPIO pins.

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number macro

```
static inline void RGPIO_TogglePinsOutput(RGPIO_Type *base, uint32_t mask)
```

Reverses the current output logic of the multiple RGPIO pins.

Deprecated:

Do not use this function. It has been superceded by RGPIO_PortToggle.

```
static inline uint32_t RGPIO_PinRead(RGPIO_Type *base, uint32_t pin)
```

Reads the current input value of the RGPIO port.

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- pin – RGPIO pin number

Return values

RGPIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

```
static inline uint32_t RGPIO_ReadPinInput(RGPIO_Type *base, uint32_t pin)
```

Reads the current input value of the RGPIO port.

Deprecated:

Do not use this function. It has been superceded by RGPIO_PinRead.

```
static inline void RGPIO_EnablePortInput(RGPIO_Type *base, uint32_t mask, bool enable)
```

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number mask
- enable – RGPIO digital input enable/disable flag.

```
void RGPIO_CheckAttributeBytes(RGPIO_Type *base, rgpio_checker_attribute_t attribute)
```

The RGPIO module supports a device-specific number of data ports, organized as 32-bit words. Each 32-bit data port includes a GACR register, which defines the byte-level attributes required for a successful access to the RGPIO programming model. The attribute controls for the 4 data bytes in the GACR follow a standard little endian data convention.

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number macro

```
static inline void RGPIO_SetPinInterruptConfig(RGPIO_Type *base, uint32_t pin,
                                             rgpio_interrupt_sel_t sel,
                                             rgpio_interrupt_config_t config)
```

Configures the gpio pin interrupt/DMA request.

Parameters

- base – RGPIO peripheral base pointer.
- pin – RGPIO pin number.
- sel – RGPIO pin interrupt selection(0-3).
- config – RGPIO pin interrupt configuration.

```
static inline void __SetMultipleInterruptPinsConfig(RGPIO_Type *base, uint32_t mask,
                                                  rgpio_interrupt_sel_t sel,
                                                  rgpio_interrupt_config_t config)
```

Sets the gpio interrupt configuration in ICR register for multiple pins.

Parameters

- base – RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
- mask – RGPIO pin number macro.
- sel – RGPIO pin interrupt selection(0-3).
- config – RGPIO pin interrupt configuration.

```
static inline uint32_t RGPIO_GetPinsInterruptFlags(RGPIO_Type *base, rgpio_interrupt_sel_t sel)
```

Reads the whole gpio status flag.

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

- base – RGPIO peripheral base pointer.
- sel – RGPIO pin interrupt selection(0-3).

Returns

Current gpio interrupt status flags, for example, 0x00010001 means the pin 0 and 16 have the interrupt.

```
static inline void RGPIO_ClearPinsInterruptFlags(RGPIO_Type *base, rgpio_interrupt_sel_t sel,
                                                uint32_t mask)
```

Clears the multiple pin interrupt status flag.

Parameters

- base – RGPIO peripheral base pointer.
- sel – RGPIO pin interrupt selection(0-3).
- mask – RGPIO pin number macro.

2.94 SAI: Serial Audio Interface

2.95 SAI Driver

`void SAI_Init(I2S_Type *base)`

Initializes the SAI peripheral.

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

Parameters

- base – SAI base pointer.

`void SAI_Deinit(I2S_Type *base)`

De-initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless SAI_TxInit or SAI_RxInit is called to enable the clock.

Parameters

- base – SAI base pointer.

`void SAI_TxReset(I2S_Type *base)`

Resets the SAI Tx.

This function enables the software reset and FIFO reset of SAI Tx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

`void SAI_RxReset(I2S_Type *base)`

Resets the SAI Rx.

This function enables the software reset and FIFO reset of SAI Rx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

`void SAI_TxEnable(I2S_Type *base, bool enable)`

Enables/disables the SAI Tx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Tx, false means disable.

`void SAI_RxEnable(I2S_Type *base, bool enable)`

Enables/disables the SAI Rx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Rx, false means disable.

`static inline void SAI_TxSetBitClockDirection(I2S_Type *base, sai_master_slave_t masterSlave)`

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference `sai_master_slave_t`.

```
static inline void SAI_RxSetBitClockDirection(I2S_Type *base, sai_master_slave_t masterSlave)
```

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

```
static inline void SAI_RxSetFrameSyncDirection(I2S_Type *base, sai_master_slave_t  
                                              masterSlave)
```

Set Rx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

```
static inline void SAI_TxSetFrameSyncDirection(I2S_Type *base, sai_master_slave_t masterSlave)
```

Set Tx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

```
void SAI_TxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate,  
                          uint32_t bitWidth, uint32_t channelNumbers)
```

Transmitter bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.
- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

```
void SAI_RxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate,  
                          uint32_t bitWidth, uint32_t channelNumbers)
```

Receiver bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.
- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

```
void SAI_TxSetBitclockConfig(I2S_Type *base, sai_master_slave_t masterSlave, sai_bit_clock_t  
                           *config)
```

Transmitter Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

```
void SAI_RxSetBitclockConfig(I2S_Type *base, sai_master_slave_t masterSlave, sai_bit_clock_t *config)
```

Receiver Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

```
void SAI_SetMasterClockConfig(I2S_Type *base, sai_master_clock_t *config)
```

Master clock configurations.

Parameters

- base – SAI base pointer.
- config – master clock configurations.

```
void SAI_TxSetFifoConfig(I2S_Type *base, sai_fifo_t *config)
```

SAI transmitter fifo configurations.

Parameters

- base – SAI base pointer.
- config – fifo configurations.

```
void SAI_RxSetFifoConfig(I2S_Type *base, sai_fifo_t *config)
```

SAI receiver fifo configurations.

Parameters

- base – SAI base pointer.
- config – fifo configurations.

```
void SAI_TxSetFrameSyncConfig(I2S_Type *base, sai_master_slave_t masterSlave, sai_frame_sync_t *config)
```

SAI transmitter Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

```
void SAI_RxSetFrameSyncConfig(I2S_Type *base, sai_master_slave_t masterSlave, sai_frame_sync_t *config)
```

SAI receiver Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

void SAI_TxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI transmitter Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_RxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI receiver Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_TxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI transmitter configurations.

Parameters

- base – SAI base pointer.
- config – transmitter configurations.

void SAI_RxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI receiver configurations.

Parameters

- base – SAI base pointer.
- config – receiver configurations.

void SAI_GetClassicI2SConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth,
sai_mono_stereo_t mode, uint32_t saiChannelMask)

Get classic I2S mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetLeftJustifiedConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth,
sai_mono_stereo_t mode, uint32_t saiChannelMask)

Get left justified mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetRightJustifiedConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth,
sai_mono_stereo_t mode, uint32_t saiChannelMask)

Get right justified mode configurations.

Parameters

- config – transceiver configurations.

- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

```
void SAI_GetTDMConfig(sai_transceiver_t *config, sai_frame_sync_len_t frameSyncWidth,
                    sai_word_width_t bitWidth, uint32_t dataWordNum, uint32_t
                    saiChannelMask)
```

Get TDM mode configurations.

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data word width.
- dataWordNum – word number in one frame.
- saiChannelMask – mask value of the channel to be enable.

```
void SAI_GetDSPConfig(sai_transceiver_t *config, sai_frame_sync_len_t frameSyncWidth,
                    sai_word_width_t bitWidth, sai_mono_stereo_t mode, uint32_t
                    saiChannelMask)
```

Get DSP mode configurations.

DSP/PCM MODE B configuration flow for TX. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
SAI_TxSetConfig(base, config)
```

Note: DSP mode is also called PCM mode which support MODE A and MODE B, DSP/PCM MODE A configuration flow. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
config->frameSync.frameSyncEarly = true;
SAI_TxSetConfig(base, config)
```

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

```
static inline uint32_t SAI_TxGetStatusFlag(I2S_Type *base)
```

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns

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

```
static inline void SAI_TxClearStatusFlags(I2S_Type *base, uint32_t mask)
```

Clears the SAI Tx status flag state.

Parameters

- base – SAI base pointer
- mask – State mask. It can be a combination of the following source if defined:
 - kSAI_WordStartFlag
 - kSAI_SyncErrorFlag
 - kSAI_FIFOErrorFlag

```
static inline uint32_t SAI_RxGetStatusFlag(I2S_Type *base)
```

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns

SAI Rx status flag value. Use the Status Mask to get the status value needed.

```
static inline void SAI_RxClearStatusFlags(I2S_Type *base, uint32_t mask)
```

Clears the SAI Rx status flag state.

Parameters

- base – SAI base pointer
- mask – State mask. It can be a combination of the following sources if defined.
 - kSAI_WordStartFlag
 - kSAI_SyncErrorFlag
 - kSAI_FIFOErrorFlag

```
void SAI_TxSoftwareReset(I2S_Type *base, sai_reset_type_t resetType)
```

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like TCR1~TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

```
void SAI_RxSoftwareReset(I2S_Type *base, sai_reset_type_t resetType)
```

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1~RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

`void SAI_TxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)`

Set the Tx channel FIFO enable mask.

Parameters

- `base` – SAI base pointer
- `mask` – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

`void SAI_RxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)`

Set the Rx channel FIFO enable mask.

Parameters

- `base` – SAI base pointer
- `mask` – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

`void SAI_TxSetDataOrder(I2S_Type *base, sai_data_order_t order)`

Set the Tx data order.

Parameters

- `base` – SAI base pointer
- `order` – Data order MSB or LSB

`void SAI_RxSetDataOrder(I2S_Type *base, sai_data_order_t order)`

Set the Rx data order.

Parameters

- `base` – SAI base pointer
- `order` – Data order MSB or LSB

`void SAI_TxSetBitClockPolarity(I2S_Type *base, sai_clock_polarity_t polarity)`

Set the Tx data order.

Parameters

- `base` – SAI base pointer
- `polarity` –

`void SAI_RxSetBitClockPolarity(I2S_Type *base, sai_clock_polarity_t polarity)`

Set the Rx data order.

Parameters

- `base` – SAI base pointer
- `polarity` –

`void SAI_TxSetFrameSyncPolarity(I2S_Type *base, sai_clock_polarity_t polarity)`

Set the Tx data order.

Parameters

- `base` – SAI base pointer
- `polarity` –

`void SAI_RxSetFrameSyncPolarity(I2S_Type *base, sai_clock_polarity_t polarity)`

Set the Rx data order.

Parameters

- `base` – SAI base pointer

- polarity –

void SAI_TxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Tx FIFO packing feature.

Parameters

- base – SAI base pointer.
- pack – FIFO pack type. It is element of *sai_fifo_packing_t*.

void SAI_RxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Rx FIFO packing feature.

Parameters

- base – SAI base pointer.
- pack – FIFO pack type. It is element of *sai_fifo_packing_t*.

static inline void SAI_TxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Tx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI_RxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Rx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI_TxEnableInterrupts(I2S_Type *base, uint32_t mask)

Enables the SAI Tx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
 - kSAI_WordStartInterruptEnable
 - kSAI_SyncErrorInterruptEnable
 - kSAI_FIFOWarningInterruptEnable
 - kSAI_FIFORequestInterruptEnable
 - kSAI_FIFOErrorInterruptEnable

static inline void SAI_RxEnableInterrupts(I2S_Type *base, uint32_t mask)

Enables the SAI Rx interrupt requests.

Parameters

- base – SAI base pointer

- **mask** – interrupt source The parameter can be a combination of the following sources if defined.
 - kSAI_WordStartInterruptEnable
 - kSAI_SyncErrorInterruptEnable
 - kSAI_FIFOWarningInterruptEnable
 - kSAI_FIFORequestInterruptEnable
 - kSAI_FIFOErrorInterruptEnable

static inline void SAI_TxDisableInterrupts(I2S_Type *base, uint32_t mask)

Disables the SAI Tx interrupt requests.

Parameters

- **base** – SAI base pointer
- **mask** – interrupt source The parameter can be a combination of the following sources if defined.
 - kSAI_WordStartInterruptEnable
 - kSAI_SyncErrorInterruptEnable
 - kSAI_FIFOWarningInterruptEnable
 - kSAI_FIFORequestInterruptEnable
 - kSAI_FIFOErrorInterruptEnable

static inline void SAI_RxDisableInterrupts(I2S_Type *base, uint32_t mask)

Disables the SAI Rx interrupt requests.

Parameters

- **base** – SAI base pointer
- **mask** – interrupt source The parameter can be a combination of the following sources if defined.
 - kSAI_WordStartInterruptEnable
 - kSAI_SyncErrorInterruptEnable
 - kSAI_FIFOWarningInterruptEnable
 - kSAI_FIFORequestInterruptEnable
 - kSAI_FIFOErrorInterruptEnable

static inline void SAI_TxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)

Enables/disables the SAI Tx DMA requests.

Parameters

- **base** – SAI base pointer
- **mask** – DMA source The parameter can be combination of the following sources if defined.
 - kSAI_FIFOWarningDMAEnable
 - kSAI_FIFORequestDMAEnable
- **enable** – True means enable DMA, false means disable DMA.

static inline void SAI_RxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)
Enables/disables the SAI Rx DMA requests.

Parameters

- base – SAI base pointer
- mask – DMA source The parameter can be a combination of the following sources if defined.
 - kSAI_FIFOWarningDMAEnable
 - kSAI_FIFORequestDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline uintptr_t SAI_TxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)
Gets the SAI Tx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- base – SAI base pointer.
- channel – Which data channel used.

Returns

data register address.

static inline uintptr_t SAI_RxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)
Gets the SAI Rx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- base – SAI base pointer.
- channel – Which data channel used.

Returns

data register address.

void SAI_WriteBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer,
uint32_t size)

Sends data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

void SAI_WriteMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask,
uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data to multi channel using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

```
static inline void SAI_WriteData(I2S_Type *base, uint32_t channel, uint32_t data)
```

Writes data into SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- data – Data needs to be written.

```
void SAI_ReadBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer,  
                     uint32_t size)
```

Receives data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

```
void SAI_ReadMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask,  
                                 uint32_t bitWidth, uint8_t *buffer, uint32_t size)
```

Receives multi channel data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

```
static inline uint32_t SAI_ReadData(I2S_Type *base, uint32_t channel)
```

Reads data from the SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.

Returns

Data in SAI FIFO.

```
void SAI_TransferTxCreateHandle(I2S_Type *base, sai_handle_t *handle, sai_transfer_callback_t  
callback, void *userData)
```

Initializes the SAI Tx handle.

This function initializes the Tx handle for the SAI Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SAI base pointer
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function

```
void SAI_TransferRxCreateHandle(I2S_Type *base, sai_handle_t *handle, sai_transfer_callback_t  
callback, void *userData)
```

Initializes the SAI Rx handle.

This function initializes the Rx handle for the SAI Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function.

```
void SAI_TransferTxSetConfig(I2S_Type *base, sai_handle_t *handle, sai_transceiver_t *config)  
SAI transmitter transfer configurations.
```

This function initializes the Tx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- config – transmitter configurations.

```
void SAI_TransferRxSetConfig(I2S_Type *base, sai_handle_t *handle, sai_transceiver_t *config)  
SAI receiver transfer configurations.
```

This function initializes the Rx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.

- config – receiver configurations.

status_t SAI_TransferSendNonBlocking(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_t* *xfer)

Performs an interrupt non-blocking send transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the SAI_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

- base – SAI base pointer.
- handle – Pointer to the *sai_handle_t* structure which stores the transfer state.
- xfer – Pointer to the *sai_transfer_t* structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SAI_TxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SAI_TransferReceiveNonBlocking(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_t* *xfer)

Performs an interrupt non-blocking receive transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the SAI_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

- base – SAI base pointer
- handle – Pointer to the *sai_handle_t* structure which stores the transfer state.
- xfer – Pointer to the *sai_transfer_t* structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SAI_RxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SAI_TransferGetSendCount(I2S_Type *base, *sai_handle_t* *handle, *size_t* *count)

Gets a set byte count.

Parameters

- base – SAI base pointer.
- handle – Pointer to the *sai_handle_t* structure which stores the transfer state.
- count – Bytes count sent.

Return values

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

`status_t` SAI_TransferGetReceiveCount(I2S_Type *base, *sai_handle_t* *handle, `size_t` *count)

Gets a received byte count.

Parameters

- `base` – SAI base pointer.
- `handle` – Pointer to the `sai_handle_t` structure which stores the transfer state.
- `count` – Bytes count received.

Return values

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

`void` SAI_TransferAbortSend(I2S_Type *base, *sai_handle_t* *handle)

Aborts the current send.

Note: This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- `base` – SAI base pointer.
- `handle` – Pointer to the `sai_handle_t` structure which stores the transfer state.

`void` SAI_TransferAbortReceive(I2S_Type *base, *sai_handle_t* *handle)

Aborts the current IRQ receive.

Note: This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- `base` – SAI base pointer
- `handle` – Pointer to the `sai_handle_t` structure which stores the transfer state.

`void` SAI_TransferTerminateSend(I2S_Type *base, *sai_handle_t* *handle)

Terminate all SAI send.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSend.

Parameters

- `base` – SAI base pointer.
- `handle` – SAI eDMA handle pointer.

void SAI_TransferTerminateReceive(I2S_Type *base, sai_handle_t *handle)

Terminate all SAI receive.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceive.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTxHandleIRQ(I2S_Type *base, sai_handle_t *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure.

void SAI_TransferRxHandleIRQ(I2S_Type *base, sai_handle_t *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure.

void SAI_DriverIRQHandler(uint32_t instance)

SAI driver IRQ handler common entry.

This function provides the common IRQ request entry for SAI.

Parameters

- instance – SAI instance.

FSL_SAI_DRIVER_VERSION

Version 2.4.10

_sai_status_t, SAI return status.

Values:

enumerator kStatus_SAI_TxBusy
SAI Tx is busy.

enumerator kStatus_SAI_RxBusy
SAI Rx is busy.

enumerator kStatus_SAI_TxError
SAI Tx FIFO error.

enumerator kStatus_SAI_RxError
SAI Rx FIFO error.

enumerator kStatus_SAI_QueueFull
SAI transfer queue is full.

enumerator kStatus_SAI_TxIdle
SAI Tx is idle

enumerator kStatus_SAI_RxIdle
SAI Rx is idle

`_sai_channel_mask`, sai channel mask value, actual channel numbers is depend soc specific
Values:

enumerator `kSAI_Channel0Mask`
channel 0 mask value

enumerator `kSAI_Channel1Mask`
channel 1 mask value

enumerator `kSAI_Channel2Mask`
channel 2 mask value

enumerator `kSAI_Channel3Mask`
channel 3 mask value

enumerator `kSAI_Channel4Mask`
channel 4 mask value

enumerator `kSAI_Channel5Mask`
channel 5 mask value

enumerator `kSAI_Channel6Mask`
channel 6 mask value

enumerator `kSAI_Channel7Mask`
channel 7 mask value

enum `_sai_protocol`

Define the SAI bus type.

Values:

enumerator `kSAI_BusLeftJustified`
Uses left justified format.

enumerator `kSAI_BusRightJustified`
Uses right justified format.

enumerator `kSAI_BusI2S`
Uses I2S format.

enumerator `kSAI_BusPCMA`
Uses I2S PCM A format.

enumerator `kSAI_BusPCMB`
Uses I2S PCM B format.

enum `_sai_master_slave`

Master or slave mode.

Values:

enumerator `kSAI_Master`
Master mode include bclk and frame sync

enumerator `kSAI_Slave`
Slave mode include bclk and frame sync

enumerator `kSAI_Bclk_Master_FrameSync_Slave`
bclk in master mode, frame sync in slave mode

enumerator kSAI_Bclk_Slave_FrameSync_Master
bclk in slave mode, frame sync in master mode

enum _sai_mono_stereo
Mono or stereo audio format.

Values:

enumerator kSAI_Stereo
Stereo sound.

enumerator kSAI_MonoRight
Only Right channel have sound.

enumerator kSAI_MonoLeft
Only left channel have sound.

enum _sai_data_order
SAI data order, MSB or LSB.

Values:

enumerator kSAI_DataLSB
LSB bit transferred first

enumerator kSAI_DataMSB
MSB bit transferred first

enum _sai_clock_polarity
SAI clock polarity, active high or low.

Values:

enumerator kSAI_PolarityActiveHigh
Drive outputs on rising edge

enumerator kSAI_PolarityActiveLow
Drive outputs on falling edge

enumerator kSAI_SampleOnFallingEdge
Sample inputs on falling edge

enumerator kSAI_SampleOnRisingEdge
Sample inputs on rising edge

enum _sai_sync_mode
Synchronous or asynchronous mode.

Values:

enumerator kSAI_ModeAsync
Asynchronous mode

enumerator kSAI_ModeSync
Synchronous mode (with receiver or transmit)

enumerator kSAI_ModeSyncWithOtherTx
Synchronous with another SAI transmit

enumerator kSAI_ModeSyncWithOtherRx
Synchronous with another SAI receiver

enum _sai_bclk_source

Bit clock source.

Values:

enumerator kSAI_BclkSourceBusclk

Bit clock using bus clock

enumerator kSAI_BclkSourceMclkOption1

Bit clock MCLK option 1

enumerator kSAI_BclkSourceMclkOption2

Bit clock MCLK option2

enumerator kSAI_BclkSourceMclkOption3

Bit clock MCLK option3

enumerator kSAI_BclkSourceMclkDiv

Bit clock using master clock divider

enumerator kSAI_BclkSourceOtherSai0

Bit clock from other SAI device

enumerator kSAI_BclkSourceOtherSai1

Bit clock from other SAI device

_sai_interrupt_enable_t, The SAI interrupt enable flag

Values:

enumerator kSAI_WordStartInterruptEnable

Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorInterruptEnable

Sync error flag, means the sync error is detected

enumerator kSAI_FIFOWarningInterruptEnable

FIFO warning flag, means the FIFO is empty

enumerator kSAI_FIFOErrorInterruptEnable

FIFO error flag

enumerator kSAI_FIFORequestInterruptEnable

FIFO request, means reached watermark

_sai_dma_enable_t, The DMA request sources

Values:

enumerator kSAI_FIFOWarningDMAEnable

FIFO warning caused by the DMA request

enumerator kSAI_FIFORequestDMAEnable

FIFO request caused by the DMA request

_sai_flags, The SAI status flag

Values:

enumerator kSAI_WordStartFlag

Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorFlag

Sync error flag, means the sync error is detected

enumerator kSAI_FIFOErrorFlag

FIFO error flag

enumerator kSAI_FIFORequestFlag

FIFO request flag.

enumerator kSAI_FIFOWarningFlag

FIFO warning flag

enum _sai_reset_type

The reset type.

Values:

enumerator kSAI_ResetTypeSoftware

Software reset, reset the logic state

enumerator kSAI_ResetTypeFIFO

FIFO reset, reset the FIFO read and write pointer

enumerator kSAI_ResetAll

All reset.

enum _sai_fifo_packing

The SAI packing mode The mode includes 8 bit and 16 bit packing.

Values:

enumerator kSAI_FifoPackingDisabled

Packing disabled

enumerator kSAI_FifoPacking8bit

8 bit packing enabled

enumerator kSAI_FifoPacking16bit

16bit packing enabled

enum _sai_sample_rate

Audio sample rate.

Values:

enumerator kSAI_SampleRate8KHz

Sample rate 8000 Hz

enumerator kSAI_SampleRate11025Hz

Sample rate 11025 Hz

enumerator kSAI_SampleRate12KHz

Sample rate 12000 Hz

enumerator kSAI_SampleRate16KHz

Sample rate 16000 Hz

enumerator kSAI_SampleRate22050Hz

Sample rate 22050 Hz

enumerator kSAI_SampleRate24KHz

Sample rate 24000 Hz

enumerator kSAI_SampleRate32KHz

Sample rate 32000 Hz

enumerator kSAI_SampleRate44100Hz

Sample rate 44100 Hz

enumerator kSAI_SampleRate48KHz

Sample rate 48000 Hz

enumerator kSAI_SampleRate96KHz

Sample rate 96000 Hz

enumerator kSAI_SampleRate192KHz

Sample rate 192000 Hz

enumerator kSAI_SampleRate384KHz

Sample rate 384000 Hz

enum _sai_word_width

Audio word width.

Values:

enumerator kSAI_WordWidth8bits

Audio data width 8 bits

enumerator kSAI_WordWidth16bits

Audio data width 16 bits

enumerator kSAI_WordWidth24bits

Audio data width 24 bits

enumerator kSAI_WordWidth32bits

Audio data width 32 bits

enum _sai_data_pin_state

sai data pin state definition

Values:

enumerator kSAI_DataPinStateTriState

transmit data pins are tri-stated when slots are masked or channels are disabled

enumerator kSAI_DataPinStateOutputZero

transmit data pins are never tri-stated and will output zero when slots are masked or channel disabled

enum _sai_fifo_combine

sai fifo combine mode definition

Values:

enumerator kSAI_FifoCombineDisabled

sai TX/RX fifo combine mode disabled

enumerator kSAI_FifoCombineModeEnabledOnRead

sai TX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnWrite

sai TX fifo combine mode enabled on FIFO write

enumerator kSAI_RxFifoCombineModeEnabledOnWrite

sai RX fifo combine mode enabled on FIFO write

```

    enumerator kSAI_RXFifoCombineModeEnabledOnRead
        sai RX fifo combine mode enabled on FIFO reads
    enumerator kSAI_FifoCombineModeEnabledOnReadWrite
        sai TX/RX fifo combined mode enabled on FIFO read/writes
enum __sai_transceiver_type
    sai transceiver type
    Values:
    enumerator kSAI_Transmitter
        sai transmitter
    enumerator kSAI_Receiver
        sai receiver
enum __sai_frame_sync_len
    sai frame sync len
    Values:
    enumerator kSAI_FrameSyncLenOneBitClk
        1 bit clock frame sync len for DSP mode
    enumerator kSAI_FrameSyncLenPerWordWidth
        Frame sync length decided by word width
typedef enum __sai_protocol sai_protocol_t
    Define the SAI bus type.
typedef enum __sai_master_slave sai_master_slave_t
    Master or slave mode.
typedef enum __sai_mono_stereo sai_mono_stereo_t
    Mono or stereo audio format.
typedef enum __sai_data_order sai_data_order_t
    SAI data order, MSB or LSB.
typedef enum __sai_clock_polarity sai_clock_polarity_t
    SAI clock polarity, active high or low.
typedef enum __sai_sync_mode sai_sync_mode_t
    Synchronous or asynchronous mode.
typedef enum __sai_bclk_source sai_bclk_source_t
    Bit clock source.
typedef enum __sai_reset_type sai_reset_type_t
    The reset type.
typedef enum __sai_fifo_packing sai_fifo_packing_t
    The SAI packing mode The mode includes 8 bit and 16 bit packing.
typedef struct __sai_config sai_config_t
    SAI user configuration structure.
typedef enum __sai_sample_rate sai_sample_rate_t
    Audio sample rate.
typedef enum __sai_word_width sai_word_width_t
    Audio word width.

```

typedef enum *_sai_data_pin_state* sai_data_pin_state_t
sai data pin state definition

typedef enum *_sai_fifo_combine* sai_fifo_combine_t
sai fifo combine mode definition

typedef enum *_sai_transceiver_type* sai_transceiver_type_t
sai transceiver type

typedef enum *_sai_frame_sync_len* sai_frame_sync_len_t
sai frame sync len

typedef struct *_sai_transfer_format* sai_transfer_format_t
sai transfer format

typedef struct *_sai_master_clock* sai_master_clock_t
master clock configurations

typedef struct *_sai_fifo* sai_fifo_t
sai fifo configurations

typedef struct *_sai_bit_clock* sai_bit_clock_t
sai bit clock configurations

typedef struct *_sai_frame_sync* sai_frame_sync_t
sai frame sync configurations

typedef struct *_sai_serial_data* sai_serial_data_t
sai serial data configurations

typedef struct *_sai_transceiver* sai_transceiver_t
sai transceiver configurations

typedef struct *_sai_transfer* sai_transfer_t
SAI transfer structure.

typedef struct *_sai_handle* sai_handle_t

typedef void (*sai_transfer_callback_t)(I2S_Type *base, sai_handle_t *handle, status_t status, void *userData)

SAI transfer callback prototype.

MCUX_SDK_SAI_ALLOW_NULL_FIFO_WATERMARK

Used to control whether SAI_RxSetFifoConfig()/SAI_TxSetFifoConfig() allows a NULL FIFO watermark.

If this macro is set to 0 then SAI_RxSetFifoConfig()/SAI_TxSetFifoConfig() will set the watermark to half of the FIFO's depth if passed a NULL watermark.

MCUX_SDK_SAI_DISABLE_IMPLICIT_CHAN_CONFIG

Disable implicit channel data configuration within SAI_TxSetConfig()/SAI_RxSetConfig().

Use this macro to control whether SAI_RxSetConfig()/SAI_TxSetConfig() will attempt to implicitly configure the channel data. By channel data we mean the startChannel, channelMask, endChannel, and channelNums fields from the sai_transciever_t structure. By default, SAI_TxSetConfig()/SAI_RxSetConfig() will attempt to compute these fields, which may not be desired in cases where the user wants to set them before the call to said functions.

SAI_XFER_QUEUE_SIZE

SAI transfer queue size, user can refine it according to use case.

FSL_SAI_HAS_FIFO_EXTEND_FEATURE

sai fifo feature


```
struct _sai_config
    #include <fsl_sai.h> SAI user configuration structure.
```

Public Members

```
sai_protocol_t protocol
    Audio bus protocol in SAI

sai_sync_mode_t syncMode
    SAI sync mode, control Tx/Rx clock sync

bool mclkOutputEnable
    Master clock output enable, true means master clock divider enabled

sai_bclk_source_t bclkSource
    Bit Clock source

sai_master_slave_t masterSlave
    Master or slave
```

```
struct _sai_transfer_format
    #include <fsl_sai.h> sai transfer format
```

Public Members

```
uint32_t sampleRate_Hz
    Sample rate of audio data

uint32_t bitWidth
    Data length of audio data, usually 8/16/24/32 bits

sai_mono_stereo_t stereo
    Mono or stereo

uint32_t masterClockHz
    Master clock frequency in Hz

uint8_t watermark
    Watermark value

uint8_t channel
    Transfer start channel

uint8_t channelMask
    enabled channel mask value, reference _sai_channel_mask

uint8_t endChannel
    end channel number

uint8_t channelNums
    Total enabled channel numbers

sai_protocol_t protocol
    Which audio protocol used

bool isFrameSyncCompact
    True means Frame sync length is configurable according to bitWidth, false means
    frame sync length is 64 times of bit clock.
```

```
struct _sai_master_clock
    #include <fsl_sai.h> master clock configurations
```

Public Members

bool mclkOutputEnable
master clock output enable

uint32_t mclkHz
target mclk frequency

uint32_t mclkSourceClkHz
mclk source frequency

struct _sai_fifo
#include <fsl_sai.h> sai fifo configurations

Public Members

bool fifoContinueOnError
fifo continues when error occur

sai_fifo_combine_t fifoCombine
fifo combine mode

sai_fifo_packing_t fifoPacking
fifo packing mode

uint8_t fifoWatermark
fifo watermark

struct _sai_bit_clock
#include <fsl_sai.h> sai bit clock configurations

Public Members

bool bclkInputDelay
bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time
.

sai_clock_polarity_t bclkPolarity
bit clock polarity

sai_bclk_source_t bclkSource
bit Clock source

struct _sai_frame_sync
#include <fsl_sai.h> sai frame sync configurations

Public Members

uint8_t frameSyncWidth
frame sync width in number of bit clocks

bool frameSyncEarly
TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame

bool frameSyncGenerateOnDemand
internal frame sync is generated when FIFO warning flag is clear

sai_clock_polarity_t frameSyncPolarity
frame sync polarity

struct *_sai_serial_data*
#include <fsl_sai.h> sai serial data configurations

Public Members

sai_data_pin_state_t dataMode
sai data pin state when slots masked or channel disabled

sai_data_order_t dataOrder
configure whether the LSB or MSB is transmitted first

uint8_t dataWord0Length
configure the number of bits in the first word in each frame

uint8_t dataWordNLength
configure the number of bits in the each word in each frame, except the first word

uint8_t dataWordLength
used to record the data length for dma transfer

uint8_t dataFirstBitShifted
Configure the bit index for the first bit transmitted for each word in the frame

uint8_t dataWordNum
configure the number of words in each frame

uint32_t dataMaskedWord
configure whether the transmit word is masked

struct *_sai_transceiver*
#include <fsl_sai.h> sai transceiver configurations

Public Members

sai_serial_data_t serialData
serial data configurations

sai_frame_sync_t frameSync
ws configurations

sai_bit_clock_t bitClock
bit clock configurations

sai_fifo_t fifo
fifo configurations

sai_master_slave_t masterSlave
transceiver is master or slave

sai_sync_mode_t syncMode
transceiver sync mode

uint8_t startChannel
Transfer start channel

uint8_t channelMask
enabled channel mask value, reference *_sai_channel_mask*

uint8_t endChannel
end channel number

uint8_t channelNums
Total enabled channel numbers

struct __sai_transfer
#include <fsl_sai.h> SAI transfer structure.

Public Members

uint8_t *data
Data start address to transfer.

size_t dataSize
Transfer size.

struct __sai_handle
#include <fsl_sai.h> SAI handle structure.

Public Members

I2S_Type *base
base address

uint32_t state
Transfer status

sai_transfer_callback_t callback
Callback function called at transfer event

void *userData
Callback parameter passed to callback function

uint8_t bitWidth
Bit width for transfer, 8/16/24/32 bits

uint8_t channel
Transfer start channel

uint8_t channelMask
enabled channel mask value, refernece `_sai_channel_mask`

uint8_t endChannel
end channel number

uint8_t channelNums
Total enabled channel numbers

sai_transfer_t saiQueue[(4U)]
Transfer queue storing queued transfer

size_t transferSize[(4U)]
Data bytes need to transfer

volatile uint8_t queueUser
Index for user to queue transfer

volatile uint8_t queueDriver
Index for driver to get the transfer data and size

uint8_t watermark
Watermark value

2.96 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

2.97 SAR_ADC: SAR_ADC Module

`void ADC_GetDefaultConfig(adc_config_t *config)`

This function is used to get available predefined configurations for the ADC initialization.

Parameters

- `config` – Pointer to the ADC configuration structure, please refer to `adc_config_t` for details.

`void ADC_Init(ADC_Type *base, const adc_config_t *config)`

This function is used to initialize the ADC.

Parameters

- `base` – ADC peripheral base address.
- `config` – Pointer to the ADC configuration structure, please refer to `adc_config_t` for details.

```
void ADC_Deinit(ADC_Type *base)
```

This function is used to de-initialize the ADC.

Parameters

- `base` – ADC peripheral base address.

```
static inline void ADC_SetPowerDownMode(ADC_Type *base, bool enable)
```

This function is used to enter or exit power-down mode.

After the release of the reset, the ADC analog module will be kept in power-down mode by default. The power-down mode can be set anytime. However, ADC can enter the power-down mode successfully only after completion of an ongoing conversion (if there is one). In scan mode, the ongoing operation should be aborted manually before or after switching mode. If the power-down mode is entered by setting MCR[PWDN], the process running in the previous mode must be restarted manually (by setting the appropriate START bit in the MCR register) after exiting power-down mode.

Note: After setting the ADC mode, it is recommended to use the function `ADC_GetAdcState` to query whether the ADC has correctly entered the mode.

Parameters

- `base` – ADC peripheral base address.
- `enable` – Indicates whether to enter or exit power-down mode.
 - **true** Request to enter power-down mode.
 - **false** When ADC status is in power-down mode (MSR[ADCSTATUS] = 001b), start ADC transition to IDLE mode.

```
static inline void ADC_SetOperatingClock(ADC_Type *base, adc_clock_frequency_t clockSelect)
```

This function is used to select the ADC operating clock.

Note: Needs to enter power-down mode before changing the ADC internal operating clock.

Parameters

- `base` – ADC peripheral base address.
- `clockSelect` – ADC clock frequency selection, please refer to `adc_clock_frequency_t` for details.

```
static inline adc_state_t ADC_GetAdcState(ADC_Type *base)
```

This function is used to get the ADC state.

Parameters

- `base` – ADC peripheral base address.

Returns

ADC state, for possible states, please refer to `adc_state_t` for details.

```
static inline bool ADC_CheckAutoClockOffEnabled(ADC_Type *base)
```

This function is used to check whether the ADC auto clock-off feature has been enabled or not.

Parameters

- `base` – ADC peripheral base address.

Returns

ADC auto clock-off feature status.

- **true** Auto clock-off feature has been enabled.
- **false** Auto clock-off feature has not been enabled.

static inline uint8_t ADC_GetCurrentConvertedChannelId(ADC_Type *base)

This function is used to get the ID of the channel that is currently being converted.

Parameters

- base – ADC peripheral base address.

Returns

ADC channel ID that is currently being converted.

static inline bool ADC_CheckSelfTestConvInProgress(ADC_Type *base)

This function is used to check whether the self-test conversion is in process or not.

Parameters

- base – ADC peripheral base address.

Returns

Self-test conversion status.

- **true** Self-test conversion is in process.
- **false** Self-test conversion is not in process.

static inline bool ADC_CheckInjectConvInProgress(ADC_Type *base)

This function is used to check whether the inject conversion is in process or not.

Parameters

- base – ADC peripheral base address.

Returns

Inject conversion status.

- **true** Inject conversion is in process.
- **false** Inject conversion is not in process.

static inline bool ADC_CheckInjectConvAborted(ADC_Type *base)

This function is used to check whether the inject conversion has been aborted or not.

Parameters

- base – ADC peripheral base address.

Returns

Inject conversion abort status.

- **true** Injected conversion has been aborted.
- **false** Injected conversion has not been aborted.

static inline bool ADC_CheckNormalConvInProgress(ADC_Type *base)

This function is used to check whether the normal conversion is in process or not.

Parameters

- base – ADC peripheral base address.

Returns

Normal conversion status.

- **true** Normal conversion is in process.
- **false** Normal conversion is not in process.

static inline bool ADC_CheckCalibrationBusy(ADC_Type *base)

This function is used to check whether the ADC is executing calibration or ready for use.

Parameters

- base – ADC peripheral base address.

Returns

Calibration process status.

- **true** ADC is busy in a calibration process.
- **false** ADC is ready for use.

static inline bool ADC_CheckCalibrationFailed(ADC_Type *base)

This function is used to check whether the calibration has failed or passed.

Note: When the user clears the calibration failed status and then reads the status, it will display the calibration passed. At this time, the calibration may not be successful. The user must read the MSR[CALBUSY] bit by function ADC_CheckCalibrationBusy to perform a double check.

Returns

Normal conversion status.

- **true** Calibration failed.
- **false** Calibration passed (must be checked with CALBUSY = 0b).

static inline void ADC_ClearCalibrationFailedFlag(ADC_Type *base)

This function is used to clear the flag of calibration.

Parameters

- base – ADC peripheral base address.

static inline bool ADC_CheckCalibrationSuccessful(ADC_Type *base)

This function is used to check whether the calibration is successful or not.

Parameters

- base – ADC peripheral base address.

Returns

Normal conversion status.

- **true** Calibrated or calibration successful.
- **false** Uncalibrated or calibration unsuccessful.

void ADC_SetConvChainConfig(ADC_Type *base, const *adc_chain_config_t* *config)

This function is used to configure the chain.

Parameters

- base – ADC peripheral base address.
- config – Pointer to the chain configuration structure, please refer to *adc_chain_config_t* for details.

static inline void ADC_EnableSpecificChannelNormalConv(ADC_Type *base, uint8_t
channelIndex)

This function is used to enable the specific ADC channel to execute normal conversion.

Parameters

- base – ADC peripheral base address.

- `channelIndex` – Channel index to enable the normal conversion.

```
static inline void ADC_DisableSpecificChannelNormalConv(ADC_Type *base, uint8_t  
channelIndex)
```

This function is used to disable the specific ADC channel to execute the normal conversion.

Parameters

- `base` – ADC peripheral base address.
- `channelIndex` – Channel index to disable the normal conversion.

```
static inline void ADC_EnableSpecificChannelInjectConv(ADC_Type *base, uint8_t channelIndex)
```

This function is used to enable the specific ADC channel to execute the inject conversion.

Parameters

- `base` – ADC peripheral base address.
- `channelIndex` – Channel index to enable the inject conversion.

```
static inline void ADC_DisableSpecificChannelInjectConv(ADC_Type *base, uint8_t channelIndex)
```

This function is used to disable the specific ADC channel to execute the inject conversion.

Parameters

- `base` – ADC peripheral base address.
- `channelIndex` – Channel index to disable the inject conversion.

```
static inline void ADC_SetConvMode(ADC_Type *base, adc_conv_mode_t convMode)
```

This function is used to set the ADC conversion mode.

Note: Before setting the conversion mode, users need to check whether the ADC is in the idle status through the function `ADC_GetAdcState`.

Parameters

- `base` – ADC peripheral base address.
- `convMode` – ADC conversion mode, please refer to `adc_conv_mode_t` for details.

```
static inline void ADC_StartConvChain(ADC_Type *base, adc_conv_mode_t convMode)
```

This function is used to start the ADC conversion chain to execute the conversion.

Note: Normal conversion supports two conversion modes, one is one-shot conversion mode, and the other is scan conversion mode. Normal conversion should usually be used to convert analog samples most of the time in an application, unless there is a special need. Inject conversion has a higher priority than normal conversion and it runs in one-shot mode only, it can be started in IDLE condition or when normal conversion is in process.

Parameters

- `base` – ADC peripheral base address.
- `convMode` – Pointer to the ADC conversion chain, please refer to `adc_conv_mode_t` for details.

static inline void ADC_StopConvChain(ADC_Type *base)

This function is used to stop scan in normal conversion scan operation mode.

Note: In scan operation mode, the MCR[NSTART] field remains high after setting. Clearing this field in scan operation mode causes the current chain conversion to finish, then stop the scan.

Parameters

- base – ADC peripheral base address.

static inline void ADC_AbortCurrentConvChain(ADC_Type *base)

This function is used to abort the conversion chain.

Abort the current chain of conversions by setting MCR[ABORTCHAIN]. In that case, the behavior of the ADC depends on MCR[MODE] (one-shot/scan conversion modes). In one-shot mode, MSR[NSTART] is automatically reset together with MCR[ABORTCHAIN], an end-of-chain interrupt is not generated in the case of an abort chain. In scan mode, a new chain is started. The end-of-conversion interrupt of the current aborted conversion is not generated but an end-of-chain interrupt is generated.

Note: Setting this field in an IDLE state (for example, no normal/inject conversion is in process) has no effect. In this case, the MCR[ABORTCHAIN] field is reset immediately.

Parameters

- base – ADC peripheral base address.

static inline void ADC_AbortCurrentConv(ADC_Type *base)

This function is used to abort the conversion channel.

Abort the current conversion and immediately start the conversion of the next channel of the chain. In the case of an abort operation, MSR[NSTART/JSTART] remains set if not in the last channel of the chain, and MCR[ABORT] is reset as soon as the channel is aborted. The end-of-conversion interrupt corresponds to the aborted channel is not generated. This behavior is true for normal or inject conversion modes. If the last channel of a chain is aborted, and the end-of-chain is reported, then an end-of-chain interrupt will be generated.

Note: This conversion can not abort while the self-test channel conversion is in process.

Parameters

- base – ADC peripheral base address.

static inline void ADC_EnableConvInt(ADC_Type *base, uint32_t mask)

This function is used to enable the ADC end-of-conversion and end-of-chain interrupts.

Parameters

- base – ADC peripheral base address.
- mask – Mask value to enable the ADC end-of-conversion and end-of-chain interrupts, please refer to _adc_conv_int_enable for details.

static inline void ADC_DisableConvInt(ADC_Type *base, uint32_t mask)

This function is used to disable the ADC end-of-conversion and end-of-chain interrupts.

Parameters

- `base` – ADC peripheral base address.
- `mask` – Mask value to disable the ADC end-of-conversion and end-of-chain interrupts, please refer to `_adc_conv_int_enable` for details.

static inline uint32_t ADC_GetConvIntStatus(ADC_Type *base)

This function is used to get the ADC end-of-conversion and end-of-chain interrupts status.

Parameters

- `base` – ADC peripheral base address.

Returns

ADC end-of-conversion and end-of-chain interrupts status mask.

static inline void ADC_ClearConvIntStatus(ADC_Type *base, uint32_t mask)

This function is used to clear the ADC end-of-conversion and end-of-chain interrupts status.

Parameters

- `base` – ADC peripheral base address.
- `mask` – Mask value for flags to be cleared, please refer to `_adc_conv_int_flag` for details.

static inline void ADC_EnableSpecificConvChannelInt(ADC_Type *base, uint8_t channelIndex)

This function is used to enable the specific ADC channel end-of-conversion interrupt.

Note: This function can only turn on the interrupt of a specific channel, if the interrupt occurs, the user also needs to turn on the global end-of-conversion interrupt by using function `ADC_EnableConvInt`

Parameters

- `base` – ADC peripheral base address.
- `channelIndex` – Channel index to enable the end-of-conversion interrupt.

static inline void ADC_DisableSpecificConvChannelInt(ADC_Type *base, uint8_t channelIndex)

This function is used to disable the specific ADC channel end-of-conversion interrupt.

Parameters

- `base` – ADC peripheral base address.
- `channelIndex` – Channel index to disable the end-of-conversion interrupt.

static inline bool ADC_CheckSpecificConvChannelInt(ADC_Type *base, uint8_t channelIndex)

This function is used to check whether the specific conversion channel's end-of-conversion interrupt has been occurred.

Parameters

- `base` – ADC peripheral base address.
- `channelIndex` – Channel index to check the end-of-conversion interrupt status.

Returns

Channel end-of-conversion interrupt status flag of the specific channel.

- **true** Channel end-of-conversion interrupt has been occurred.
- **false** Channel end-of-conversion interrupt has not been occurred.

```
static inline void ADC__ClearSpecificConvChannelInt(ADC_Type *base, uint8_t channelIndex)
```

This function is used to clear specific channel end-of-conversion interrupt flag.

Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to clear the end-of-conversion interrupt flag.

```
static inline void ADC__EnableDmaTransfer(ADC_Type *base)
```

This function is used to enable the DMA transfer function.

Note: This function is a master switch used to control whether the data converted by the ADC is transmitted through DMA. If the user configures DMA to transmit the conversion data of the channel during the chain channel configuration process, then the main switch of the DMA transmission must be turned on, otherwise, data transmission cannot be performed.

Parameters

- base – ADC peripheral base address.

```
static inline void ADC__DisableDmaTransfer(ADC_Type *base)
```

This function is used to disable the DMA transfer function.

Parameters

- base – ADC peripheral base address.

```
static inline void ADC__EnableSpecificConvChannelDmaTransfer(ADC_Type *base, uint8_t  
                                                            channelIndex)
```

This function is used to enable the specific ADC conversion channel's DMA transfer function.

Note: This function can only turn on the DMA transfer of a specific channel, before using the DMA transfer, the user needs to turn on the global DMA transfer by using the function `ADC_EnableDmaTransfer`.

Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to enable the DMA transfer function.

```
static inline void ADC__DisableSpecificConvChannelDmaTransfer(ADC_Type *base, uint8_t  
                                                            channelIndex)
```

This function is used to disable the specific ADC conversion channel's DMA transfer function.

Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to disable the DMA transfer function.

```
static inline void ADC__EnableSpecificConvChannelPresample(ADC_Type *base, uint8_t  
                                                            channelIndex)
```

This function is used to enable the specific ADC conversion channel's pre-sample function.

Parameters

- base – ADC peripheral base address.

- `channelIndex` – Channel index to enable the pre-sample function.

```
static inline void ADC__DisableSpecificConvChannelPresample(ADC_Type *base, uint8_t  
                                                           channelIndex)
```

This function is used to disable the specific ADC conversion channel's pre-sample function.

Parameters

- `base` – ADC peripheral base address.
- `channelIndex` – Channel index to disable the pre-sample function.

```
void ADC__SetAnalogWdgConfig(ADC_Type *base, const adc_wdg_config_t *config)
```

This function is used to configure the analog watchdog.

The analog watchdogs are used to monitor the conversion result to see if it is within defined limits, specified by a higher and a lower threshold value. After the conversion of the selected channel, a comparison is performed between the converted value and the threshold values. If the converted value is outside the threshold values, then a corresponding threshold violation interrupt is generated.

Parameters

- `base` – ADC peripheral base address.
- `config` – Pointer to the analog watchdog configuration structure, please refer to `adc_wdg_config_t` for details.

```
static inline void ADC__EnableSpecificConvChannelAnalogWdg(ADC_Type *base, uint8_t  
                                                           channelIndex)
```

This function is used to enable the specific ADC conversion channel's analog watchdog function.

Parameters

- `base` – ADC peripheral base address.
- `channelIndex` – Conversion channel index to enable the analog watchdog function.

```
static inline void ADC__DisableSpecificConvChannelAnalogWdg(ADC_Type *base, uint8_t  
                                                           channelIndex)
```

This function is used to disable the specific ADC conversion channel's analog watchdog function.

Parameters

- `base` – ADC peripheral base address.
- `channelIndex` – Conversion channel index to disable the analog watchdog function.

```
static inline bool ADC__CheckSpecificConvChannelOutOfRange(ADC_Type *base, uint8_t  
                                                           channelIndex)
```

This function is used to check whether the specific conversion channel's converted data is out of range.

Parameters

- `base` – ADC peripheral base address.
- `channelIndex` – Channel index to check the converted data out of range status.

Returns

Converted data out of range status of the specific conversion channel.

- **true** Channel converted data is out of range.

- **false** Channel converted data is in range.

```
static inline void ADC__ClearSpecificConvChannelOutOfRange(ADC_Type *base, uint8_t  
                                                         channelIndex)
```

This function is used to clear the specific conversion channel's converted data out-of-range flag.

Parameters

- `base` – ADC peripheral base address.
- `channelIndex` – Channel index to clear the converted data out of range flag.

```
static inline void ADC__EnableWdgThresholdInt(ADC_Type *base, uint32_t mask)
```

This function is used to enable the analog watchdog threshold low or/and high interrupts.

Parameters

- `base` – ADC peripheral base address.
- `mask` – Mask value to enable the analog watchdog threshold low or/and high interrupts, please refer to `_adc_wdg_threshold_int_enable` for details.

```
static inline void ADC__DisableWdgThresholdInt(ADC_Type *base, uint32_t mask)
```

This function is used to disable the analog watchdog threshold low or/and high interrupts.

Parameters

- `base` – ADC peripheral base address.
- `mask` – Mask value to disable the analog watchdog threshold low or/and high interrupts, please refer to `_adc_wdg_threshold_int_enable` for details.

```
static inline uint32_t ADC__GetWdgThresholdIntStatus(ADC_Type *base)
```

This function is used to get the analog watchdog threshold interrupts status.

Parameters

- `base` – ADC peripheral base address.

Returns

Analog watchdog threshold interrupts status mask.

```
static inline void ADC__ClearWdgThresholdIntStatus(ADC_Type *base, uint32_t mask)
```

This function is used to clear the analog watchdog threshold low or/and high interrupts status.

Parameters

- `base` – ADC peripheral base address.
- `mask` – Mask value for flags to be cleared, please refer to `_adc_wdg_threshold_int_flag` for details.

```
bool ADC__DoCalibration(ADC_Type *base, const adc_calibration_config_t *config)
```

This function is used to do the calibration.

The calibration is used to reduce or eliminate the various errors. In the calibration process, the calibration values for offset, gain, and capacitor mismatch are obtained. These calibration values (except gain calibration) are used in a result post-processing step to reduce or eliminate the various errors contribution effects. The gain calibration is used during the sample phase to define the additional charge to be loaded in order to compensate for the gain failure. Calibration must be performed after every power-up reset and whenever required in runtime operation. It is also recommended to run calibration if the operating conditions (particularly `VrefH`) change. Never apply functional reset during the calibration process. If applied, calibration must be rerun after exiting a reset condition; otherwise, the calibration-generated values and conversion results may be unspecified.

Note: This function executes a calibration sequence, it is recommended to run this sequence before using the ADC converter. The maximum clock frequency for the calibration is 40 MHz. Before calling this function, the user needs to ensure that the input clock is within 40MHz. The results of individual steps are also updated in the CALSTAT register (CALSTAT[STAT_n]). The result of the last failed step is dynamically updated in the same register.

Parameters

- `base` – ADC peripheral base address.
- `config` – Pointer to the calibration configuration structure, please refer to `adc_calibration_config_t` for details.

Returns

Status whether calibration is running passed or failed.

- **true** Calibration successful.
- **false** Calibration unsuccessful.

```
void ADC_SetUserOffsetAndGainConfig(ADC_Type *base, const adc_user_offset_gain_config_t *config)
```

This function is used to configure the user gain and offset.

Parameters

- `base` – ADC peripheral base address.
- `config` – Pointer to the user offset and gain configuration structure, please refer to `adc_user_offset_gain_config_t` for details.

```
void ADC_SetSelfTestConfig(ADC_Type *base, const adc_self_test_config_t *config)
```

This function is used to configure the ADC self-test.

The self-test is used to check at regular intervals whether ADC is operating correctly. When self-test is enabled, ADC automatically checks its components and flags any errors it finds. The test can be enabled to check the supply voltage (VDD), reference voltage (VrefH), and calibrated values.

Note: Before calling this function, please ensure the functional conversion is one-shot conversion mode normal conversion type and the operating clock are equal to bus frequency. ADC self-test should be run with MCR[ADCLKSE] bit set to 1. Self-test with ADCLKSE bit set to 0 can give erroneous results.

Parameters

- `base` – ADC peripheral base address.
- `config` – Pointer to the self-test configuration structure, please refer to `adc_self_test_config_t` for details.

```
void ADC_SetSelfTestWdgConfig(ADC_Type *base, const adc_self_test_wdg_config_t *config)
```

This function is used to configure the ADC self-test watchdog.

Parameters

- `base` – ADC peripheral base address.
- `config` – Pointer to the self-test watchdog configuration structure, please refer to `adc_self_test_wdg_config_t` for details.

```
void ADC_GetCalibrationLastFailedTestResult(ADC_Type *base, int16_t *result)
```

This function is used to get the test result for the last failed test.

Parameters

- base – ADC peripheral base address.
- result – Points to a 16-bit signed variable, and it is used to store the test result for the last failing test.

```
static inline uint16_t ADC_GetCalibrationStepsStatus(ADC_Type *base)
```

This function is used to get the status of the calibration steps.

Note: The status of calibration steps (step 0 to step 12) is stored in the CALSTAT register, and only the lower 12 bits are available in the returned result.

Parameters

- base – ADC peripheral base address.

Returns

ADC self-test interrupt status mask.

```
static inline void ADC_EnableSelfTest(ADC_Type *base)
```

This function is used to enable the ADC self-test.

Decides whether to enable the ADC self-test. The self-test test is enabled by setting STCR2[EN], this field must be set before starting normal conversion and should not be changed while the conversion is in process. This field should only be reset after the end-of-conversion of the last self-test channel has been received.

Note: ADC self-test should be run with MCR[ADCLKSE] bit set to 1. Self-test with ADCLKSE bit set to 0 can give erroneous results. In the case of Inject Conversion mode, test channel conversion is not performed. It is performed only during normal conversions.

Parameters

- base – ADC peripheral base address.

```
static inline void ADC_DisableSelfTest(ADC_Type *base)
```

This function is used to disable the ADC self-test.

Parameters

- base – ADC peripheral base address.

```
static inline void ADC_EnableSelfTestWdgThreshold(ADC_Type *base,  
                                                  adc_self_test_wdg_threshold_t wdgID)
```

This function is used to enable the ADC self-test watchdog threshold for algorithm S step 0/1/2 and algorithm C.

The user can pass `kADC_SelfTestWdgThresholdForAlgSStep0` as parameter 'wdgID' to enable the self-test watchdog threshold function for algorithm S step 0; pass `kADC_SelfTestWdgThresholdForAlgSStep1Integer` as parameter 'wdgID' to enable the self-test watchdog threshold function for algorithm S step 1; pass `kADC_SelfTestWdgThresholdForAlgSStep2` as parameter 'wdgID' to enable the self-test watchdog threshold function for algorithm S step 2; pass `kADC_SelfTestWdgThresholdForAlgCStep0` as parameter 'wdgID' to enable the self-test watchdog threshold function for algorithm C; Other enumerations in `adc_self_test_wdg_threshold_t` have no use.

Parameters

- base – ADC peripheral base address.
- wdgID – Watchdog threshold index to enable, please refer to `adc_self_test_wdg_threshold_t` for details.

```
static inline void ADC_DisableSelfTestWdgThreshold(ADC_Type *base,
                                                  adc_self_test_wdg_threshold_t wdgID)
```

This function is used to disable the ADC self-test watchdog threshold for algorithm S step 0/1/2 and algorithm C.

The user can pass `kADC_SelfTestWdgThresholdForAlgSStep0` as parameter 'wdgID' to disable the self-test watchdog threshold function for algorithm S step 0; pass `kADC_SelfTestWdgThresholdForAlgSStep1Integer` as parameter 'wdgID' to disable the self-test watchdog threshold function for algorithm S step 1; pass `kADC_SelfTestWdgThresholdForAlgSStep2` as parameter 'wdgID' to disable the self-test watchdog threshold function for algorithm S step 2; pass `kADC_SelfTestWdgThresholdForAlgCStep0` as parameter 'wdgID' to disable the self-test watchdog threshold function for algorithm C; Other enumerations in `adc_self_test_wdg_threshold_t` have no use.

Parameters

- base – ADC peripheral base address.
- wdgID – Watchdog threshold index to disable, please refer to `adc_self_test_wdg_threshold_t` for details.

```
static inline void ADC_EnableSelfTestWdgTimer(ADC_Type *base, adc_alg_type_t
                                              wdgTimerType)
```

This function is used to enable the ADC self-test watchdog timer for algorithm S or/and algorithm C.

The user can pass `kADC_SelfTestForAlgS` as parameter 'wdgTimerType' to enable the watchdog timer for algorithm S; pass `kADC_SelfTestForAlgC` as parameter 'wdgTimerType' to enable the watchdog timer for algorithm C; pass `kADC_SelfTestForAlgSAndC` as parameter 'wdgTimerType' to enable the watchdog timer for algorithm S and C.

Parameters

- base – ADC peripheral base address.
- wdgTimerType – Watchdog timer type to enable, please refer to `adc_alg_type_t` for details.

```
static inline void ADC_DisableSelfTestWdgTimer(ADC_Type *base, adc_alg_type_t
                                              wdgTimerType)
```

This function is used to disable the ADC self-test watchdog timer.

The user can pass `kADC_SelfTestForAlgS` as parameter 'wdgTimerType' to disable the watchdog timer for algorithm S; pass `kADC_SelfTestForAlgC` as parameter 'wdgTimerType' to disable the watchdog timer for algorithm C; pass `kADC_SelfTestForAlgSAndC` as parameter 'wdgTimerType' to disable the watchdog timer for algorithm S and C.

Parameters

- base – ADC peripheral base address.
- wdgTimerType – Watchdog timer type to disable, please refer to `adc_alg_type_t` for details.

```
static inline void ADC_SetSelfTestWdgTimerVal(ADC_Type *base, adc_wdg_timer_val_t
                                              wdgTimerVal)
```

This function is used to set the ADC self-test watchdog timer value.

Parameters

- base – ADC peripheral base address.
- wdgTimerVal – Watchdog timer value, please refer to `adc_wdg_timer_val_t` for details.

```
static inline uint16_t ADC_GetSelfTestChannelConvFailedData(ADC_Type *base,  
                                                           adc_self_test_wdg_threshold_t  
                                                           type)
```

This function is used to get the ADC self-test channel converted data when ERR_S0/ERR_S1_INTEGER/ ERR_S1_FRACTION/ERR_S2/ERR_C occurred.

Note: The user can pass `kADC_SelfTestWdgThresholdForAlgSSStep0` as parameter 'type' to get the converted data when ERR_S0 occurred; pass `kADC_SelfTestWdgThresholdForAlgSSStep1Integer` as parameter 'type' to get the converted data when ERR_S1_INTEGER occurred; pass `kADC_SelfTestWdgThresholdForAlgSSStep1Fraction` as parameter 'type' to get the converted data when ERR_S1_FRACTION occurred; pass `kADC_SelfTestWdgThresholdForAlgSSStep2` as parameter 'type' to get the converted data when ERR_S2 occurred; pass `kADC_SelfTestWdgThresholdForAlgCStep0` as parameter 'type' to get the converted data when ERR_C occurred; Other enumerations in `adc_self_test_wdg_threshold_t` have no use.

Parameters

- base – ADC peripheral base address.
- type – Watchdog threshold index to get the ADC self-test channel converted data, please refer to `adc_self_test_wdg_threshold_t` for details.

```
static inline void ADC_EnableSelfTestInt(ADC_Type *base, uint32_t mask)
```

This function is used to enable the ADC self-test-related interrupts.

Note: Watchdog timer feature is applicable only for scan operation mode and not for one-shot operation mode.

Parameters

- base – ADC peripheral base address.
- mask – Mask value to enable the ADC self-test related interrupts, please refer to `_adc_self_test_int_enable` for details.

```
static inline void ADC_DisableSelfTestInt(ADC_Type *base, uint32_t mask)
```

This function is used to disable the ADC self-test interrupt.

Parameters

- base – ADC peripheral base address.
- mask – Mask value to disable the ADC self-test related interrupts, please refer to `_adc_self_test_int_enable` for details.

```
static inline uint32_t ADC_GetSelfTestIntStatus(ADC_Type *base)
```

This function is used to get the ADC self-test interrupts status.

Parameters

- base – ADC peripheral base address.

Returns

ADC self-test related interrupts status mask.

```
static inline void ADC_ClearSelfTestIntStatus(ADC_Type *base, uint32_t mask)
```

This function is used to clear the ADC self-test interrupts status.

Parameters

- `base` – ADC peripheral base address.
- `mask` – Mask value for flags to be cleared, please refer to `_adc_self_test_int_flag` for details.

```
bool ADC_GetChannelConvResult(ADC_Type *base, adc_conv_result_t *result, uint8_t
                             channelIndex)
```

This function is used to get the specific ADC channel's conversion result.

CDR[VALID] indicates whether a new conversion is available, this field is automatically reset to 0 when the data is read. CDR[OVERW] Indicates whether the previous conversion data was overwritten without having been read, in which case the overwritten data is lost.

Parameters

- `base` – SAR ADC peripheral base address.
- `result` – Pointer to SAR ADC channels conversion result structure, please refer to `adc_conv_result_t` for details.
- `channelIndex` – Channel index to get the conversion result.

Returns

Indicates whether the acquisition of the specific channel conversion result is successful or not.

- **true** Obtaining the specific channel conversion result successfully, and the conversion result is stored in the input parameter `result`.
- **false** Obtaining the specific channel conversion result failed.

```
bool ADC_GetSelfTestChannelConvData(ADC_Type *base, adc_self_test_conv_result_t *result)
```

This function is used to get the test channel converted data when algorithm S step 0, algorithm S step 2, or algorithm C step executes.

Parameters

- `base` – ADC peripheral base address.
- `result` – Pointer to the SAR ADC self-test channel conversion result structure, please refer to `adc_self_test_conv_result_t` for details.

Returns

Indicates whether the acquisition of the self-test channel conversion result is successful or not.

- **true** Obtaining the self-test channel conversion result successfully, and the conversion result is stored in the input parameter 'result'.
- **false** Obtaining the self-test channel conversion result failed.

```
bool ADC_GetSelfTestChannelConvDataForAlgSStep1(ADC_Type *base,
                                                  adc_self_test_conv_result_t *result)
```

This function is used to get the test channel converted data when algorithm S step 1 executes.

Parameters

- `base` – ADC peripheral base address.

- **result** – Pointer to the SAR ADC self-test channel conversion result structure, please refer to `adc_self_test_conv_result_t` for details.

Returns

Indicates whether the acquisition of the self-test channel conversion result is successful or not.

- **true** Obtaining the self-test channel conversion result successfully, and the conversion result is stored in the input parameter 'result'.
- **false** Obtaining the self-test channel conversion result failed.

FSL_SAR_ADC_DRIVER_VERSION

SAR ADC driver version 2.3.0.

enum _adc_conv_int_enable

This enumeration provides the mask for the ADC end-of-conversion and end-of-chain interrupts enabling.

Values:

enumerator kADC_NormalConvChainEndIntEnable

Enable end of normal chain conversion interrupt.

enumerator kADC_NormalConvEndIntEnable

Enable end of normal conversion interrupt.

enumerator kADC_InjectConvChainEndIntEnable

Enable end of inject chain conversion interrupt.

enumerator kADC_InjectConvEndIntEnable

Enable end of inject conversion interrupt.

enum _adc_wdg_threshold_int_enable

This enumeration provides the mask for the ADC analog watchdog threshold interrupts enabling.

Values:

enumerator kADC_wdg0LowThresholdIntEnable

Enable watchdog 0 low threshold interrupt.

enumerator kADC_wdg0HighThresholdIntEnable

Enable watchdog 0 high threshold interrupt.

enumerator kADC_wdg1LowThresholdIntEnable

Enable watchdog 1 low threshold interrupt.

enumerator kADC_wdg1HighThresholdIntEnable

Enable watchdog 1 high threshold interrupt.

enumerator kADC_wdg2LowThresholdIntEnable

Enable watchdog 2 low threshold interrupt.

enumerator kADC_wdg2HighThresholdIntEnable

Enable watchdog 2 high threshold interrupt.

enumerator kADC_wdg3LowThresholdIntEnable

Enable watchdog 3 low threshold interrupt.

enumerator kADC_wdg3HighThresholdIntEnable

Enable watchdog 3 high threshold interrupt.

enumerator kADC__wdg4LowThresholdIntEnable
Enable watchdog 4 low threshold interrupt.

enumerator kADC__wdg4HighThresholdIntEnable
Enable watchdog 4 high threshold interrupt.

enumerator kADC__wdg5LowThresholdIntEnable
Enable watchdog 5 low threshold interrupt.

enumerator kADC__wdg5HighThresholdIntEnable
Enable watchdog 5 high threshold interrupt.

enumerator kADC__wdg6LowThresholdIntEnable
Enable watchdog 6 low threshold interrupt.

enumerator kADC__wdg6HighThresholdIntEnable
Enable watchdog 6 high threshold interrupt.

enumerator kADC__wdg7LowThresholdIntEnable
Enable watchdog 7 low threshold interrupt.

enumerator kADC__wdg7HighThresholdIntEnable
Enable watchdog 7 high threshold interrupt.

enum _adc_self_test_int_enable

This enumeration provides the mask for the ADC self-test related interrupts enabling.

Values:

enumerator kADC__AlgSStep0ErrIntEnable
Enable self-test algorithm S step0 error interrupt.

enumerator kADC__AlgSStep1ErrIntEnable
Enable self-test algorithm S step1 error interrupt.

enumerator kADC__AlgSStep2ErrIntEnable
Enable self-test algorithm S step2 error interrupt.

enumerator kADC__AlgCErrIntEnable
Enable self-test algorithm C error interrupt.

enumerator kADC__AlgSEndIntEnable
Enable self-test algorithm S end interrupt.

enumerator kADC__AlgCEndIntEnable
Enable self-test algorithm C end interrupt.

enumerator kADC__ConvEndIntEnable
Enable self-test conversion end interrupt.

enumerator kADC__WdgTimeErrIntEnable
Enable watchdog time error interrupt.

enumerator kADC__WdgSequenceErrIntEnable
Enable watchdog sequence error interrupt.

enum _adc_conv_int_flag

This enumeration provides the mask for the ADC end-of-conversion and end-of-chain interrupts flag.

Values:

enumerator kADC__NormalConvChainEndIntFlag
Indicates whether the end of normal chain conversion interrupt has occurred.

enumerator kADC_NormalConvEndIntFlag

Indicates whether the end of conversion interrupt has occurred.

enumerator kADC_InjectConvChainEndIntFlag

Indicates whether the end of inject chain conversion interrupt has occurred.

enumerator kADC_InjectConvEndIntFlag

Indicates whether the end of inject conversion interrupt has occurred.

enum _adc_wdg_threshold_int_flag

This enumeration provides the mask for the ADC analog watchdog threshold interrupts flag.

Values:

enumerator kADC_wdg0LowThresholdIntFlag

Indicates whether the watchdog 0 low threshold interrupt has occurred.

enumerator kADC_wdg0HighThresholdIntFlag

Indicates whether the watchdog 0 high threshold interrupt has occurred.

enumerator kADC_wdg1LowThresholdIntFlag

Indicates whether the watchdog 1 low threshold interrupt has occurred.

enumerator kADC_wdg1HighThresholdIntFlag

Indicates whether the watchdog 1 high threshold interrupt has occurred.

enumerator kADC_wdg2LowThresholdIntFlag

Indicates whether the watchdog 2 low threshold interrupt has occurred.

enumerator kADC_wdg2HighThresholdIntFlag

Indicates whether the watchdog 2 high threshold interrupt has occurred.

enumerator kADC_wdg3LowThresholdIntFlag

Indicates whether the watchdog 3 low threshold interrupt has occurred.

enumerator kADC_wdg3HighThresholdIntFlag

Indicates whether the watchdog 3 high threshold interrupt has occurred.

enumerator kADC_wdg4LowThresholdIntFlag

Indicates whether the watchdog 4 low threshold interrupt has occurred.

enumerator kADC_wdg4HighThresholdIntFlag

Indicates whether the watchdog 4 high threshold interrupt has occurred.

enumerator kADC_wdg5LowThresholdIntFlag

Indicates whether the watchdog 5 low threshold interrupt has occurred.

enumerator kADC_wdg5HighThresholdIntFlag

Indicates whether the watchdog 5 high threshold interrupt has occurred.

enumerator kADC_wdg6LowThresholdIntFlag

Indicates whether the watchdog 6 low threshold interrupt has occurred.

enumerator kADC_wdg6HighThresholdIntFlag

Indicates whether the watchdog 6 high threshold interrupt has occurred.

enumerator kADC_wdg7LowThresholdIntFlag

Indicates whether the watchdog 7 low threshold interrupt has occurred.

enumerator kADC_wdg7HighThresholdIntFlag

Indicates whether the watchdog 7 high threshold interrupt has occurred.

enum _adc_self_test_int_flag

This enumeration provides the mask for the ADC self-test-related interrupts flag.

Values:

enumerator kADC__AlgSStep0ErrIntFlag

Indicates whether the self-test algorithm S step0 error interrupt has occurred.

enumerator kADC__AlgSStep1ErrIntFlag

Indicates whether the self-test algorithm S step1 error interrupt has occurred.

enumerator kADC__AlgSStep2ErrIntFlag

Indicates whether the self-test algorithm S step2 error interrupt has occurred.

enumerator kADC__AlgCErrIntFlag

Indicates whether the self-test algorithm C error interrupt has occurred.

enumerator kADC__AlgSEndIntFlag

Indicates whether the algorithm S end interrupt has completed.

enumerator kADC__AlgCEndIntFlag

Indicates whether the algorithm C end interrupt has completed.

enumerator kADC__SelfTestConvEndIntFlag

Indicates whether the self-test end-of-conversion interrupt has completed.

enumerator kADC__OverWriteErrIntFlag

Indicates whether the overwrite error interrupt has occurred.

enumerator kADC__WdgTimeErrIntFlag

Indicates whether the watchdog time error interrupt has occurred.

enumerator kADC__WdgSequenceErrIntFlag

Indicates whether the watchdog sequence error interrupt has occurred.

enum _adc_ext_trig

This enumeration provides the selection of the ADC external trigger type.

Values:

enumerator kADC__ExtTrigDisable

Normal trigger input does not start a conversion.

enumerator kADC__ExtTrigFallingEdge

Normal trigger (falling edge) input starts a conversion.

enumerator kADC__ExtTrigRisingEdge

Normal trigger (rising edge) input starts a conversion.

enum _adc_state

This enumeration provides the selection of the ADC state.

Values:

enumerator kADC__AdcIdle

Indicates the ADC is in the IDLE state.

enumerator kADC__AdcPowerdown

Indicates the ADC is in the power-down state.

enumerator kADC__AdcWait

Indicates the ADC is in the wait state.

enumerator kADC_AdcBusyInCalibration

Indicates the ADC is in the calibration busy state.

enumerator kADC_AdcSample

Indicates the ADC is in the sample state.

enumerator kADC_AdcConv

Indicates the ADC is in the conversion state.

enum _adc_conv_mode

This enumeration provides the selection of the ADC conversion mode, including normal conversion one-shot mode, normal conversion scan mode, and inject conversion one-shot mode.

Values:

enumerator kADC_NormalConvOneShotMode

Normal conversion one-shot mode.

enumerator kADC_NormalConvScanMode

Normal conversion scan mode.

enumerator kADC_InjectConvOneShotMode

Inject conversion one-shot mode.

enum _adc_clock_frequency

This enumeration provides the selection of the ADC operating clock frequency, including half-bus frequency and full bus frequency.

Values:

enumerator kADC_HalfBusFrequency

Half of bus clock frequency.

enumerator kADC_FullBusFrequency

Equal to bus clock frequency.

enum _adc_conv_data_align

This enumeration provides the selection of the ADC conversion data alignment, including the right alignment and left alignment.

Values:

enumerator kADC_ConvDataRightAlign

Conversion data is right aligned.

enumerator kADC_ConvDataLeftAlign

Conversion data is left aligned.

enum _adc_presample_voltage_src

This enumeration provides the selection of the ADC internal analog input voltage sources for pre-sample, including DVDD0P8/2, AVDD1P8/4, VREFL_1p8 and VREFH_1p8.

Values:

enumerator kADC_PresampleVoltageSrcVREL

Use VREL as pre-sample voltage source.

enumerator kADC_PresampleVoltageSrcVREH

Use VREH as pre-sample voltage source.

enum _adc_dma_request_clear_src

This enumeration provides the selection of the DMA request clear sources, including clear by acknowledgment from the DMA controller and clear on a read of the data register.

Values:

enumerator kADC_DMAResultClearByAck

DMA request cleared by acknowledgment from DMA controller.

enumerator kADC_DMAResultClearOnRead

DMA request cleared on a read of the data register.

enum _adc_average_sample_numbers

This enumeration provides the selection of the ADC calibration averaging sample numbers, including 16, 32, 128, and 512 averaging samples.

Values:

enumerator kADC_AverageSampleNumbers16

Use 16 averaging samples during calibration.

enumerator kADC_AverageSampleNumbers32

Use 32 averaging samples during calibration.

enumerator kADC_AverageSampleNumbers128

Use 128 averaging samples during calibration.

enumerator kADC_AverageSampleNumbers512

Use 512 averaging samples during calibration.

enum _adc_sample_time

This enumeration provides the selection of the ADC sample time of calibration conversions, including 22, 8, 16 and 32 cycles of ADC_CLK.

Values:

enumerator kADC_SampleTime22

Use 22 cycles of ADC_CLK as sample time of calibration conversions.

enumerator kADC_SampleTime8

Use 8 cycles of ADC_CLK as sample time of calibration conversions.

enumerator kADC_SampleTime16

Use 16 cycles of ADC_CLK as sample time of calibration conversions.

enumerator kADC_SampleTime32

Use 32 cycles of ADC_CLK as sample time of calibration conversions.

enum _adc_wdg_threshold_int

This enumeration provides the selection of the ADC analog watchdog threshold low and high interrupt enable.

Values:

enumerator kADC_LowHighThresholdIntDisable

Enable the ADC analog watchdog low and high threshold interrupts.

enumerator kADC_LowThresholdIntEnable

Enable the ADC analog watchdog low threshold interrupt.

enumerator kADC_HighThresholdIntEnable

Enable the ADC analog watchdog high threshold interrupt.

enumerator kADC_LowHighThresholdIntEnable

Enable the ADC analog watchdog low and high threshold interrupts.

enum _adc_alg_type

This enumeration provides the selection of the ADC self-test algorithm type, including algorithm S, algorithm C and algorithm S and C.

Note: The meaning of enumeration member 'kADC_SelfTestForAlgSAndC' in different conversion modes is different, in the one-shot conversion mode, it means executing algorithm S; in the scan conversion mode, it means executing algorithm S and C.

Values:

enumerator kADC_SelfTestForAlgS

Use algorithm S for self-test.

enumerator kADC_SelfTestForAlgC

Use algorithm C for self-test.

enumerator kADC_SelfTestForAlgSAndC

Use algorithm S for one-shot conversion mode self-test, use algorithm S and algorithm C for scan conversion mode self-test.

enum _adc_self_test_wdg_threshold

This enumeration provides the selection of the ADC self-test watchdog thresholds for algorithm S step 0 - 2, and watchdog thresholds for algorithm C step 0 and step x (x = 1 - 11).

Values:

enumerator kADC_SelfTestWdgThresholdForAlgSStep0

Self-test watchdog threshold for the algorithm S step 0.

enumerator kADC_SelfTestWdgThresholdForAlgSStep1

Self-test watchdog threshold for the algorithm S step 1 fraction part.

enumerator kADC_SelfTestWdgThresholdForAlgSStep2

Self-test watchdog threshold for the algorithm S step 2.

enumerator kADC_SelfTestWdgThresholdForAlgCStep0

Self-test watchdog threshold for the algorithm C step 0.

enumerator kADC_SelfTestWdgThresholdForAlgCStepx

Self-test watchdog threshold for the algorithm C step x.

enum _adc_wdg_timer_val

This enumeration provides the selection of the ADC self-test watchdog timer value, including 0.1ms, 0.5ms, 1ms, 2ms, 5ms, 10ms, 20ms and 50ms.

Values:

enumerator kADC_SelfTestWdgTimerVal0

0.1ms ((0008h × Prescaler) cycles at 80 MHz).

enumerator kADC_SelfTestWdgTimerVal1

0.5ms ((0027h × Prescaler) cycles at 80 MHz).

enumerator kADC_SelfTestWdgTimerVal2

1ms ((004Eh × Prescaler) cycles at 80 MHz).

enumerator kADC_SelfTestWdgTimerVal3

2ms ((009Ch × Prescaler) cycles at 80 MHz).

enumerator kADC_SelfTestWdgTimerVal4

5ms ((0187h × Prescaler) cycles at 80 MHz).

enumerator kADC_SelfTestWdgTimerVal5

10ms ((030Dh × Prescaler) cycles at 80 MHz).

enumerator kADC_SelfTestWdgTimerVal6

20ms ((061Ah × Prescaler) cycles at 80 MHz).

enumerator kADC_SelfTestWdgTimerVal7

50ms ((0F42h × Prescaler) cycles at 80 MHz).

typedef enum *_adc_ext_trig* adc_ext_trig_t

This enumeration provides the selection of the ADC external trigger type.

typedef enum *_adc_state* adc_state_t

This enumeration provides the selection of the ADC state.

typedef enum *_adc_conv_mode* adc_conv_mode_t

This enumeration provides the selection of the ADC conversion mode, including normal conversion one-shot mode, normal conversion scan mode, and inject conversion one-shot mode.

typedef enum *_adc_clock_frequency* adc_clock_frequency_t

This enumeration provides the selection of the ADC operating clock frequency, including half-bus frequency and full bus frequency.

typedef enum *_adc_conv_data_align* adc_conv_data_align_t

This enumeration provides the selection of the ADC conversion data alignment, including the right alignment and left alignment.

typedef enum *_adc_presample_voltage_src* adc_presample_voltage_src_t

This enumeration provides the selection of the ADC internal analog input voltage sources for pre-sample, including DVDD0P8/2, AVDD1P8/4, VREFL_1p8 and VREFH_1p8.

typedef enum *_adc_dma_request_clear_src* adc_dma_request_clear_src_t

This enumeration provides the selection of the DMA request clear sources, including clear by acknowledgment from the DMA controller and clear on a read of the data register.

typedef enum *_adc_average_sample_numbers* adc_average_sample_numbers_t

This enumeration provides the selection of the ADC calibration averaging sample numbers, including 16, 32, 128, and 512 averaging samples.

typedef enum *_adc_sample_time* adc_sample_time_t

This enumeration provides the selection of the ADC sample time of calibration conversions, including 22, 8, 16 and 32 cycles of ADC_CLK.

typedef enum *_adc_wdg_threshold_int* adc_wdg_threshold_int_t

This enumeration provides the selection of the ADC analog watchdog threshold low and high interrupt enable.

typedef enum *_adc_alg_type* adc_alg_type_t

This enumeration provides the selection of the ADC self-test algorithm type, including algorithm S, algorithm C and algorithm S and C.

Note: The meaning of enumeration member 'kADC_SelfTestForAlgSAndC' in different conversion modes is different, in the one-shot conversion mode, it means executing algorithm S; in the scan conversion mode, it means executing algorithm S and C.

`typedef enum _adc_self_test_wdg_threshold adc_self_test_wdg_threshold_t`

This enumeration provides the selection of the ADC self-test watchdog thresholds for algorithm S step 0 - 2, and watchdog thresholds for algorithm C step 0 and step x (x = 1 - 11).

`typedef enum _adc_wdg_timer_val adc_wdg_timer_val_t`

This enumeration provides the selection of the ADC self-test watchdog timer value, including 0.1ms, 0.5ms, 1ms, 2ms, 5ms, 10ms, 20ms and 50ms.

`typedef struct _adc_config adc_config_t`

This structure is used to configure the ADC module.

`typedef struct _adc_channel_config adc_channel_config_t`

This structure is used to configure the ADC conversion channel.

`typedef struct _adc_chain_config adc_chain_config_t`

This structure is used to configure the ADC conversion chain.

`typedef struct _adc_wdg_config adc_wdg_config_t`

This structure is used to configure the ADC analog watchdog.

`typedef struct _adc_calibration_config adc_calibration_config_t`

This structure is used to configure the ADC calibration.

`typedef struct _adc_user_offset_gain_config adc_user_offset_gain_config_t`

This structure is used to configure the ADC user offset and gain.

`typedef struct _adc_self_test_config adc_self_test_config_t`

This structure is used to configure the ADC self-test.

`typedef struct _adc_self_test_wdg_config adc_self_test_wdg_config_t`

This structure is used to configure the ADC self-test watchdog for algorithm steps.

Note: The algorithm S step 2 only has the 'LowThresholdVal'.

`typedef struct _adc_conv_result adc_conv_result_t`

This structure is used to save the result information when obtaining the conversion result.

`typedef struct _adc_self_test_conv_result adc_self_test_conv_result_t`

This structure is used to save the result information when obtaining the self-test channel conversion result.

Note: The member 'convData' is used to store self-test channel conversion results. Only when executing step 1 of algorithm S, member 'convDataFraction' will be used to store the fractional part data. When executing other algorithms, this member will not be used.

ADC_GROUP_COUNTS

ADC_THRESHOLD_COUNTS

ADC_SELF_TEST_THRESHOLD_COUNTS

GET_REGINDEX(channelIndex)

GET_BITINDEX(channelIndex)

REGISTER_READWRITE(baseRegister, shiftIndex)

REGISTER_READONLY(baseRegister, shiftIndex)

NCMR_IO(base, registerIndex)
JCMR_IO(base, registerIndex)
PSR_IO(base, registerIndex)
DMAR_IO(base, registerIndex)
CWSELR_IO(base, registerIndex)
CWENR_IO(base, registerIndex)
CIMR_IO(base, registerIndex)
CEOCFR_IO(base, registerIndex)
AWORR_IO(base, registerIndex)
STAWR_IO(base, registerIndex)
CEOCFR_I(base, registerIndex)
AWORR_I(base, registerIndex)
CDR_I(base, registerIndex)
WDG_SELECT_MASK(shiftIndex)
WDG_SELECT_SHIFT(shiftIndex)
WDG_SELECT(val, shiftIndex)
ADC_CDR_VALID_MASK
ADC_CDR_VALID_SHIFT
ADC_CDR_OVERW_MASK
ADC_CDR_OVERW_SHIFT
ADC_CDR_RESULT_MASK
ADC_CDR_RESULT_SHIFT
ADC_CDR_CDATA_MASK
ADC_CDR_CDATA_SHIFT
ADC_STAWR_AWDE_MASK
ADC_STAWR_THRL_MASK
ADC_STAWR_THRL_SHIFT
ADC_STAWR_THRL(val)
ADC_STAWR_THRH_MASK
ADC_STAWR_THRH_SHIFT
ADC_STAWR_THRH(val)
ADC_CALSTAT_MAX
ADC_CALSTAT_SIGN
struct _adc_config
 #include <fsl_sar_adc.h> This structure is used to configure the ADC module.

Public Members

`bool enableAutoClockOff`

Decides whether to enable the ADC auto clock-off function, when set to true, the internal ADC clock is automatically switched off during IDLE mode to reduce power consumption (without going into power-down mode).

`bool enableOverWrite`

Decides whether to enable the latest conversion to overwrite the current value in the data registers.

`bool enableConvertPresampleVal`

Decides whether to convert the pre-sampled value, if enabled, pre-sampling is followed by the conversion, sampling will be bypassed and conversion of the pre-sampled data will be done.

`adc_ext_trig_t extTrig`

Specifies whether the normal trigger (with trigger type) input starts a conversion.

`adc_conv_data_align_t convDataAlign`

Selects the conversion data alignment.

`adc_clock_frequency_t clockFrequency`

Selects the ADC clock frequency.

`adc_dma_request_clear_src_t dmaRequestClearSrc`

Selects DMA request clear source.

`adc_presample_voltage_src_t presampleVoltageSrc[1]`

Selects analog input voltages for group 0 (corresponding to channel 0 to channel 31) and group 32 (corresponding to channel 32 to channel 63) pre-sampling.

`uint8_t samplePhaseDuration[1]`

Sets the sample phase duration in terms of the ADC controller clock for group 0 (corresponding to channel 0 to channel 31) and group 32 (corresponding to channel 32 to channel 63), the minimum acceptable value is 8, configuring to a value lower than 8 sets the sample period to 8 cycles.

`struct _adc_channel_config`

#include <fsl_sar_adc.h> This structure is used to configure the ADC conversion channel.

Public Members

`uint8_t channelIndex`

Sets the conversion channel index.

`bool enableInt`

Decides Whether to enable the interrupt function of the current conversion channel.

`bool enablePresample`

Decides whether to enable the pre-sample function of the current conversion channel.

`bool enableDmaTransfer`

Decides whether to enable the DMA transfer function of the current conversion channel.

`bool enableWdg`

Decides whether to enable the analog watchdog function of the current conversion channel.

uint8_t wdgIndex

Indicates which analog watchdog to provide the low and high threshold value.

struct _adc_chain_config

#include <fsl_sar_adc.h> This structure is used to configure the ADC conversion chain.

Public Members

adc_conv_mode_t convMode

Selects conversion mode.

bool enableGlobalChannelConvEndInt

Global control function to determine whether to enable the interrupt function of conversion channels.

bool enableChainConvEndInt

Decides whether to enable the current chain end-of-conversion interrupt.

uint8_t channelCount

Indicates the channel counts.

adc_channel_config_t *channelConfig

Chain channels configuration.

struct _adc_wdg_config

#include <fsl_sar_adc.h> This structure is used to configure the ADC analog watchdog.

Public Members

uint8_t wdgIndex

Indicates the analog watchdog index

adc_wdg_threshold_int_t wdgThresholdInt

Selects watchdog threshold low or/and high interrupt to enable/disable.

uint16_t lowThresholdVal

Sets the ADC analog watchdog low threshold value.

uint16_t highThresholdVal

Sets the ADC analog watchdog high threshold value.

struct _adc_calibration_config

#include <fsl_sar_adc.h> This structure is used to configure the ADC calibration.

Public Members

bool enableAverage

Decides whether to enable averaging of calibration time.

adc_sample_time_t sampleTime

Selects sample time of calibration conversions.

adc_average_sample_numbers_t averageSampleNumbers

Selects calibration averaging sample numbers.

struct _adc_user_offset_gain_config

#include <fsl_sar_adc.h> This structure is used to configure the ADC user offset and gain.

Public Members

`int8_t userOffset`
Sets user defined gain value.

`int16_t userGain`
Sets user defined offset value.

`struct __adc_self_test_config`
#include <fsl_sar_adc.h> This structure is used to configure the ADC self-test.

Public Members

`adc_alg_type_t algType`
Selects the self-test algorithm.

`uint8_t algSteps`
Sets the self-test algorithm steps, it should be programmed to zero in scan mode.

`uint8_t algSSamplePhaseDuration`
Sets the self-test algorithm S conversion sampling phase duration.

`uint8_t algCSamplePhaseDuration`
Sets the self-test algorithm C conversion sampling phase duration.

`uint8_t baudRate`
Sets the baud rate for the selected algorithm in scan mode, must write to this field before enabling a self-test channel.

`struct __adc_self_test_wdg_config`
#include <fsl_sar_adc.h> This structure is used to configure the ADC self-test watchdog for algorithm steps.

Note: The algorithm S step 2 only has the 'LowThrsholdVal'.

Public Members

`adc_self_test_wdg_threshold_t wdgThresholdId`
Indicates the self-test watchdog index

`uint16_t lowThrsholdVal`
Sets the self-test watchdog low threshold value.

`uint16_t highThrsholdVal`
Sets the self-test watchdog high threshold value.

`struct __adc_conv_result`
#include <fsl_sar_adc.h> This structure is used to save the result information when obtaining the conversion result.

Public Members

`bool overWrittenFlag`
Indicates when conversion data was overwritten by a newer result, the new data is written or discarded according to MCR[OWREN].

uint8_t convMode

Indicates the mode of conversion for the corresponding channel.

uint16_t convData

Stores the conversion data corresponding to the internal channel.

struct __adc_self_test_conv_result

#include <fsl_sar_adc.h> This structure is used to save the result information when obtaining the self-test channel conversion result.

Note: The member ‘convData’ is used to store self-test channel conversion results. Only when executing step 1 of algorithm S, member ‘convDataFraction’ will be used to store the fractional part data. When executing other algorithms, this member will not be used.

Public Members

bool overWrittenFlag

Indicates when conversion data is overwritten by a newer result.

uint16_t convData

Stores the conversion data corresponding to the internal self-test channel.

uint16_t convDataFraction

This field is only used to store the fractional part conversion result.

2.98 SEMA42: Hardware Semaphores Driver

FSL_SEMA42_DRIVER_VERSION

SEMA42 driver version.

SEMA42 status return codes.

Values:

enumerator kStatus_SEMA42_Busy

SEMA42 gate has been locked by other processor.

enumerator kStatus_SEMA42_Reseting

SEMA42 gate reseting is ongoing.

enum __sema42_gate_status

SEMA42 gate lock status.

Values:

enumerator kSEMA42_Unlocked

The gate is unlocked.

enumerator kSEMA42_LockedByProc0

The gate is locked by processor 0.

enumerator kSEMA42_LockedByProc1

The gate is locked by processor 1.

enumerator kSEMA42_LockedByProc2

The gate is locked by processor 2.

enumerator kSEMA42_LockedByProc3

The gate is locked by processor 3.

enumerator kSEMA42_LockedByProc4

The gate is locked by processor 4.

enumerator kSEMA42_LockedByProc5

The gate is locked by processor 5.

enumerator kSEMA42_LockedByProc6

The gate is locked by processor 6.

enumerator kSEMA42_LockedByProc7

The gate is locked by processor 7.

enumerator kSEMA42_LockedByProc8

The gate is locked by processor 8.

enumerator kSEMA42_LockedByProc9

The gate is locked by processor 9.

enumerator kSEMA42_LockedByProc10

The gate is locked by processor 10.

enumerator kSEMA42_LockedByProc11

The gate is locked by processor 11.

enumerator kSEMA42_LockedByProc12

The gate is locked by processor 12.

enumerator kSEMA42_LockedByProc13

The gate is locked by processor 13.

enumerator kSEMA42_LockedByProc14

The gate is locked by processor 14.

typedef enum *_sema42_gate_status* sema42_gate_status_t

SEMA42 gate lock status.

void SEMA42_Init(SEMA42_Type *base)

Initializes the SEMA42 module.

This function initializes the SEMA42 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either SEMA42_ResetGate or SEMA42_ResetAllGates function.

Parameters

- base – SEMA42 peripheral base address.

void SEMA42_Deinit(SEMA42_Type *base)

De-initializes the SEMA42 module.

This function de-initializes the SEMA42 module. It only disables the clock.

Parameters

- base – SEMA42 peripheral base address.

status_t SEMA42_TryLock(SEMA42_Type *base, uint8_t gateNum, uint8_t procNum)

Tries to lock the SEMA42 gate.

This function tries to lock the specific SEMA42 gate. If the gate has been locked by another processor, this function returns an error code.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

Return values

- kStatus_Success – Lock the sema42 gate successfully.
- kStatus_SEMA42_Busy – Sema42 gate has been locked by another processor.

status_t SEMA42_Lock(SEMA42_Type *base, uint8_t gateNum, uint8_t procNum)

Locks the SEMA42 gate.

This function locks the specific SEMA42 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

If SEMA42_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

Return values

- kStatus_Success – The gate was successfully locked.
- kStatus_Timeout – Timeout occurred while waiting for the gate to be unlocked.

Returns

status_t

static inline void SEMA42_Unlock(SEMA42_Type *base, uint8_t gateNum)

Unlocks the SEMA42 gate.

This function unlocks the specific SEMA42 gate. It only writes unlock value to the SEMA42 gate register. However, it does not check whether the SEMA42 gate is locked by the current processor or not. As a result, if the SEMA42 gate is not locked by the current processor, this function has no effect.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to unlock.

static inline *sema42_gate_status_t* SEMA42_GetGateStatus(SEMA42_Type *base, uint8_t gateNum)

Gets the status of the SEMA42 gate.

This function checks the lock status of a specific SEMA42 gate.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number.

Returns

status_t Current status.

`status_t SEMA42_ResetGate(SEMA42_Type *base, uint8_t gateNum)`

Resets the SEMA42 gate to an unlocked status.

This function resets a SEMA42 gate to an unlocked status.

Parameters

- `base` – SEMA42 peripheral base address.
- `gateNum` – Gate number.

Return values

- `kStatus_Success` – SEMA42 gate is reset successfully.
- `kStatus_SEMA42_Reseting` – Some other reset process is ongoing.

`static inline status_t SEMA42_ResetAllGates(SEMA42_Type *base)`

Resets all SEMA42 gates to an unlocked status.

This function resets all SEMA42 gate to an unlocked status.

Parameters

- `base` – SEMA42 peripheral base address.

Return values

- `kStatus_Success` – SEMA42 is reset successfully.
- `kStatus_SEMA42_Reseting` – Some other reset process is ongoing.

`SEMA42_GATE_NUM_RESET_ALL`

The number to reset all SEMA42 gates.

`SEMA42_GATEn(base, n)`

SEMA42 gate `n` register address.

The SEMA42 gates are sorted in the order 3, 2, 1, 0, 7, 6, 5, 4, ... not in the order 0, 1, 2, 3, 4, 5, 6, 7, ... The macro `SEMA42_GATEn` gets the SEMA42 gate based on the gate index.

The input gate index is XOR'ed with 3U: $0 \wedge 3 = 3$, $1 \wedge 3 = 2$, $2 \wedge 3 = 1$, $3 \wedge 3 = 0$, $4 \wedge 3 = 7$, $5 \wedge 3 = 6$, $6 \wedge 3 = 5$, $7 \wedge 3 = 4$...

`SEMA42_BUSY_POLL_COUNT`

Maximum polling iterations for SEMA42 waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the SEMA42 driver code before timing out and returning an error.

It applies to all waiting loops in SEMA42 driver, such as waiting for a gate to be unlocked, waiting for a reset to complete, or waiting for a resource to become available.

This is a count of loop iterations, not a time-based value.

If defined as 0, polling loops will continue indefinitely until their exit condition is met, which could potentially cause the system to hang if hardware doesn't respond or if a resource is never released.

2.99 SFA: Signal Frequency Analyser

`void SFA_GetDefaultConfig(sfa_config_t *config)`

Fill the SFA configuration structure with default settings.

The default values are:

```

config->mode = kSFA_FrequencyMeasurement0;
config->cutSelect = kSFA_CUTSelect0;
config->refSelect = kSFA_REFSelect0;
config->prediv = 0U;
config->trigStart = kSFA_TriggerStartSelect0;
config->startPolarity = kSFA_TriggerStartPolarityRiseEdge;
config->trigEnd = kSFA_TriggerEndSelect0;
config->endPolarity = kSFA_TriggerEndPolarityRiseEdge;
config->enableTrigMeasurement = false;
config->enableCUTPin = false;
config->cutTarget = 0xffffU;
config->refTarget = 0xffffffffU;

```

Parameters

- config – Pointer to the user configuration structure.

void SFA_Init(SFA_Type *base)

Initialize SFA.

Parameters

- base – SFA peripheral base address.

status_t SFA_Deinit(SFA_Type *base)

Clear counter, disable SFA and gate the SFA clock.

Parameters

- base – SFA peripheral base address.

Return values

- kStatus_Success – run success.
- kStatus_Timeout – timeout occurs.

static inline void SFA_EnableCUTPin(SFA_Type *base, bool enable)

Control the connection of the clock under test to an external pin.

Parameters

- base – SFA peripheral base address.
- enable – true: connect the clock under test and external pin. false: Disconnect the clock under test and external pin.

static inline uint32_t SFA_GetStatusFlags(SFA_Type *base)

Get SFA status flags.

Parameters

- base – SFA peripheral base address.

void SFA_ClearStatusFlag(SFA_Type *base, uint32_t mask)

Clear the SFA status flags.

Note: To clear kSFA_RefStoppedFlag, kSFA_CUTStoppedFlag, kSFA_MeasurementStartedFlag, and kSFA_ReferenceCounterTimeOutFlag, each counter will also be cleared.

Parameters

- base – SFA peripheral base address.

- mask – SFA status flag mask (see `_sfa_status_flags` for bit definition).

static inline void SFA__EnableInterrupts(SFA_Type *base, uint32_t mask)

Enable the selected SFA interrupt.

Parameters

- base – SFA peripheral base address.
- mask – The interrupt to enable (see `_sfa_interrupts_enable` for definition).

static inline void SFA__DisableInterrupts(SFA_Type *base, uint32_t mask)

Disable the selected SFA interrupt.

Parameters

- base – SFA peripheral base address.
- mask – The interrupt to disable (see `_sfa_interrupts_enable` for definition).

static inline uint8_t SFA__GetMode(SFA_Type *base)

Get SFA measurement mode.

Parameters

- base – SFA peripheral base address.

static inline uint8_t SFA__GetCUTPredivide(SFA_Type *base)

Get CUT predivide value.

Parameters

- base – SFA peripheral base address.

void SFA__InstallCallback(SFA_Type *base, *sfa_callback_t* function)

Install the callback function to be called when IRQ happens or measurement completes.

Parameters

- base – SFA peripheral base address.
- function – the SFA measure completed callback function.

void SFA__SetMeasureConfig(SFA_Type *base, const *sfa_config_t* *config)

Set Measurement options with the passed in configuration structure.

Parameters

- base – SFA peripheral base address.
- config – SFA configuration structure.

status_t SFA__MeasureBlocking(SFA_Type *base)

Start SFA measurement in blocking mode.

Parameters

- base – SFA peripheral base address.

Return values

- kStatus_SFA_MeasurementCompleted – SFA measure completes.
- kStatus_SFA_ReferenceCounterTimeout – reference counter timeout error happens.
- kStatus_SFA_CUTCounterTimeout – CUT counter time out happens.

void SFA_MeasureNonBlocking(SFA_Type *base)

Start measure sequence in NonBlocking mode.

This function performs nonblocking measurement by enabling sfa interrupt (Please enable the FreqGreaterThanMax and FreqLessThanMin interrupts individually as needed). The callback function must be installed before invoking this function.

Note: This function has different functions for different instances.

Parameters

- base – SFA peripheral base address.

void SFA_AbortMeasureSequence(SFA_Type *base)

Abort SFA measurement sequence.

Parameters

- base – SFA peripheral base address.

uint32_t SFA_CalculateFrequencyOrPeriod(SFA_Type *base, uint32_t refFrequency)

Calculate the frequency or period.

Parameters

- base – SFA peripheral base address.
- refFrequency – The reference clock frequency(BUS clock recommended).

static inline uint32_t SFA_GetCUTCOUNTER(SFA_Type *base)

Get current count of the clock under test.

Parameters

- base – SFA peripheral base address.

static inline void SFA_SetCUTTargetCount(SFA_Type *base, uint32_t count)

Set the target count for the clock under test.

Parameters

- base – SFA peripheral base address.
- count – target count for CUT.

static inline uint32_t SFA_GetCUTTargetCount(SFA_Type *base)

Get the target count of the clock under test.

Parameters

- base – SFA peripheral base address.

static inline void SFA_SetCUTLowLimitClockCount(SFA_Type *base, uint32_t count)

Set CUT low limit clock count.

Parameters

- base – SFA peripheral base address.
- count – low limit count for CUT clock.

static inline uint32_t SFA_GetCUTLowLimitClockCount(SFA_Type *base)

Get CUT low limit clock count.

Parameters

- base – SFA peripheral base address.

static inline void SFA_SetCUTHighLimitClockCount(SFA_Type *base, uint32_t count)
Set CUT high limit clock count.

Parameters

- base – SFA peripheral base address.
- count – high limit count for CUT clock.

static inline uint32_t SFA_GetCUTHighLimitClockCount(SFA_Type *base)
Get CUT high limit clock count.

Parameters

- base – SFA peripheral base address.

static inline uint32_t SFA_GetREFCounter(SFA_Type *base)
Get current count of the reference clock.

Parameters

- base – SFA peripheral base address.

static inline void SFA_SetREFTargetCount(SFA_Type *base, uint32_t count)
Set the target count for the reference clock.

Parameters

- base – SFA peripheral base address.
- count – target count for reference clock.

static inline uint32_t SFA_GetREFTargetCount(SFA_Type *base)
Get the target count of the reference clock.

Parameters

- base – SFA peripheral base address.

static inline uint32_t SFA_GetREFStartCount(SFA_Type *base)
Get saved reference clock counter which is loaded when measurement start.

Parameters

- base – SFA peripheral base address.

static inline uint32_t SFA_GetREFEndCount(SFA_Type *base)
Get saved reference clock counter which is loaded when measurement complete.

Parameters

- base – SFA peripheral base address.

static inline void SFA_SetREFLowLimitClockCount(SFA_Type *base, uint32_t count)
Set REF low limit clock count.

Parameters

- base – SFA peripheral base address.
- count – low limit count for REF clock.

static inline uint32_t SFA_GetREFLowLimitClockCount(SFA_Type *base)
Get REF low limit clock count.

Parameters

- base – SFA peripheral base address.

static inline void SFA_SetREFHighLimitClockCount(SFA_Type *base, uint32_t count)

Set REF high limit clock count.

Parameters

- base – SFA peripheral base address.
- count – high limit count for REF clock.

static inline uint32_t SFA_GetREFHighLimitClockCount(SFA_Type *base)

Get REF high limit clock count.

Parameters

- base – SFA peripheral base address.

FSL_SFA_DRVIER_VERSION

SFA driver version 2.1.3.

SFA_MEASUREMENT_START_TIMEOUT

Max loops to wait for SFA measurement started.

This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

SFA_CUT_COUNTER_STOP_TIMEOUT

Max loops to wait for SFA CUT counter has stopped.

This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

SFA_REF_COUNTER_STOP_TIMEOUT

Max loops to wait for SFA REF counter has stopped.

This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

SFA status return codes.

enumeration _sfa_status

Values:

enumerator kStatus_SFA_MeasurementCompleted

Measurement completed

enumerator kStatus_SFA_ReferenceCounterTimeout

Reference counter timeout

enumerator kStatus_SFA_CUTCounterTimeout

CUT counter timeout

enumerator kStatus_SFA_CUTClockFreqLessThanMinLimit

CUT clock frequency less than minimum limit

enumerator kStatus_SFA_CUTClockFreqGreaterThanMaxLimit

CUT clock frequency greater than maximum limit

enum _sfa_status_flags

List of SFA status flags.

The following status register flags can be cleared on any write to REF_CNT.

- kSFA_RefStoppedFlag
- kSFA_CutStoppedFlag

- kSFA_MeasurementStartedFlag
- kSFA_ReferenceCounterTimeOutFlag

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kSFA_RefStoppedFlag

Reference counter stopped flag

enumerator kSFA_CutStoppedFlag

CUT counter stopped flag

enumerator kSFA_MeasurementStartedFlag

Measurement Started flag

enumerator kSFA_ReferenceCounterTimeOutFlag

Reference counter time out flag

enumerator kSFA_InterruptRequestFlag

SFA interrupt request flag

enumerator kSFA_FreqGreaterThanMaxInterruptFlag

FREQ_GT_MAX interrupt flag

enumerator kSFA_FreqLessThanMinInterruptFlag

FREQ_LT_MIN interrupt flag

enumerator kSFA_AllStatusFlags

enum _sfa_interrupts_enable

List of SFA interrupt.

Values:

enumerator kSFA_InterruptEnable

SFA interrupt enable

enumerator kSFA_FreqGreaterThanMaxInterruptEnable

FREQ_GT_MAX interrupt enable

enumerator kSFA_FreqLessThanMinInterruptEnable

FREQ_LT_MIN interrupt enable

enum _sfa_measurement_mode

List of SFA measurement mode(Please check the mode configuration according to the manual).

Values:

enumerator kSFA_FrequencyMeasurement0

Frequency measurement performed with REF frequency > CUT frequency

enumerator kSFA_FrequencyMeasurement1

Frequency measurement performed with REF frequency < CUT frequency

enumerator kSFA_CUTPeriodMeasurement

CUT period measurement performed

enumerator kSFA_TriggerBasedMeasurement

Trigger based measurement performed

enum _sfa_cut_select

List of CUT which is connected to the CUT counter (Please refer to the manual for configuration).

Values:

enumerator kSFA__CUTSelect0

enumerator kSFA__CUTSelect1

enumerator kSFA__CUTSelect2

enumerator kSFA__CUTSelect3

enumerator kSFA__CUTSelect4

enumerator kSFA__CUTSelect5

enumerator kSFA__CUTSelect6

enumerator kSFA__CUTSelect7

enumerator kSFA__CUTSelect8

enumerator kSFA__CUTSelect9

enumerator kSFA__CUTSelect10

enumerator kSFA__CUTSelect11

enumerator kSFA__CUTSelect12

enumerator kSFA__CUTSelect13

enumerator kSFA__CUTSelect14

enumerator kSFA__CUTSelect15

enum _sfa_ref_select

List of REF which is connected to the REF counter (Please refer to the manual for configuration).

Values:

enumerator kSFA__REFSelect0

enumerator kSFA__REFSelect1

enumerator kSFA__REFSelect2

enum _sfa_trigger_start_select

List of Signal MUX for Trigger Based Measurement Start.

Values:

enumerator kSFA__TriggerStartSelect0

enumerator kSFA__TriggerStartSelect1

enum _sfa_trigger_end_select

List of Signal MUX for Trigger Based Measurement End.

Values:

enumerator kSFA__TriggerEndSelect0

enumerator kSFA_TriggerEndSelect1

enum _sfa_trigger_start_polarity

List of Trigger Start Polarity.

Values:

enumerator kSFA_TriggerStartPolarityRiseEdge

Rising edge will begin the measurement sequence

enumerator kSFA_TriggerStartPolarityFallEdge

Falling edge will begin the measurement sequence

enum _sfa_trigger_end_polarity

List of Trigger End Polarity.

Values:

enumerator kSFA_TriggerEndPolarityRiseEdge

Rising edge will end the measurement sequence

enumerator kSFA_TriggerEndPolarityFallEdge

Falling edge will end the measurement sequence

typedef void (*sfa_callback_t)(status_t status)

sfa measure completion callback function pointer type

This callback can be used in non blocking IRQHandler. Specify the callback you want in the call to SFA_InstallCallback().

Param base

SFA peripheral base address.

Param status

The runtime measurement status. kStatus_SFA_MeasurementCompleted: The measurement completes. kStatus_SFA_ReferenceCounterTimeout: Reference counter timeout happens. kStatus_SFA_CUTCounterTimeout: CUT counter timeout happens.

typedef enum _sfa_measurement_mode sfa_measurement_mode_t

List of SFA measurement mode(Please check the mode configuration according to the manual).

typedef enum _sfa_cut_select sfa_cut_select_t

List of CUT which is connected to the CUT counter (Please refer to the manual for configuration).

typedef enum _sfa_ref_select sfa_ref_select_t

List of REF which is connected to the REF counter (Please refer to the manual for configuration).

typedef enum _sfa_trigger_start_select sfa_trigger_start_select_t

List of Signal MUX for Trigger Based Measurement Start.

typedef enum _sfa_trigger_end_select sfa_trigger_end_select_t

List of Signal MUX for Trigger Based Measurement End.

typedef enum _sfa_trigger_start_polarity sfa_trigger_start_polarity_t

List of Trigger Start Polarity.

typedef enum _sfa_trigger_end_polarity sfa_trigger_end_polarity_t

List of Trigger End Polarity.

```
typedef struct _sfa_init_config sfa_config_t
```

Structure with setting to initialize the SFA module.

This structure holds configuration setting for the SFA peripheral. To initialize this structure to reasonable defaults, call the SFA_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

```
SFA__CUT__CLK__Enable(val)
```

```
struct _sfa_init_config
```

```
#include <fsl_sfa.h> Structure with setting to initialize the SFA module.
```

This structure holds configuration setting for the SFA peripheral. To initialize this structure to reasonable defaults, call the SFA_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

Public Members

```
sfa_measurement_mode_t mode
```

measurement mode

```
sfa_cut_select_t cutSelect
```

Select clock connected to the clock under test counter

```
sfa_ref_select_t refSelect
```

Select REF connected the bus clock

```
uint8_t prediv
```

Integer divide of the Input CUT signal

```
sfa_trigger_start_select_t trigStart
```

Select the signal will be used to end a trigger based measurement

```
sfa_trigger_start_polarity_t startPolarity
```

Select the polarity of the start trigger signal

```
sfa_trigger_end_select_t trigEnd
```

Select the signal will be used to commence a trigger based measurement

```
sfa_trigger_end_polarity_t endPolarity
```

Select the polarity of the end trigger signal

```
bool enableTrigMeasurement
```

false: The measurement will start by default with a dummy write to the CUT counter;
true : The measurement will start after receiving a dummy write to the REF_CNT followed by receiving the trigger edge

```
bool enableCUTPin
```

Control the connection of the clock under test to an external pin.

```
uint32_t refTarget
```

Reference counter target counts

```
uint32_t cutTarget
```

CUT counter target counts

2.100 TEMPSSENSOR: Temperature Sensor Module

void TMPSNS_GetDefaultConfig(*tmplsns_config_t* *config)

This function is used to get predefined configurations for the tempsensor initialization.

Parameters

- config – Pointer to the tempsensor configuration structure, please refer to *tmplsns_config_t* for details.

void TMPSNS_Init(TMPSNS_Type *base, const *tmplsns_config_t* *config)

This function is used to initialize the tempsensor.

Parameters

- base – Tempsensor peripheral base address.
- config – Pointer to the tempsensor configuration structure, please refer to *tmplsns_config_t* for details.

void TMPSNS_Deinit(TMPSNS_Type *base)

This function is used to de-initialize the tempsensor.

Parameters

- base – Tempsensor peripheral base address.

void TMPSNS_NonSecure_DoThresholdConfig(TMPSNS_Type *base, const *tmplsns_threshold_config_t* *config)

This function is used to configure the thresholds.

Parameters

- base – Tempsensor peripheral base address.
- config – Pointer to the threshold configuration structure, please refer *tmplsns_threshold_config_t* for details.

void TMPSNS_Secure_DoThresholdConfig(TMPSNS_Type *base, const *tmplsns_threshold_config_t* *config)

This function is used to configure the thresholds.

Parameters

- base – Tempsensor peripheral base address.
- config – Pointer to the threshold configuration structure, please refer *tmplsns_threshold_config_t* for details.

float TMPSNS_NonSecure_GetCurrentTemperature(TMPSNS_Type *base)

This function is used to get the current temperature.

Parameters

- base – TMPSNS base pointer

Returns

Current temperature with degrees Celsius.

float TMPSNS_Secure_GetCurrentTemperature(TMPSNS_Type *base)

This function is used to get the current temperature.

Parameters

- base – TMPSNS base pointer

Returns

Current temperature with degrees Celsius.

static inline void TMPSNS_EnableTmpsns(TMPSNS_Type *base)

This function is used to enable the tempsensor.

Parameters

- base – Tempsensor peripheral base address.

static inline void TMPSNS_DisableTmpsns(TMPSNS_Type *base)

This function is used to disable the tempsensor.

Parameters

- base – Tempsensor peripheral base address.

static inline void TMPSNS_StartMeasurement(TMPSNS_Type *base)

This function is used to start the temperature measurement process.

Parameters

- base – Tempsensor peripheral base address.

static inline void TMPSNS_StopMeasurement(TMPSNS_Type *base)

This function is used to stop the temperature measurement process.

Parameters

- base – Tempsensor peripheral base address.

static inline void TMPSNS_NonSecure_EnableTmpsnsInt(TMPSNS_Type *base, uint32_t mask)

This function is used to enable the tempsensor interrupts.

Parameters

- base – Tempsensor peripheral base address.
- mask – Mask value for interrupts, please refer `_tmpsns_nonsecure_int_enable` for details.

static inline void TMPSNS_Secure_EnableTmpsnsInt(TMPSNS_Type *base, uint32_t mask)

This function is used to enable the tempsensor interrupts.

Parameters

- base – Tempsensor peripheral base address.
- mask – Mask value for interrupts, please refer `_tmpsns_secure_int_enable` for details.

static inline void TMPSNS_NonSecure_DisableTmpsnsInt(TMPSNS_Type *base, uint32_t mask)

This function is used to disable the tempsensor interrupts.

Parameters

- base – Tempsensor peripheral base address.
- mask – Mask value for interrupts, please refer `_tmpsns_nonsecure_int_enable` for details.

static inline void TMPSNS_Secure_DisableTmpsnsInt(TMPSNS_Type *base, uint32_t mask)

This function is used to disable the tempsensor interrupts.

Parameters

- base – Tempsensor peripheral base address.
- mask – Mask value for interrupts, please refer `_tmpsns_secure_int_enable` for details.

```
static inline void TMPSNS_NonSecure_ClearTmpsnsIntStatus(TMPSNS_Type *base, uint32_t mask)
```

This function clears the tempsensor threshold interrupt status flags.

Note: The mask can using member `kTMPSNS_NonSecure_Threshold0IntStatus` to clean threshold 0 interrupt flag, using member `kTMPSNS_NonSecure_Threshold1IntStatus` to clean threshold 1 interrupt flag, and use member `kTMPSNS_NonSecure_Threshold2IntStatus` to clear threshold 2 interrupt. The data ready 0 interrupt flag cannot be cleaned using this function.

Parameters

- `base` – Tempsensor peripheral base address.
- `mask` – Mask value for flags to be cleared, please refer to `_tmpsns_nonsecure_status` for details.

```
static inline void TMPSNS_Secure_ClearTmpsnsIntStatus(TMPSNS_Type *base, uint32_t mask)
```

This function clears the tempsensor threshold interrupt status flags.

Note: The mask can using member `kTMPSNS_Secure_Threshold4IntStatus` to clean threshold 4 interrupt flag and using member `kTMPSNS_Secure_Threshold5IntStatus` to clear threshold 5 interrupt. The data ready 1 interrupt flag cannot be cleaned using this function.

Parameters

- `base` – Tempsensor peripheral base address.
- `mask` – Mask value for flags to be cleared, please refer to `_tmpsns_secure_status` for details.

```
static inline uint32_t TMPSNS_NonSecure_GetTmpsnsStatus(TMPSNS_Type *base)
```

This function is used to get the tempsensor status.

Parameters

- `base` – Tempsensor peripheral base address.

Returns

tempsensor status mask.

```
static inline uint32_t TMPSNS_Secure_GetTmpsnsStatus(TMPSNS_Type *base)
```

This function is used to get the tempsensor status.

Parameters

- `base` – Tempsensor peripheral base address.

Returns

tempsensor status mask.

```
static inline void TMPSNS_NonSecure_SetFilterLength(TMPSNS_Type *base, uint8_t length)
```

This function is used to set the tempsensor filter length.

Parameters

- `base` – Tempsensor peripheral base address.
- `length` – Filter length to be set.

```
static inline void TMPSNS_Secure_SetFilterLength(TMPSNS_Type *base, uint8_t length)
```

This function is used to set the tempsensor filter length.

Parameters

- `base` – Tempsensor peripheral base address.
- `length` – Filter length to be set.

```
static inline void TMPSNS_NonSecure_ClearFilterCount(TMPSNS_Type *base, uint32_t mask)
```

This function is used to clear the tempsensor filter count.

Parameters

- `base` – Tempsensor peripheral base address.
- `mask` – Mask value for flags to be cleared, please refer `_tmpsns_nonsecure_filter_count_clear` for details.

```
static inline void TMPSNS_Secure_ClearFilterCount(TMPSNS_Type *base, uint32_t mask)
```

This function is used to clear the tempsensor filter count.

Parameters

- `base` – Tempsensor peripheral base address.
- `mask` – Mask value for flags to be cleared, please refer `_tmpsns_secure_filter_count_clear` for details.

```
static inline void TMPSNS_Secure_SetTrimVal(TMPSNS_Type *base, uint16_t valA, int16_t valB,  
                                             uint16_t valAlpha, int16_t valOffset)
```

This function is used to set the trim values.

Parameters

- `base` – Tempsensor peripheral base address.
- `valA` – Controls trimming code, VAL_A, for the analog block.
- `valB` – Controls trimming code, VAL_B, for the analog block.
- `valAlpha` – Controls trimming code, VAL_ALPHA, for the analog block.
- `valOffset` – Controls trimming code, VAL_OFFSET, for the analog block.

```
FSL_TEMPSENSOR_DRIVER_VERSION
```

Tempsensor driver version 2.0.1.

```
enum _tmpsns_nonsecure_int_enable
```

This enumeration provides the type of tempsensor interrupt enablement.

Note: The threshold 0 interrupt is connected to the SRC.

Values:

```
enumerator kTMPSNS_NonSecure_Threshold0CmpIntEnable
```

Enable interrupt output event for the threshold0 comparator event.

```
enumerator kTMPSNS_NonSecure_Threshold1CmpIntEnable
```

Enable interrupt output event for the threshold1 comparator event.

```
enumerator kTMPSNS_NonSecure_Threshold2CmpIntEnable
```

Enable interrupt output event for the threshold2 comparator event.

enumerator kTMPSNS_NonSecure_DataReady0IntEnable

Enable interrupt output event when output data0 is ready after measurement completion.

enumerator kTMPSNS_NonSecure_AllIntEnable

Enable threshold 0, 1, 2, and data ready 0 interrupts.

enum _tmplsns_secure_int_enable

This enumeration provides the type of tempsensor interrupt enablement.

Values:

enumerator kTMPSNS_Secure_Threshold4CmpIntEnable

Enable interrupt output event for the threshold4 comparator event.

enumerator kTMPSNS_Secure_Threshold5CmpIntEnable

Enable interrupt output event for the threshold5 comparator event.

enumerator kTMPSNS_Secure_DataReady1IntEnable

Enable interrupt output event when output data1 is ready after measurement completion.

enumerator kTMPSNS_Secure_AllIntEnable

Enable thresholds 4, 5, and data ready 1 interrupt.

enum _tmplsns_nonsecure_status

This enumeration provides tempsensor status flags.

Values:

enumerator kTMPSNS_NonSecure_Idle0

Indicate whether the tempsensor is in an idle state or undergoing conversion.

enumerator kTMPSNS_NonSecure_DataReady0

Indicate whether new data is available on DATA0[DATA_VAL].

enumerator kTMPSNS_NonSecure_Threshold0CmpStatus

Indicate the state of the threshold0 comparator.

enumerator kTMPSNS_NonSecure_Threshold1CmpStatus

Indicate the state of the threshold1 comparator.

enumerator kTMPSNS_NonSecure_Threshold2CmpStatus

Indicate the state of the threshold2 comparator.

enumerator kTMPSNS_NonSecure_Threshold0IntStatus

Indicate whether a threshold0 comparator event (interrupt) occurred.

enumerator kTMPSNS_NonSecure_Threshold1IntStatus

Indicate whether a threshold1 comparator event (interrupt) occurred.

enumerator kTMPSNS_NonSecure_Threshold2IntStatus

Indicate whether a threshold2 comparator event (interrupt) occurred.

enum _tmplsns_secure_status

This enumeration provides tempsensor status flags.

Values:

enumerator kTMPSNS_Secure_Idle1

Indicate whether the tempsensor is in an idle state or undergoing conversion.

enumerator kTMPSNS_Secure_DataReady1

Indicate whether new data is available on DATA1[DATA_VAL].

enumerator kTMPSNS_Secure_Threshold4CmpStatus

Indicate the state of the threshold4 comparator.

enumerator kTMPSNS_Secure_Threshold5CmpStatus

Indicate the state of the threshold5 comparator.

enumerator kTMPSNS_Secure_Threshold4IntStatus

Indicate whether a threshold4 comparator event (interrupt) occurred.

enumerator kTMPSNS_Secure_Threshold5IntStatus

Indicate whether a threshold5 comparator event (interrupt) occurred.

enum _tmplsns_nonsecure_filter_count_clear

This enumeration provides the type of tempsensor filter counter clear.

Values:

enumerator kTMPSNS_NonSecure_Filter0CountClear

Clear the internal counter that counts the number of threshold violations corresponding to THR_CTRL01[TEMPERATURE_THRESHOLD0].

enumerator kTMPSNS_NonSecure_Filter1CountClear

Clear the internal counter that counts the number of threshold violations corresponding to THR_CTRL01[TEMPERATURE_THRESHOLD1].

enumerator kTMPSNS_NonSecure_Filter2CountClear

Clear the internal counter that counts the number of threshold violations corresponding to THR_CTRL23[TEMPERATURE_THRESHOLD2].

enumerator kTMPSNS_NonSecure_AllFilterCountClear

Clear threshold 0, 1, and 2 filter count.

enum _tmplsns_secure_filter_count_clear

This enumeration provides the type of tempsensor filter counter clear.

Values:

enumerator kTMPSNS_Secure_Filter4CountClear

Clears the internal counter that counts the number of threshold violations corresponding to THR_CTRL45[TEMPERATURE_THRESHOLD4].

enumerator kTMPSNS_Secure_Filter5CountClear

Clear the internal counter that counts the number of threshold violations corresponding to THR_CTRL45[TEMPERATURE_THRESHOLD5].

enumerator kTMPSNS_Secure_AllFilterCountClear

Clear thresholds 4, and 5 filter count.

enum _tmplsns_resolution_mode

This enumeration provides the selection of the tempsensor resolution modes, including mode 0 (0.59325ms), mode 1 (1.10525ms), mode 2 (2.12925ms), mode 3 (4.17725ms).

Values:

enumerator kTMPSNS_ResolutionMode0

0.59325ms.

enumerator kTMPSNS_ResolutionMode1

1.10525ms.

enumerator kTMPSNS_ResolutionMode2

2.12925ms.

enumerator kTMPSNS_ResolutionMode3
4.17725ms.

enum _tmplsns_measurement_mode

This enumeration provides the selection of the tempsensor measurement modes, including single one-shot mode, continuous mode and periodic one-shot mode.

Values:

enumerator kTMPSNS_SingleOneShotMode
Measures single temperature data after you start a measurement.

enumerator kTMPSNS_ContinuousMode
Continues to measure temperature data repeatedly at the conversion time rate.

enumerator kTMPSNS_PeriodicOneShotMode
Measures temperature data repeatedly at a periodic interval of time.

enum _tmplsns_thresholds

This enumeration provides the selection of the tempsensor thresholds, including thresholds 0, 1, 2, 4, and 5.

Note: Thresholds 4 and 5 are used only when secure registers can be accessed.

Values:

enumerator kTMPSNS_ThresholdCmp0
Threshold comparator 0.

enumerator kTMPSNS_ThresholdCmp1
Threshold comparator 1.

enumerator kTMPSNS_ThresholdCmp2
Threshold comparator 2.

enumerator kTMPSNS_ThresholdCmp4
Threshold comparator 4.

enumerator kTMPSNS_ThresholdCmp5
Threshold comparator 5.

enum _tmplsns_threshold_cmp_mode

This enumeration provides the selection of the tempsensor threshold comparator modes, including mode 0 (Less than or equal to the threshold), mode 1 (Greater than the threshold), mode 2 (High to low-temperature data transition at threshold) and mode 3 (Low to high-temperature data transition at threshold).

Values:

enumerator kTMPSNS_ThresholdCmpMode0
Less than or equal to the threshold.

enumerator kTMPSNS_ThresholdCmpMode1
Greater than threshold.

enumerator kTMPSNS_ThresholdCmpMode2
High to low temperature data transition at the threshold.

enumerator kTMPSNS_ThresholdCmpMode3
Low to high temperature data transition at the threshold.

`typedef enum _tmpsns_resolution_mode tmpsns_resolution_mode_t`

This enumeration provides the selection of the tempsensor resolution modes, including mode 0 (0.59325ms), mode 1 (1.10525ms), mode 2 (2.12925ms), mode 3 (4.17725ms).

`typedef enum _tmpsns_measurement_mode tmpsns_measurement_mode_t`

This enumeration provides the selection of the tempsensor measurement modes, including single one-shot mode, continuous mode and periodic one-shot mode.

`typedef enum _tmpsns_thresholds tmpsns_thresholds_t`

This enumeration provides the selection of the tempsensor thresholds, including thresholds 0, 1, 2, 4, and 5.

Note: Thresholds 4 and 5 are used only when secure registers can be accessed.

`typedef enum _tmpsns_threshold_cmp_mode tmpsns_threshold_cmp_mode_t`

This enumeration provides the selection of the tempsensor threshold comparator modes, including mode 0 (Less than or equal to the threshold), mode 1 (Greater than the threshold), mode 2 (High to low-temperature data transition at threshold) and mode 3 (Low to high-temperature data transition at threshold).

`typedef struct _tmpsns_config tmpsns_config_t`

This structure is used to configure the tempsensor module.

Note: All configurations of the tempsensor need to access secure registers (register address is offset by more than 0x200 relative to the peripheral base address). Before configuring the tempsensor, make sure you have permission to access these registers.

`typedef struct _tmpsns_threshold_config tmpsns_threshold_config_t`

This structure is used to configure the tempsensor threshold.

Note: Thresholds 4 and 5 are used only when secure registers can be accessed.

`TMPSNS_CTRL_THR_MODE_MASK(index)`

`TMPSNS_CTRL_THR_MODE_SHIFT(index)`

`TMPSNS_CTRL_THR_MODE(index, mode)`

`TMPSNS_THR_CTRL_TEMPERATURE_THRESHOLD_MASK(index)`

`TMPSNS_THR_CTRL_TEMPERATURE_THRESHOLD_SHIFT(index)`

`TMPSNS_THR_CTRL_TEMPERATURE_THRESHOLD(index, value)`

`struct _tmpsns_config`

`#include <fsl_tempsensor.h>` This structure is used to configure the tempsensor module.

Note: All configurations of the tempsensor need to access secure registers (register address is offset by more than 0x200 relative to the peripheral base address). Before configuring the tempsensor, make sure you have permission to access these registers.

Public Members

bool enableClockDivider

Decides whether to enable the clock divider.

uint8_t refClockDivider

Specifies the divider value for generating CONV_CLK from MODULE_CLK, the output clock frequency = input clock frequency / (DIV + 1).

uint8_t powerUpDealy

Sets the power-up delay on CONV_CLK for the analog block, see the chip data sheet for the tempsensor power-up time.

tmplsns_resolution_mode_t resolutionMode

Specifies the conversion time for different resolution modes, this parameter helps the user configure the length of the conversion time, longer conversion time translates into higher resolution.

tmplsns_measurement_mode_t measurementMode

Selects the measurement mode for the tempsensor, the mode could be changed only when the tempsensor does not measure, that is, when STAT[IDLE] = 1.

uint32_t measurementPeriod

Sets the target count (measurement period) on CONV_CLK for periodic trigger in periodic One-Shot conversion mode.

struct _tmplsns_threshold_config

#include <fsl_tempsensor.h> This structure is used to configure the tempsensor threshold.

Note: Thresholds 4 and 5 are used only when secure registers can be accessed.

Public Members

tmplsns_thresholds_t thresholdIndex

Indicates which threshold is selected.

tmplsns_threshold_cmp_mode_t thresholdCmpMode

Indicates which threshold comparator mode is selected.

float thresholdVal

Sets selected threshold's value.

2.101 TPM: Timer PWM Module

uint32_t TPM_GetInstance(TPM_Type *base)

Gets the instance from the base address.

Parameters

- base – TPM peripheral base address

Returns

The TPM instance

void TPM_Init(TPM_Type *base, const *tpm_config_t* *config)

Ungates the TPM clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the TPM driver.

Parameters

- base – TPM peripheral base address
- config – Pointer to user's TPM config structure.

void TPM_Deinit(TPM_Type *base)

Stops the counter and gates the TPM clock.

Parameters

- base – TPM peripheral base address

void TPM_GetDefaultConfig(*tpm_config_t* *config)

Fill in the TPM config struct with the default settings.

The default values are:

```
config->prescale = kTPM_Prescale_Divide_1;
config->useGlobalTimeBase = false;
config->syncGlobalTimeBase = false;
config->dozeEnable = false;
config->dbgMode = false;
config->enableReloadOnTrigger = false;
config->enableStopOnOverflow = false;
config->enableStartOnTrigger = false;
#if FSL_FEATURE_TPM_HAS_PAUSE_COUNTER_ON_TRIGGER
config->enablePauseOnTrigger = false;
#endif
config->triggerSelect = kTPM_Trigger_Select_0;
#if FSL_FEATURE_TPM_HAS_EXTERNAL_TRIGGER_SELECTION
config->triggerSource = kTPM_TriggerSource_External;
config->extTriggerPolarity = kTPM_ExtTrigger_Active_High;
#endif
#if defined(FSL_FEATURE_TPM_HAS_POL) && FSL_FEATURE_TPM_HAS_POL
config->chnlPolarity = 0U;
#endif
```

Parameters

- config – Pointer to user's TPM config structure.

tpm_clock_prescale_t TPM_CalculateCounterClkDiv(TPM_Type *base, uint32_t counterPeriod_Hz, uint32_t srcClock_Hz)

Calculates the counter clock prescaler.

This function calculates the values for SC[PS].

return Calculated clock prescaler value.

Parameters

- base – TPM peripheral base address
- counterPeriod_Hz – The desired frequency in Hz which corresponding to the time when the counter reaches the mod value
- srcClock_Hz – TPM counter clock in Hz

status_t TPM_SetupPwm(TPM_Type *base, const *tpm_chnl_pwm_signal_param_t* *chnlParams, uint8_t numOfChnls, *tpm_pwm_mode_t* mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz)

Configures the PWM signal parameters.

User calls this function to configure the PWM signals period, mode, dutycycle and edge. Use this function to configure all the TPM channels that will be used to output a PWM signal

Parameters

- base – TPM peripheral base address
- chnlParams – Array of PWM channel parameters to configure the channel(s)
- numOfChnls – Number of channels to configure, this should be the size of the array passed in
- mode – PWM operation mode, options available in enumeration `tpm_pwm_mode_t`
- pwmFreq_Hz – PWM signal frequency in Hz
- srcClock_Hz – TPM counter clock in Hz

Returns

`kStatus_Success` PWM setup successful
`kStatus_Error` PWM setup failed
`kStatus_Timeout` PWM setup timeout when write register CnV or MOD

`status_t` TPM_UpdatePwmDutycycle(TPM_Type *base, *tpm_chnl_t* chnlNumber, *tpm_pwm_mode_t* currentPwmMode, uint8_t dutyCyclePercent)

Update the duty cycle of an active PWM signal.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number. In combined mode, this represents the channel pair number
- currentPwmMode – The current PWM mode set during PWM setup
- dutyCyclePercent – New PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle)

Returns

`kStatus_Success` if the PWM setup was successful, `kStatus_Error` on failure

`void` TPM_UpdateChnlEdgeLevelSelect(TPM_Type *base, *tpm_chnl_t* chnlNumber, uint8_t level)

Update the edge level selection for a channel.

Note: When the TPM has PWM pause level select feature (FSL_FEATURE_TPM_HAS_PAUSE_LEVEL_SELECT = 1), the PWM output cannot be turned off by selecting the output level. In this case, must use TPM_DisableChannel API to close the PWM output.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- level – The level to be set to the ELSnB:ELSnA field; valid values are 00, 01, 10, 11. See the appropriate SoC reference manual for details about this field.

`static inline uint8_t` TPM_GetChannelControlBits(TPM_Type *base, *tpm_chnl_t* chnlNumber)

Get the channel control bits value (mode, edge and level bit fields).

This function disable the channel by clear all mode and level control bits.

Parameters

- base – TPM peripheral base address

- `chnlNumber` – The channel number

Returns

The control bits value. This is the logical OR of members of the enumeration `tpm_chnl_control_bit_mask_t`.

```
static inline status_t TPM_DisableChannel(TPM_Type *base, tpm_chnl_t chnlNumber)
```

Disable the channel.

This function disable the channel by clear all mode and level control bits.

Parameters

- `base` – TPM peripheral base address
- `chnlNumber` – The channel number

Returns

`kStatus_Success` PWM setup successful `kStatus_Timeout` PWM setup timeout when write register `CnSC`

```
static inline status_t TPM_EnableChannel(TPM_Type *base, tpm_chnl_t chnlNumber, uint8_t control)
```

Enable the channel according to mode and level configs.

This function enable the channel output according to input mode/level config parameters.

Parameters

- `base` – TPM peripheral base address
- `chnlNumber` – The channel number
- `control` – The control bits value. This is the logical OR of members of the enumeration `tpm_chnl_control_bit_mask_t`.

Returns

`kStatus_Success` PWM setup successful `kStatus_Timeout` PWM setup timeout when write register `CnSC`

```
void TPM_SetupInputCapture(TPM_Type *base, tpm_chnl_t chnlNumber, tpm_input_capture_edge_t captureMode)
```

Enables capturing an input signal on the channel using the function parameters.

When the edge specified in the `captureMode` argument occurs on the channel, the TPM counter is captured into the `CnV` register. The user has to read the `CnV` register separately to get this value.

Parameters

- `base` – TPM peripheral base address
- `chnlNumber` – The channel number
- `captureMode` – Specifies which edge to capture

```
status_t TPM_SetupOutputCompare(TPM_Type *base, tpm_chnl_t chnlNumber, tpm_output_compare_mode_t compareMode, uint32_t compareValue)
```

Configures the TPM to generate timed pulses.

When the TPM counter matches the value of `compareVal` argument (this is written into `CnV` reg), the channel output is changed based on what is specified in the `compareMode` argument.

Parameters

- `base` – TPM peripheral base address

- `chnlNumber` – The channel number
- `compareMode` – Action to take on the channel output when the compare condition is met
- `compareValue` – Value to be programmed in the CnV register.

Returns

`kStatus_Success` PWM setup successful
`kStatus_Timeout` PWM setup timeout when write register CnV

```
void TPM_SetupDualEdgeCapture(TPM_Type *base, tpm_chnl_t chnlPairNumber, const  
                             tpm_dual_edge_capture_param_t *edgeParam, uint32_t  
                             filterValue)
```

Configures the dual edge capture mode of the TPM.

This function allows to measure a pulse width of the signal on the input of channel of a channel pair. The filter function is disabled if the filterVal argument passed is zero.

Parameters

- `base` – TPM peripheral base address
- `chnlPairNumber` – The TPM channel pair number; options are 0, 1, 2, 3
- `edgeParam` – Sets up the dual edge capture function
- `filterValue` – Filter value, specify 0 to disable filter.

```
void TPM_SetupQuadDecode(TPM_Type *base, const tpm_phase_params_t *phaseAParams,  
                        const tpm_phase_params_t *phaseBParams,  
                        tpm_quad_decode_mode_t quadMode)
```

Configures the parameters and activates the quadrature decode mode.

Parameters

- `base` – TPM peripheral base address
- `phaseAParams` – Phase A configuration parameters
- `phaseBParams` – Phase B configuration parameters
- `quadMode` – Selects encoding mode used in quadrature decoder mode

```
static inline void TPM_SetChannelPolarity(TPM_Type *base, tpm_chnl_t chnlNumber, bool  
                                         enable)
```

Set the input and output polarity of each of the channels.

Parameters

- `base` – TPM peripheral base address
- `chnlNumber` – The channel number
- `enable` – `true`: Set the channel polarity to active high; `false`: Set the channel polarity to active low;

```
static inline void TPM_EnableChannelExtTrigger(TPM_Type *base, tpm_chnl_t chnlNumber, bool  
                                              enable)
```

Enable external trigger input to be used by channel.

In input capture mode, configures the trigger input that is used by the channel to capture the counter value. In output compare or PWM mode, configures the trigger input used to modulate the channel output. When modulating the output, the output is forced to the channel initial value whenever the trigger is not asserted.

Note: No matter how many external trigger sources there are, only input trigger 0 and 1 are used. The even numbered channels share the input trigger 0 and the odd numbered channels share the second input trigger 1.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- enable – true: Configures trigger input 0 or 1 to be used by channel; false: Trigger input has no effect on the channel

void TPM_EnableInterrupts(TPM_Type *base, uint32_t mask)

Enables the selected TPM interrupts.

Parameters

- base – TPM peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration `tpm_interrupt_enable_t`

void TPM_DisableInterrupts(TPM_Type *base, uint32_t mask)

Disables the selected TPM interrupts.

Parameters

- base – TPM peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration `tpm_interrupt_enable_t`

uint32_t TPM_GetEnabledInterrupts(TPM_Type *base)

Gets the enabled TPM interrupts.

Parameters

- base – TPM peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration `tpm_interrupt_enable_t`

void TPM_RegisterCallBack(TPM_Type *base, *tpm_callback_t* callback)

Register callback.

If channel or overflow interrupt is enabled by the user, then a callback can be registered which will be invoked when the interrupt is triggered.

Parameters

- base – TPM peripheral base address
- callback – Callback function

static inline uint32_t TPM_GetChannelValue(TPM_Type *base, *tpm_chnl_t* chnlNumber)

Gets the TPM channel value.

Note: The TPM channel value contain the captured TPM counter value for the input modes or the match value for the output modes.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number

Returns

The channle CnV regisyer value.

static inline uint32_t TPM_GetStatusFlags(TPM_Type *base)

Gets the TPM status flags.

Parameters

- base – TPM peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration `tpm_status_flags_t`

static inline void TPM_ClearStatusFlags(TPM_Type *base, uint32_t mask)

Clears the TPM status flags.

Parameters

- base – TPM peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration `tpm_status_flags_t`

static inline *status_t* TPM_SetTimerPeriod(TPM_Type *base, uint32_t ticks)

Sets the timer period in units of ticks.

Timers counts from 0 until it equals the count value set here. The count value is written to the MOD register.

Note:

- This API allows the user to use the TPM module as a timer. Do not mix usage of this API with TPM's PWM setup API's.
 - Call the utility macros provided in the `fsl_common.h` to convert usec or msec to ticks.
-

Parameters

- base – TPM peripheral base address
- ticks – A timer period in units of ticks, which should be equal or greater than 1.

Returns

`kStatus_Success` PWM setup successful
`kStatus_Timeout` PWM setup timeout when write register CnSC

static inline uint32_t TPM_GetCurrentTimerCount(TPM_Type *base)

Reads the current timer counting value.

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note: Call the utility macros provided in the `fsl_common.h` to convert ticks to usec or msec.

Parameters

- base – TPM peripheral base address

Returns

The current counter value in ticks

```
static inline void TPM_StartTimer(TPM_Type *base, tpm_clock_source_t clockSource)
```

Starts the TPM counter.

Parameters

- base – TPM peripheral base address
- clockSource – TPM clock source; once clock source is set the counter will start running

```
static inline status_t TPM_StopTimer(TPM_Type *base)
```

Stops the TPM counter.

Parameters

- base – TPM peripheral base address

Returns

kStatus_Success PWM setup successful
kStatus_Timeout PWM setup timeout when write register CnSC

```
FSL_TPM_DRIVER_VERSION
```

TPM driver version 2.4.0.

```
enum _tpm_chnl
```

List of TPM channels.

Note: Actual number of available channels is SoC dependent

Values:

```
enumerator kTPM_Chnl_0
    TPM channel number 0
```

```
enumerator kTPM_Chnl_1
    TPM channel number 1
```

```
enumerator kTPM_Chnl_2
    TPM channel number 2
```

```
enumerator kTPM_Chnl_3
    TPM channel number 3
```

```
enumerator kTPM_Chnl_4
    TPM channel number 4
```

```
enumerator kTPM_Chnl_5
    TPM channel number 5
```

```
enumerator kTPM_Chnl_6
    TPM channel number 6
```

```
enumerator kTPM_Chnl_7
    TPM channel number 7
```

```
enum _tpm_pwm_mode
```

TPM PWM operation modes.

Values:

```
enumerator kTPM_EdgeAlignedPwm
    Edge aligned PWM
```

enumerator kTPM_CenterAlignedPwm

Center aligned PWM

enumerator kTPM_CombinedPwm

Combined PWM (Edge-aligned, center-aligned, or asymmetrical PWMs can be obtained in combined mode using different software configurations)

enum _tpm_pwm_level_select

TPM PWM output pulse mode: high-true, low-true or no output.

Note: When the TPM has PWM pause level select feature, the PWM output cannot be turned off by selecting the output level. In this case, the channel must be closed to close the PWM output.

Values:

enumerator kTPM_HighTrue

High true pulses

enumerator kTPM_LowTrue

Low true pulses

enum _tpm_pwm_pause_level_select

TPM PWM output when first enabled or paused: set or clear.

Values:

enumerator kTPM_ClearOnPause

Clear Output when counter first enabled or paused.

enumerator kTPM_SetOnPause

Set Output when counter first enabled or paused.

enum _tpm_chnl_control_bit_mask

List of TPM channel modes and level control bit mask.

Values:

enumerator kTPM_ChnlELSnAMask

Channel ELSA bit mask.

enumerator kTPM_ChnlELSnBMask

Channel ELSE bit mask.

enumerator kTPM_ChnlMSAMask

Channel MSA bit mask.

enumerator kTPM_ChnlMSBMask

Channel MSB bit mask.

enum _tpm_trigger_select

Trigger sources available.

This is used for both internal & external trigger sources (external trigger sources available in certain SoC's)

Note: The actual trigger sources available is SoC-specific.

Values:

enumerator kTPM_Trigger_Select_0
enumerator kTPM_Trigger_Select_1
enumerator kTPM_Trigger_Select_2
enumerator kTPM_Trigger_Select_3
enumerator kTPM_Trigger_Select_4
enumerator kTPM_Trigger_Select_5
enumerator kTPM_Trigger_Select_6
enumerator kTPM_Trigger_Select_7
enumerator kTPM_Trigger_Select_8
enumerator kTPM_Trigger_Select_9
enumerator kTPM_Trigger_Select_10
enumerator kTPM_Trigger_Select_11
enumerator kTPM_Trigger_Select_12
enumerator kTPM_Trigger_Select_13
enumerator kTPM_Trigger_Select_14
enumerator kTPM_Trigger_Select_15

enum _tpm_trigger_source

Trigger source options available.

Note: This selection is available only on some SoC's. For SoC's without this selection, the only trigger source available is internal trigger.

Values:

enumerator kTPM_TriggerSource_External

Use external trigger input

enumerator kTPM_TriggerSource_Internal

Use internal trigger (channel pin input capture)

enum _tpm_ext_trigger_polarity

External trigger source polarity.

Note: Selects the polarity of the external trigger source.

Values:

enumerator kTPM_ExtTrigger_Active_High

External trigger input is active high

enumerator kTPM_ExtTrigger_Active_Low

External trigger input is active low

enum _tpm_output_compare_mode

TPM output compare modes.

Values:

enumerator kTPM_NoOutputSignal

No channel output when counter reaches CnV

enumerator kTPM_ToggleOnMatch

Toggle output

enumerator kTPM_ClearOnMatch

Clear output

enumerator kTPM_SetOnMatch

Set output

enumerator kTPM_HighPulseOutput

Pulse output high

enumerator kTPM_LowPulseOutput

Pulse output low

enum _tpm_input_capture_edge

TPM input capture edge.

Values:

enumerator kTPM_RisingEdge

Capture on rising edge only

enumerator kTPM_FallingEdge

Capture on falling edge only

enumerator kTPM_RiseAndFallEdge

Capture on rising or falling edge

enum _tpm_quad_decode_mode

TPM quadrature decode modes.

Note: This mode is available only on some SoC's.

Values:

enumerator kTPM_QuadPhaseEncode

Phase A and Phase B encoding mode

enumerator kTPM_QuadCountAndDir

Count and direction encoding mode

enum _tpm_phase_polarity

TPM quadrature phase polarities.

Values:

enumerator kTPM_QuadPhaseNormal

Phase input signal is not inverted

enumerator kTPM_QuadPhaseInvert

Phase input signal is inverted

enum _tpm_clock_source

TPM clock source selection.

Values:

enumerator kTPM_SystemClock

System clock

enumerator kTPM_ExternalClock

External TPM_EXTCLK pin clock

enumerator kTPM_ExternalInputTriggerClock

Selected external input trigger clock

enum _tpm_clock_prescale

TPM prescale value selection for the clock source.

Values:

enumerator kTPM_Prescale_Divide_1

Divide by 1

enumerator kTPM_Prescale_Divide_2

Divide by 2

enumerator kTPM_Prescale_Divide_4

Divide by 4

enumerator kTPM_Prescale_Divide_8

Divide by 8

enumerator kTPM_Prescale_Divide_16

Divide by 16

enumerator kTPM_Prescale_Divide_32

Divide by 32

enumerator kTPM_Prescale_Divide_64

Divide by 64

enumerator kTPM_Prescale_Divide_128

Divide by 128

enum _tpm_interrupt_enable

List of TPM interrupts.

Values:

enumerator kTPM_Chnl0InterruptEnable

Channel 0 interrupt.

enumerator kTPM_Chnl1InterruptEnable

Channel 1 interrupt.

enumerator kTPM_Chnl2InterruptEnable

Channel 2 interrupt.

enumerator kTPM_Chnl3InterruptEnable

Channel 3 interrupt.

enumerator kTPM_Chnl4InterruptEnable

Channel 4 interrupt.

enumerator kTPM_Chnl5InterruptEnable
Channel 5 interrupt.

enumerator kTPM_Chnl6InterruptEnable
Channel 6 interrupt.

enumerator kTPM_Chnl7InterruptEnable
Channel 7 interrupt.

enumerator kTPM_TimeOverflowInterruptEnable
Time overflow interrupt.

enum _tpm_status_flags
List of TPM flags.

Values:

enumerator kTPM_Chnl0Flag
Channel 0 flag

enumerator kTPM_Chnl1Flag
Channel 1 flag

enumerator kTPM_Chnl2Flag
Channel 2 flag

enumerator kTPM_Chnl3Flag
Channel 3 flag

enumerator kTPM_Chnl4Flag
Channel 4 flag

enumerator kTPM_Chnl5Flag
Channel 5 flag

enumerator kTPM_Chnl6Flag
Channel 6 flag

enumerator kTPM_Chnl7Flag
Channel 7 flag

enumerator kTPM_TimeOverflowFlag
Time overflow flag

typedef enum _tpm_chnl tpm_chnl_t
List of TPM channels.

Note: Actual number of available channels is SoC dependent

typedef enum _tpm_pwm_mode tpm_pwm_mode_t
TPM PWM operation modes.

typedef enum _tpm_pwm_level_select tpm_pwm_level_select_t
TPM PWM output pulse mode: high-true, low-true or no output.

Note: When the TPM has PWM pause level select feature, the PWM output cannot be turned off by selecting the output level. In this case, the channel must be closed to close the PWM output.

typedef enum *_tpm_pwm_pause_level_select* tpm_pwm_pause_level_select_t

TPM PWM output when first enabled or paused: set or clear.

typedef enum *_tpm_chnl_control_bit_mask* tpm_chnl_control_bit_mask_t

List of TPM channel modes and level control bit mask.

typedef struct *_tpm_chnl_pwm_signal_param* tpm_chnl_pwm_signal_param_t

Options to configure a TPM channel's PWM signal.

typedef enum *_tpm_trigger_select* tpm_trigger_select_t

Trigger sources available.

This is used for both internal & external trigger sources (external trigger sources available in certain SoC's)

Note: The actual trigger sources available is SoC-specific.

typedef enum *_tpm_trigger_source* tpm_trigger_source_t

Trigger source options available.

Note: This selection is available only on some SoC's. For SoC's without this selection, the only trigger source available is internal trigger.

typedef enum *_tpm_ext_trigger_polarity* tpm_ext_trigger_polarity_t

External trigger source polarity.

Note: Selects the polarity of the external trigger source.

typedef enum *_tpm_output_compare_mode* tpm_output_compare_mode_t

TPM output compare modes.

typedef enum *_tpm_input_capture_edge* tpm_input_capture_edge_t

TPM input capture edge.

typedef struct *_tpm_dual_edge_capture_param* tpm_dual_edge_capture_param_t

TPM dual edge capture parameters.

Note: This mode is available only on some SoC's.

typedef enum *_tpm_quad_decode_mode* tpm_quad_decode_mode_t

TPM quadrature decode modes.

Note: This mode is available only on some SoC's.

typedef enum *_tpm_phase_polarity* tpm_phase_polarity_t

TPM quadrature phase polarities.

typedef struct *_tpm_phase_param* tpm_phase_params_t

TPM quadrature decode phase parameters.

typedef enum *_tpm_clock_source* tpm_clock_source_t

TPM clock source selection.

`typedef enum _tpm_clock_prescale tpm_clock_prescale_t`

TPM prescale value selection for the clock source.

`typedef struct _tpm_config tpm_config_t`

TPM config structure.

This structure holds the configuration settings for the TPM peripheral. To initialize this structure to reasonable defaults, call the `TPM_GetDefaultConfig()` function and pass a pointer to your config structure instance.

The config struct can be made `const` so it resides in flash

`typedef enum _tpm_interrupt_enable tpm_interrupt_enable_t`

List of TPM interrupts.

`typedef enum _tpm_status_flags tpm_status_flags_t`

List of TPM flags.

`typedef void (*tpm_callback_t)(TPM_Type *base)`

TPM callback function pointer.

Param base

TPM peripheral base address.

`static inline void TPM_Reset(TPM_Type *base)`

Performs a software reset on the TPM module.

Reset all internal logic and registers, except the Global Register. Remains set until cleared by software.

Note: TPM software reset is available on certain SoC's only

Parameters

- base – TPM peripheral base address

`void TPM_DriverIRQHandler(uint32_t instance)`

TPM driver IRQ handler common entry.

This function provides the common IRQ request entry for TPM.

Parameters

- instance – TPM instance.

`TPM_TIMEOUT`

Max loops to wait for writing register.

When writing MOD CnV CnSC and SC register, driver will wait until register is updated. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

`TPM_MAX_COUNTER_VALUE(x)`

Help macro to get the max counter value.

`struct _tpm_chnl_pwm_signal_param`

`#include <fsl_tpm.h>` Options to configure a TPM channel's PWM signal.

Public Members

tpm_chnl_t chnlNumber

TPM channel to configure. In combined mode (available in some SoC's), this represents the channel pair number

tpm_pwm_pause_level_select_t pauseLevel

PWM output level when counter first enabled or paused

tpm_pwm_level_select_t level

PWM output active level select

uint8_t dutyCyclePercent

PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=always active signal (100% duty cycle)

uint8_t firstEdgeDelayPercent

Used only in combined PWM mode to generate asymmetrical PWM. Specifies the delay to the first edge in a PWM period. If unsure, leave as 0. Should be specified as percentage of the PWM period, (dutyCyclePercent + firstEdgeDelayPercent) value should be not greater than 100.

bool enableComplementary

Used only in combined PWM mode. true: The combined channels output complementary signals; false: The combined channels output same signals;

tpm_pwm_pause_level_select_t secPauseLevel

Used only in combined PWM mode. Define the second channel output level when counter first enabled or paused

uint8_t deadTimeValue[2]

The dead time value for channel n and n+1 in combined complementary PWM mode. Deadtime insertion is disabled when this value is zero, otherwise deadtime insertion for channel n/n+1 is configured as (deadTimeValue * 4) clock cycles. deadTimeValue's available range is 0 ~ 15.

struct _tpm_dual_edge_capture_param

#include <fsl_tpm.h> TPM dual edge capture parameters.

Note: This mode is available only on some SoC's.

Public Members

bool enableSwap

true: Use channel n+1 input, channel n input is ignored; false: Use channel n input, channel n+1 input is ignored

tpm_input_capture_edge_t currChanEdgeMode

Input capture edge select for channel n

tpm_input_capture_edge_t nextChanEdgeMode

Input capture edge select for channel n+1

struct _tpm_phase_param

#include <fsl_tpm.h> TPM quadrature decode phase parameters.

Public Members

uint32_t phaseFilterVal

Filter value, filter is disabled when the value is zero

tpm_phase_polarity_t phasePolarity

Phase polarity

struct __tpm_config

#include <fsl_tpm.h> TPM config structure.

This structure holds the configuration settings for the TPM peripheral. To initialize this structure to reasonable defaults, call the TPM_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

Public Members

tpm_clock_prescale_t prescale

Select TPM clock prescale value

bool useGlobalTimeBase

true: The TPM channels use an external global time base (the local counter still use for generate overflow interrupt and DMA request); false: All TPM channels use the local counter as their timebase

bool syncGlobalTimeBase

true: The TPM counter is synchronized to the global time base; false: disabled

tpm_trigger_select_t triggerSelect

Input trigger to use for controlling the counter operation

tpm_trigger_source_t triggerSource

Decides if we use external or internal trigger.

tpm_ext_trigger_polarity_t extTriggerPolarity

when using external trigger source, need selects the polarity of it.

bool enableDoze

true: TPM counter is paused in doze mode; false: TPM counter continues in doze mode

bool enableDebugMode

true: TPM counter continues in debug mode; false: TPM counter is paused in debug mode

bool enableReloadOnTrigger

true: TPM counter is reloaded on trigger; false: TPM counter not reloaded

bool enableStopOnOverflow

true: TPM counter stops after overflow; false: TPM counter continues running after overflow

bool enableStartOnTrigger

true: TPM counter only starts when a trigger is detected; false: TPM counter starts immediately

bool enablePauseOnTrigger

true: TPM counter will pause while trigger remains asserted; false: TPM counter continues running

uint8_t chnlPolarity

Defines the input/output polarity of the channels in POL register

2.102 TRDC: Trusted Resource Domain Controller

void TRDC_Init(*TRDC_Type* *base)

Initializes the TRDC module.

This function enables the TRDC clock.

Parameters

- base – TRDC peripheral base address.

void TRDC_Deinit(*TRDC_Type* *base)

De-initializes the TRDC module.

This function disables the TRDC clock.

Parameters

- base – TRDC peripheral base address.

static inline uint8_t TRDC_GetCurrentMasterDomainId(*TRDC_Type* *base)

Gets the domain ID of the current bus master.

Parameters

- base – TRDC peripheral base address.

Returns

Domain ID of current bus master.

void TRDC_GetHardwareConfig(*TRDC_Type* *base, *trdc_hardware_config_t* *config)

Gets the TRDC hardware configuration.

This function gets the TRDC hardware configurations, including number of bus masters, number of domains, number of MRCs and number of PACs.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the structure to get the configuration.

static inline void TRDC_SetDacGlobalValid(*TRDC_Type* *base)

Sets the TRDC DAC(Domain Assignment Controllers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- base – TRDC peripheral base address.

static inline void TRDC_LockMasterDomainAssignment(*TRDC_Type* *base, uint8_t master)

Locks the bus master domain assignment register.

This function locks the master domain assignment. After it is locked, the register can't be changed until next reset.

Parameters

- base – TRDC peripheral base address.
- master – Which master to configure.

static inline void TRDC_SetMasterDomainAssignmentValid(*TRDC_Type* *base, uint8_t master, bool valid)

Sets the master domain assignment as valid or invalid.

This function sets the master domain assignment as valid or invalid.

Parameters

- base – TRDC peripheral base address.
- master – Which master to configure.
- valid – True to set valid, false to set invalid.

```
void TRDC_GetDefaultProcessorDomainAssignment(trdc_processor_domain_assignment_t  
                                              *domainAssignment)
```

Gets the default master domain assignment for the processor bus master.

This function gets the default master domain assignment for the processor bus master. It should only be used for the processor bus masters, such as CORE0. This function sets the assignment as follows:

```
assignment->domainId      = 0U;  
assignment->domainIdSelect = kTRDC_DidMda;  
assignment->lock           = 0U;
```

Parameters

- domainAssignment – Pointer to the assignment structure.

```
void TRDC_GetDefaultNonProcessorDomainAssignment(trdc_non_processor_domain_assignment_t  
                                                *domainAssignment)
```

Gets the default master domain assignment for non-processor bus master.

This function gets the default master domain assignment for non-processor bus master. It should only be used for the non-processor bus masters, such as DMA. This function sets the assignment as follows:

```
assignment->domainId      = 0U;  
assignment->privilegeAttr  = kTRDC_ForceUser;  
assignment->secureAttr    = kTRDC_ForceSecure;  
assignment->bypassDomainId = 0U;  
assignment->lock           = 0U;
```

Parameters

- domainAssignment – Pointer to the assignment structure.

```
void TRDC_SetProcessorDomainAssignment(TRDC_Type *base, const  
                                       trdc_processor_domain_assignment_t  
                                       *domainAssignment)
```

Sets the processor bus master domain assignment.

This function sets the processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter `assignIndex` specifies which assignment register to set.

Example: Set domain assignment for core 0.

```
trdc_processor_domain_assignment_t processorAssignment;  
  
TRDC_GetDefaultProcessorDomainAssignment(&processorAssignment);  
  
processorAssignment.domainId = 0;  
processorAssignment.xxx      = xxx;  
TRDC_SetMasterDomainAssignment(TRDC, &processorAssignment);
```

Parameters

- base – TRDC peripheral base address.
- domainAssignment – Pointer to the assignment structure.

```
static inline void TRDC_EnableProcessorDomainAssignment(TRDC_Type *base, bool enable)
```

Enables the processor bus master domain assignment.

Parameters

- base – TRDC peripheral base address.
- enable – True to enable, false to disable.

```
void TRDC_SetNonProcessorDomainAssignment(TRDC_Type *base, uint8_t master, const
                                         trdc_non_processor_domain_assignment_t
                                         *domainAssignment)
```

Sets the non-processor bus master domain assignment.

This function sets the non-processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter `assignIndex` specifies which assignment register to set.

Example: Set domain assignment for DMA0.

```
trdc_non_processor_domain_assignment_t nonProcessorAssignment;

TRDC_GetDefaultNonProcessorDomainAssignment(&nonProcessorAssignment);
nonProcessorAssignment.domainId = 1;
nonProcessorAssignment.xxx      = xxx;

TRDC_SetMasterDomainAssignment(TRDC, kTrdcMasterDma0, 0U, &nonProcessorAssignment);
```

Parameters

- base – TRDC peripheral base address.
- master – Which master to configure, refer to `trdc_master_t` in processor header file.
- domainAssignment – Pointer to the assignment structure.

```
void TRDC_GetDefaultIDAUConfig(trdc_idau_config_t *idauConfiguration)
```

Gets the default IDAU(Implementation-Defined Attribution Unit) configuration.

```
config->lockSecureVTOR   = false;
config->lockNonsecureVTOR = false;
config->lockSecureMPU     = false;
config->lockNonsecureMPU  = false;
config->lockSAU           = false;
```

Parameters

- idauConfiguration – Pointer to the configuration structure.

```
void TRDC_SetIDAU(TRDC_Type *base, const trdc_idau_config_t *idauConfiguration)
```

Sets the IDAU(Implementation-Defined Attribution Unit) control configuration.

Example: Lock the secure and non-secure MPU registers.

```
trdc_idau_config_t idauConfiguration;

TRDC_GetDefaultIDAUConfig(&idauConfiguration);

idauConfiguration.lockSecureMPU = true;
idauConfiguration.lockNonsecureMPU = true;
TRDC_SetIDAU(TRDC, &idauConfiguration);
```

Parameters

- base – TRDC peripheral base address.
- idauConfiguration – Pointer to the configuration structure.

static inline void TRDC_EnableFlashLogicalWindow(*TRDC_Type* *base, bool enable)

Enables/disables the FLW(flash logical window) function.

Parameters

- base – TRDC peripheral base address.
- enable – True to enable, false to disable.

static inline void TRDC_LockFlashLogicalWindow(*TRDC_Type* *base)

Locks FLW registers. Once locked the registers can not be updated until next reset.

Parameters

- base – TRDC peripheral base address.

static inline uint32_t TRDC_GetFlashLogicalWindowPbase(*TRDC_Type* *base)

Gets the FLW physical base address.

Parameters

- base – TRDC peripheral base address.

Returns

Physical address of the FLW function.

static inline void TRDC_GetSetFlashLogicalWindowSize(*TRDC_Type* *base, uint16_t size)

Sets the FLW size.

Parameters

- base – TRDC peripheral base address.
- size – Size of the FLW in unit of 32k bytes.

void TRDC_GetDefaultFlashLogicalWindowConfig(*trdc_flw_config_t* *flwConfiguration)

Gets the default FLW(Flash Logical Window) configuration.

```
config->blockCount    = false;
config->arrayBaseAddr = false;
config->lock          = false;
config->enable        = false;
```

Parameters

- flwConfiguration – Pointer to the configuration structure.

void TRDC_SetFlashLogicalWindow(*TRDC_Type* *base, const *trdc_flw_config_t* *flwConfiguration)

Sets the FLW function's configuration.

```
trdc_flw_config_t flwConfiguration;

TRDC_GetDefaultIDAUConfig(&flwConfiguration);

flwConfiguration.blockCount = 32U;
flwConfiguration.arrayBaseAddr = 0xFFFFFFFF;
TRDC_SetIDAU(TRDC, &flwConfiguration);
```

Parameters

- base – TRDC peripheral base address.
- flwConfiguration – Pointer to the configuration structure.

status_t TRDC_GetAndClearFirstDomainError(*TRDC_Type* *base, *trdc_domain_error_t* *error)

Gets and clears the first domain error of the current domain.

This function gets the first access violation information for the current domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

Parameters

- base – TRDC peripheral base address.
- error – Pointer to the error information.

Returns

If the access violation is captured, this function returns the *kStatus_Success*. The error information can be obtained from the parameter error. If no access violation is captured, this function returns the *kStatus_NoData*.

status_t TRDC_GetAndClearFirstSpecificDomainError(*TRDC_Type* *base, *trdc_domain_error_t* *error, *uint8_t* domainId)

Gets and clears the first domain error of the specific domain.

This function gets the first access violation information for the specific domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

Parameters

- base – TRDC peripheral base address.
- error – Pointer to the error information.
- domainId – The error of which domain to get and clear.

Returns

If the access violation is captured, this function returns the *kStatus_Success*. The error information can be obtained from the parameter error. If no access violation is captured, this function returns the *kStatus_NoData*.

static inline void TRDC_SetMrcGlobalValid(*TRDC_Type* *base)

Sets the TRDC MRC(Memory Region Checkers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- base – TRDC peripheral base address.

static inline *uint8_t* TRDC_GetMrcRegionNumber(*TRDC_Type* *base, *uint8_t* mrcIdx)

Gets the TRDC MRC(Memory Region Checkers) region number valid.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.

Returns

the region number of the given MRC instance

void TRDC_MrcSetMemoryAccessConfig(*TRDC_Type* *base, const *trdc_memory_access_control_config_t* *config, *uint8_t* mrcIdx, *uint8_t* regIdx)

Sets the memory access configuration for one of the access control register of one MRC.

Example: Enable the secure operations and lock the configuration for MRC0 region 1.

```
trdc_memory_access_control_config_t config;

config.securePrivX = true;
config.securePrivW = true;
config.securePrivR = true;
config.lock = true;
TRDC_SetMrcMemoryAccess(TRDC, &config, 0, 1);
```

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the configuration structure.
- mrcIdx – MRC index.
- regIdx – Register number.

void TRDC_MrcEnableDomainNseUpdate(*TRDC_Type* *base, uint8_t mrcIdx, uint16_t domainMask, bool enable)

Enables the update of the selected domains.

After the domains' update are enabled, their regions' NSE bits can be set or clear.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- domainMask – Bit mask of the domains to be enabled.
- enable – True to enable, false to disable.

void TRDC_MrcRegionNseSet(*TRDC_Type* *base, uint8_t mrcIdx, uint16_t regionMask)

Sets the NSE bits of the selected regions for domains.

This function sets the NSE bits for the selected regions for the domains whose update are enabled.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- regionMask – Bit mask of the regions whose NSE bits to set.

void TRDC_MrcRegionNseClear(*TRDC_Type* *base, uint8_t mrcIdx, uint16_t regionMask)

Clears the NSE bits of the selected regions for domains.

This function clears the NSE bits for the selected regions for the domains whose update are enabled.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- regionMask – Bit mask of the regions whose NSE bits to clear.

void TRDC_MrcDomainNseClear(*TRDC_Type* *base, uint8_t mrcIdx, uint16_t domainMask)

Clears the NSE bits for all the regions of the selected domains.

This function clears the NSE bits for all regions of selected domains whose update are enabled.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- domainMask – Bit mask of the domains whose NSE bits to clear.

```
void TRDC__MrcSetRegionDescriptorConfig(TRDC_Type *base, const  
                                         trdc_mrc_region_descriptor_config_t *config)
```

Sets the configuration for one of the region descriptor per domain per MRC instance.

This function sets the configuration for one of the region descriptor, including the start and end address of the region, memory access control policy and valid.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to region descriptor configuration structure.

```
static inline void TRDC__SetMbcGlobalValid(TRDC_Type *base)
```

Sets the TRDC MBC(Memory Block Checkers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- base – TRDC peripheral base address.

```
void TRDC__GetMbcHardwareConfig(TRDC_Type *base, trdc_slave_memory_hardware_config_t  
                               *config, uint8_t mbcIdx, uint8_t slvIdx)
```

Gets the hardware configuration of the one of two slave memories within each MBC(memory block checker).

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the structure to get the configuration.
- mbcIdx – MBC number.
- slvIdx – Slave number.

```
void TRDC__MbcSetNseUpdateConfig(TRDC_Type *base, const trdc_mbc_nse_update_config_t  
                                *config, uint8_t mbcIdx)
```

Sets the NSR update configuration for one of the MBC instance.

After set the NSE configuration, the configured memory area can be update by NSE set/clear.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to NSE update configuration structure.
- mbcIdx – MBC index.

```
void TRDC__MbcWordNseSet(TRDC_Type *base, uint8_t mbcIdx, uint32_t bitMask)
```

Sets the NSE bits of the selected configuration words according to NSE update configuration.

This function sets the NSE bits of the word for the configured region, memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- bitMask – Mask of the bits whose NSE bits to set.

```
void TRDC_MbcWordNseClear(TRDC_Type *base, uint8_t mbcIdx, uint32_t bitMask)
```

Clears the NSE bits of the selected configuration words according to NSE update configuration.

This function sets the NSE bits of the word for the configured regio, memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- bitMask – Mask of the bits whose NSE bits to clear.

```
void TRDC_MbcNseClearAll(TRDC_Type *base, uint8_t mbcIdx, uint16_t domainMask, uint8_t slaveMask)
```

Clears all configuration words' NSE bits of the selected domain and memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- domainMask – Mask of the domains whose NSE bits to clear, 0b110 means clear domain 1&2.
- slaveMask – Mask of the slaves whose NSE bits to clear, 0x11 means clear all slave 0&1's NSE bits.

```
void TRDC_MbcSetMemoryAccessConfig(TRDC_Type *base, const trdc_memory_access_control_config_t *config, uint8_t mbcIdx, uint8_t rgdIdx)
```

Sets the memory access configuration for one of the region descriptor of one MBC.

Example: Enable the secure operations and lock the configuration for MRC0 region 1.

```
trdc_memory_access_control_config_t config;

config.securePrivX = true;
config.securePrivW = true;
config.securePrivR = true;
config.lock = true;
TRDC_SetMbcMemoryAccess(TRDC, &config, 0, 1);
```

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the configuration structure.
- mbcIdx – MBC index.
- rgdIdx – Region descriptor number.

```
void TRDC_MbcSetMemoryBlockConfig(TRDC_Type *base, const trdc_mbc_memory_block_config_t *config)
```

Sets the configuration for one of the memory block per domain per MBC instance.

This function sets the configuration for one of the memory block, including the memory access control policy and nse enable.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to memory block configuration structure.

enum _trdc_did_sel

TRDC domain ID select method, the register bit TRDC_MDA_W0_0_DFMT0[DIDS], used for domain hit evaluation.

Values:

enumerator kTRDC_DidMda

Use MDAn[2:0] as DID.

enumerator kTRDC_DidInput

Use the input DID (DID_in) as DID.

enumerator kTRDC_DidMdaAndInput

Use MDAn[2] concatenated with DID_in[1:0] as DID.

enumerator kTRDC_DidReserved

Reserved.

enum _trdc_secure_attr

TRDC secure attribute, the register bit TRDC_MDA_W0_0_DFMT0[SA], used for bus master domain assignment.

Values:

enumerator kTRDC_ForceSecure

Force the bus attribute for this master to secure.

enumerator kTRDC_ForceNonSecure

Force the bus attribute for this master to non-secure.

enumerator kTRDC_MasterSecure

Use the bus master's secure/nonsecure attribute directly.

enumerator kTRDC_MasterSecure1

Use the bus master's secure/nonsecure attribute directly.

enum _trdc_privilege_attr

TRDC privileged attribute, the register bit TRDC_MDA_W0_x_DFMT1[PA], used for non-processor bus master domain assignment.

Values:

enumerator kTRDC_ForceUser

Force the bus attribute for this master to user.

enumerator kTRDC_ForcePrivilege

Force the bus attribute for this master to privileged.

enumerator kTRDC_MasterPrivilege

Use the bus master's attribute directly.

enumerator kTRDC_MasterPrivilege1

Use the bus master's attribute directly.

enum _trdc_controller

TRDC controller definition for domain error check. Each TRDC instance may have different MRC or MBC count, call TRDC_GetHardwareConfig to get the actual count.

Values:

enumerator kTRDC_MemBlockController0

Memory block checker 0.

enumerator kTRDC_MemBlockController1
Memory block checker 1.

enumerator kTRDC_MemBlockController2
Memory block checker 2.

enumerator kTRDC_MemBlockController3
Memory block checker 3.

enumerator kTRDC_MemRegionChecker0
Memory region checker 0.

enumerator kTRDC_MemRegionChecker1
Memory region checker 1.

enumerator kTRDC_MemRegionChecker2
Memory region checker 2.

enumerator kTRDC_MemRegionChecker3
Memory region checker 3.

enumerator kTRDC_MemRegionChecker4
Memory region checker 4.

enumerator kTRDC_MemRegionChecker5
Memory region checker 5.

enumerator kTRDC_MemRegionChecker6
Memory region checker 6.

enum _trdc_error_state

TRDC domain error state	definition	TRDC_MBCn_DERR_W1[EST]	or
TRDC_MRCn_DERR_W1[EST].			

Values:

enumerator kTRDC_ErrorStateNone
No access violation detected.

enumerator kTRDC_ErrorStateNone1
No access violation detected.

enumerator kTRDC_ErrorStateSingle
Single access violation detected.

enumerator kTRDC_ErrorStateMulti
Multiple access violation detected.

enum _trdc_error_attr

TRDC domain error attribute	definition	TRDC_MBCn_DERR_W1[EATR]	or
TRDC_MRCn_DERR_W1[EATR].			

Values:

enumerator kTRDC_ErrorSecureUserInst
Secure user mode, instruction fetch access.

enumerator kTRDC_ErrorSecureUserData
Secure user mode, data access.

enumerator kTRDC_ErrorSecurePrivilegeInst
Secure privileged mode, instruction fetch access.

enumerator kTRDC_ErrorSecurePrivilegeData

Secure privileged mode, data access.

enumerator kTRDC_ErrorNonSecureUserInst

NonSecure user mode, instruction fetch access.

enumerator kTRDC_ErrorNonSecureUserData

NonSecure user mode, data access.

enumerator kTRDC_ErrorNonSecurePrivilegeInst

NonSecure privileged mode, instruction fetch access.

enumerator kTRDC_ErrorNonSecurePrivilegeData

NonSecure privileged mode, data access.

enum _trdc_error_type

TRDC domain error access type definition TRDC_DERR_W1_n[ERW].

Values:

enumerator kTRDC_ErrorTypeRead

Error occurs on read reference.

enumerator kTRDC_ErrorTypeWrite

Error occurs on write reference.

enum _trdc_region_descriptor

The region descriptor enumeration, used to form a mask to set/clear the NSE bits for one or several regions.

Values:

enumerator kTRDC_RegionDescriptor0

Region descriptor 0.

enumerator kTRDC_RegionDescriptor1

Region descriptor 1.

enumerator kTRDC_RegionDescriptor2

Region descriptor 2.

enumerator kTRDC_RegionDescriptor3

Region descriptor 3.

enumerator kTRDC_RegionDescriptor4

Region descriptor 4.

enumerator kTRDC_RegionDescriptor5

Region descriptor 5.

enumerator kTRDC_RegionDescriptor6

Region descriptor 6.

enumerator kTRDC_RegionDescriptor7

Region descriptor 7.

enumerator kTRDC_RegionDescriptor8

Region descriptor 8.

enumerator kTRDC_RegionDescriptor9

Region descriptor 9.

enumerator kTRDC_RegionDescriptor10

Region descriptor 10.

enumerator kTRDC_RegionDescriptor11

Region descriptor 11.

enumerator kTRDC_RegionDescriptor12

Region descriptor 12.

enumerator kTRDC_RegionDescriptor13

Region descriptor 13.

enumerator kTRDC_RegionDescriptor14

Region descriptor 14.

enumerator kTRDC_RegionDescriptor15

Region descriptor 15.

enum _trdc_MRC_domain

The MRC domain enumeration, used to form a mask to enable/disable the update or clear all NSE bits of one or several domains.

Values:

enumerator kTRDC_MrcDomain0

Domain 0.

enumerator kTRDC_MrcDomain1

Domain 1.

enumerator kTRDC_MrcDomain2

Domain 2.

enumerator kTRDC_MrcDomain3

Domain 3.

enumerator kTRDC_MrcDomain4

Domain 4.

enumerator kTRDC_MrcDomain5

Domain 5.

enumerator kTRDC_MrcDomain6

Domain 6.

enumerator kTRDC_MrcDomain7

Domain 7.

enumerator kTRDC_MrcDomain8

Domain 8.

enumerator kTRDC_MrcDomain9

Domain 9.

enumerator kTRDC_MrcDomain10

Domain 10.

enumerator kTRDC_MrcDomain11

Domain 11.

enumerator kTRDC_MrcDomain12

Domain 12.

enumerator kTRDC_MrcDomain13

Domain 13.

enumerator kTRDC_MrcDomain14
Domain 14.

enumerator kTRDC_MrcDomain15
Domain 15.

enum _trdc_MBC_domain

The MBC domain enumeration, used to form a mask to enable/disable the update or clear NSE bits of one or several domains.

Values:

enumerator kTRDC_MbcDomain0
Domain 0.

enumerator kTRDC_MbcDomain1
Domain 1.

enumerator kTRDC_MbcDomain2
Domain 2.

enumerator kTRDC_MbcDomain3
Domain 3.

enumerator kTRDC_MbcDomain4
Domain 4.

enumerator kTRDC_MbcDomain5
Domain 5.

enumerator kTRDC_MbcDomain6
Domain 6.

enumerator kTRDC_MbcDomain7
Domain 7.

enum _trdc_MBC_memory

The MBC slave memory enumeration, used to form a mask to enable/disable the update or clear NSE bits of one or several memory block.

Values:

enumerator kTRDC_MbcSlaveMemory0
Memory 0.

enumerator kTRDC_MbcSlaveMemory1
Memory 1.

enumerator kTRDC_MbcSlaveMemory2
Memory 2.

enumerator kTRDC_MbcSlaveMemory3
Memory 3.

enum _trdc_MBC_bit

The MBC bit enumeration, used to form a mask to set/clear configured words' NSE.

Values:

enumerator kTRDC_MbcBit0
Bit 0.

enumerator kTRDC_MbcBit1
Bit 1.

enumerator kTRDC_MbcBit2
Bit 2.

enumerator kTRDC_MbcBit3
Bit 3.

enumerator kTRDC_MbcBit4
Bit 4.

enumerator kTRDC_MbcBit5
Bit 5.

enumerator kTRDC_MbcBit6
Bit 6.

enumerator kTRDC_MbcBit7
Bit 7.

enumerator kTRDC_MbcBit8
Bit 8.

enumerator kTRDC_MbcBit9
Bit 9.

enumerator kTRDC_MbcBit10
Bit 10.

enumerator kTRDC_MbcBit11
Bit 11.

enumerator kTRDC_MbcBit12
Bit 12.

enumerator kTRDC_MbcBit13
Bit 13.

enumerator kTRDC_MbcBit14
Bit 14.

enumerator kTRDC_MbcBit15
Bit 15.

enumerator kTRDC_MbcBit16
Bit 16.

enumerator kTRDC_MbcBit17
Bit 17.

enumerator kTRDC_MbcBit18
Bit 18.

enumerator kTRDC_MbcBit19
Bit 19.

enumerator kTRDC_MbcBit20
Bit 20.

enumerator kTRDC_MbcBit21
Bit 21.

enumerator kTRDC_MbcBit22
Bit 22.

enumerator `kTRDC_MbcBit23`

Bit 23.

enumerator `kTRDC_MbcBit24`

Bit 24.

enumerator `kTRDC_MbcBit25`

Bit 25.

enumerator `kTRDC_MbcBit26`

Bit 26.

enumerator `kTRDC_MbcBit27`

Bit 27.

enumerator `kTRDC_MbcBit28`

Bit 28.

enumerator `kTRDC_MbcBit29`

Bit 29.

enumerator `kTRDC_MbcBit30`

Bit 30.

enumerator `kTRDC_MbcBit31`

Bit 31.

typedef struct *_trdc_hardware_config* trdc_hardware_config_t

TRDC hardware configuration.

typedef struct *_trdc_slave_memory_hardware_config* trdc_slave_memory_hardware_config_t

Hardware configuration of the two slave memories within each MBC(memory block checker).

typedef enum *_trdc_did_sel* trdc_did_sel_t

TRDC domain ID select method, the register bit `TRDC_MDA_W0_0_DFMT0[DIDS]`, used for domain hit evaluation.

typedef enum *_trdc_secure_attr* trdc_secure_attr_t

TRDC secure attribute, the register bit `TRDC_MDA_W0_0_DFMT0[SA]`, used for bus master domain assignment.

typedef struct *_trdc_processor_domain_assignment* trdc_processor_domain_assignment_t

Domain assignment for the processor bus master.

typedef enum *_trdc_privilege_attr* trdc_privilege_attr_t

TRDC privileged attribute, the register bit `TRDC_MDA_W0_x_DFMT1[PA]`, used for non-processor bus master domain assignment.

typedef struct *_trdc_non_processor_domain_assignment*

trdc_non_processor_domain_assignment_t

Domain assignment for the non-processor bus master.

typedef struct *_trdc_idau_config* trdc_idau_config_t

IDAU(Implementation-Defined Attribution Unit) configuration for TZ-M function control.

typedef struct *_trdc_flw_config* trdc_flw_config_t

FLW(Flash Logical Window) configuration.

typedef enum *_trdc_controller* trdc_controller_t

TRDC controller definition for domain error check. Each TRDC instance may have different MRC or MBC count, call `TRDC_GetHardwareConfig` to get the actual count.

```
typedef enum _trdc_error_state trdc_error_state_t
    TRDC domain error state definition TRDC_MBCn_DERR_W1[EST] or
    TRDC_MRCn_DERR_W1[EST].
```

```
typedef enum _trdc_error_attr trdc_error_attr_t
    TRDC domain error attribute definition TRDC_MBCn_DERR_W1[EATR] or
    TRDC_MRCn_DERR_W1[EATR].
```

```
typedef enum _trdc_error_type trdc_error_type_t
    TRDC domain error access type definition TRDC_DERR_W1_n[ERW].
```

```
typedef struct _trdc_domain_error trdc_domain_error_t
    TRDC domain error definition.
```

```
typedef struct _trdc_memory_access_control_config trdc_memory_access_control_config_t
    Memory access control configuration for MBC/MRC.
```

```
typedef struct _trdc_mrc_region_descriptor_config trdc_mrc_region_descriptor_config_t
    The configuration of each region descriptor per domain per MRC instance.
```

```
typedef struct _trdc_mbc_nse_update_config trdc_mbc_nse_update_config_t
    The configuration of MBC NSE update.
```

```
typedef struct _trdc_mbc_memory_block_config trdc_mbc_memory_block_config_t
    The configuration of each memory block per domain per MBC instance.
```

FSL_TRDC_DRIVER_VERSION

FSL_TRDC_DRIVER_VERSION

```
struct _trdc_hardware_config
    #include <fsl_trdc.h> TRDC hardware configuration.
```

Public Members

uint8_t masterNumber
Number of bus masters.

uint8_t domainNumber
Number of domains.

uint8_t mbcNumber
Number of MBCs.

uint8_t mrcNumber
Number of MRCs.

```
struct _trdc_slave_memory_hardware_config
    #include <fsl_trdc.h> Hardware configuration of the two slave memories within each
    MBC(memory block checker).
```

Public Members

uint32_t blockNum
Number of blocks.

uint32_t blockSize
Block size.

```
struct _trdc_processor_domain_assignment
    #include <fsl_trdc.h> Domain assignment for the processor bus master.
```

Public Members

uint32_t domainId

Domain ID.

uint32_t domainIdSelect

Domain ID select method, see trdc_did_sel_t.

uint32_t __pad0__

Reserved.

uint32_t secureAttr

Secure attribute, see trdc_secure_attr_t.

uint32_t __pad1__

Reserved.

uint32_t lock

Lock the register.

uint32_t __pad2__

Reserved.

struct __trdc_non_processor_domain_assignment

#include <fsl_trdc.h> Domain assignment for the non-processor bus master.

Public Members

uint32_t domainId

Domain ID.

uint32_t privilegeAttr

Privileged attribute, see trdc_privilege_attr_t.

uint32_t secureAttr

Secure attribute, see trdc_secure_attr_t.

uint32_t bypassDomainId

Bypass domain ID.

uint32_t __pad0__

Reserved.

uint32_t lock

Lock the register.

uint32_t __pad1__

Reserved.

struct __trdc_idau_config

#include <fsl_trdc.h> IDAU(Implementation-Defined Attribution Unit) configuration for TZ-M function control.

Public Members

uint32_t __pad0__

Reserved.

uint32_t lockSecureVTOR

Disable writes to secure VTOR(Vector Table Offset Register).

uint32_t lockNonsecureVTOR

Disable writes to non-secure VTOR, Application interrupt and Reset Control Registers.

uint32_t lockSecureMPU

Disable writes to secure MPU(Memory Protection Unit) from software or from a debug agent connected to the processor in Secure state.

uint32_t lockNonsecureMPU

Disable writes to non-secure MPU(Memory Protection Unit) from software or from a debug agent connected to the processor.

uint32_t lockSAU

Disable writes to SAU(Security Attribution Unit) registers.

uint32_t __pad1__

Reserved.

struct __trdc_flw_config

#include <fsl_trdc.h> FLW(Flash Logical Window) configuration.

Public Members

uint16_t blockCount

Block count of the Flash Logic Window in 32KByte blocks.

uint32_t arrayBaseAddr

Flash array base address of the Flash Logical Window.

bool lock

Disable writes to FLW registers.

bool enable

Enable FLW function.

struct __trdc_domain_error

#include <fsl_trdc.h> TRDC domain error definition.

Public Members

trdc_controller_t controller

Which controller captured access violation.

uint32_t address

Access address that generated access violation.

trdc_error_state_t errorState

Error state.

trdc_error_attr_t errorAttr

Error attribute.

trdc_error_type_t errorType

Error type.

uint8_t errorPort

Error port.

uint8_t domainId

Domain ID.

uint8_t slaveMemoryIdx

The slave memory index. Only apply when violation in MBC.

struct __trdc_memory_access_control_config

#include <fsl_trdc.h> Memory access control configuration for MBC/MRC.

Public Members

uint32_t nonsecureUsrX

Allow nonsecure user execute access.

uint32_t nonsecureUsrW

Allow nonsecure user write access.

uint32_t nonsecureUsrR

Allow nonsecure user read access.

uint32_t __pad0__

Reserved.

uint32_t nonsecurePrivX

Allow nonsecure privilege execute access.

uint32_t nonsecurePrivW

Allow nonsecure privilege write access.

uint32_t nonsecurePrivR

Allow nonsecure privilege read access.

uint32_t __pad1__

Reserved.

uint32_t secureUsrX

Allow secure user execute access.

uint32_t secureUsrW

Allow secure user write access.

uint32_t secureUsrR

Allow secure user read access.

uint32_t __pad2__

Reserved.

uint32_t securePrivX

Allownonsecure privilege execute access.

uint32_t securePrivW

Allownonsecure privilege write access.

uint32_t securePrivR

Allownonsecure privilege read access.

uint32_t __pad3__

Reserved.

uint32_t lock

Lock the configuration until next reset, only apply to access control register 0.

struct __trdc_mrc_region_descriptor_config

#include <fsl_trdc.h> The configuration of each region descriptor per domain per MRC instance.

Public Members

uint8_t memoryAccessControlSelect

Select one of the 8 access control policies for this region, for access control policies see `trdc_memory_access_control_config_t`.

uint32_t startAddr

Physical start address.

bool valid

Lock the register.

bool nseEnable

Enable non-secure accesses and disable secure accesses.

uint32_t endAddr

Physical start address.

uint8_t mrcIdx

The index of the MRC for this configuration to take effect.

uint8_t domainIdx

The index of the domain for this configuration to take effect.

uint8_t regionIdx

The index of the region for this configuration to take effect.

struct `_trdc_mbc_nse_update_config`

#include <fsl_trdc.h> The configuration of MBC NSE update.

Public Members

uint32_t __pad0__

Reserved.

uint32_t wordIdx

MBC configuration word index to be updated.

uint32_t __pad1__

Reserved.

uint32_t memorySelect

Bit mask of the selected memory to be updated. `_trdc_MBC_memory`.

uint32_t __pad2__

Reserved.

uint32_t domainSelect

Bit mask of the selected domain to be updated. `_trdc_MBC_domain`.

uint32_t __pad3__

Reserved.

uint32_t autoIncrement

Whether to increment the word index after current word is updated using this configuration.

struct `_trdc_mbc_memory_block_config`

#include <fsl_trdc.h> The configuration of each memory block per domain per MBC instance.

Public Members

uint32_t memoryAccessControlSelect

Select one of the 8 access control policies for this memory block, for access control policies see trdc_memory_access_control_config_t.

uint32_t nseEnable

Enable non-secure accesses and disable secure accesses.

uint32_t mbcIdx

The index of the MBC for this configuration to take effect.

uint32_t domainIdx

The index of the domain for this configuration to take effect.

uint32_t slaveMemoryIdx

The index of the slave memory for this configuration to take effect.

uint32_t memoryBlockIdx

The index of the memory block for this configuration to take effect.

2.103 Trdc_core

typedef struct *_TRDC_General_Type* TRDC_General_Type

TRDC general configuration register definition.

typedef struct *_TRDC_FLW_Type* TRDC_FLW_Type

TRDC flash logical control register definition.

typedef struct *_TRDC_DomainError_Type* TRDC_DomainError_Type

TRDC domain error register definition.

typedef struct *_TRDC_DomainAssignment_Type* TRDC_DomainAssignment_Type

TRDC master domain assignment register definition.

typedef struct *_TRDC_MBC_Type* TRDC_MBC_Type

TRDC MBC control register definition.

typedef struct *_TRDC_MRC_Type* TRDC_MRC_Type

TRDC MRC control register definition. MRC_DOM0_RGD_W[region][word].

TRDC_GENERAL_BASE(base)

TRDC base address convert macro.

TRDC_FLW_BASE(base)

TRDC_DOMAIN_ERROR_BASE(base)

TRDC_DOMAIN_ASSIGNMENT_BASE(base)

TRDC_MBC_BASE(base, instance)

TRDC_MRC_BASE(base, instance)

struct *_TRDC_General_Type*

#include <fsl_trdc_core.h> TRDC general configuration register definition.

Public Members

__IO uint32_t TRDC_CR
TRDC Register, offset: 0x0

__I uint32_t TRDC_HWCFG0
TRDC Hardware Configuration Register 0, offset: 0xF0

__I uint32_t TRDC_HWCFG1
TRDC Hardware Configuration Register 1, offset: 0xF4

__I uint32_t TRDC_HWCFG2
TRDC Hardware Configuration Register 2, offset: 0xF8

__I uint32_t TRDC_HWCFG3
TRDC Hardware Configuration Register 3, offset: 0xFC

__I uint8_t DACFG [8]
Domain Assignment Configuration Register, array offset: 0x100, array step: 0x1

__IO uint32_t TRDC_IDAU_CR
TRDC IDAU Control Register, offset: 0x1C0

struct _TRDC_FLW_Type
#include <fsl_trdc_core.h> TRDC flash logical control register definition.

Public Members

__IO uint32_t TRDC_FLW_CTL
TRDC FLW Control, offset: 0x1E0

__I uint32_t TRDC_FLW_PBASE
TRDC FLW Physical Base, offset: 0x1E4

__IO uint32_t TRDC_FLW_ABASE
TRDC FLW Array Base, offset: 0x1E8

__IO uint32_t TRDC_FLW_BCNT
TRDC FLW Block Count, offset: 0x1EC

struct _TRDC_DomainError_Type
#include <fsl_trdc_core.h> TRDC domain error register definition.

Public Members

__IO uint32_t TRDC_FDIS
TRDC Fault Domain ID, offset: 0x1FC

__I uint32_t TRDC_DERRLOC [16]
TRDC Domain Error Location Register, array offset: 0x200, array step: 0x4

struct _TRDC_DomainAssignment_Type
#include <fsl_trdc_core.h> TRDC master domain assignment register definition.

Public Members

__IO uint32_t PID [8]
Process Identifier, array offset: 0x700, array step: 0x4

struct _TRDC_MBC_Type
#include <fsl_trdc_core.h> TRDC MBC control register definition.

Public Members

- ___I uint32_t MBC_MEM_GLBCFG [4]
MBC Global Configuration Register, array offset: 0x10000, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_NSE_BLK_INDEX
MBC NonSecure Enable Block Index, array offset: 0x10010, array step: 0x2000
- ___O uint32_t MBC_NSE_BLK_SET
MBC NonSecure Enable Block Set, array offset: 0x10014, array step: 0x2000
- ___O uint32_t MBC_NSE_BLK_CLR
MBC NonSecure Enable Block Clear, array offset: 0x10018, array step: 0x2000
- ___O uint32_t MBC_NSE_BLK_CLR_ALL
MBC NonSecure Enable Block Clear All, array offset: 0x1001C, array step: 0x2000
- ___IO uint32_t MBC_MEMN_GLBAC [8]
MBC Global Access Control, array offset: 0x10020, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10040, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10140, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10180, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x101A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x101A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x101C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x101D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x101F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10240, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10340, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM1_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10380, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x103A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x103A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x103C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x103D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x103F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10440, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10540, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10580, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x105A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x105A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x105C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x105D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x105F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10640, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM3_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10740, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10780, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x107A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x107A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x107C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x107D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x107F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10840, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10940, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10980, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x109A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x109A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x109C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x109D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x109F0, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM5_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10A40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10B40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10B80, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10BA0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10BA8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10BC8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10BD0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10BF0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10C40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10D40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10D80, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10DA0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10DA8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10DC8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10DD0, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM6_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10DF0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10E40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10F40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10F80, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10FA0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10FA8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10FC8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10FD0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10FF0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11040, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11140, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11180, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x111A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x111A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x111C8, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM8_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x111D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x111F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11240, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11340, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11380, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x113A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x113A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x113C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x113D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x113F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11440, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11540, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11580, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x115A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x115A8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM10_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x115C8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM10_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x115D0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM10_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x115F0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11640, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11740, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11780, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x117A0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x117A8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x117C8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x117D0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x117F0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM12_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11840, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM12_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11940, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM12_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11980, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM12_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x119A0, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM12_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x119A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM12_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x119C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM12_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x119D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM12_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x119F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11A40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11B40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11B80, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x11BA0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11BA8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x11BC8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11BD0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x11BF0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM14_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11C40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM14_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11D40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM14_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11D80, array step: index*0x2000, index2*0x4

```

__IO uint32_t MBC_DOM14_MEM1_BLK_NSE_W [2]
    MBC Memory Block NonSecure Enable Word, array offset: 0x11DA0, array step: in-
    dex*0x2000, index2*0x4

__IO uint32_t MBC_DOM14_MEM2_BLK_CFG_W [8]
    MBC Memory Block Configuration Word, array offset: 0x11DA8, array step: in-
    dex*0x2000, index2*0x4

__IO uint32_t MBC_DOM14_MEM2_BLK_NSE_W [2]
    MBC Memory Block NonSecure Enable Word, array offset: 0x11DC8, array step: in-
    dex*0x2000, index2*0x4

__IO uint32_t MBC_DOM14_MEM3_BLK_CFG_W [8]
    MBC Memory Block Configuration Word, array offset: 0x11DD0, array step: in-
    dex*0x2000, index2*0x4

__IO uint32_t MBC_DOM14_MEM3_BLK_NSE_W [2]
    MBC Memory Block NonSecure Enable Word, array offset: 0x11DF0, array step: in-
    dex*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM0_BLK_CFG_W [64]
    MBC Memory Block Configuration Word, array offset: 0x11E40, array step: in-
    dex*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM0_BLK_NSE_W [16]
    MBC Memory Block NonSecure Enable Word, array offset: 0x11F40, array step: in-
    dex*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM1_BLK_CFG_W [8]
    MBC Memory Block Configuration Word, array offset: 0x11F80, array step: in-
    dex*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM1_BLK_NSE_W [2]
    MBC Memory Block NonSecure Enable Word, array offset: 0x11FA0, array step: in-
    dex*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM2_BLK_CFG_W [8]
    MBC Memory Block Configuration Word, array offset: 0x11FA8, array step: in-
    dex*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM2_BLK_NSE_W [2]
    MBC Memory Block NonSecure Enable Word, array offset: 0x11FC8, array step: in-
    dex*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM3_BLK_CFG_W [8]
    MBC Memory Block Configuration Word, array offset: 0x11FD0, array step: in-
    dex*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM3_BLK_NSE_W [2]
    MBC Memory Block NonSecure Enable Word, array offset: 0x11FF0, array step: in-
    dex*0x2000, index2*0x4

struct _TRDC_MRC_Type
#include <fsl_trdc_core.h> TRDC MRC control register definition.
MRC_DOM0_RGD_W[region][word].

```

Public Members

```

__IO uint32_t MRC_GLBCFG
    MRC Global Configuration Register, array offset: 0x14000, array step: 0x1000

```

___IO uint32_t MRC_NSE_RGN_INDIRECT
MRC NonSecure Enable Region Indirect, array offset: 0x14010, array step: 0x1000

___O uint32_t MRC_NSE_RGN_SET
MRC NonSecure Enable Region Set, array offset: 0x14014, array step: 0x1000

___O uint32_t MRC_NSE_RGN_CLR
MRC NonSecure Enable Region Clear, array offset: 0x14018, array step: 0x1000

___O uint32_t MRC_NSE_RGN_CLR_ALL
MRC NonSecure Enable Region Clear All, array offset: 0x1401C, array step: 0x1000

___IO uint32_t MRC_GLBAC [8]
MRC Global Access Control, array offset: 0x14020, array step: index*0x1000, index2*0x4

___IO uint32_t MRC_DOM0_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14040, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM0_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x140C0, array step: 0x1000

___IO uint32_t MRC_DOM1_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14140, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM1_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x141C0, array step: 0x1000

___IO uint32_t MRC_DOM2_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14240, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM2_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x142C0, array step: 0x1000

___IO uint32_t MRC_DOM3_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14340, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM3_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x143C0, array step: 0x1000

___IO uint32_t MRC_DOM4_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14440, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM4_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x144C0, array step: 0x1000

___IO uint32_t MRC_DOM5_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14540, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM5_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x145C0, array step: 0x1000

___IO uint32_t MRC_DOM6_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14640, array step: index*0x1000, index2*0x8, index3*0x4

```

__IO uint32_t MRC_DOM6_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x146C0, array step: 0x1000
__IO uint32_t MRC_DOM7_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14740,
    array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM7_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x147C0, array step: 0x1000
__IO uint32_t MRC_DOM8_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14840,
    array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM8_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x148C0, array step: 0x1000
__IO uint32_t MRC_DOM9_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14940,
    array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM9_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x149C0, array step: 0x1000
__IO uint32_t MRC_DOM10_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14A40,
    array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM10_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x14AC0, array step: 0x1000
__IO uint32_t MRC_DOM11_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14B40,
    array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM11_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x14BC0, array step: 0x1000
__IO uint32_t MRC_DOM12_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14C40,
    array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM12_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x14CC0, array step: 0x1000
__IO uint32_t MRC_DOM13_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14D40,
    array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM13_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x14DC0, array step: 0x1000
__IO uint32_t MRC_DOM14_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14E40,
    array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM14_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x14EC0, array step: 0x1000
__IO uint32_t MRC_DOM15_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14F40,
    array step: index*0x1000, index2*0x8, index3*0x4

```

__IO uint32_t MRC_DOM15_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x14FC0, array step: 0x1000
struct MBC_DERR

Public Members

__I uint32_t W0
MBC Domain Error Word0 Register, array offset: 0x400, array step: 0x10
__I uint32_t W1
MBC Domain Error Word1 Register, array offset: 0x404, array step: 0x10
__O uint32_t W3
MBC Domain Error Word3 Register, array offset: 0x40C, array step: 0x10
struct MRC_DERR

Public Members

__I uint32_t W0
MRC Domain Error Word0 Register, array offset: 0x480, array step: 0x10
__I uint32_t W1
MRC Domain Error Word1 Register, array offset: 0x484, array step: 0x10
__O uint32_t W3
MRC Domain Error Word3 Register, array offset: 0x48C, array step: 0x10
union __unnamed217__

Public Members

struct *_TRDC_DomainAssignment_Type* MDA_DFMT0[8]
struct *_TRDC_DomainAssignment_Type* MDA_DFMT1[8]
struct MDA_DFMT0

Public Members

__IO uint32_t MDA_W_DFMT0 [8]
DAC Master Domain Assignment Register, array offset: 0x800, array step: index*0x20,
index2*0x4
struct MDA_DFMT1

Public Members

__IO uint32_t MDA_W_DFMT1 [1]
DAC Master Domain Assignment Register, array offset: 0x800, array step: index*0x20,
index2*0x4

2.104 Trdc_soc

enum _trdc_master

Enumeration for TRDC master mapping.

Defines the enumeration for TRDC master resource collections.

Values:

enumerator kTRDC1_MasterReserved
Reserved

enumerator kTRDC1_MasterCM33
CM33

enumerator kTRDC1_MasterEDMA1
EDMA1

enumerator kTRDC1_MasterMTR_FBX
MTR FBX

enumerator kTRDC1_MasterMTR
MTR

enumerator kTRDC2_MasterReserved0
Reserved0

enumerator kTRDC2_MasterReserved1
Reserved1

enumerator kTRDC2_MasterDAP
DAP AHB_AP_SYS

enumerator kTRDC2_MasterCoreSight
CoreSight

enumerator kTRDC2_MasterEDMA2
EDMA2

enumerator kTRDC3_MasterUSDHC1
uSDHC1

enumerator kTRDC3_MasterUSDHC2
uSDHC2

enumerator kTRDC3_MasterTestPort
Test port

enumerator kTRDC3_MasterUSDHC3
USDHC3

enumerator kTRDC3_MasterENET0
ENET0

enumerator kTRDC3_MasterENET1
ENET1

enumerator kTRDC3_MasterENETQos
ENET Qos

enumerator kTRDC3_MasterCA55Read
CA55 read channel

enumerator kTRDC3_MasterCA55Write
CA55 write channel

enumerator kTRDC3_MasterNPUm0
NPU m0

enumerator kTRDC3_MasterNPUm1
NPU m1

enumerator kTRDC_MediaMix_MasterISI_M
ISI M

enumerator kTRDC_MediaMix_MasterISI_U_V
ISI U&V

enumerator kTRDC_MediaMix_MasterPXP
PXP

enumerator kTRDC_MediaMix_MasterLCDIF
Lcdif

enumerator kTRDC_HSIOMix_MasterUSB1
USB1

enumerator kTRDC_HSIOMix_MasterUSB2
USB2

typedef enum *_trdc_master* trdc_master_t
Enumeration for TRDC master mapping.

Defines the enumeration for TRDC master resource collections.

typedef void TRDC_Type
TRDC typedef.

FSL_TRDC_SOC_DRIVER_VERSION
Driver version 2.0.0.

TRDC_DACFG_NCM_MASK

TRDC_MBC_MEM_GLBCFG_NBLKS_MASK

TRDC_MBC_MEM_GLBCFG_SIZE_LOG2_MASK

TRDC_MBC_MEM_GLBCFG_SIZE_LOG2_SHIFT

TRDC_MBC_NSE_BLK_CLR_ALL_DID_SEL0_SHIFT

TRDC_MDA_W_DFMT0_LK1_MASK

TRDC_MDA_W_DFMT0_VLD_MASK

TRDC_MDA_W_DFMT1_LK1_MASK

TRDC_MDA_W_DFMT1_VLD_MASK

TRDC_MRC_DOM0_RGD_W_MRACSEL_MASK

TRDC_MRC_DOM0_RGD_W_NSE_MASK

TRDC_MRC_DOM0_RGD_W_VLD_MASK

TRDC_MRC_DOM0_RGD_W_END_ADDR_MASK

TRDC_MRC_DOM0_RGD_W_STRT_ADDR_MASK

TRDC_MRC_GLBCFG_NRGNS_MASK

TRDC_MRC_GLBCFG_NRGNS_SHIFT

TRDC_TRDC_CR_GVLDB_MASK

TRDC_TRDC_CR_GVLDM_MASK

TRDC_TRDC_CR_GVLDR_MASK

TRDC_TRDC_FLW_CTL_LK_MASK

TRDC_TRDC_FLW_CTL_V_MASK

TRDC_TRDC_HWCFG0_NDID_MASK

TRDC_TRDC_HWCFG0_NDID_SHIFT

TRDC_TRDC_HWCFG0_NMBC_MASK

TRDC_TRDC_HWCFG0_NMBC_SHIFT

TRDC_TRDC_HWCFG0_NMRC_MASK

TRDC_TRDC_HWCFG0_NMRC_SHIFT

TRDC_TRDC_HWCFG0_NMSTR_MASK

TRDC_TRDC_HWCFG0_NMSTR_SHIFT

TRDC_TRDC_HWCFG1_DID_MASK

TRDC_TRDC_HWCFG1_DID_SHIFT

TRDC_TRDC_IDAU_CR_VLD_MASK

TRDC_W1_EATR_MASK

TRDC_W1_EATR_SHIFT

TRDC_W1_EDID_MASK

TRDC_W1_EPORT_MASK

TRDC_W1_EPORT_SHIFT

TRDC_W1_ERW_MASK

TRDC_W1_ERW_SHIFT

TRDC_W1_EST_MASK

TRDC_W1_EST_SHIFT

TRDC_MBC_NSE_BLK_CLR_ALL_MEMSEL

TRDC_MRC_DOM0_RGD_W_MRACSEL

TRDC_MRC_DOM0_RGD_W_NSE

TRDC_MRC_DOM0_RGD_W_VLD

TRDC_TRDC_FDID_FDID

TRDC_TRDC_FLW_CTL_LK

TRDC_TRDC_FLW_CTL_V

TRDC_W3_RECR

TRDC_BASE_PTRS

TRDC base table.

TRDC_GENERAL_OFFSET

TRDC base address convert macro.

TRDC_FLW_OFFSET

TRDC_DOMAIN_ERROR_OFFSET

TRDC_DOMAIN_ASSIGNMENT_OFFSET

TRDC_MBC_OFFSET(**x**)

TRDC_MBC_ARRAY_STEP

TRDC_MRC_OFFSET(**x**)

TRDC_MRC_ARRAY_STEP

2.105 TRGMUX: Trigger Mux Driver

`static inline void TRGMUX_LockRegister(TRGMUX_Type *base, uint32_t index)`

Sets the flag of the register which is used to mark writeable.

The function sets the flag of the register which is used to mark writeable. Example:

```
TRGMUX_LockRegister(TRGMUX0, kTRGMUX_Trgmux0Dmamux0);
```

Parameters

- `base` – TRGMUX peripheral base address.
- `index` – The index of the TRGMUX register, see the enum `trgmux_device_t` defined in `<SOC>.h`.

`status_t TRGMUX_SetTriggerSource(TRGMUX_Type *base, uint32_t index, trgmux_trigger_input_t input, uint32_t trigger_src)`

Configures the trigger source of the appointed peripheral.

The function configures the trigger source of the appointed peripheral. Example:

```
TRGMUX_SetTriggerSource(TRGMUX0, kTRGMUX_Trgmux0Dmamux0, kTRGMUX_TriggerInput0,  
↪ kTRGMUX_SourcePortPin);
```

Parameters

- `base` – TRGMUX peripheral base address.
- `index` – The index of the TRGMUX register, see the enum `trgmux_device_t` defined in `<SOC>.h`.
- `input` – The MUX select for peripheral trigger input
- `trigger_src` – The trigger inputs for various peripherals. See the enum `trgmux_source_t` defined in `<SOC>.h`.

Return values

- `kStatus_Success` – Configured successfully.
- `kStatus_TRGMUX_Locked` – Configuration failed because the register is locked.

`FSL_TRGMUX_DRIVER_VERSION`
TRGMUX driver version.

TRGMUX configure status.

Values:

enumerator `kStatus_TRGMUX_Locked`
Configure failed for register is locked

enum `_trgmux_trigger_input`
Defines the MUX select for peripheral trigger input.

Values:

enumerator `kTRGMUX_TriggerInput0`
The MUX select for peripheral trigger input 0

enumerator `kTRGMUX_TriggerInput1`
The MUX select for peripheral trigger input 1

enumerator `kTRGMUX_TriggerInput2`
The MUX select for peripheral trigger input 2

enumerator `kTRGMUX_TriggerInput3`
The MUX select for peripheral trigger input 3

typedef enum `_trgmux_trigger_input` `trgmux_trigger_input_t`
Defines the MUX select for peripheral trigger input.

2.106 TSTMR: Timestamp Timer Driver

`void TSTMR_Init(TSTMR_Type *base)`
Init TSTMR.
This function initializes the TSTMR module.

Parameters

- `base` – TSTMR peripheral base address.

`void TSTMR_Deinit(TSTMR_Type *base)`
Deinit TSTMR.
This function deinitializes the TSTMR module.

Parameters

- `base` – TSTMR peripheral base address.

`FSL_TSTMR_DRIVER_VERSION`
Version 2.1.0

```
static inline uint64_t TSTMR_ReadTimeStamp(TSTMR_Type *base)
```

Reads the time stamp.

This function reads the low and high registers and returns the 56-bit free running counter value. This can be read by software at any time to determine the software ticks. TSTMR registers can be read with 32-bit accesses only. The TSTMR LOW read should occur first, followed by the TSTMR HIGH read.

Parameters

- base – TSTMR peripheral base address.

Returns

The 56-bit time stamp value.

```
void TSTMR_DelayUs(TSTMR_Type *base, uint64_t delayInUs)
```

Delays for a specified number of microseconds.

This function repeatedly reads the timestamp register and waits for the user-specified delay value.

Parameters

- base – TSTMR peripheral base address.
- delayInUs – Delay value in microseconds.

2.107 CACHE: CACHE Memory Controller

```
uint32_t XCACHE_GetInstanceByAddr(uint32_t address)
```

brief Returns an instance number given physical memory address.

param address The physical memory address.

Returns

XCACHE instance number starting from 0.

```
void XCACHE_EnableCache(XCACHE_Type *base)
```

Enables the cache.

Parameters

- base – XCACHE peripheral base address.

```
void XCACHE_DisableCache(XCACHE_Type *base)
```

Disables the cache.

Parameters

- base – XCACHE peripheral base address.

```
void XCACHE_InvalidateCache(XCACHE_Type *base)
```

Invalidates the cache.

Parameters

- base – XCACHE peripheral base address.

```
void XCACHE_InvalidateCacheByRange(uint32_t address, uint32_t size_byte)
```

Invalidates cache by range.

Note: Address and size should be aligned to “XCACHE_LINESIZE_BYTE”. The startAddr here will be forced to align to XCACHE_LINESIZE_BYTE if startAddr is not aligned. For the

size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address of cache.
- size_byte – size of the memory to be invalidated, should be larger than 0, better to align with cache line size.

void XCACHE_CleanCache(XCACHE_Type *base)

Cleans the cache.

Parameters

- base – XCACHE peripheral base address.

void XCACHE_CleanCacheByRange(uint32_t address, uint32_t size_byte)

Cleans cache by range.

Note: Address and size should be aligned to “XCACHE_LINESIZE_BYTE”. The startAddr here will be forced to align to XCACHE_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address of cache.
- size_byte – size of the memory to be cleaned, should be larger than 0, better to align with cache line size.

void XCACHE_CleanInvalidateCache(XCACHE_Type *base)

Cleans and invalidates the cache.

Parameters

- base – XCACHE peripheral base address.

void XCACHE_CleanInvalidateCacheByRange(uint32_t address, uint32_t size_byte)

Cleans and invalidate cache by range.

Note: Address and size should be aligned to “XCACHE_LINESIZE_BYTE”. The startAddr here will be forced to align to XCACHE_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address of cache.
- size_byte – size of the memory to be Cleaned and Invalidated, should be larger than 0, better to align with cache line size.

static inline void ICACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates instruction cache by range.

Note: Address and size should be aligned to XCACHE_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_XCACHE_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application

should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be invalidated, should be larger than 0, better to align with cache line size.

static inline void DCACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates data cache by range.

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

Parameters

- address – The physical address.
- size_byte – size of the memory to be invalidated, should be larger than 0, better to align with cache line size.

static inline void DCACHE_CleanByRange(uint32_t address, uint32_t size_byte)

Clean data cache by range.

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

Parameters

- address – The physical address.
- size_byte – size of the memory to be cleaned, should be larger than 0, better to align with cache line size.

static inline void DCACHE_CleanInvalidateByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates data cache by range.

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

Parameters

- address – The physical address.
- size_byte – size of the memory to be Cleaned and Invalidated, should be larger than 0, better to align with cache line size.

FSL_CACHE_DRIVER_VERSION

cache driver version.

XCACHE_LINESIZE_BYTE

cache line size.

Chapter 3

Middleware

Chapter 4

RTOS

4.1 FreeRTOS

4.1.1 FreeRTOS kernel

Open source RTOS kernel for small devices.

[FreeRTOS kernel for MCUXpresso SDK Readme](#)

[FreeRTOS kernel for MCUXpresso SDK ChangeLog](#)

[FreeRTOS kernel Readme](#)

4.1.2 FreeRTOS drivers

This is set of NXP provided FreeRTOS reentrant bus drivers.

4.1.3 backoffalgorithm

Algorithm for calculating exponential backoff with jitter for network retry attempts.

[Readme](#)

4.1.4 corehttp

C language HTTP client library designed for embedded platforms.

4.1.5 corejson

JSON parser.

Readme

4.1.6 coremqtt

MQTT publish/subscribe messaging library.

4.1.7 corepkcs11

PKCS #11 key management library.

Readme

4.1.8 freertos-plus-tcp

Open source RTOS FreeRTOS Plus TCP.

Readme