



MCUXpresso SDK Documentation

Release 25.12.00-pvw2



NXP
Nov 14, 2025



Table of contents

1	MIMXRT1160-EVK	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	44
1.3.1	Getting Started with MCUXpresso SDK Repository	44
1.4	Release Notes	57
1.4.1	MCUXpresso SDK Release Notes	57
1.5	ChangeLog	64
1.5.1	MCUXpresso SDK Changelog	64
1.6	Driver API Reference Manual	198
1.7	Middleware Documentation	198
1.7.1	VG-Lite GPU Library	198
1.7.2	Multicore	198
1.7.3	MCU Boot	198
1.7.4	Audio Voice components	198
1.7.5	Maestro Audio Framework for MCU	198
1.7.6	eIQ	199
1.7.7	FreeMASTER	199
1.7.8	AWS IoT	199
1.7.9	NXP Wi-Fi	199
1.7.10	FreeRTOS	199
1.7.11	lwIP	199
1.7.12	File systemFatfs	199
2	MIMXRT1166	201
2.1	ACMP: Analog Comparator Driver	201
2.2	ADC_ETC: ADC External Trigger Control	209
2.3	AOI: Crossbar AND/OR/INVERT Driver	213
2.4	ASRC: Asynchronous sample rate converter	216
2.5	ASRC Driver	216
2.6	ASRC EDMA Driver	232
2.7	CAAM: Cryptographic Acceleration and Assurance Module	236
2.8	CAAM AES driver	245
2.9	CAAM Key Blankening driver	254
2.10	CAAM Blob driver	254
2.11	CAAM CRC driver	258
2.12	CAAM DES driver	260
2.13	Caam_driver_ecc	270
2.14	CAAM HASH driver	272
2.15	Caam_driver_hmac	275
2.16	CAAM PKHA driver	276
2.17	CAAM RNG driver	284
2.18	Caam_driver_rsa	287
2.19	CAAM Blocking APIs	289
2.20	CAAM Non-blocking APIs	289

2.21	CAAM Non-blocking AES driver	289
2.22	CAAM Non-blocking DES driver	299
2.23	CAAM Non-blocking HASH driver	310
2.24	Caam_nonblocking_driver_hmac	312
2.25	CAAM Non-blocking RNG driver	312
2.26	CACHE: ARMV7-M7 CACHE Memory Controller	313
2.27	CACHE: LMEM CACHE Memory Controller	316
2.28	CDOG	320
2.29	Clock Driver	324
2.30	MIPI CSI2 RX: MIPI CSI2 RX Driver	399
2.31	CSI: CMOS Sensor Interface	405
2.32	DAC12: 12-bit Digital-to-Analog Converter Driver	414
2.33	DCIC: Display Content Integrity Checker	420
2.34	DCIC	420
2.35	DMAMUX: Direct Memory Access Multiplexer Driver	425
2.36	eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver	427
2.37	eLCDIF: Enhanced LCD Interface	446
2.38	ENC: Quadrature Encoder/Decoder	456
2.39	ENET: Ethernet MAC Driver	467
2.40	EWM: External Watchdog Monitor Driver	497
2.41	FlexCAN: Flex Controller Area Network Driver	500
2.42	FlexCAN Driver	500
2.43	FlexCAN eDMA Driver	541
2.44	FlexIO: FlexIO Driver	543
2.45	FlexIO Driver	543
2.46	FlexIO eDMA I2S Driver	561
2.47	FlexIO eDMA SPI Driver	564
2.48	FlexIO eDMA UART Driver	567
2.49	FlexIO I2C Master Driver	570
2.50	FlexIO I2S Driver	579
2.51	FlexIO SPI Driver	589
2.52	FlexIO UART Driver	602
2.53	FLEXRAM: on-chip RAM manager	613
2.54	FLEXSPI: Flexible Serial Peripheral Interface Driver	621
2.55	FLEXSPI eDMA Driver	638
2.56	GPIO: General-Purpose Input/Output Driver	641
2.57	GPT: General Purpose Timer	645
2.58	IEE: Inline Encryption Engine	652
2.59	Ieer	656
2.60	IOMUXC: IOMUX Controller	657
2.61	Key_manager	701
2.62	KPP: KeyPad Port Driver	704
2.63	Common Driver	706
2.64	LCDIFv2: LCD Interface v2	719
2.65	Lin_lpuart_driver	731
2.66	LPADC: 12-bit SAR Analog-to-Digital Converter Driver	738
2.67	LPI2C: Low Power Inter-Integrated Circuit Driver	757
2.68	LPI2C Master Driver	758
2.69	LPI2C Master DMA Driver	773
2.70	LPI2C Slave Driver	775
2.71	LPSPi: Low Power Serial Peripheral Interface	786
2.72	LPSPi Peripheral driver	786
2.73	LPSPi eDMA Driver	807
2.74	LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver	814
2.75	LPUART Driver	814
2.76	LPUART eDMA Driver	833
2.77	MCM: Miscellaneous Control Module	836
2.78	MECC: internal error correction code	840

2.79	MIPI DSI Driver	848
2.80	MIPI_DSI: MIPI DSI Host Controller	866
2.81	MU: Messaging Unit	866
2.82	OCOTP: On Chip One-Time Programmable controller.	874
2.83	OTFAD: On The Fly AES-128 Decryption Driver	878
2.84	PDM: Microphone Interface	880
2.85	PDM Driver	880
2.86	PDM EDMA Driver	898
2.87	PIT: Periodic Interrupt Timer	902
2.88	PUF: Physical Unclonable Function	906
2.89	PWM: Pulse Width Modulator	908
2.90	PXP: Pixel Pipeline	933
2.91	QTMR: Quad Timer Driver	969
2.92	RDC: Resource Domain Controller	979
2.93	RDC_SEMA42: Hardware Semaphores Driver	985
2.94	Romapi	988
2.95	RTWDOG: 32-bit Watchdog Timer	1004
2.96	SAI: Serial Audio Interface	1010
2.97	SAI Driver	1010
2.98	SAI EDMA Driver	1036
2.99	SEMA4: Hardware Semaphores Driver	1043
2.100	SEMC: Smart External DRAM Controller Driver	1046
2.101	Smart Card	1071
2.102	Smart Card EMVSIM Driver	1079
2.103	Smart Card PHY Driver	1082
2.104	Smart Card PHY EMVSIM Driver	1083
2.105	Smart Card PHY TDA8035 Driver	1083
2.106	SNVS: Secure Non-Volatile Storage	1083
2.107	Secure Non-Volatile Storage High-Power	1083
2.108	Secure Non-Volatile Storage Low-Power	1092
2.109	Soc_mipi_csi2rx	1101
2.110	Soc_mipi_dsi	1102
2.111	SPDIF: Sony/Philips Digital Interface	1102
2.112	SPDIF eDMA Driver	1116
2.113	SSARC: State Save and Restore Controller	1119
2.114	TEMPSENSOR: Temperature Sensor Module	1126
2.115	USDHC: Ultra Secured Digital Host Controller Driver	1129
2.116	WDOG: Watchdog Timer Driver	1158
2.117	XBARA: Inter-Peripheral Crossbar Switch	1163
2.118	XBARB: Inter-Peripheral Crossbar Switch	1165
2.119	XECC: external error correction code controller	1166
2.120	XRDC2: Extended Resource Domain Controller 2	1170
3	Middleware	1183
3.1	Boot	1183
3.1.1	MCUXpresso SDK : mcuxsdk-middleware-mcuboot_opensource	1183
3.1.2	MCUboot	1184
3.2	eIQ	1185
3.2.1	eIQ	1185
3.3	Motor Control	1213
3.3.1	FreeMASTER	1214
3.4	Multimedia	1251
3.4.1	Audio Voice	1251
4	RTOS	1305
4.1	FreeRTOS	1305
4.1.1	FreeRTOS kernel	1305
4.1.2	FreeRTOS drivers	1305

4.1.3	backoffalgorithm	1305
4.1.4	corehttp	1305
4.1.5	corejson	1305
4.1.6	coremqtt	1306
4.1.7	corepkcs11	1306
4.1.8	freertos-plus-tcp	1306

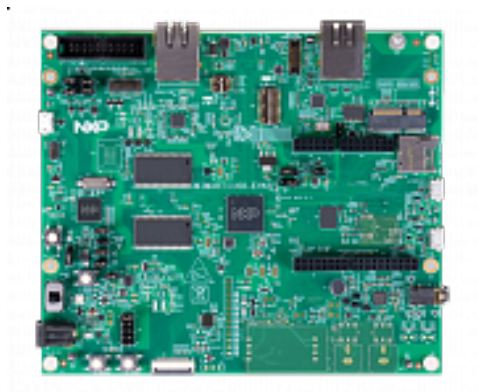
This documentation contains information specific to the evkmimxrt1160 board.

Chapter 1

MIMXRT1160-EVK

1.1 Overview

i.MX RT1160 crossover MCUs are part of the EdgeVerse edge computing platform, achieving 600MHz performance. This MCU family combines superior computing power and multiple media capabilities with ease of use and real-time functionality. The dual core i.MX RT1160 MCU runs on the Arm Cortex-M7 core at 600 MHz and Arm Cortex-M4 at 240 MHz, while providing excellent security. The i.MX RT1160 MCU offers support over a wide temperature range and is designed for consumer, industrial and automotive markets. The i.MX RT1160 evaluation kit (EVK) provides a high-performance solution enabled by a 6-layer PCB with through-hole design for better EMC performance - all at a low cost.



MCU device and part on board is shown below:

- Device: MIMXRT1166
- PartNumber: MIMXRT1166DVM6A

1.2 Getting Started with MCUXpresso SDK Package

1.2.1 Getting Started with Package

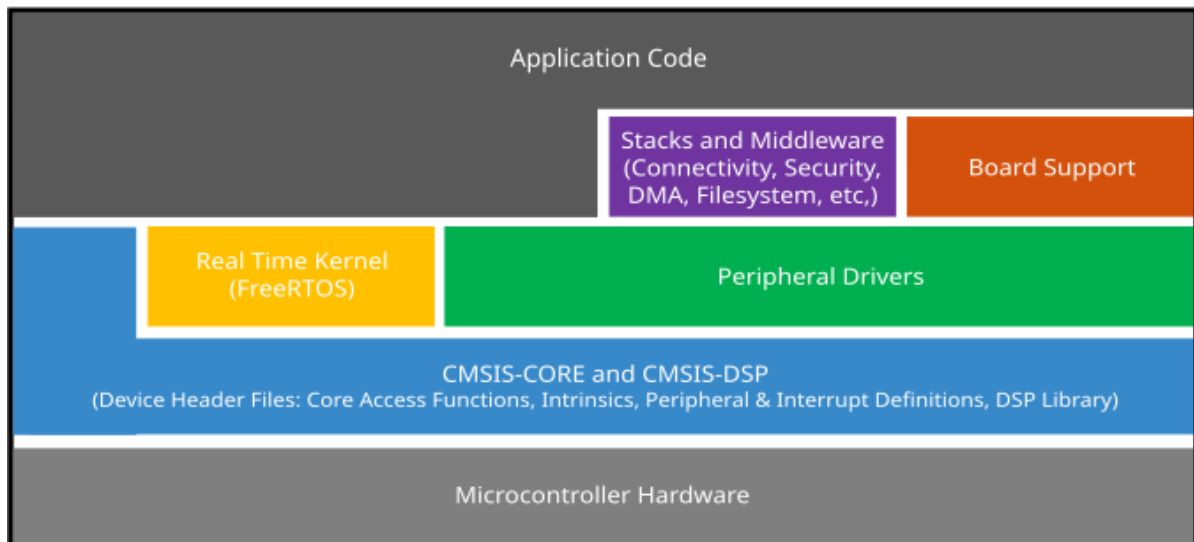
Overview

The MCUXpresso Software Development Kit (SDK) provides comprehensive software support for Kinetis and LPC Microcontrollers. The MCUXpresso SDK includes a flexible set of peripheral drivers designed to speed up and simplify development of embedded applications. Along with the peripheral drivers, the MCUXpresso SDK provides an extensive and rich set of example

applications covering everything from basic peripheral use case examples to full demo applications. The MCUXpresso SDK contains FreeRTOS and various other middleware to support rapid development.

For supported toolchain versions, see *MCUXpresso SDK Release Notes for MIMXRT1160-EVK* (document MCUXSDKMIMXRT116XRN).

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



MCUXpresso SDK board support package folders

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

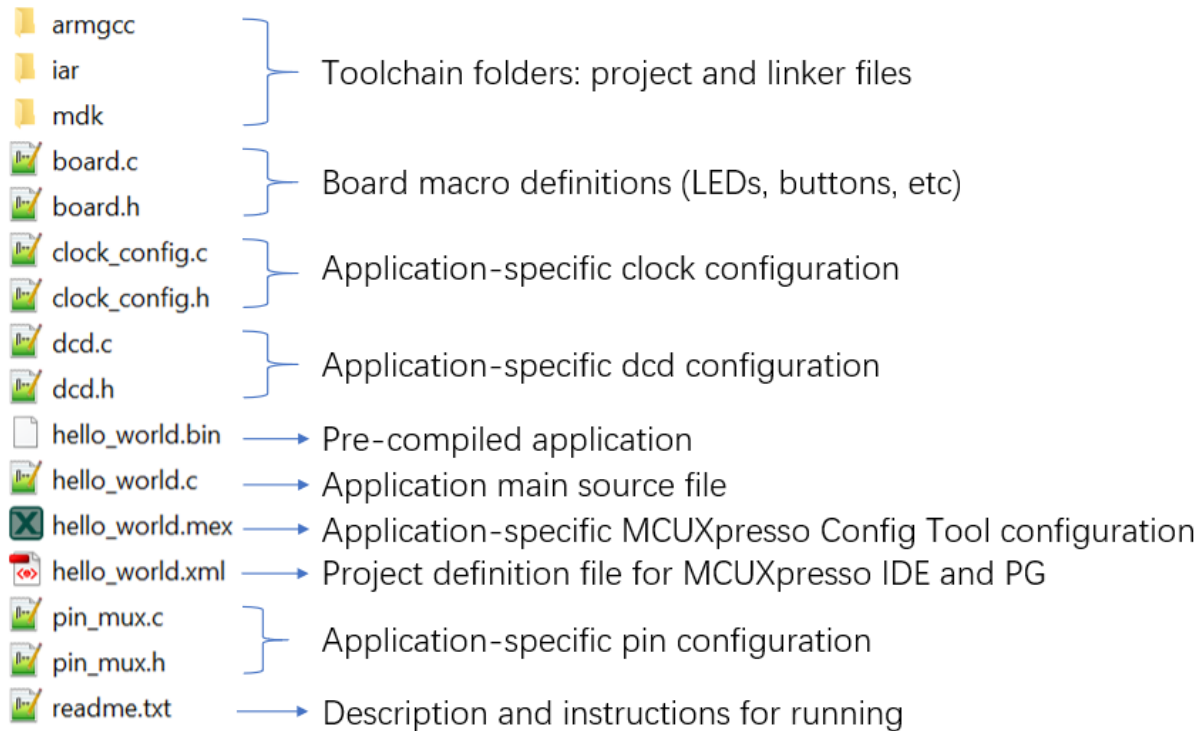
- `cmsis_driver_examples`: Simple applications intended to show how to use CMSIS drivers.
- `demo_apps`: Full-featured applications that highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- `driver_examples`: Simple applications that show how to use the MCUXpresso SDK's peripheral drivers for a single use case. These applications typically only use a single peripheral but there are cases where multiple peripherals are used (for example, SPI conversion using DMA).
- `rtos_examples`: Basic FreeRTOS™ OS examples that show the use of various RTOS objects (semaphores, queues, and so on) and interfaces with the MCUXpresso SDK's RTOS drivers.
- `usb_examples`: Applications that use the USB host/device/OTG stack.
- `multicore_examples`: Applications for both cores showing the usage of multicore software components and the interaction between cores.
- `Other_examples`: See detail in package *boards/evkmimxrt1160*.

Example application structure This section describes how the various types of example applications interact with the other components in the MCUXpresso SDK. To get a comprehensive

understanding of all MCUXpresso SDK components and folder structure, see *MCUXpresso SDK API Reference Manual*.

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

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



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

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

Locating example application source files When opening an example application in any of the supported IDEs, 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>/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 package folders](#)

Run a demo using MCUXpresso IDE

Note:

Most MCUXpresso projects provide two targets (debug and release). For CM7 projects, they are actually flash target. For CM4 projects, they are linked to RAM. To debug and run the CM7 examples, set **SW1[1:4]** to 0010 as internal flash boot mode. Currently, MCUXpresso IDE does not support CM4 download/debug.

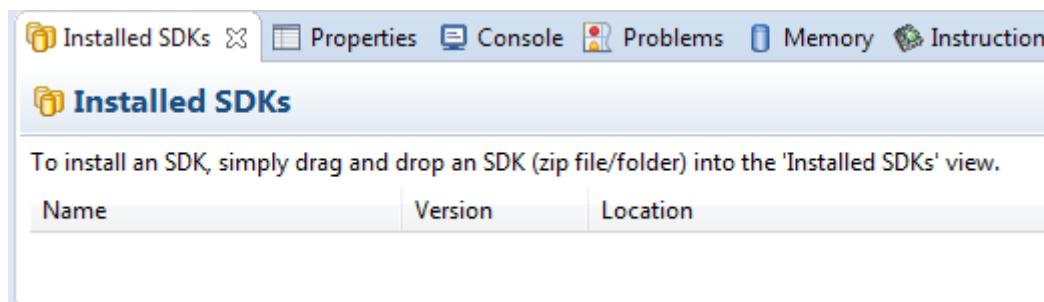
This section describes the steps required to configure MCUXpresso IDE to build, run, and debug example applications. The `hello_world` demo application targeted for the MIMXRT1160-EVK hardware platform is used as an example, though these steps can be applied to any example application in the MCUXpresso SDK.

Select the workspace location Every time MCUXpresso IDE launches, it prompts the user to select a workspace location. MCUXpresso IDE is built on top of Eclipse which uses workspace to store information about its current configuration, and in some use cases, source files for the projects are in the workspace. The location of the workspace can be anywhere, but it is recommended that the workspace be located outside of the MCUXpresso SDK tree.

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

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

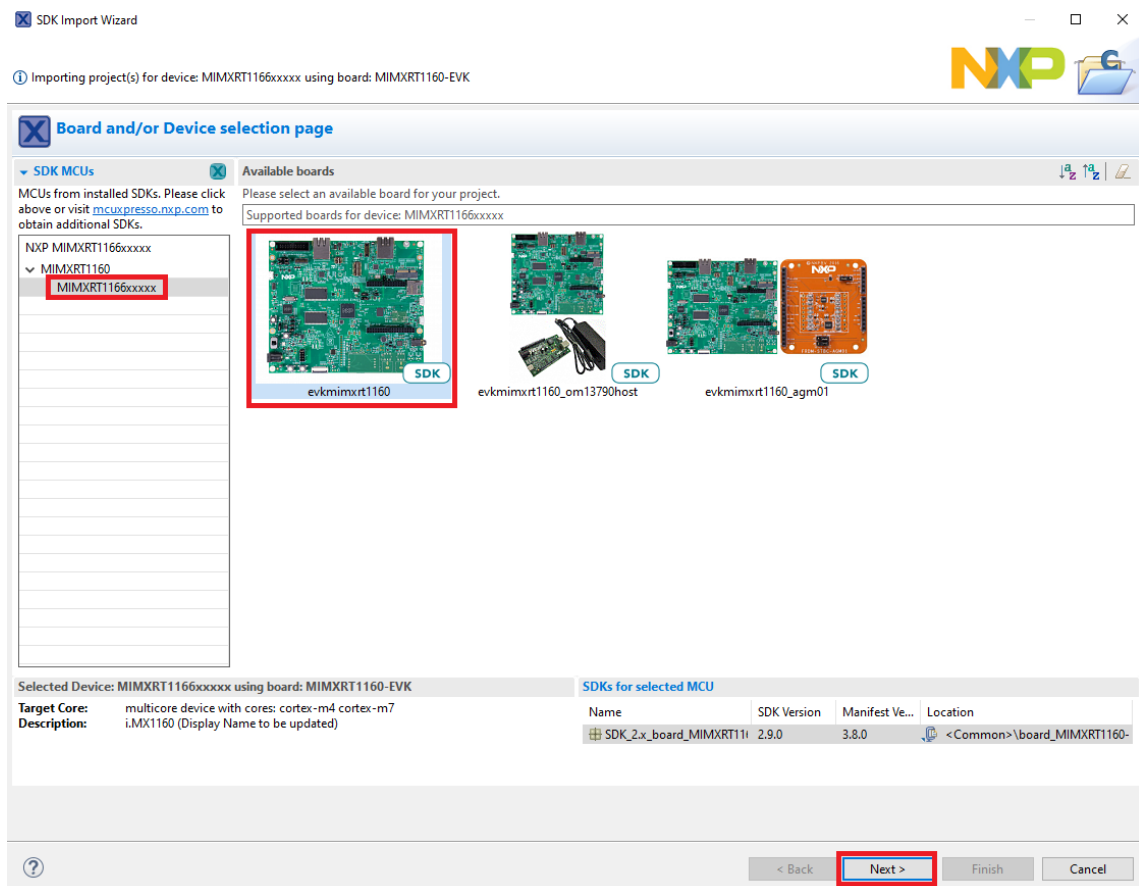
1. Drag and drop the SDK zip file into the **Installed SDKs** view to install an SDK. In the window that appears, click **OK** and wait until the import has finished.



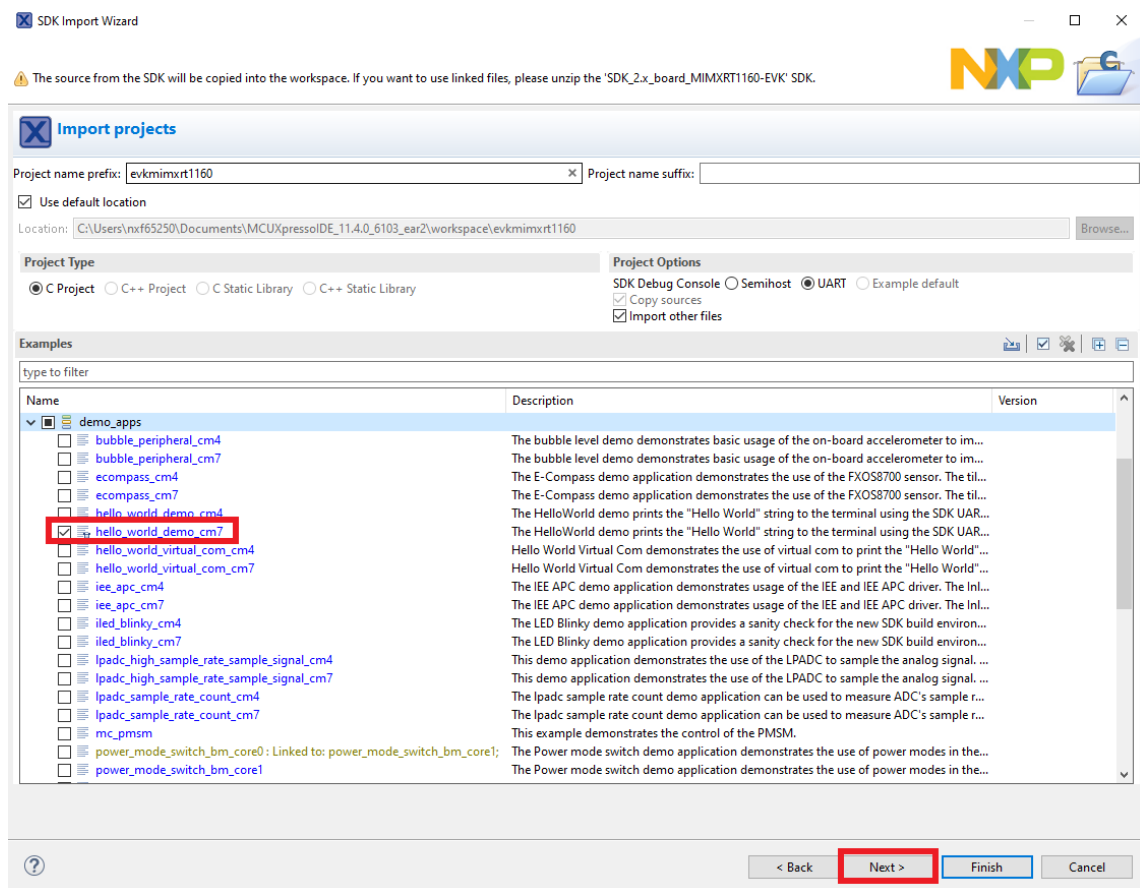
2. On the **Quickstart Panel**, click **Import SDK example(s)....**



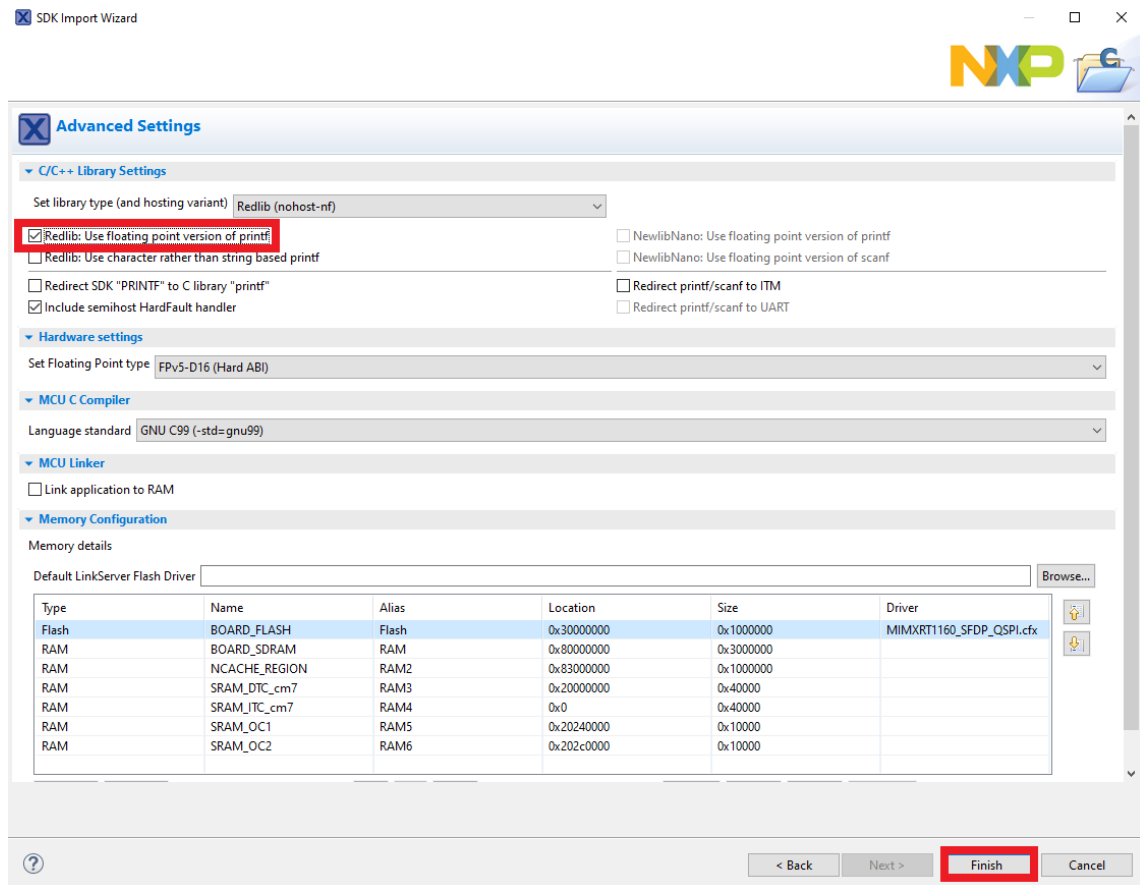
3. In the window that appears, select **MIMXRT1166xxxxx**. Then, select **evkmimxrt1160** and click **Next**.



4. Expand the *demo_apps* folder and select *hello_world*. Then, click **Next**.



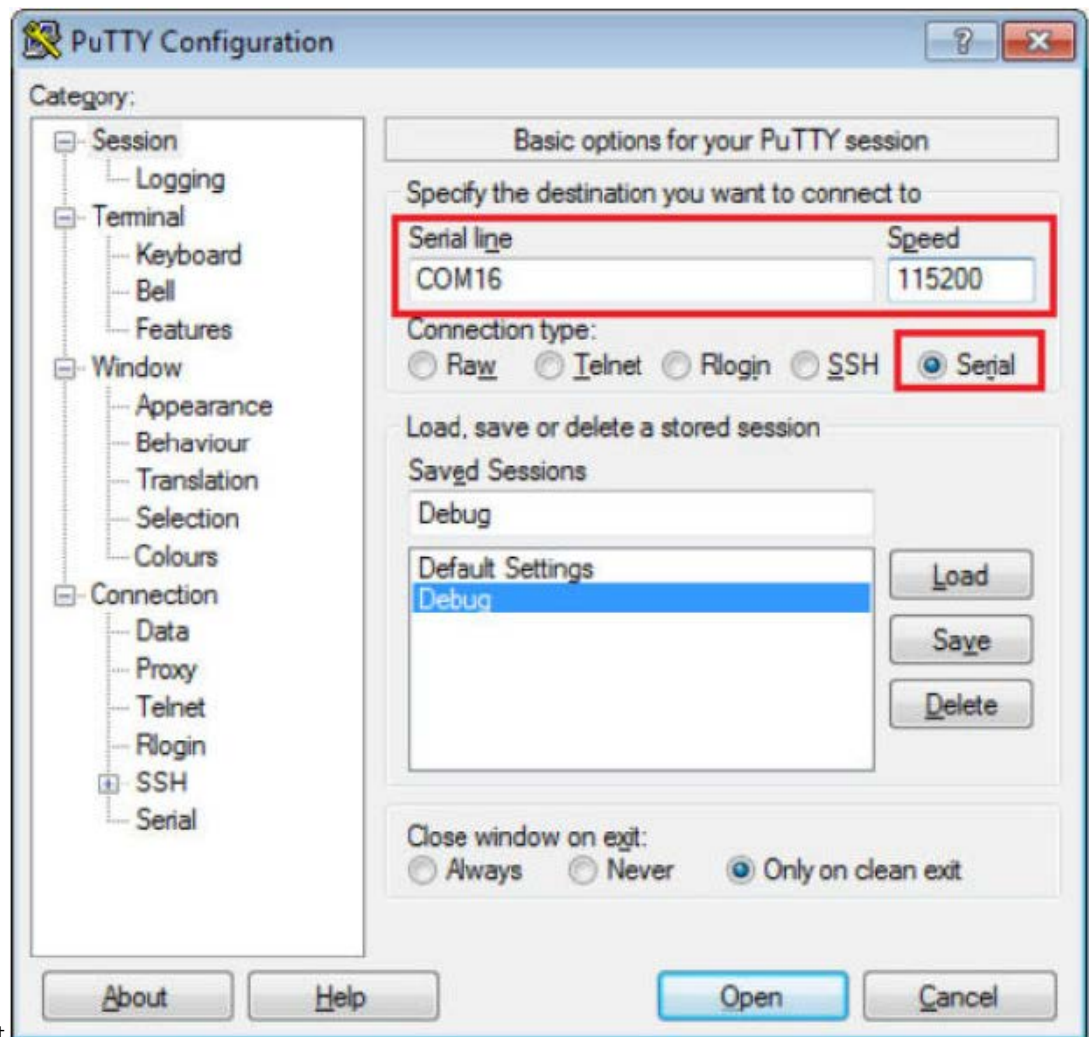
5. Ensure **Redlib: Use floating point version of printf** is selected if the example prints floating point numbers on the terminal for demo applications such as `adc_basic`, `adc_burst`, `adc_dma`, and `adc_interrupt`. Otherwise, it is not necessary to select this option. Then, click **Finish**.



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

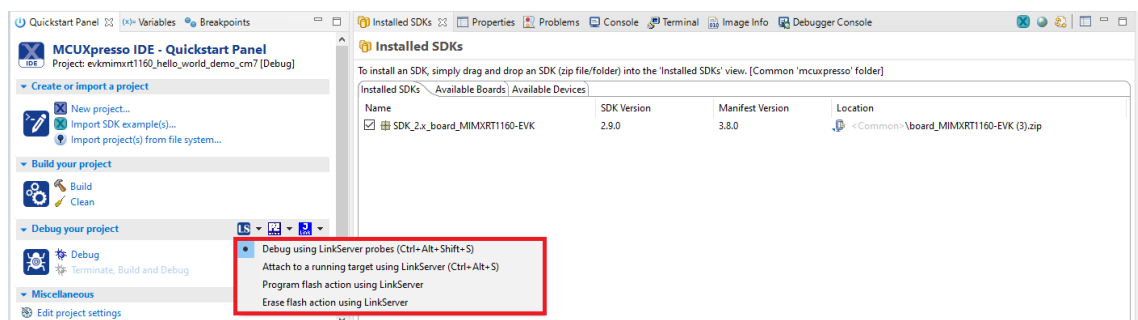
Run an example application To download and run the application, perform the following steps:

- See [Table 1](#) to determine the debug interface that comes loaded on your specific hardware platform.
 - If using J-Link with either a standalone debug pod or OpenSDA, install the J-Link software (drivers and utilities) from [SEGGER](#).
 - For boards with the OSJTAG interface, install the driver from [KEIL](#).
- Connect the development platform to your PC via a USB cable.
- Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number. To determine the COM port number, see [How to determine COM port](#). Configure the terminal with these settings:
 - 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the *board.h* file)
 - No parity
 - 8 data bits

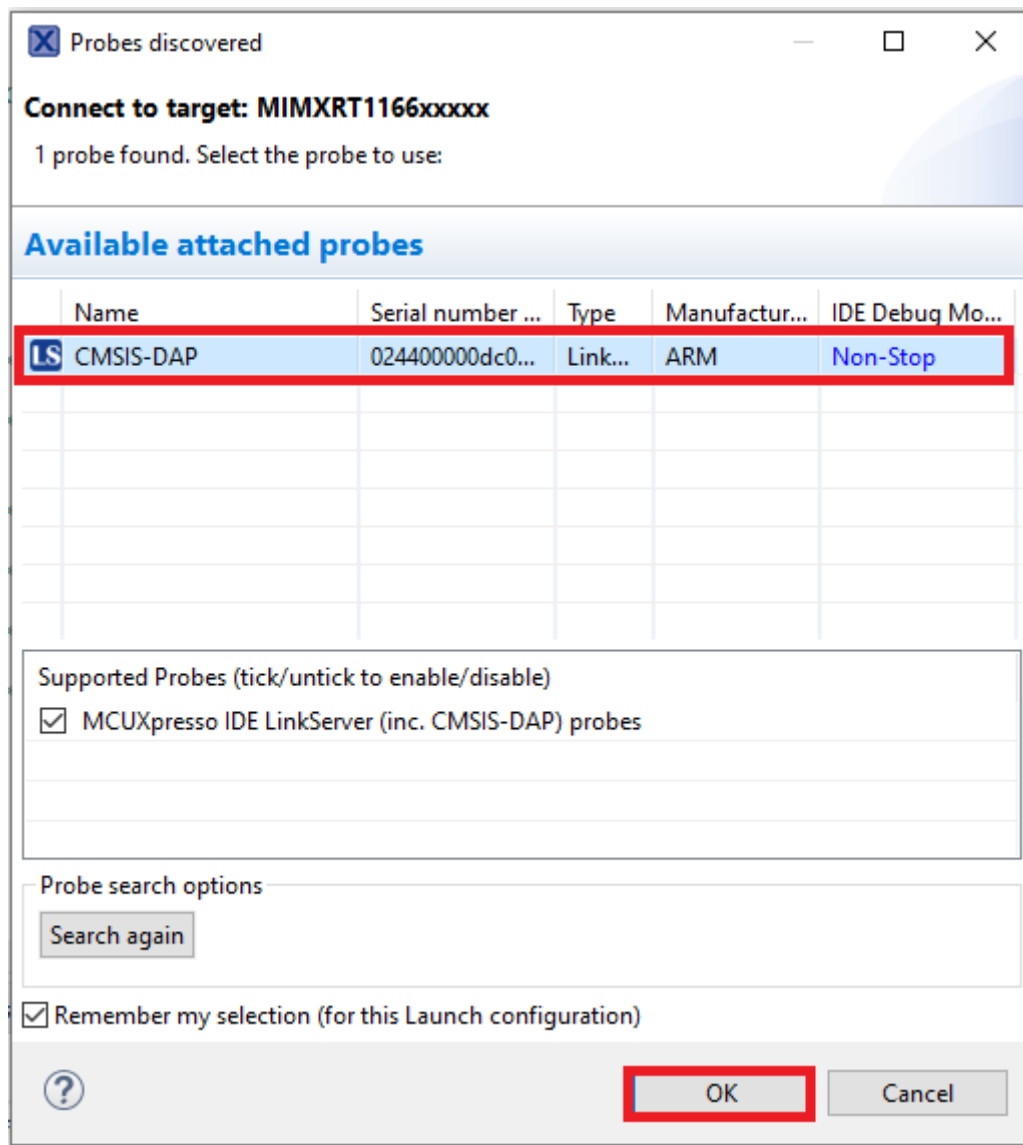


4. 1 stop bit

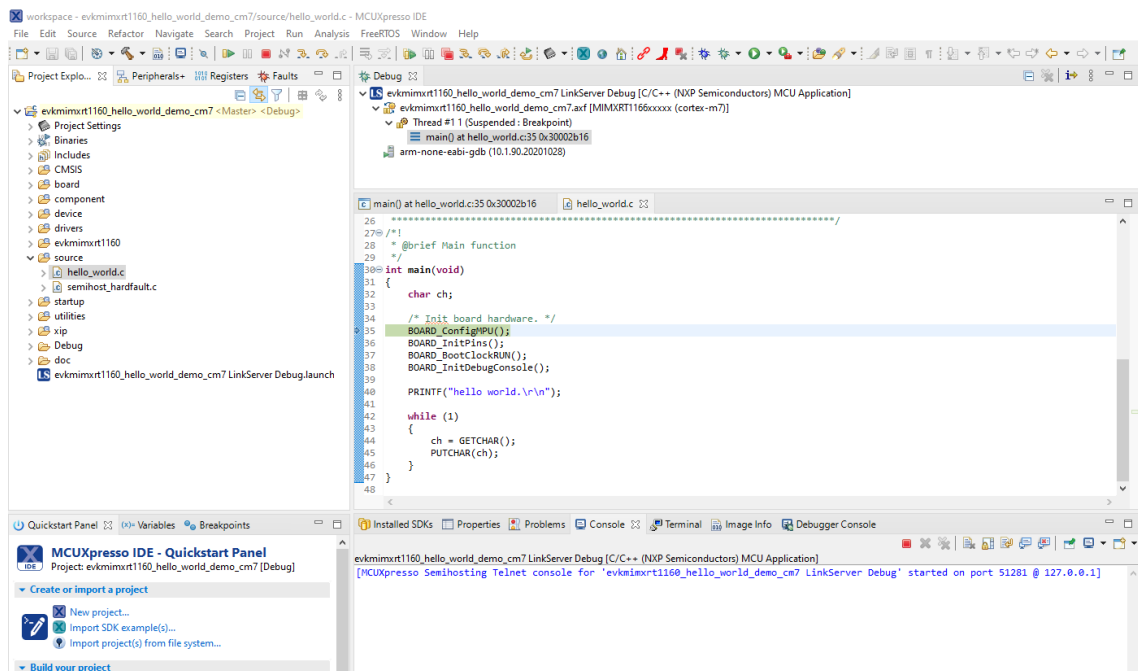
4. On the **Quickstart Panel**, click **Debug** 'evkmimxrt1160_demo_apps_hello_world' [Debug].



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



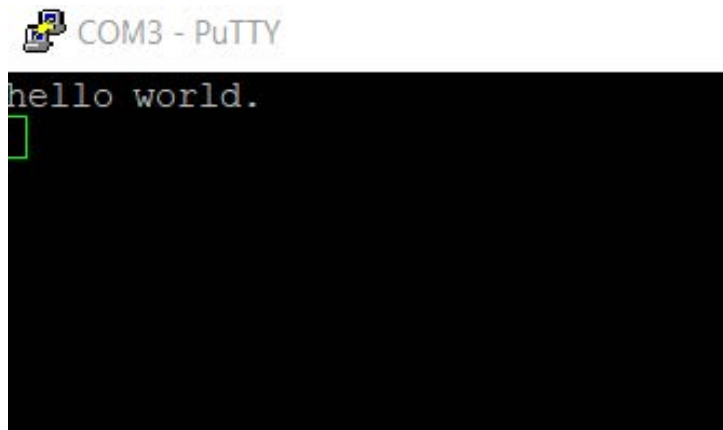
- The application is downloaded to the target and automatically runs to `main()`.



7. Start the application by clicking **Resume**.



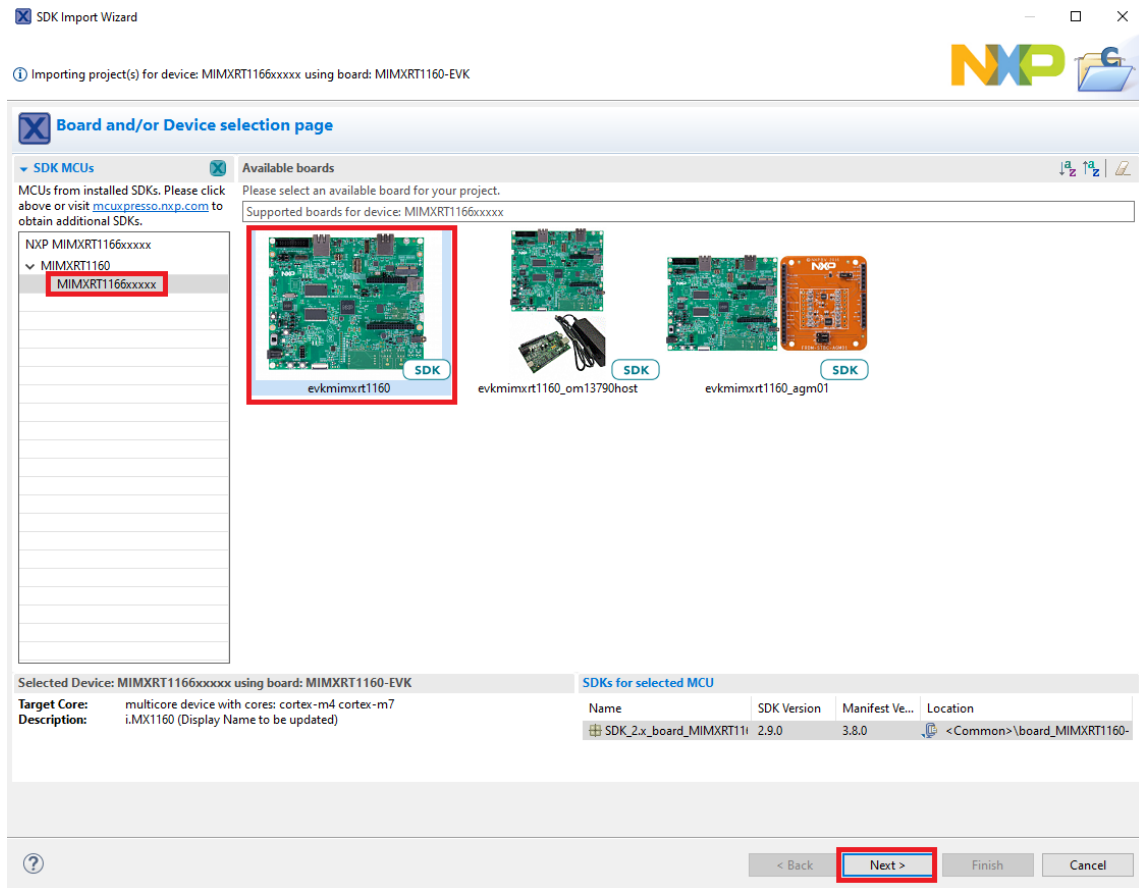
The hello_world application is now running and a banner is displayed on the terminal. If this is not the case, check your terminal settings and connections.



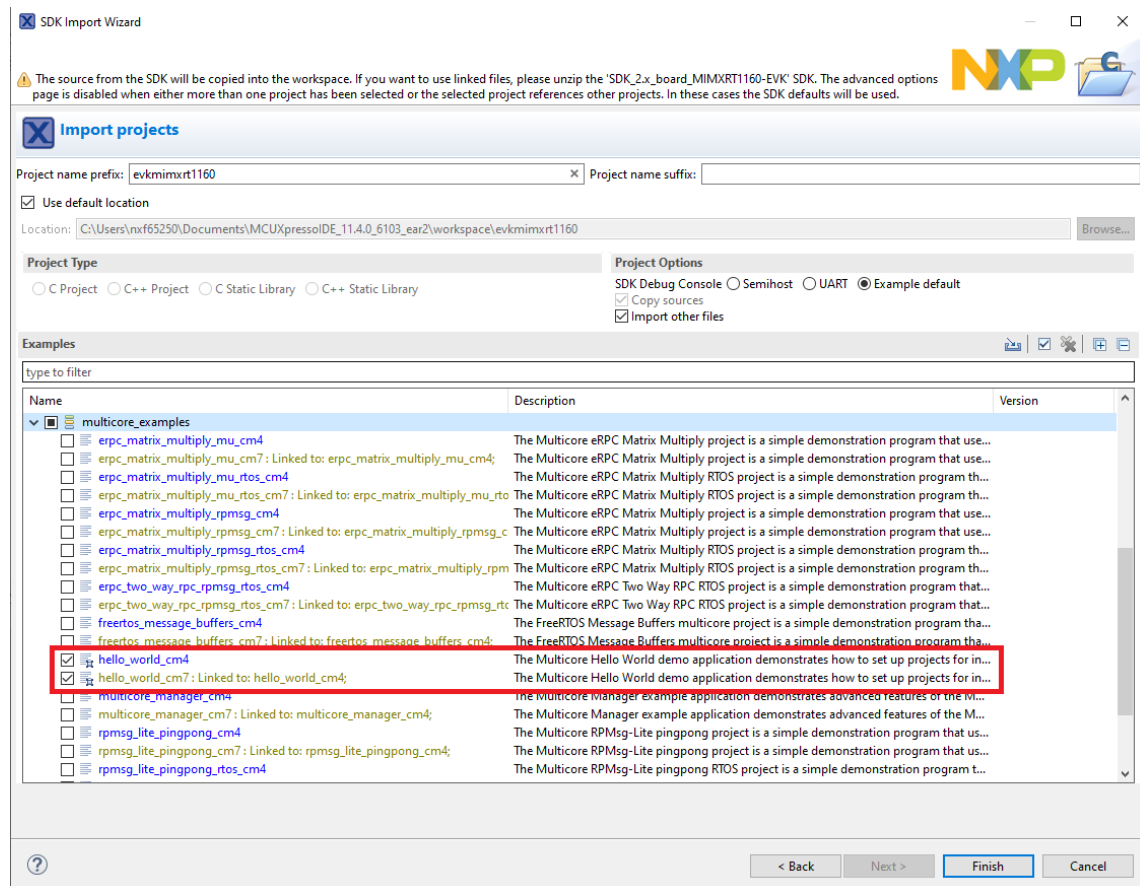
Parent topic: [Run a demo using MCUXpresso IDE](#)

Build a multicore example application This section describes the steps required to configure MCUXpresso IDE to build, run, and debug multicore example applications. The following steps can be applied to any multicore example application in the MCUXpresso SDK. Here, the dual-core version of hello_world example application targeted for the **evkmimxrt1160** hardware platform is used as an example.

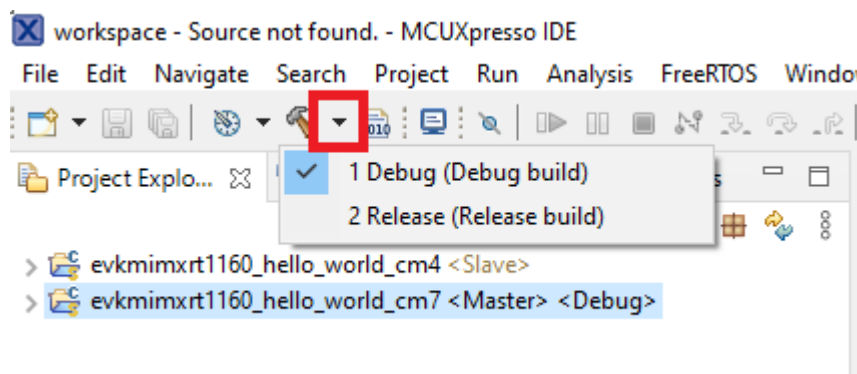
1. Multicore examples are imported into the workspace in a similar way as single core applications, explained in Build an example application. When the SDK zip package for **evkmimxrt1160** is installed and available in the **Installed SDKs** view, click **Import SDK example(s)...** on the Quickstart Panel. In the window that appears, select **MIMXRT1166xxxxxx**. Then, select **evkmimxrt1160** and click **Next**.



- Expand the `multicore_examples` folder and select **hello_world_cm7**. The `hello_world_cm4` counterpart project is automatically imported with the `cm7` project, because the multicore examples are linked together and there is no need to select it explicitly. Click **Finish**.



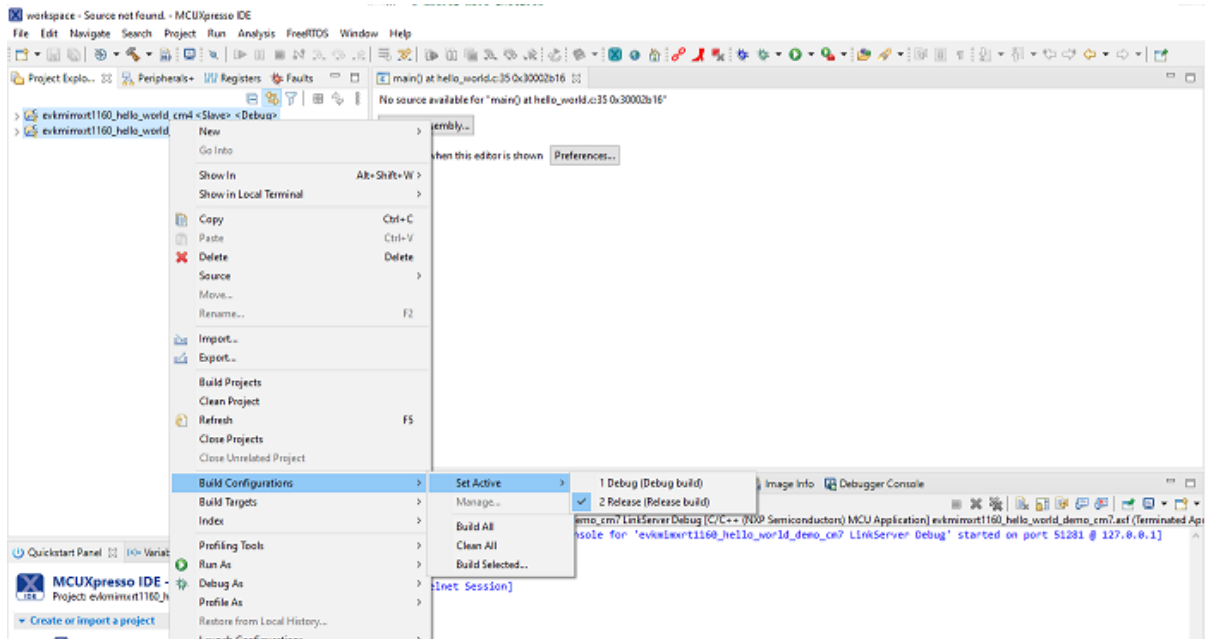
3. Now, two projects should be imported into the workspace. To start building the multi-core application, highlight the `hello_world_cm7` project (multicore master project) in the Project Explorer. Then choose the appropriate build target, **Debug** or **Release**, by clicking the downward facing arrow next to the hammer icon, as shown in Figure 3. For this example, select **Debug**.



Press the **Build** button to start the multi-core project build. Because of the project reference settings in multicore projects, triggering the build of the primary core application (cm7) also makes the referenced auxiliary core application (cm4) to build.

Note:

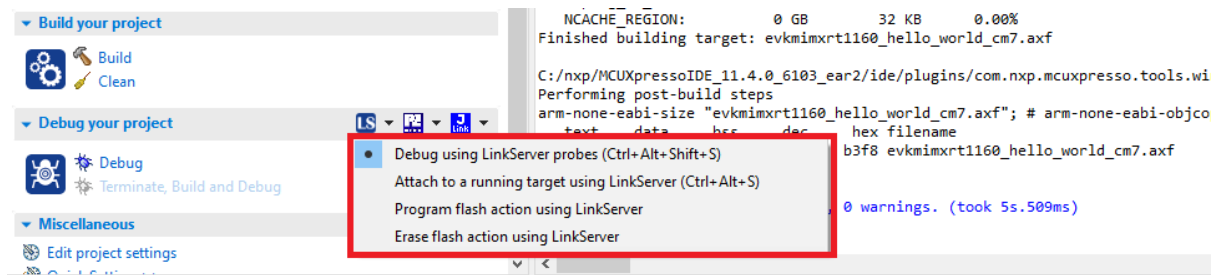
When the **Release** build is requested, it is necessary to change the build configuration of both the primary and auxiliary core application projects first. To do this, select both projects in the Project Explorer view and then right click which displays the context-sensitive menu. Select **Build Configurations** -> **Set Active** -> **Release**. This alternate navigation using the menu item is **Project** -> **Build Configuration** -> **Set Active** -> **Release**. After switching to the **Release** build configuration, the build of the multicore example can be started by triggering the primary core application (cm7) build.

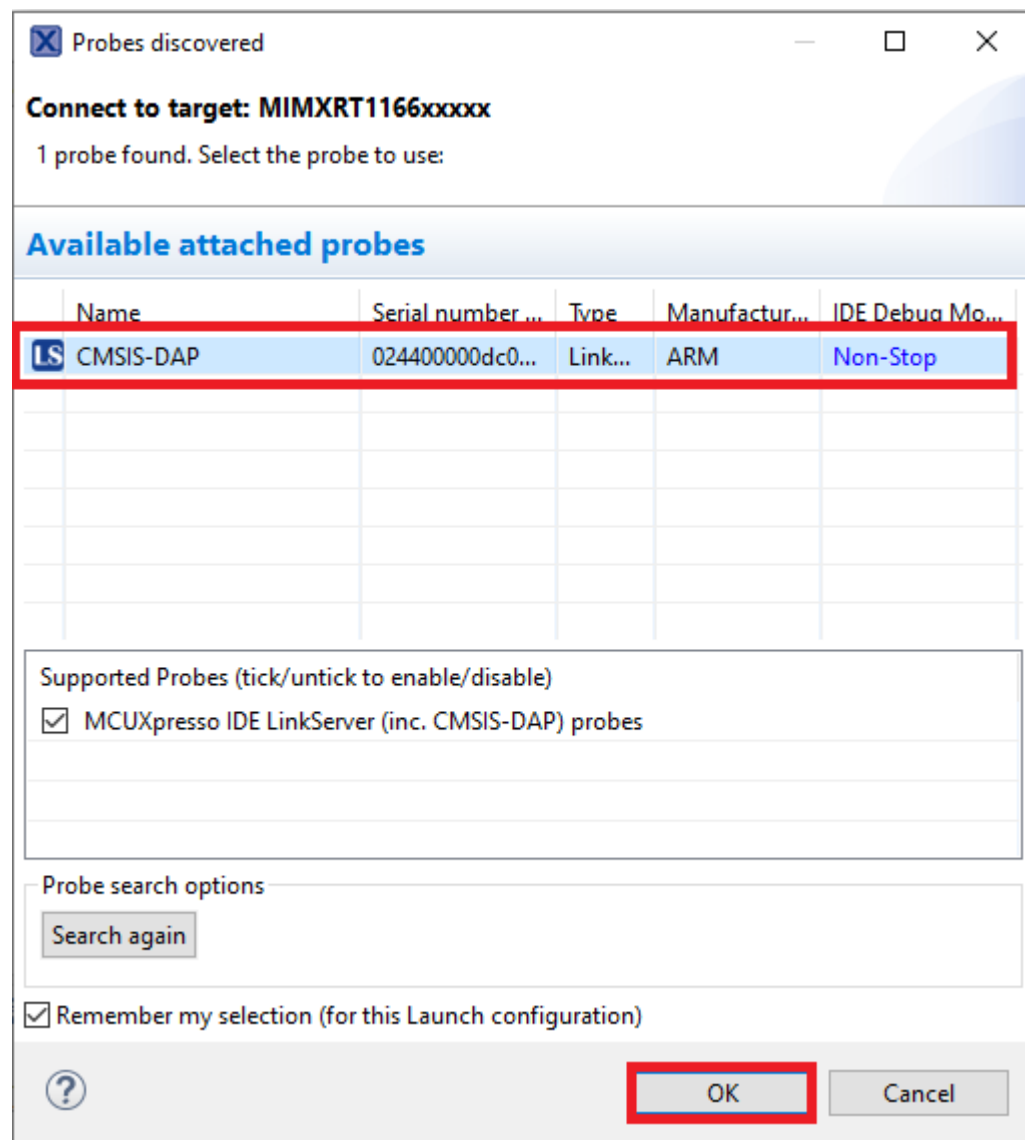


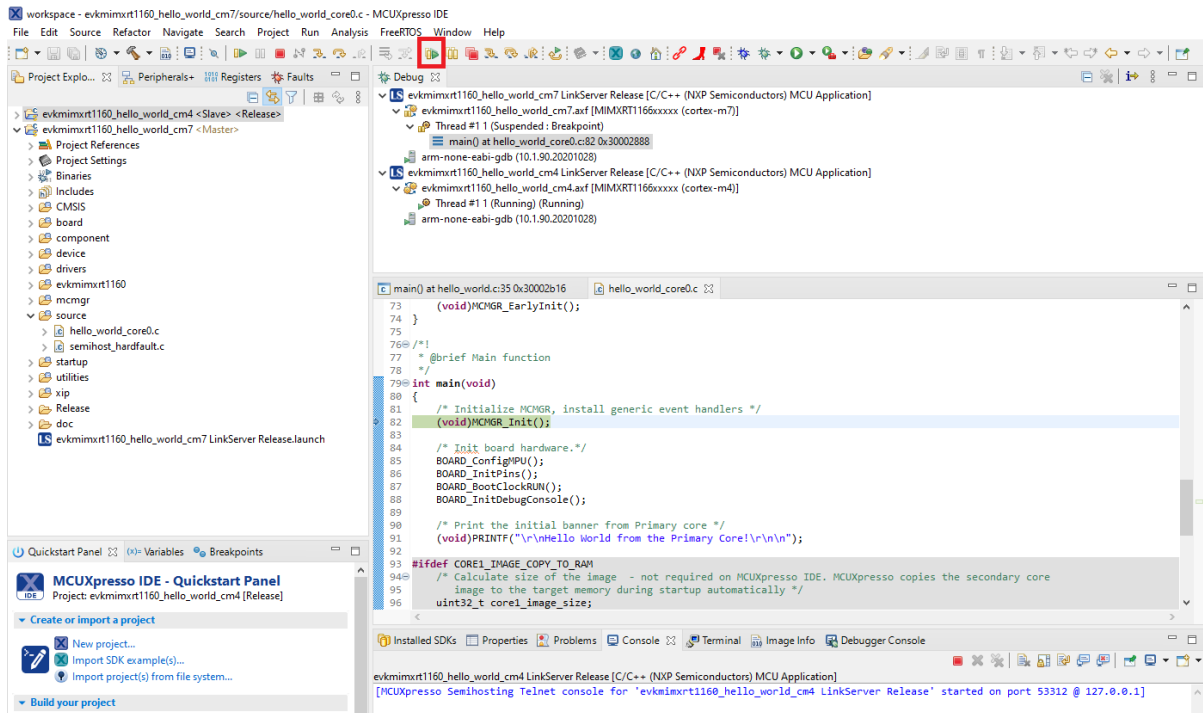
Parent topic: [Run a demo using MCUXpresso IDE](#)

Run a multicore example application The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform all steps as described in [Run an example application](#). These steps are common for both single-core applications and the primary side of dual-core applications, ensuring both sides of the multicore application are properly loaded and started. However, there is one additional dialogue that is specific to multicore examples which requires selecting the target core. See Figure 1 to Figure 4 as reference.

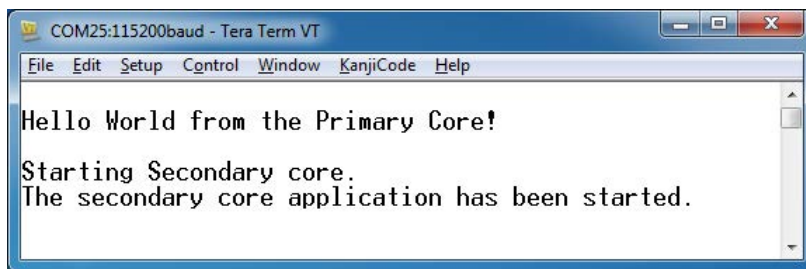
Note: On MCUXpresso IDE, the feature to simultaneously debug two cores is only supported by CMSIS-DAP debugger.







After clicking **Resume All Debug** sessions, the `hello_world` multicore application runs and a banner is displayed on the terminal. If this is not the case, check your terminal settings and connections



Note: There are some limitations on MCUXpresso IDE debugging. For details, see **Section 8.5 MCUXpresso IDE limitation** in *MCUXpresso SDK Release Notes for MIMXRT1160-EVK* (document MCUXSDKMIMXRT116XRN).

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

Run a demo application using IAR

Note:

When erasing flash on IAR, IAR will show all range that can connect to flash. Please only check address flash connect to practically. Take the evkmimxrt1160 for example:

- M7: 0x30000000-0x3fffffff
- M4: 0x80000000-0x17ffffff

When using IAR download/debug flexspi_nor related targets, make sure the boot switch is put to internal flash boot mode **SW1[1:4]:0010**.

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK. The `hello_world` demo application targeted for the MIMXRT1160-EVK hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

Build an example application Do the following steps to build the `hello_world` demo application.

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

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

Using the MIMXRT1160-EVK hardware platform as an example, the `hello_world` workspace is located in:

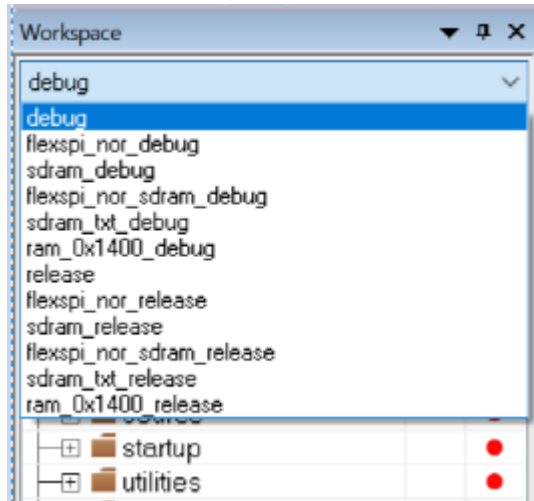
`<install_dir>/boards/evkmimxrt1160/demo_apps/hello_world/cm7/iar/hello_world_demo_cm7.eww`

Other example applications may have additional folders in their path.

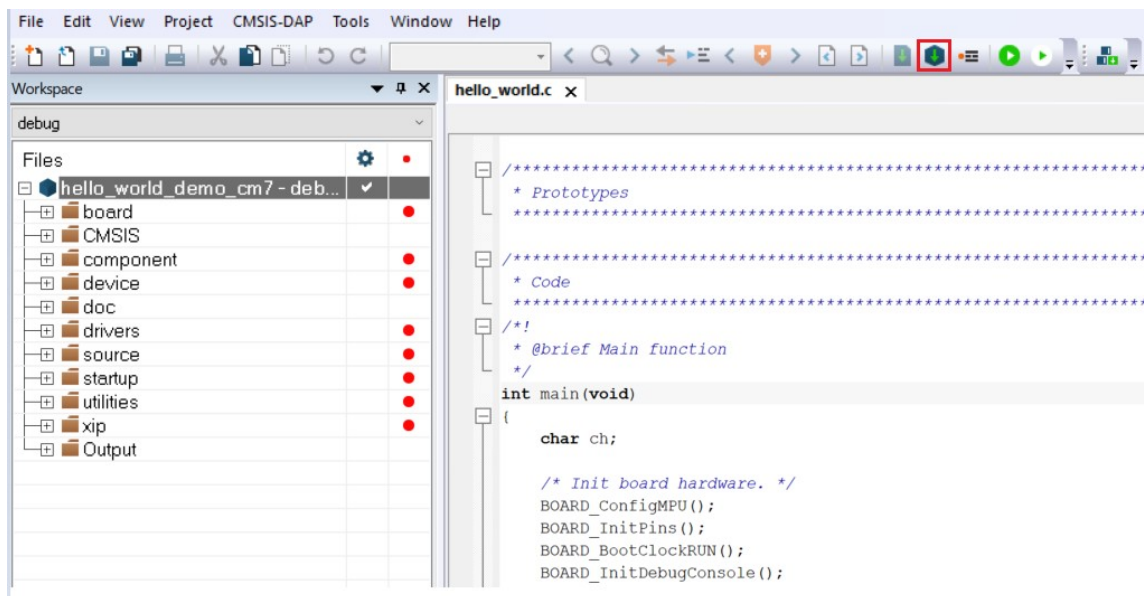
2. Select the desired build target from the drop-down menu.

There are twelve project configurations (build targets) supported across MCUXpresso SDK projects:

- **Debug**– Compiler optimization is set to low, and debug information is generated for the executable. The linker file is `RAMlinker`, where text and data section is put in internal TCM.
- **Release**– Compiler optimization is set to high, and debug information is not generated. The linker file is `RAM linker`, where text and data section is put in internal TCM.
- `ram_0x1400_debug`– Project configuration is same as the debug target. The linker file is `RAM_0x1400linker`, where text is put in ITCM with offset 0x1400 and data put in DTCM.
- `ram_0x1400_release`– Project configuration is same as the release target. The linker file is `RAM_0x1400 linker`, where text is put in ITCM with offset 0x1400 and data put in DTCM.
- `sdram_debug`– Project configuration is same as the debug target. The linker file is `SDRAMlinker`, where text is put in internal TCM and data put in SDRAM.
- `sdram_release`– Project configuration is same as the release target. The linker file is `SDRAMlinker`, where text is put in internal TCM and data put in SDRAM.
- `sdram_txt_debug`– Project configuration is same as the debug target. The linker file is `SDRAM_txtlinker`, where text is put in SDRAM and data put in OCRAM.
- `sdram_txt_release`– Project configuration is same as the release target. The linker file is `SDRAM_txtlinker`, where text is put in SDRAM and data put in OCRAM.
- `flexspi_nor_debug`– Project configuration is same as the debug target. The linker file is `flexspi_nor linker`, where text is put in flash and data put in TCM.
- `flexspi_nor_release`– Project configuration is same as the release target. The linker file is `flexspi_nor linker`, where text is put in flash and data put in TCM.
- `flexspi_nor_sdram_release`– Project configuration is same as the release target. The linker file is `flexspi_nor_sdramlinker`, where text is put in flash and data put in SDRAM.
- `flexspi_nor_sdram_debug`– Project configuration is same as the debug target. The linker file is `flexspi_nor_sdramlinker`, where text is put in flash and data put in SDRAM. For some examples need large data memory, only `sdram_debug` and `sdram_release` targets are supported. For this example, select **hello_world– debug**.



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

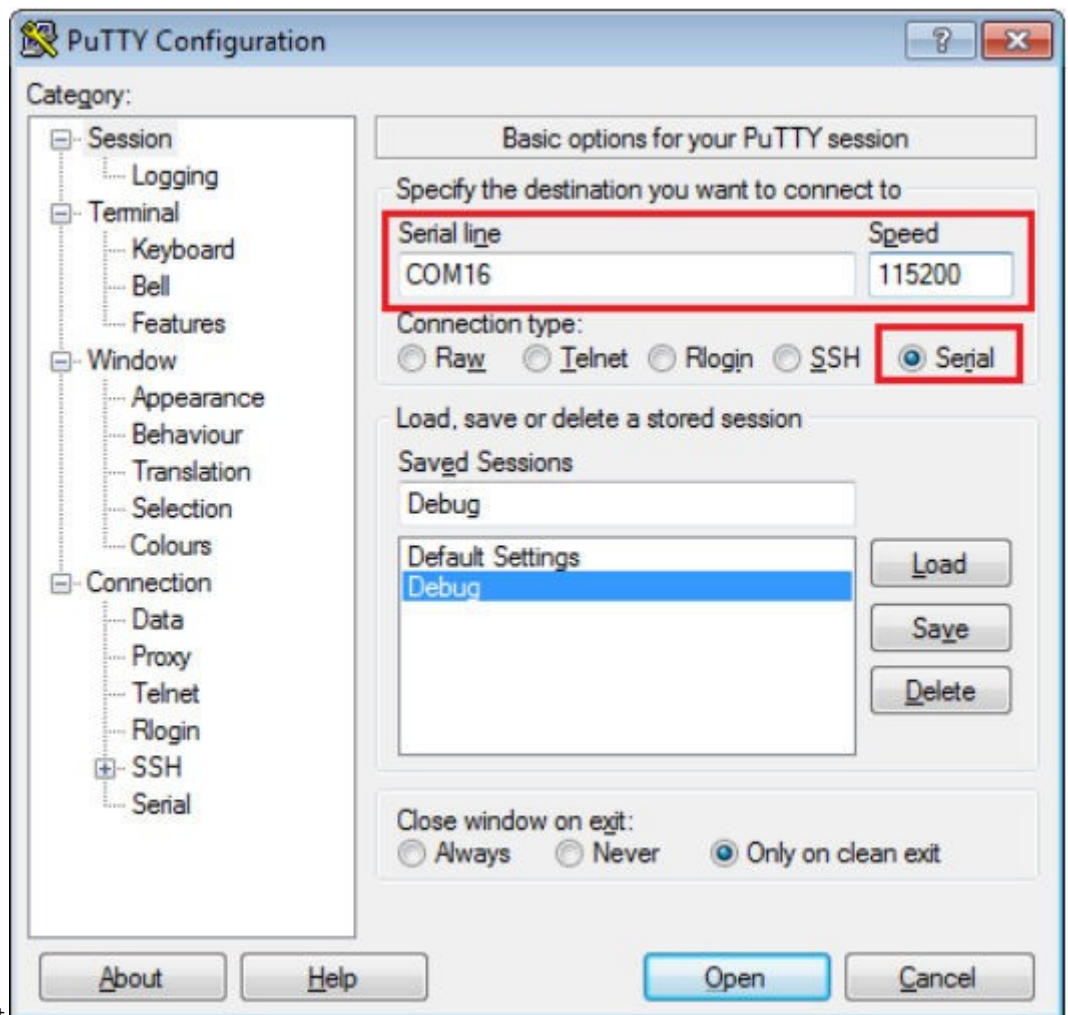


4. The build completes without errors.

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

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

1. This board supports the CMSIS-DAP/mbd/DAPLink debug probe by default. Visit [MBED](#) and follow the instructions to install the Windows® operating system serial driver. If running on Linux OS, this step is not required.
2. Connect the development platform to your PC via USB cable. Connect the USB cable to J11 and make sure SW1[1:4] is **0010b**.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (to determine the COM port number, see [How to determine COM port](#)). Configure the terminal with these settings:
 1. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the *board.h* file)
 2. No parity
 3. 8 data bits

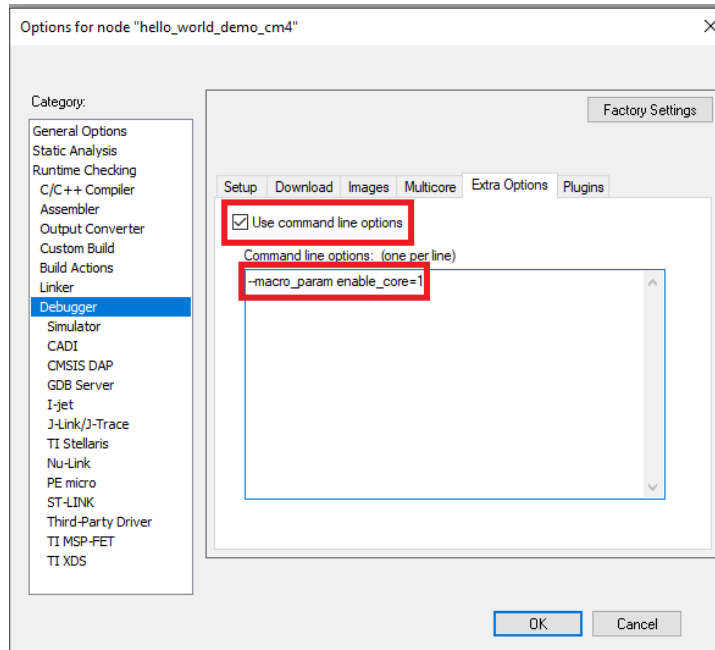


4. 1 stop bit

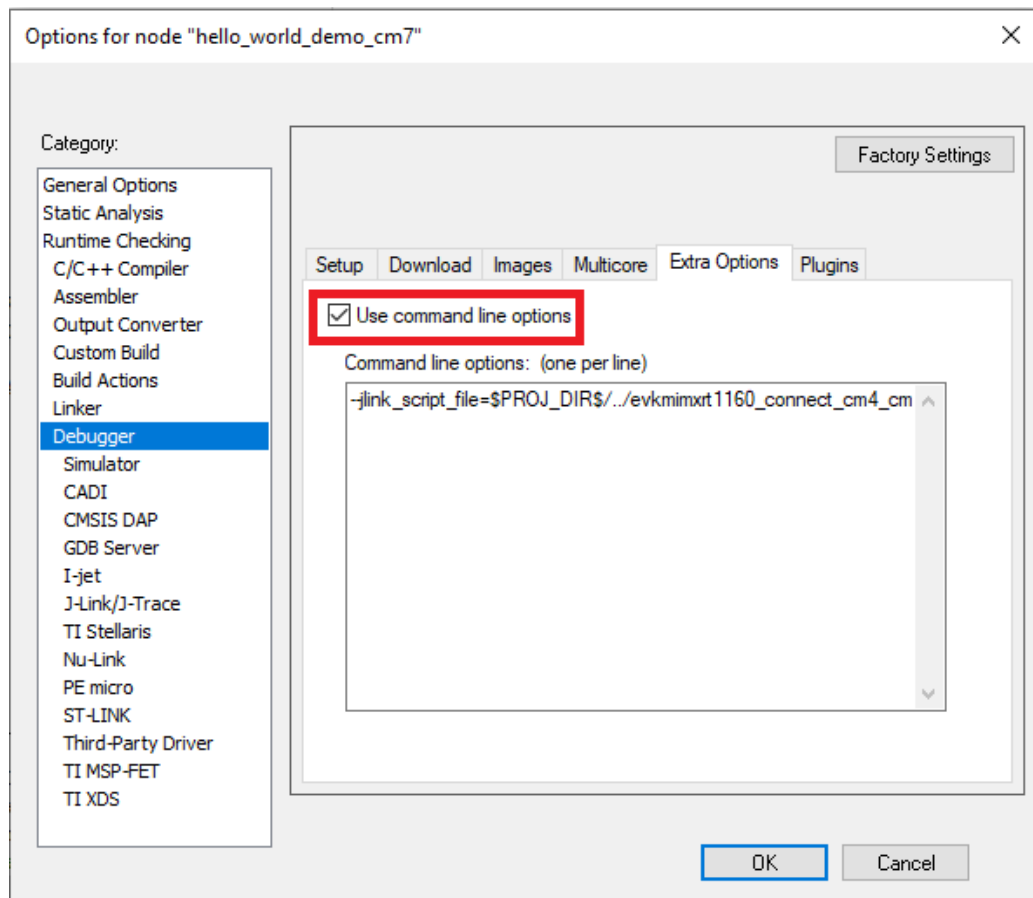
4. In IAR, click the **Download and Debug** button to download the application to the target.



- When using CMSIS-DAP to debug cm4 project on IAR, an extra option needs to be specified in debugger settings. Check and fill in `-macro_param enable_core=1` in **Debugger -> Extra Options -> Command line options**, as shown in Figure 3.



- If debugging with JLINK as probe, jlinkscript file is needed.
 - When downloading the cm7 project, check **Use command line options**, as shown in Figure 4.



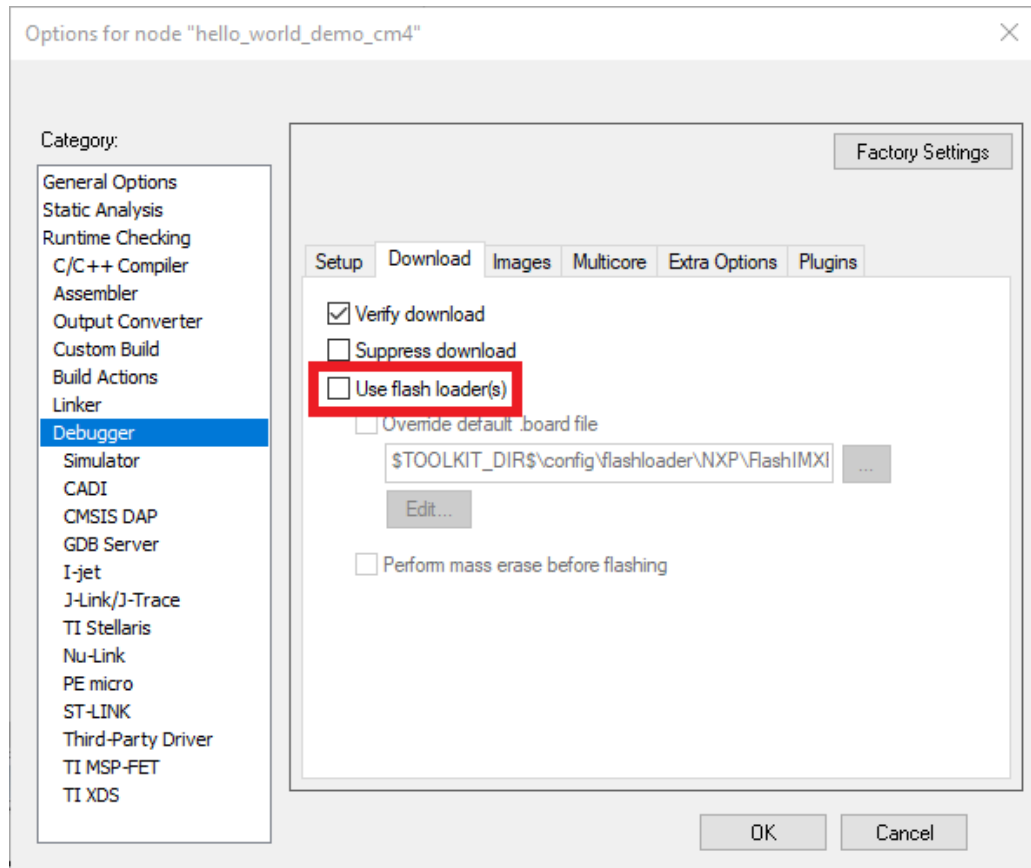
- When downloading the cm4 project, uncheck **Use flash loader(s)**, as shown in Figure 5, and change the contents of command line options as below:

* Target with SDRAM

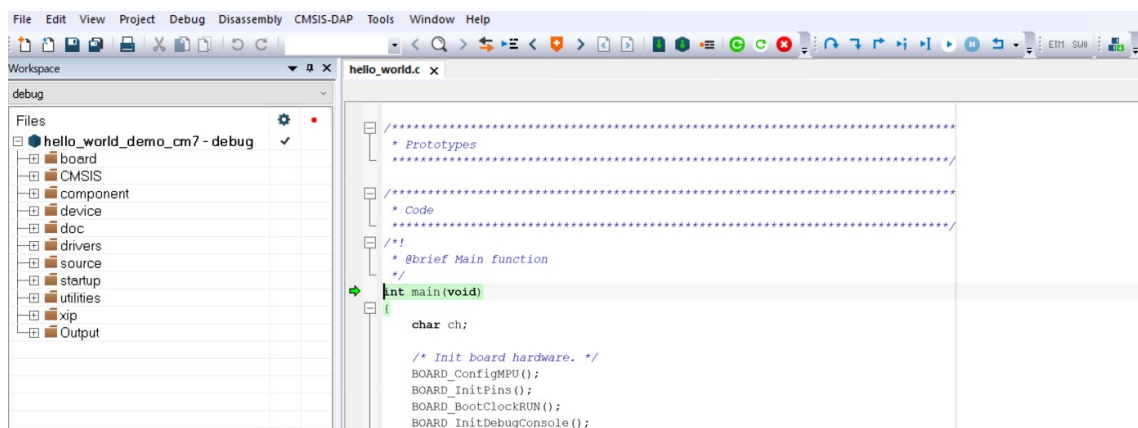
```
--jlink_script_file=$PROJ_DIR$/../evkmimxrt1160_connect_cm4_cm4side_sdram.  
↩ jlinkscript
```

* Other target

```
--jlink_script_file=$PROJ_DIR$/../evkmimxrt1160_connect_cm4_cm4side.jlinkscript
```



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

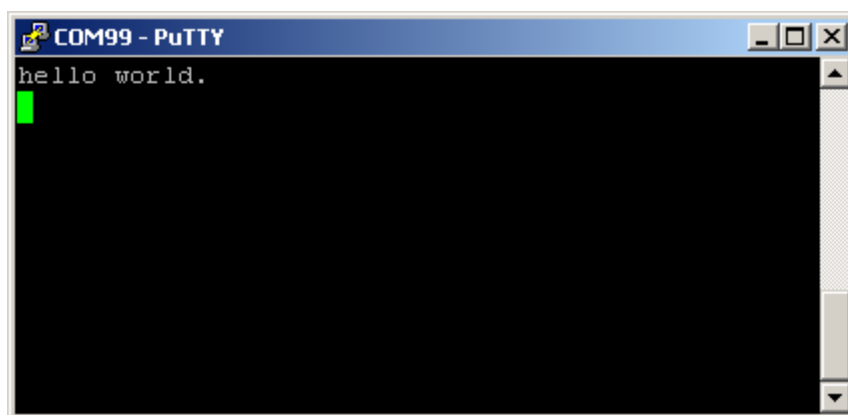


6. Run the code by clicking the **Go** button to start the application.



7. The `hello_world` application is now running and a banner is displayed on the terminal. If

this is not true, check your terminal settings and connections.



Note: There are some limitations on MCUXpresso IDE debugging. For details, see **Section 8.6 IAR debug limitation** in *MCUXpresso SDK Release Notes for MIMXRT1160-EVK* (document MCUXSD-KMIMXRT116XRN).

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

Build a multicore example application This section describes the steps to build and run a dual-core application. The demo applications workspace files are located in this folder:

```
<install_dir>/boards/<board_name>/multicore_examples/<application_name>/<core_type>/iar
```

Begin with a simple dual-core version of the Hello World application. The multicore Hello World IAR workspaces are located in this folder:

```
<install_dir>/boards/evkmimxrt1160/multicore_examples/hello_world/cm4/iar/hello_world_cm4.eww
<install_dir>/boards/evkmimxrt1160/multicore_examples/hello_world/cm7/iar/hello_world_cm7.eww
```

Build both applications separately by clicking the **Make** button. Build the application for the auxiliary core (cm4) first, because the primary core application project (cm7) needs to know the auxiliary core application binary when running the linker. It is not possible to finish the primary core linker when the auxiliary core application binary is not ready.

Because the auxiliary core runs always from RAM, only debug and release RAM targets are present in the project. When building the primary core project, it is possible to select either *debug/release* RAM targets or *flexspi_nor_debug/flexspi_nor_release* Flash targets. When choosing Flash targets (preferred) the auxiliary core binary is linked with the primary core image and stored in the external SPI Flash memory. During the primary core execution the auxiliary core image is copied from flash into the CM4 RAM and executed.

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

Run a multicore example application The primary core debugger handles flashing both primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 4 as described in *Run an example application*. These steps are common for both single core and dual-core applications in IAR.

After clicking the **Download and Debug** button, the auxiliary core project is opened in the separate EWARM instance. Both the primary and auxiliary image are loaded into the device flash memory and the primary core application is executed. It stops at the default C language entry point in the `main()` function.

Run both cores by clicking the **Start all cores** button to start the multicore application.



During the primary core code execution, the auxiliary core is released from the reset. The `hello_world` multicore application is now running and a banner is displayed on the terminal. If this does not appear, check the terminal settings and connections.

```
COM21:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help

Hello World from the Primary Core!
Copy Secondary core image to address: 0x20200000, size: 17124
Starting Secondary core.
The secondary core application has been started.
```

An LED controlled by the auxiliary core starts flashing, indicating that the auxiliary core has been released from the reset and is running correctly. When both cores are running, use the **Stop all cores** and **Start all cores** buttons to stop or run both cores simultaneously.



Note: On IAR, the feature to simultaneously debug two cores is only supported by CMSIS-DAP debugger.

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

Run a demo using Keil® MDK/μVision

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK.

Install CMSIS device pack After the MDK tools are installed, Cortex® Microcontroller Software Interface Standard (CMSIS) device packs must be installed to fully support the device from a debug perspective. These packs include things such as memory map information, register definitions and flash programming algorithms. Follow these steps to install the MIMXRT116x CMSIS pack.

1. Download the MIMXRT1165 and MIMXRT1166 packs.
2. After downloading the DFP, double click to install it. Be patient when the DFP is installed. It will take approximate 15 minutes for the installation to complete.

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

Build an example application

1. Open the desired example application workspace in:

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

The workspace file is named as `<demo_name>.uvmpw`. For this specific example, the actual path is:

`<install_dir>/boards/evkmimxrt1160/demo_apps/hello_world/cm7/mdk/hello_world_demo_cm7.uvmpw`

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

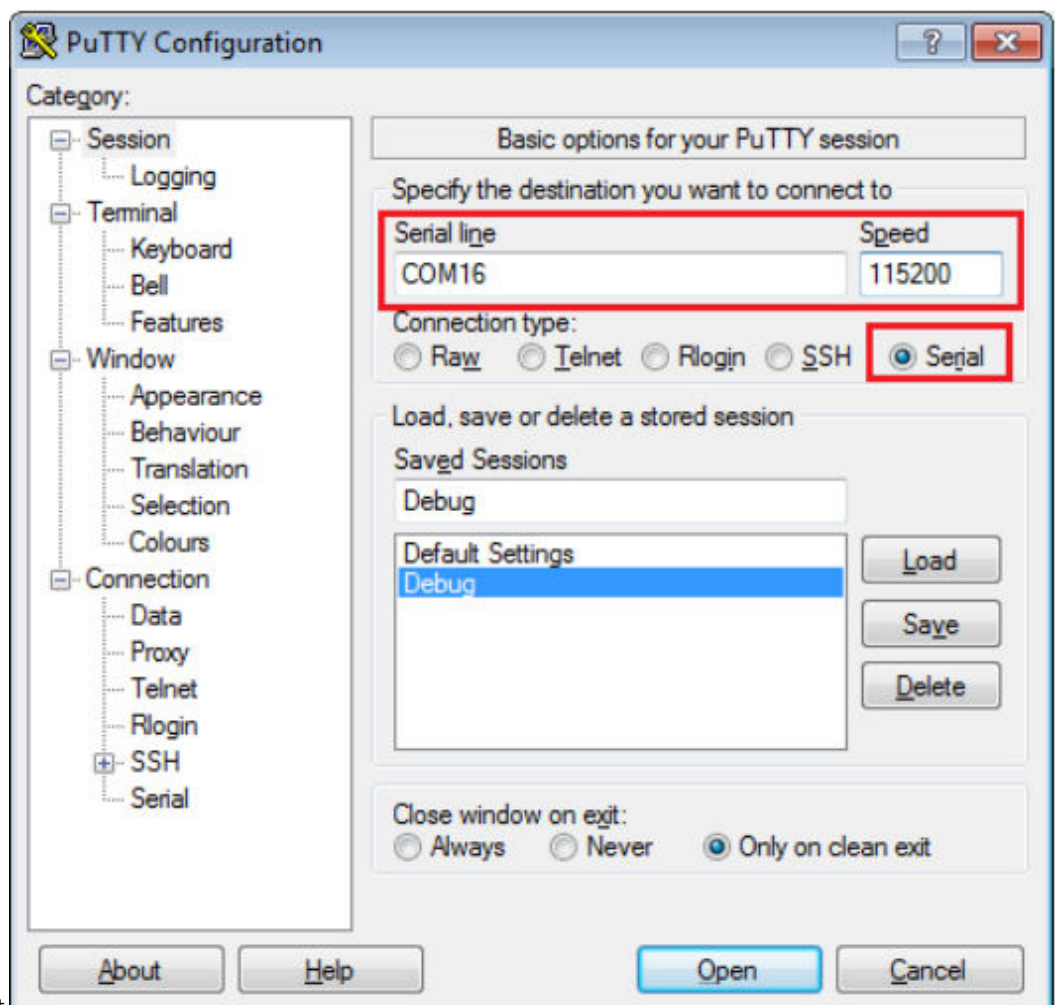


3. The build completes without errors.

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

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

1. This board supports the CMSIS-DAP/mbed/DAPLink debug probe by default. Visit [MBED](#) serial-configuration and follow the instructions to install the Windows® operating system serial driver. If running on Linux OS, this step is not required.
2. Connect the development platform to your PC via USB cable.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number. To determine the COM port number, see [How to determine COM port](#). Configure the terminal with these settings:
 1. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the *board.h* file)
 2. No parity
 3. 8 data bits



4. 1 stop bit
4. To debug the application, click **load** or press the **F8** key for flexspi target which need to download the program to flash memory. Then, click the **Start/Stop Debug Ses-**

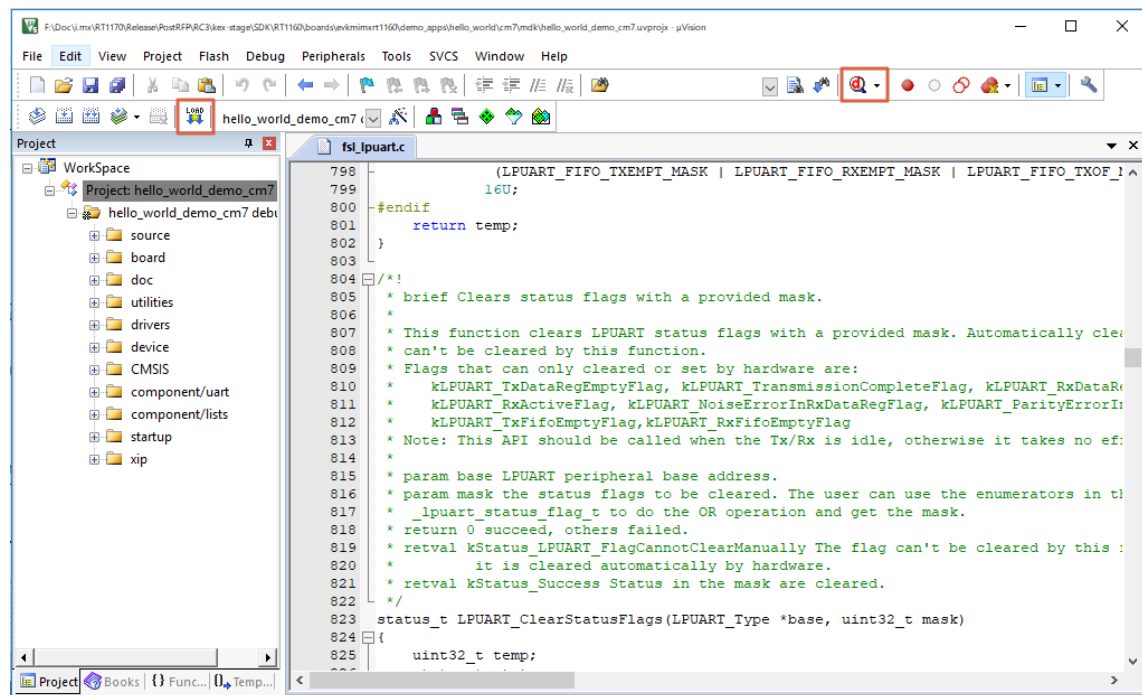
sionbutton, highlighted in red in Figure 2. If using J-Link as the debugger, click **Project option>Debug>Settings>Debug >Port**, and select SW.

Note: If debugging with JLINK, device selection window will be popped when you click the **Settings** button under the **Debug** tab. Users need to choose **MIMXRT1165/MIMXRT1166** device manually.

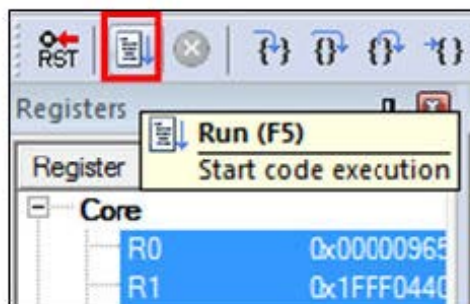
Note:

When using jlink in MDK for cm4 projects, it expects one jlinkscript file named JLinkSettings.JLinkScript in the folder where the uVision project files are located. Please refer to [Segger Wiki](#) for more information.

For the contents in this JlinkSettings.JLinkScript, use contents in evkmimxrt1160_connect_cm4_cm4side.jlinkscript (non-sdram targets) and evkmimxrt1160_connect_cm4_cm4side_sdram.jlinkscript (sdram targets).



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



The hello_world application is now running and a banner is displayed on the terminal, as shown in Figure 4. If this is not true, check your terminal settings and connections.



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

Build a multicore example application This section describes the particular steps that need to be done in order to build and run a dual-core application. The demo applications workspace files are located in this folder:

`<install_dir>/boards/evkmimxrt1160/multicore_examples/<application_name>/<core_type>/mdk`

Begin with a simple dual-core version of the Hello World application. The multicore Hello World MDK workspaces are located in this folder:

`<install_dir>/boards/evkmimxrt1160/multicore_examples/hello_world/cm4/mdk/hello_world_cm4.uvmpw`

`<install_dir>/boards/evkmimxrt1160/multicore_examples/hello_world/cm7/mdk/hello_world_cm7.uvmpw`

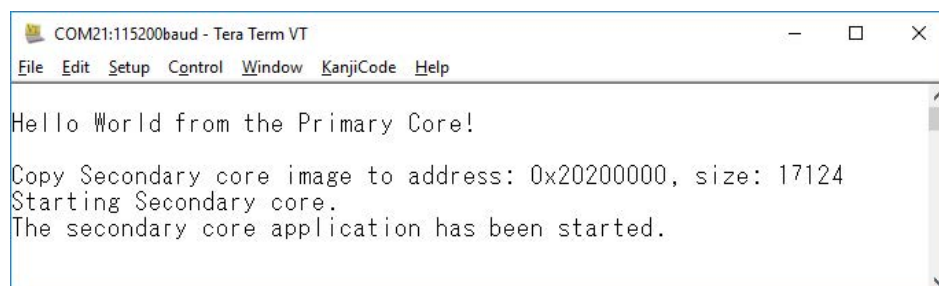
Build both applications separately by clicking the **Rebuild** button. Build the application for the auxiliary core (cm4) first, because the primary core application project (cm7) needs to know the auxiliary core application binary when running the linker. It is not possible to finish the primary core linker when the auxiliary core application binary is not ready.

Because the auxiliary core runs always from RAM, debug and release RAM targets are present in the project only. When building the primary core project, it is possible to select `flexspi_nor_debug/flexspi_nor_release` Flash targets. When choosing Flash targets the auxiliary core binary is linked with the primary core image and stored in the external SPI Flash memory. During the primary core execution the auxiliary core image is copied from flash into the CM4 RAM and executed.

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

Run a multicore example application The primary core debugger flashes both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 5 as described in [Run an example application](#). These steps are common for both single-core and dual-core applications in μVision.

Both the primary and the auxiliary image is loaded into the flash memory. After clicking **Run**, the primary core application is executed. During the primary core code execution, the auxiliary core code is re-allocated from the SPI flash memory to the RAM, and the auxiliary core is released from the reset. The `hello_world` multicore application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.



```

COM21:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help

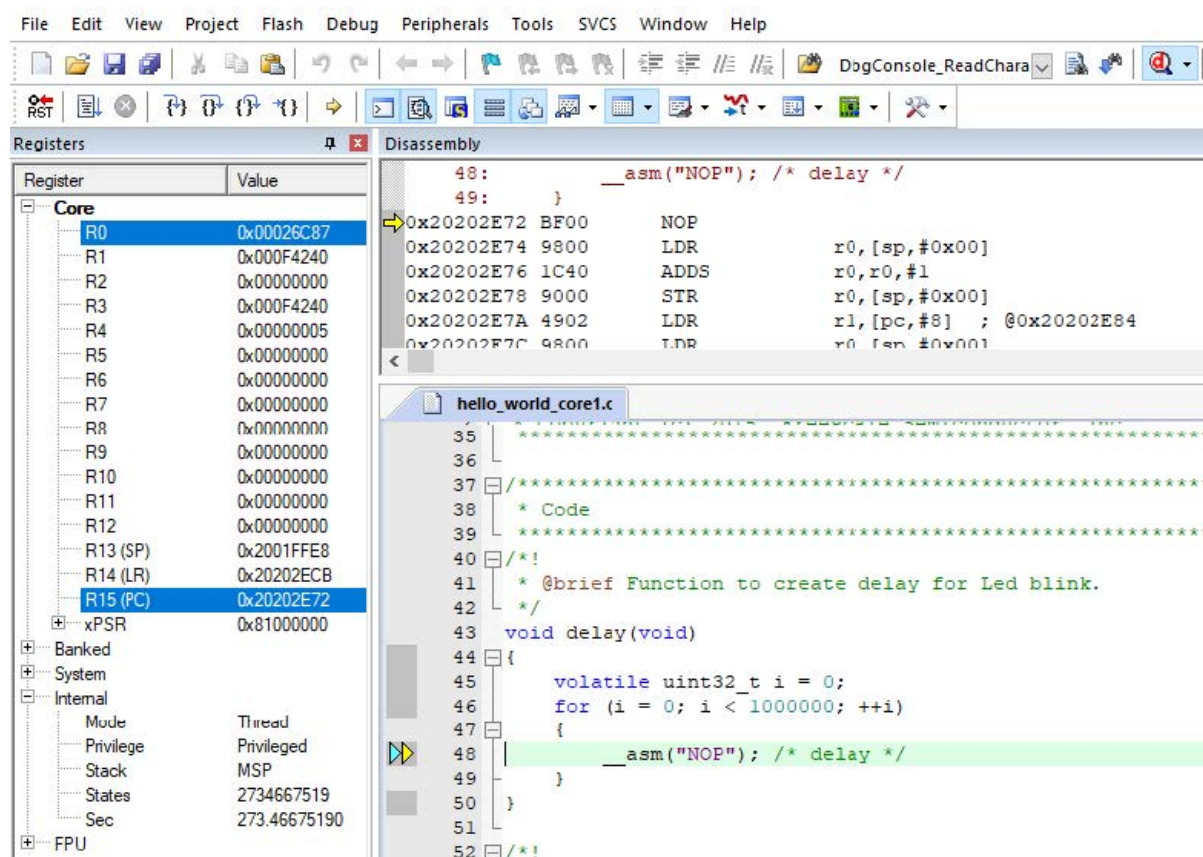
Hello World from the Primary Core!

Copy Secondary core image to address: 0x20200000, size: 17124
Starting Secondary core.
The secondary core application has been started.

```

An LED controlled by the auxiliary core starts flashing indicating that the auxiliary core has been released from the reset and is running correctly.

Attach the running application of the auxiliary core by opening the auxiliary core project in the second μ Vision instance and clicking the **Start/Stop Debug Session** button. After this, the second debug session is opened and the auxiliary core application can be debugged.



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

Run a demo using Arm® GCC

This section describes the steps to configure the command line Arm® GCC tools to build, run, and debug demo applications and necessary driver libraries provided in the MCUXpresso SDK. The `hello_worlddemo` application is targeted which is used as an example.

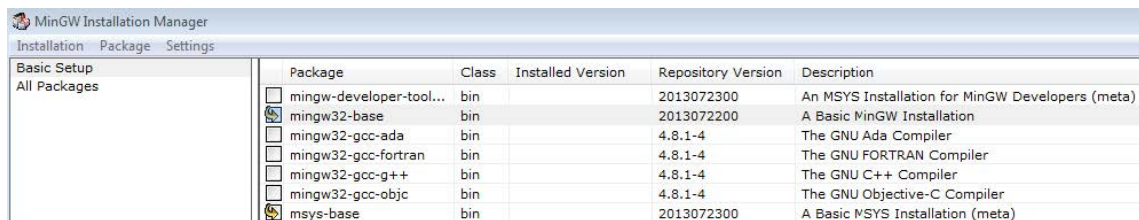
Set up toolchain This section contains the steps to install the necessary components required to build and run an MCUXpresso SDK demo application with the Arm GCC toolchain, as supported by the MCUXpresso SDK. There are many ways to use Arm GCC tools, but this example focuses on a Windows operating system environment.

Install GCC Arm Embedded tool chain Download and run the installer from [GNU Arm Embedded Toolchain](#). This is the actual toolset (in other words, compiler, linker, etc.). The GCC toolchain should correspond to the latest supported version, as described in *MCUXpresso SDK Release Notes for MIMXRT1160-EVK* (document MCUXSDKMIMXRT116XRN).

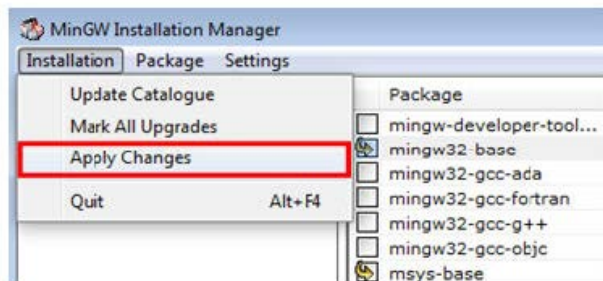
Parent topic:Set up toolchain

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

1. Download the latest MinGW mingw-get-setup installer from [SOURCEFORGE](#).
 2. Run the installer. The recommended installation path is *C:|MinGW*. However, you may install to any location.
- Note:** The installation path cannot contain any spaces.
3. Ensure that the **mingw32-base** and **msys-base** are selected under **Basic Setup**.



4. In the **Installation** menu, click **Apply Changes** and follow the remaining instructions to complete the installation.

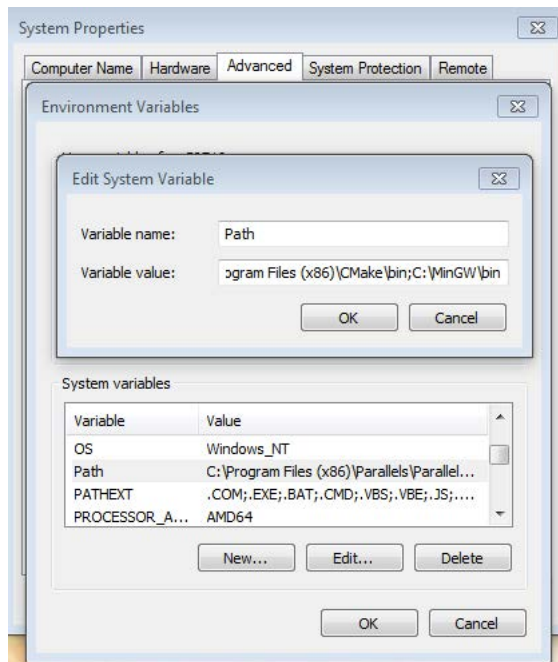


5. Add the appropriate item to the Windows operating system path environment variable. It can be found under **Control Panel->System and Security->System->Advanced System Settings** in the ****Environment Variables...**b> section. The path is:**

`<mingw_install_dir>|bin`

Assuming the default installation path is *C:|MinGW*, an example is as shown in Figure 3. If the path is not set correctly, the toolchain will not work.

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



Parent topic:Set up toolchain

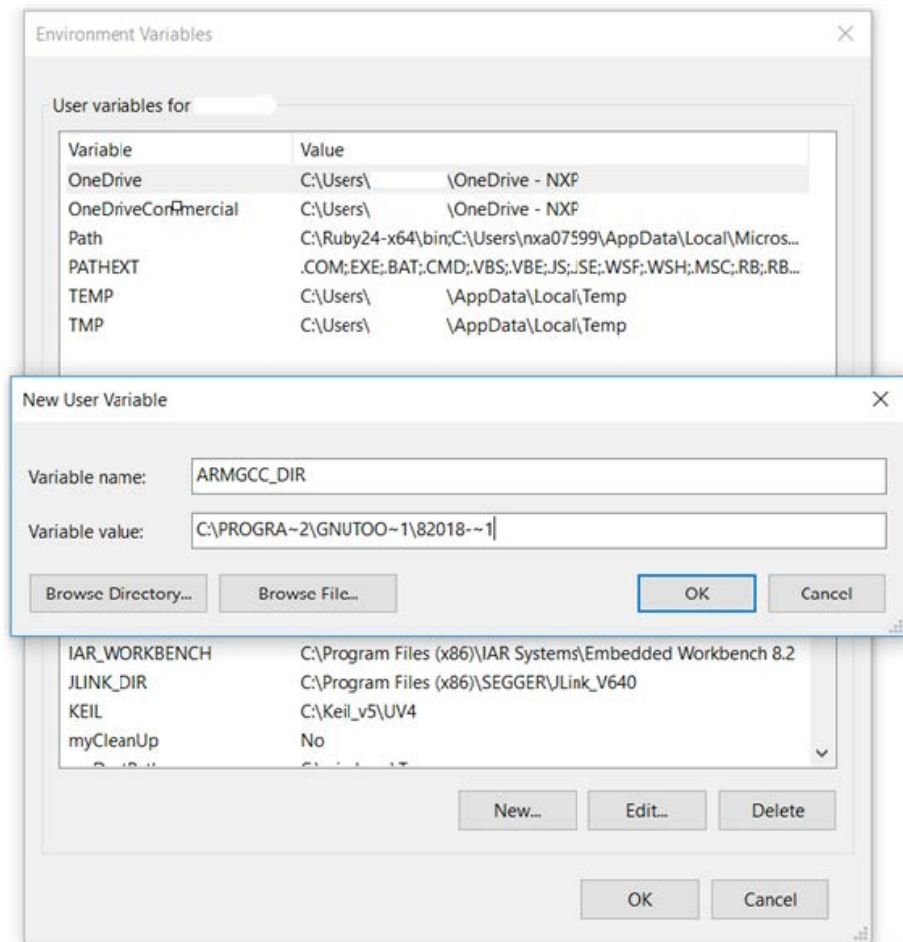
Add a new system environment variable for ARMGCC_DIR Create a new system environment variable and name it as ARMGCC_DIR. The value of this variable should point to the Arm GCC Embedded tool chain installation path. For this example, the path is:

C:\Program Files (x86)\GNU Tools ARM Embedded\8 2018-q4-major

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

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

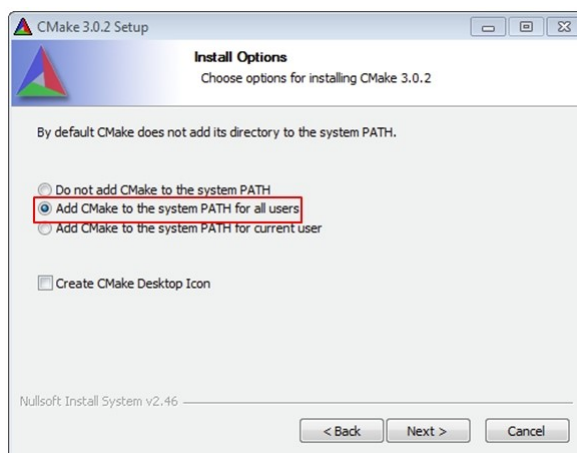
```
C:\Program Files (x86)\GNU Tools ARM Embedded\8 2018-q4-major>for %I in (.) do echo %~sI
C:\Program Files (x86)\GNU Tools ARM Embedded\8 2018-q4-major>echo C:\PROGRA~2\GNUTOO~1\82018~1
C:\PROGRA~2\GNUTOO~1\82018~1
```

Parent topic: Set up toolchain

Install CMake

1. Download CMake 3.0.x from [CMAKE](#).
2. Install CMake, ensuring that the option **Add CMake to system PATH** is selected when installing. The user chooses to select whether it is installed into the PATH for all users or just the current user. In this example, it is installed for all users.



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

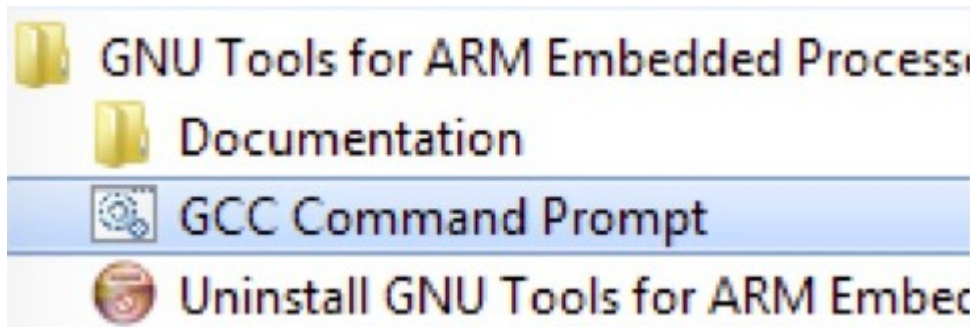
5. Make sure sh.exe is not in the Environment Variable PATH. This is a limitation of mingw32-make.

Parent topic: Set up toolchain

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

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

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



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

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

For this example, the exact path is:

```
<install_dir>/examples/evkmimxrt1160/demo_apps/hello_world/cm7/armgcc
```

Note: To change directories, use the `cd` command.

3. Type **build_debug.bat** on the command line or double click on **build_debug.bat** file in Windows Explorer to build it. The output is as shown in Figure 2.

```
[ 86%] Building C object CMakeFiles/hello_world_demo_cm7.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1160/drivers/cm7/fsl_cache.c.obj
Building C object CMakeFiles/hello_world_demo_cm7.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1160/drivers/fsl_pmu.c.obj [ 91%] Building C object CMakeFiles/hello_world_demo_cm7.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1160/drivers/fsl_dcdc.c.obj
Building C object CMakeFiles/hello_world_demo_cm7.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1160/xip/fsl_flexspi_nor_boot.c.obj
[ 95%] Building C object CMakeFiles/hello_world_demo_cm7.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1160/utilities/fsl_sbrk.c.obj
[100%] Linking C executable debug\hello_world_demo_cm7.elf
[100%] Built target hello_world_demo_cm7.elf
```

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

Run an example application This section describes steps to run a demo application using J-Link GDB Server application. To perform this exercise, make sure that either:

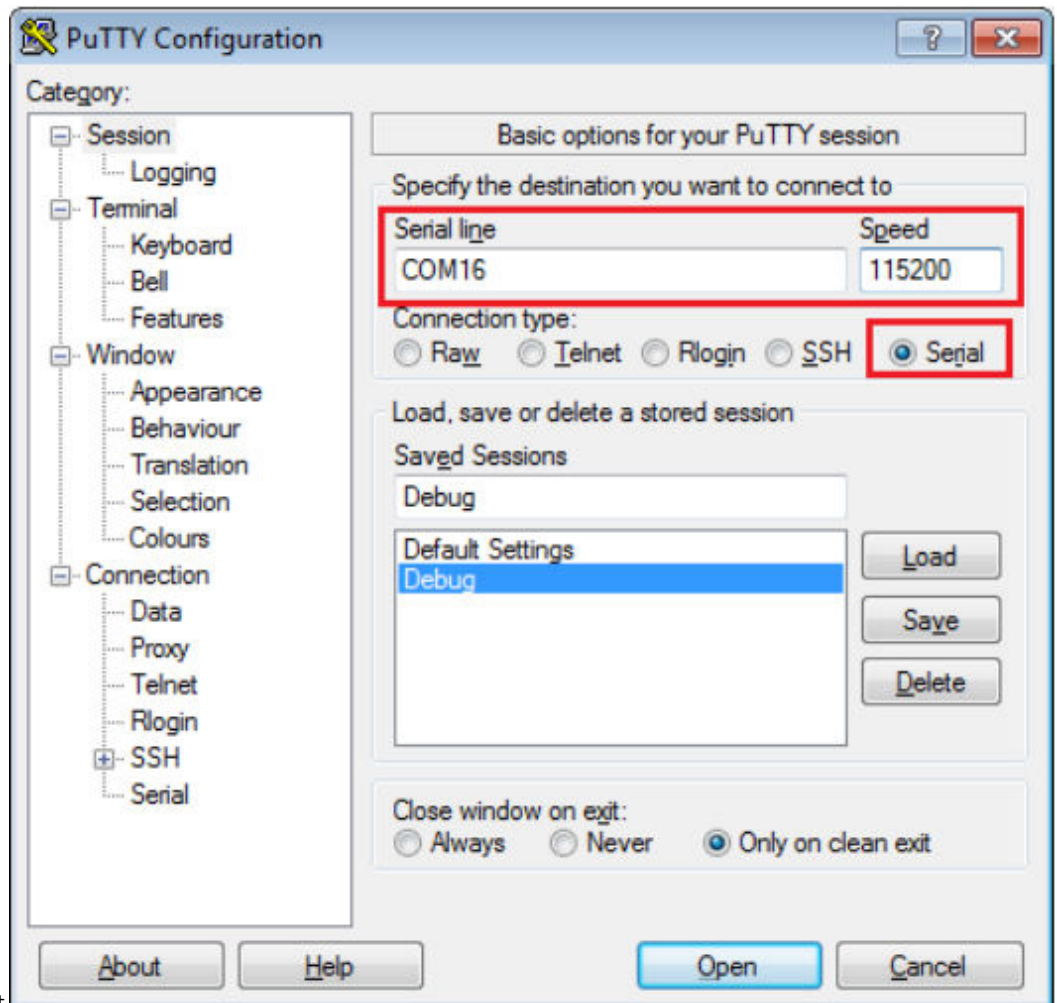
- The OpenSDA interface on your board is programmed with the J-Link OpenSDA firmware. If your board does not support OpenSDA, then a standalone J-Link pod is required.
- You have a standalone J-Link pod that is connected to the debug interface of your board.

Note: Some hardware platforms require hardware modification in order to function correctly with an external debug interface.

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

1. This board supports the J-Link debug probe. Before using it, install SEGGER software, which can be downloaded from [SEGGER](#).

2. Connect the development platform to your PC via USB cable between the OpenSDA USB connector and the PC USB connector. If using a standalone J-Link debug pod, also connect it to the SWD/JTAG connector of the board.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see [How to determine COM port](#)). Configure the terminal with these settings:
 1. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the *board.h* file)
 2. No parity
 3. 8 data bits

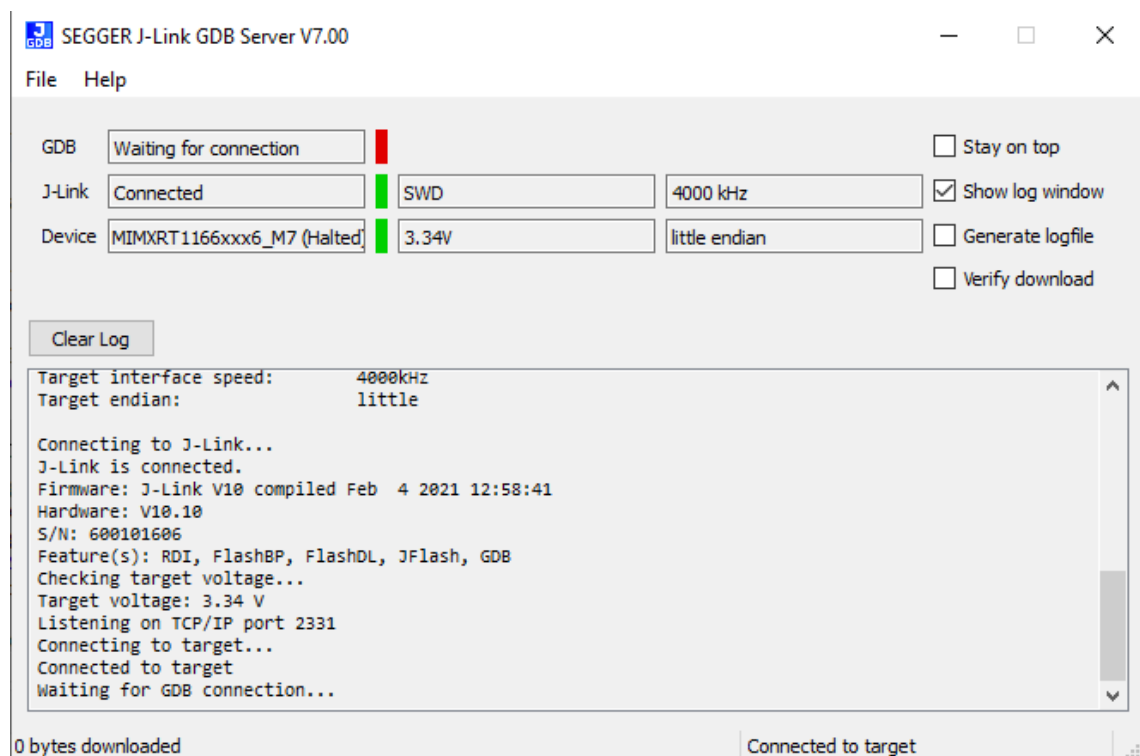


4. 1 stop bit
4. Open the J-Link GDB Server application. Go to the SEGGER install folder. For example, `C:\Program Files(x86)\SEGGER\JLink_Vxxx`. Open the command windows. For Debug and Release targets, use the `JLinkGDBServer.exe` command. For the `sdram_debug`, `sdram_release`, `flexspi_nor_sdram_debug`, and `flexspi_nor_sdram_release` targets, use the `JLinkGDBServer.exe -jlinkscriptfile <install_dir>/boards/evkmimxrt1160/demo_apps/hello_world/cm7/evkmimxrt1160_connect_cm4_cm7side.jlinkscript` command.
5. The target device selection chosen for this example is **MIMXRT1166xxx6_M7**.
6. After it is connected, the screen should resemble Figure 2.

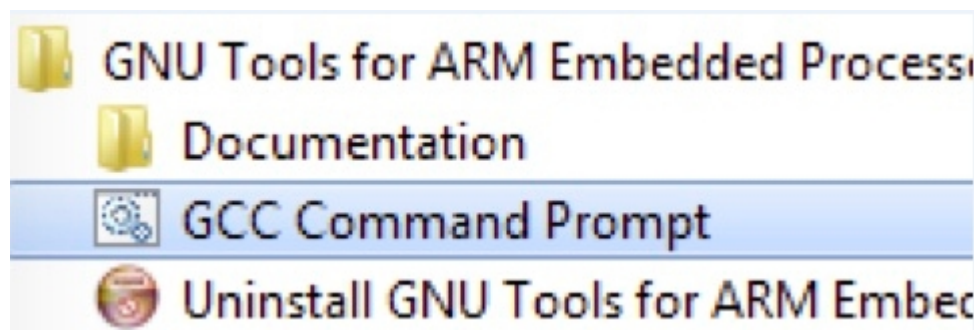
Note:

When download the cm4 project, use **MIMXRT1166xxx6_M4** as the target device. Use the `evkmimxrt1160_connect_cm4_cm4side.jlinkscript` file for non-sdram targets and `evk-`

mimxrt1160_connect_cm4_cm4side_sdram.jlinkscript for sdram targets.



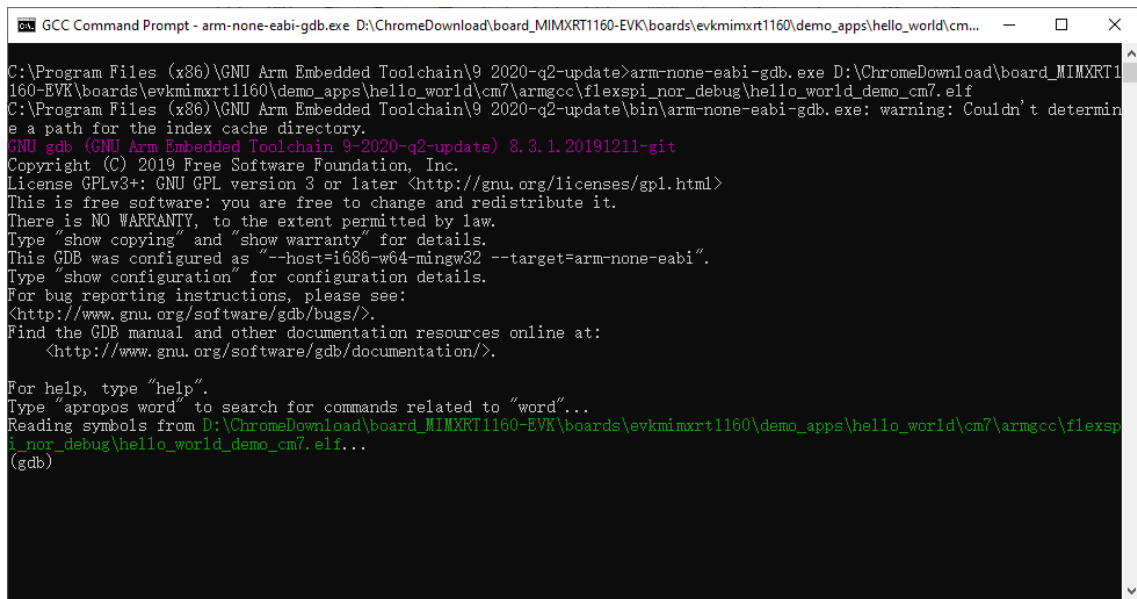
7. If not already running, open a GCC ARM Embedded tool chain command window. To launch the window, from the Windows operating system **Start menu**, go to **Programs** > ****GNU Tools ARM Embedded <version>**** and select **GCC Command Prompt**.



8. Change to the directory that contains the example application output. The output can be found in using one of these paths, depending on the build target selected:
`<install_dir>/boards/<board_name>/<example_type>/<application_name>/cm7/armgcc/debug`
`<install_dir>/boards/<board_name>/<example_type>/<application_name>/cm7/armgcc/release`
 For this example, the path is:

`<install_dir>/boards/evkmimxrt1160/demo_apps/hello_world/cm7/armgcc/debug`

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



```

C:\Program Files (x86)\GNU Arm Embedded Toolchain\9 2020-q2-update>arm-none-eabi-gdb.exe D:\ChromeDownload\board_MIMXRT1160-EVK\boards\evkmimxrt1160\demo_apps\hello_world\cm7\armgcc\flexspi_nor_debug\hello_world_demo_cm7.elf
C:\Program Files (x86)\GNU Arm Embedded Toolchain\9 2020-q2-update\bin\arm-none-eabi-gdb.exe: warning: Couldn't determine a path for the index cache directory.
GNU gdb (GNU Arm Embedded Toolchain 9-2020-q2-update) 8.3.1.20191211-git
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from D:\ChromeDownload\board_MIMXRT1160-EVK\boards\evkmimxrt1160\demo_apps\hello_world\cm7\armgcc\flexspi_nor_debug\hello_world_demo_cm7.elf...
(gdb)

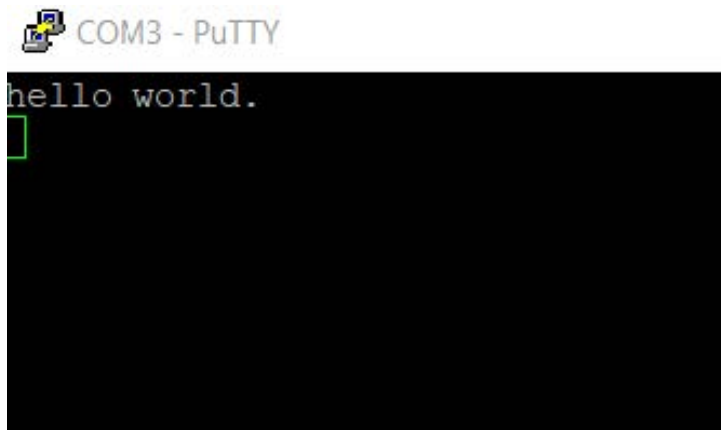
```

10. Run these commands:

```
target remote localhost:2331
load
```

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

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



```

COM3 - PuTTY
hello world.

```

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

Build a multicore example application This section describes the steps to build and run a dual-core application. The demo application build scripts are located in this folder:

```
<install_dir>/boards/evkmimxrt1160/multicore_examples/<application_name>/<core_type>/armgcc
```

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

```
<install_dir>/boards/evkmimxrt1160/multicore_examples/hello_world/cm4/armgcc/build_debug.bat
```

```
<install_dir>/boards/evkmimxrt1160/multicore_examples/hello_world/cm7/armgcc/build_flexspi_nor_debug.bat
```

Build both applications separately following steps for single core examples as described in [Build an example application](#).


```

C:\WINDOWS\system32\cmd.exe
[ 44%] [ 43%] [ 51%] MIMXRT1166/drivers/fsl_clock.c.obj Building C object CMakeFiles/hello_world_cm4.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1166/drivers/fsl_lpuart.c.obj [ 55%] [ 59%]
Building C object CMakeFiles/hello_world_cm4.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1166/utilities/debug_console_lite/fsl_assert.c.obj Building C object CMakeFiles/hello_world_cm4.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/components/lists/fsl_component_generic_list.c.obj [ 62%]
Building ASM object CMakeFiles/hello_world_cm4.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1166/gcc/startup_MIMXRT1166_cm4.S.obj [ 66%] Building C object CMakeFiles/hello_world_cm4.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/components/uart/fsl_adapter_lpuart.c.obj
[ 70%] Building C object CMakeFiles/hello_world_cm4.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1166/utilities/debug_console_lite/fsl_debug_console.c.obj
Building C object CMakeFiles/hello_world_cm4.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/sl_gpio.c.obj
[ 74%] Building C object CMakeFiles/hello_world_cm4.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/bjor_boot.c.c
[ 81%] [ 81%] Building C object CMakeFiles/hello_world_cm4.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/boards/evkmimxrt1160/xip/evkmimxrt1160_flexspi_nor_config.c.obj [ 85%] Building C object CMakeFiles/hello_world_cm4.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1166/system_MIMXRT1166_cm4.c.obj
Building C object CMakeFiles/hello_world_cm4.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1166/drivers/fsl_pmu.c.obj
[ 88%] [ 96%] Building C object CMakeFiles/hello_world_cm4.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1166/utilities/fsl_sbrk.c.obj [ 96%]
Building C object CMakeFiles/hello_world_cm4.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1166/drivers/cm4/fsl_cache.c.obj Building C object CMakeFiles/hello_world_cm4.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1166/drivers/fsl_dcdc.c.obj
[100%] Linking C executable debug\hello_world_cm4.elf
[100%] Built target hello_world_cm4.elf

C:\WINDOWS\system32\cmd.exe
[ 57%] Building C object CMakeFiles/hello_world_cm7.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1166/drivers/fsl_lpuart.c.obj
[ 60%] Building C object CMakeFiles/hello_world_cm7.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1166/utilities/debug_console_lite/fsl_assert.c.obj
[ 64%] Building C object CMakeFiles/hello_world_cm7.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1166/system_MIMXRT1166_cm7.c.obj
[ 67%] Building ASM object CMakeFiles/hello_world_cm7.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1166/utilities/incbin/fsl_incbin.S.obj
[ 71%] Building C object CMakeFiles/hello_world_cm7.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/o.c.obj sl_gpio.c
[ 82%] [ 82%] [ 82%] Building C object CMakeFiles/hello_world_cm7.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1166/xip/fsl_flexspi_nor_boot.c.obj [ 85%] Building ASM object CMakeFiles/hello_world_cm7.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1166/gcc/startup_MIMXRT1166_cm7.S.obj Building C object CMakeFiles/hello_world_cm7.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/boards/evkmimxrt1160/xip/evkmimxrt1160_flexspi_nor_config.c.obj [ 89%]
Building C object CMakeFiles/hello_world_cm7.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1166/drivers/fsl_pmu.c.obj Building C object CMakeFiles/hello_world_cm7.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1166/drivers/fsl_dcdc.c.obj
[ 92%] [ 96%] Building C object CMakeFiles/hello_world_cm7.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1166/drivers/cm7/fsl_cache.c.obj [ Building C object CMakeFiles/hello_world_cm7.elf.dir/D:/ChromeDownload/board_MIMXRT1160-EVK/devices/MIMXRT1166/utilities/fsl_sbrk.c.obj
[100%] Linking C executable flexspi_nor_debug\hello_world_cm7.elf
[100%] Built target hello_world_cm7.elf

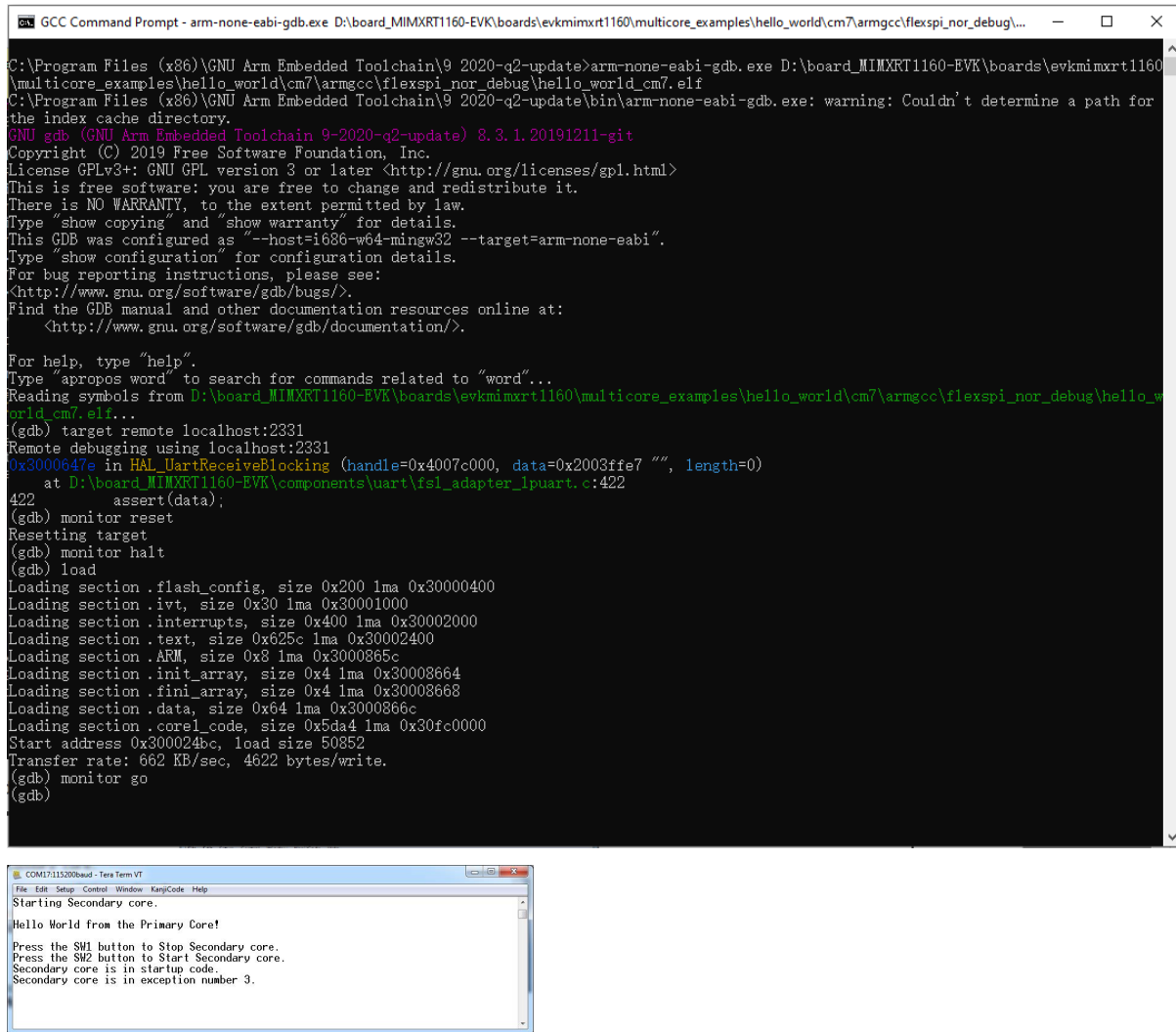
```

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

Run a multicore example application When running a multicore application, the same prerequisites for J-Link/J-Link OpenSDA firmware, and the serial console as for the single-core application, applies, as described in [Run an example application](#).

The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 to 10, as described in [Run an example application](#). These steps are common for both single-core and dual-core applications in Arm GCC.

Both the primary and the auxiliary image is loaded into the SPI flash memory. After execution of the monitor go command, the primary core application is executed. During the primary core code execution, the auxiliary core code is re-allocated from the SPI flash memory to the RAM, and the auxiliary core is released from the reset. The `hello_world` multicore application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.



The image shows two windows. The top window is a 'GCC Command Prompt' running 'arm-none-eabi-gdb.exe'. It displays the GDB startup sequence, including the license and target configuration. The user enters 'target remote localhost:2331' and 'monitor reset'. The GDB interface shows the loading of the ELF file 'hello_world_cm7.elf' and the execution of the program. The bottom window is a 'Tera Term' window titled 'COM17:115200baud - Tera Term VT'. It shows the output of the program: 'Hello World from the Primary Core!'. It also displays instructions for using the SW1 and SW2 buttons to stop and start the secondary core, and a message indicating the secondary core is in exception number 3.

```

C:\Program Files (x86)\GNU Arm Embedded Toolchain\9 2020-q2-update>arm-none-eabi-gdb.exe D:\board_MIMXRT1160-EVK\boards\evkmimxrt1160\multicore_examples\hello_world\cm7\armgcc\flexspi_nor_debug\hello_world_cm7.elf
C:\Program Files (x86)\GNU Arm Embedded Toolchain\9 2020-q2-update\bin\arm-none-eabi-gdb.exe: warning: Couldn't determine a path for the index cache directory.
GNU gdb (GNU Arm Embedded Toolchain 9-2020-q2-update) 8.3.1.20191211-git
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from D:\board_MIMXRT1160-EVK\boards\evkmimxrt1160\multicore_examples\hello_world\cm7\armgcc\flexspi_nor_debug\hello_world_cm7.elf...
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
0x30000647e in HAL_UartReceiveBlocking (handle=0x4007c000, data=0x2003ffe7 "", length=0)
    at D:\board_MIMXRT1160-EVK\components\uart\fsi_adapter_lpuart.c:422
422      assert(data);
(gdb) monitor reset
Resetting target
(gdb) monitor halt
(gdb) load
Loading section .flash_config, size 0x200 lma 0x30000400
Loading section .ivt, size 0x30 lma 0x30001000
Loading section .interrupts, size 0x400 lma 0x30002000
Loading section .text, size 0x625c lma 0x30002400
Loading section .ARM, size 0x8 lma 0x3000865c
Loading section .init_array, size 0x4 lma 0x30008664
Loading section .fini_array, size 0x4 lma 0x30008668
Loading section .data, size 0x64 lma 0x3000866c
Loading section .corel_code, size 0x5da4 lma 0x30fc0000
Start address 0x300024bc, load size 50852
Transfer rate: 662 KB/sec, 4622 bytes/write.
(gdb) monitor go
(gdb)

```

```

COM17:115200baud - Tera Term VT
File Edit Setup Control Window Keyboard Help
Starting Secondary core.
Hello World from the Primary Core!
Press the SW1 button to Stop Secondary core.
Press the SW2 button to Start Secondary core.
Secondary core is in startup code.
Secondary core is in exception number 3.

```

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

MCUXpresso config tools

MCUXpresso Config Tools can help configure the processor and generate initialization code for the on chip peripherals. The tools are able to modify any existing example project, or create a new configuration for the selected board or processor. The generated code is designed to be used with MCUXpresso SDK version 2.x.

Table 1 describes the tools included in the MCUXpresso config tools.

Config tool	Description	Image
Pins tool	For configuration of pin routing and pin electrical properties.	
Clock tool	For system clock configuration	
Peripherals tools	For configuration of other peripherals	
TEE tool	Configures access policies for memory area and peripherals helping to protect and isolate sensitive parts of the application.	
Device Configuration tool	Configures Device Configuration Data (DCD) contained in the program image that the Boot ROM code interprets to setup various on-chip peripherals prior the program launch.	

MCUXpresso Config Tools can be accessed in the following products:

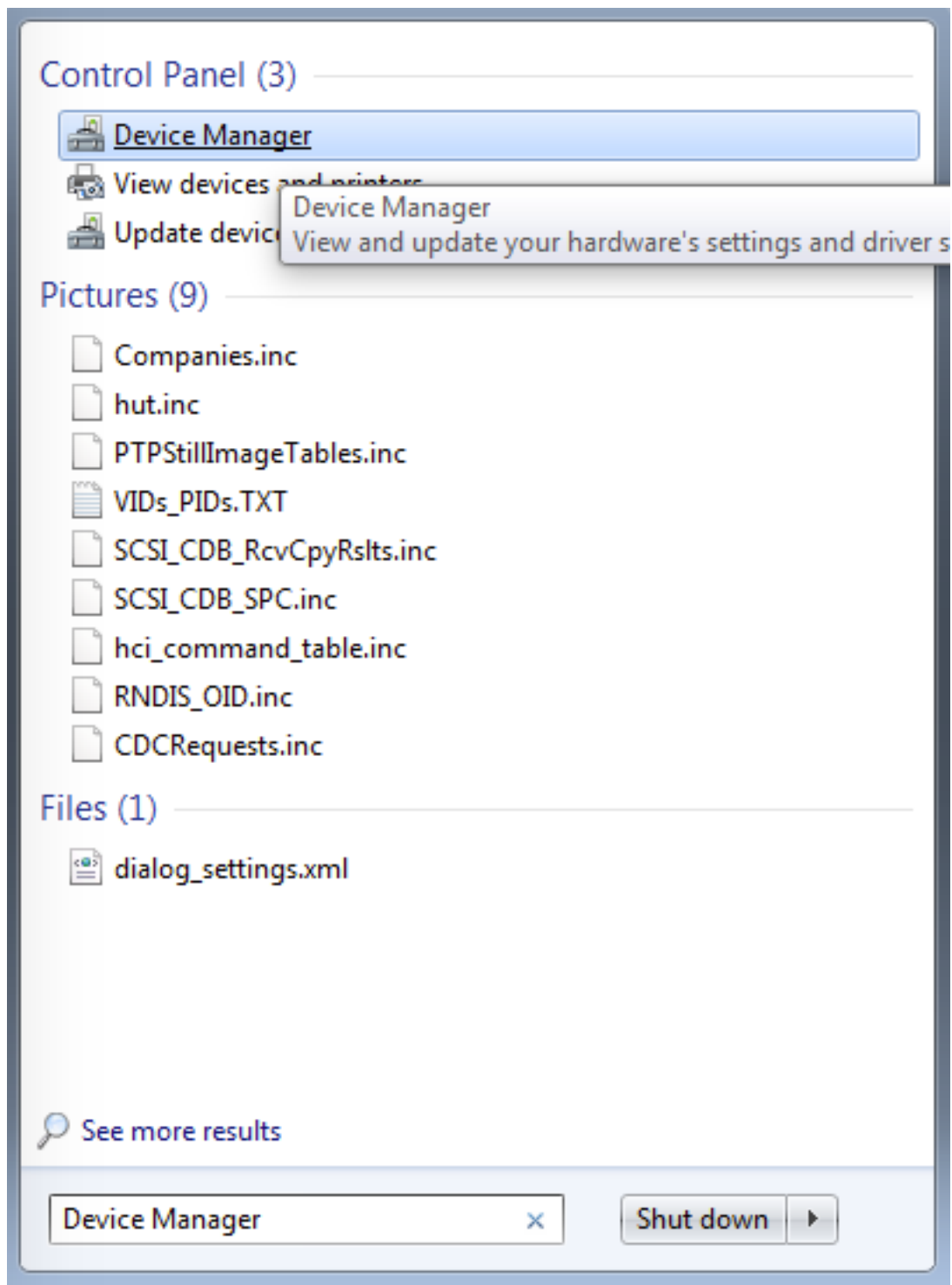
- **Integrated** in the MCUXpresso IDE. Config tools are integrated with both compiler and debugger which makes it the easiest way to begin the development.
- **Standalone version** available for download from [MCUXPRESSO](#). Recommended for customers using IAR Embedded Workbench, Keil MDK μ Vision, or Arm GCC.
- **Online version** available on [MCUXPRESSO](#). Recommended to do a quick evaluation of the processor or use the tool without installation.

Each version of the product contains a specific *Quick Start Guide* document MCUXpresso IDE Config Tools installation folder that can help start your work.

How to determine COM port

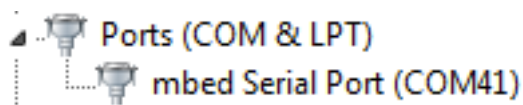
This section describes the steps necessary to determine the debug COM port number of your NXP hardware development platform.

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



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

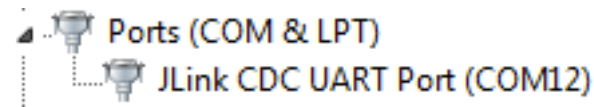
1. OpenSDA – CMSIS-DAP/mbed/DAPLink interface:



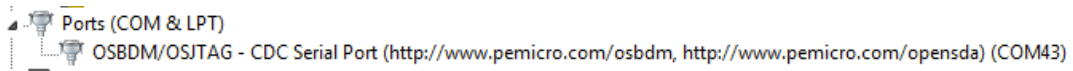
2. OpenSDA – P&E Micro:



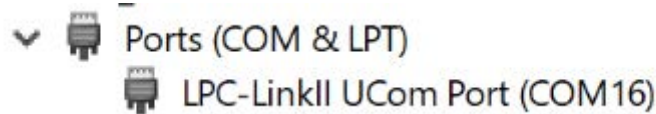
3. OpenSDA – J-Link:



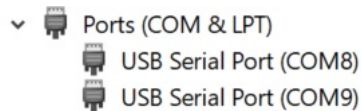
4. P&E Micro OSJTAG:



5. LPC-Link2:



6. FTDI UART:



Default debug interfaces

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

Hardware platform	Default interface	OpenSDA details ¹
EVK-MC56F83000	P&E Micro OSJTAG	N/A
EVK-MIMXRT595	CMSIS-DAP	N/A
EVK-MIMXRT685	CMSIS-DAP	N/A
FRDM-K22F	CMSIS-DAP/mbd/DAPLink	OpenSDA v2.1
FRDM-K28F	DAPLink	OpenSDA v2.1
FRDM-K32L2A4S	CMSIS-DAP	OpenSDA v2.1
FRDM-K32L2B	CMSIS-DAP	OpenSDA v2.1
FRDM-K32W042	CMSIS-DAP	N/A
FRDM-K64F	CMSIS-DAP/mbd/DAPLink	OpenSDA v2.0
FRDM-K66F	J-Link OpenSDA	OpenSDA v2.1
FRDM-K82F	CMSIS-DAP	OpenSDA v2.1
FRDM-KE15Z	DAPLink	OpenSDA v2.1
FRDM-KE16Z	CMSIS-DAP/mbd/DAPLink	OpenSDA v2.2
FRDM-KL02Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL03Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL25Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL26Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL27Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL28Z	P&E Micro OpenSDA	OpenSDA v2.1
FRDM-KL43Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL46Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL81Z	CMSIS-DAP	OpenSDA v2.0

continues on next page

Table 1 – continued from previous page

Hardware platform	Default interface	OpenSDA details ¹
FRDM-KL82Z	CMSIS-DAP	OpenSDA v2.0
FRDM-KV10Z	CMSIS-DAP	OpenSDA v2.1
FRDM-KV11Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KV31F	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KW24	CMSIS-DAP/mbd/DAPLink	OpenSDA v2.1
FRDM-KW36	DAPLink	OpenSDA v2.2
FRDM-KW41Z	CMSIS-DAP/DAPLink	OpenSDA v2.1 or greater
Hexiwear	CMSIS-DAP/mbd/DAPLink	OpenSDA v2.0
HVP-KE18F	DAPLink	OpenSDA v2.2
HVP-KV46F150M	P&E Micro OpenSDA	OpenSDA v1
HVP-KV11Z75M	CMSIS-DAP	OpenSDA v2.1
HVP-KV58F	CMSIS-DAP	OpenSDA v2.1
HVP-KV31F120M	P&E Micro OpenSDA	OpenSDA v1
JN5189DK6	CMSIS-DAP	N/A
LPC54018 IoT Module	N/A	N/A
LPCXpresso54018	CMSIS-DAP	N/A
LPCXpresso54102	CMSIS-DAP	N/A
LPCXpresso54114	CMSIS-DAP	N/A
LPCXpresso51U68	CMSIS-DAP	N/A
LPCXpresso54608	CMSIS-DAP	N/A
LPCXpresso54618	CMSIS-DAP	N/A
LPCXpresso54628	CMSIS-DAP	N/A
LPCXpresso54S018M	CMSIS-DAP	N/A
LPCXpresso55s16	CMSIS-DAP	N/A
LPCXpresso55s28	CMSIS-DAP	N/A
LPCXpresso55s69	CMSIS-DAP	N/A
MAPS-KS22	J-Link OpenSDA	OpenSDA v2.0
MIMXRT1160-EVK	CMSIS-DAP	N/A
MIMXRT1170-EVK	CMSIS-DAP	N/A
TWR-K21D50M	P&E Micro OSJTAG	N/A OpenSDA v2.0
TWR-K21F120M	P&E Micro OSJTAG	N/A
TWR-K22F120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K24F120M	CMSIS-DAP/mbd	OpenSDA v2.1
TWR-K60D100M	P&E Micro OSJTAG	N/A
TWR-K64D120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K64F120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K65D180M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K65D180M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV10Z32	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K80F150M	CMSIS-DAP	OpenSDA v2.1
TWR-K81F150M	CMSIS-DAP	OpenSDA v2.1
TWR-KE18F	DAPLink	OpenSDA v2.1
TWR-KL28Z72M	P&E Micro OpenSDA	OpenSDA v2.1
TWR-KL43Z48M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KL81Z72M	CMSIS-DAP	OpenSDA v2.0
TWR-KL82Z72M	CMSIS-DAP	OpenSDA v2.0
TWR-KM34Z75M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KM35Z75M	DAPLink	OpenSDA v2.2
TWR-KV10Z32	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV11Z75M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV31F120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV46F150M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV58F220M	CMSIS-DAP	OpenSDA v2.1
TWR-KW24D512	P&E Micro OpenSDA	OpenSDA v1.0
USB-KW24D512	N/A External probe	N/A

continues on next page

Table 1 – continued from previous page

Hardware platform	Default interface	OpenSDA details ¹
USB-KW41Z	CMSIS-DAP\DAPLink	OpenSDA v2.1 or greater

¹ The OpenSDA details is not applicable to LPC.

How to add or remove boot header for XIP targets

The MCUXpresso SDK for i.MX RT1160 provides `flexspi_nor_debug` and `flexspi_nor_release` targets for each example and/or demo which supports XIP (eXecute-In-Place). These two targets add `XIP_BOOT_HEADER` to the image by default. Because of this, ROM can boot and run this image directly on external flash.

Macros for the boot leader:

- The following three macros are added in `flexspi_nor` targets to support XIP, as described in *Table 1*.

^

| **XIP_EXTERNAL_FLASH** | 1: Exclude the code which changes the clock of FLEXSPI. | 0: Make no changes. | | **XIP_BOOT_HEADER_ENABLE** | 1: Add FLEXSPI configuration block, image vector table, boot data, and device configuration data (optional) to the image by default. | 0: Add nothing to the image by default. | | **XIP_BOOT_HEADER_DCD_ENABLE** | 1: Add device configuration data to the image. | 0: Do **NOT** add device configuration data to the image. |

- Table 2* shows the different effect on the built image with a different combination of these macros.

^

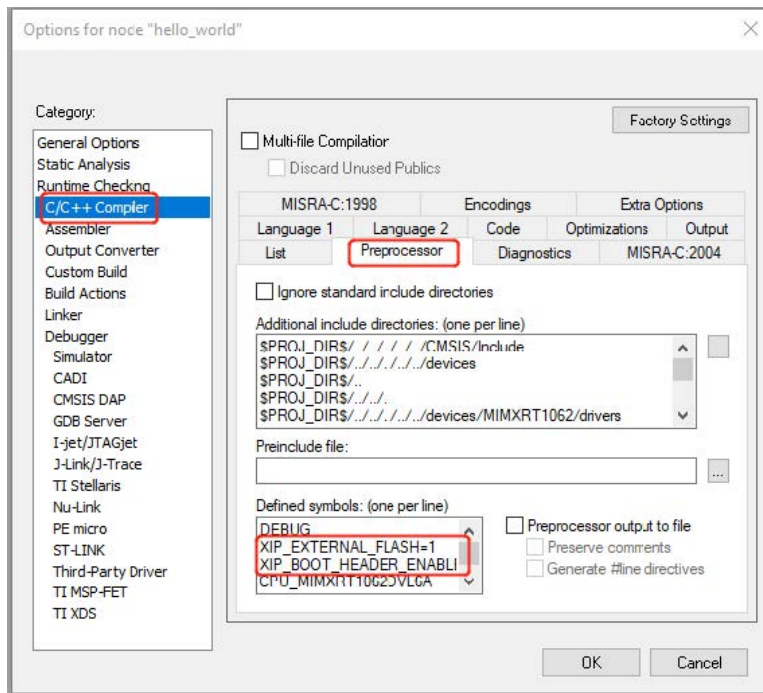
XIP_BOOT_HEADER XIP_BOOT_HEADER_DCD_ENABLE=0
XIP_EXTERNAL_FLASH XIP_BOOT_HEADER_ENABLE - Can be programmed to qspi flash by IDE and can run after POR reset if qspi flash is the boot source.

- SDRAM will be initialized. | - Can be programmed to qspi flash by IDE, and can run after POR reset if qspi flash is the boot source.
- SDRAM will **NOT** be initialized. | | **XIP_BOOT_HEADER_ENABLE=0** | - **CANNOT** run after POR reset if it is programmed by IDE, even if qspi flash is the boot source. | — | | **XIP_EXTERNAL_FLASH=0** | - This image **CANNOT** complete XIP because when this macro is set to 1, it excludes the code, which changes the clock for FLEXSPI. |

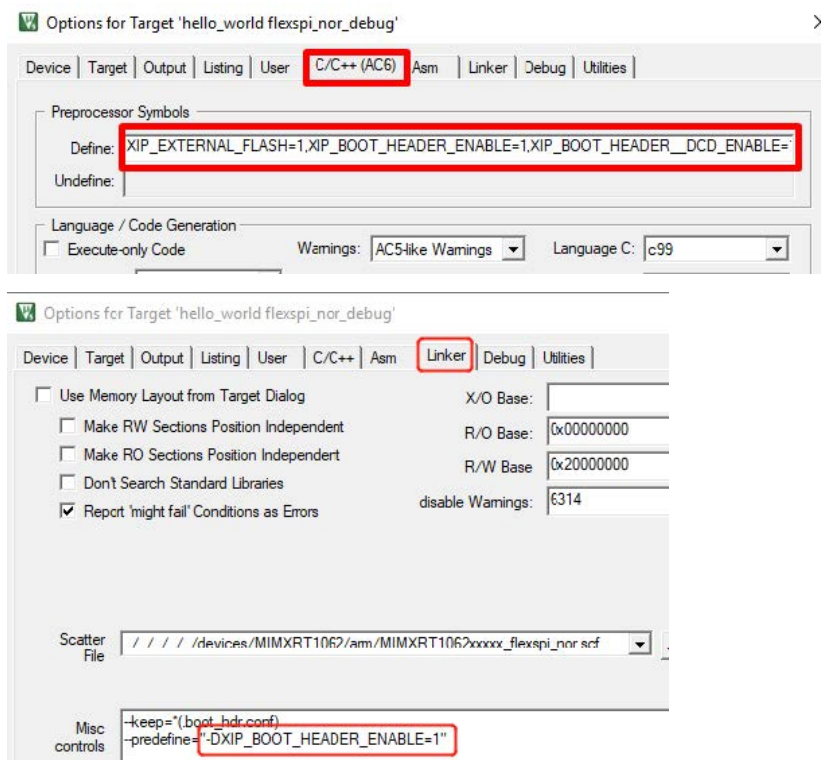
Where to change the macros for each toolchain in MCUXpresso SDK?

Take `hello_world` as an example:

- IAR**



• MDK



• ARMGCC

Change the configuration in *CMakeLists.txt*.

```
SET(CMAKE_C_FLAGS_SDRAM_RELEASE "${CMAKE_C_FLAGS_SDRAM_RELEASE} -std=gnu99")

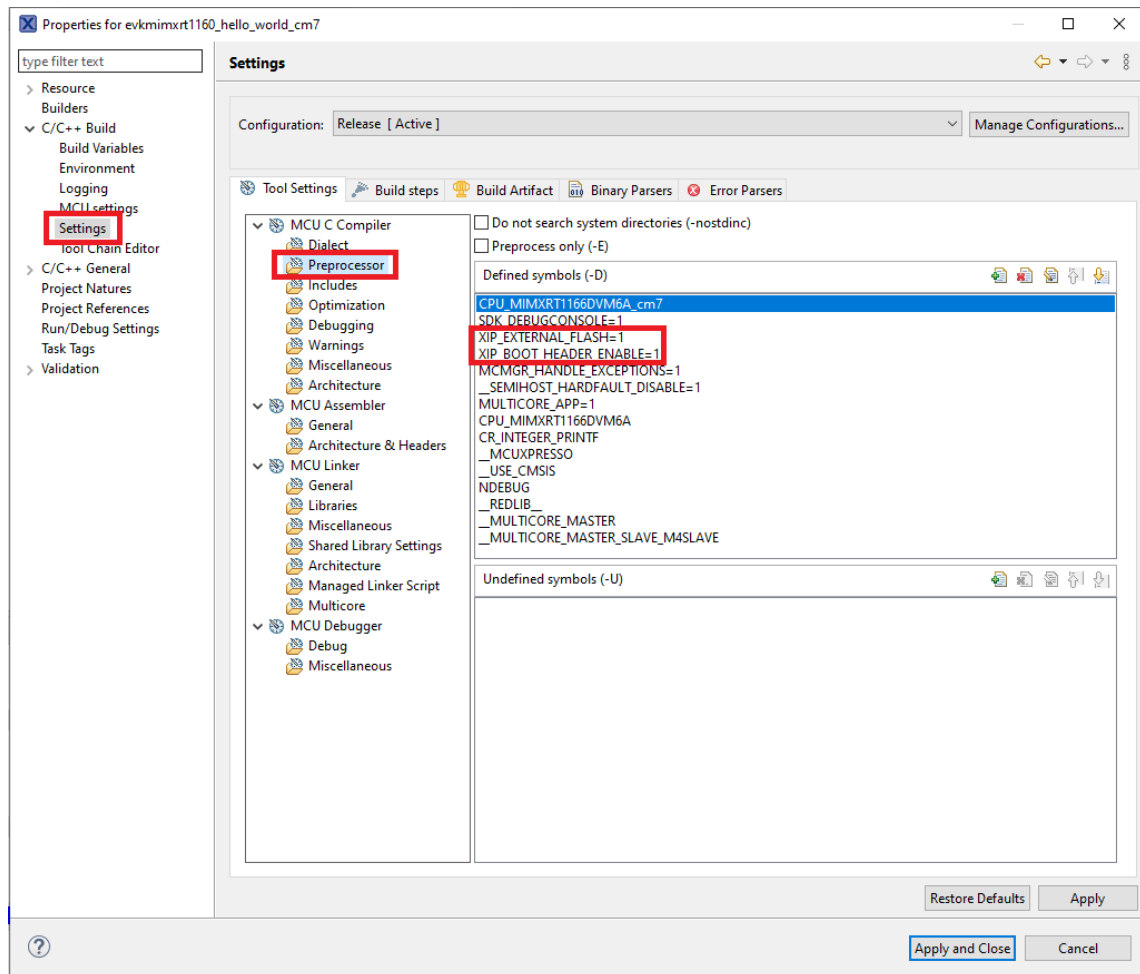
SET(CMAKE_C_FLAGS_FLEXSPI_NOR_DEBUG "${CMAKE_C_FLAGS_FLEXSPI_NOR_DEBUG} -DXIP_EXTERNAL_FLASH=1")

SET(CMAKE_C_FLAGS_FLEXSPI_NOR_DEBUG "${CMAKE_C_FLAGS_FLEXSPI_NOR_DEBUG} -DXIP_BOOT_HEADER_ENABLE=1")

SET(CMAKE_C_FLAGS_FLEXSPI_NOR_DEBUG "${CMAKE_C_FLAGS_FLEXSPI_NOR_DEBUG} -DXIP_BOOT_HEADER_DCD_ENABLE=1")

SET(CMAKE_C_FLAGS_FLEXSPI_NOR_DEBUG "${CMAKE_C_FLAGS_FLEXSPI_NOR_DEBUG} -DCPU_MIMXRT1052DVL6A")
```

- MCUX



1.3 Getting Started with MCUXpresso SDK GitHub

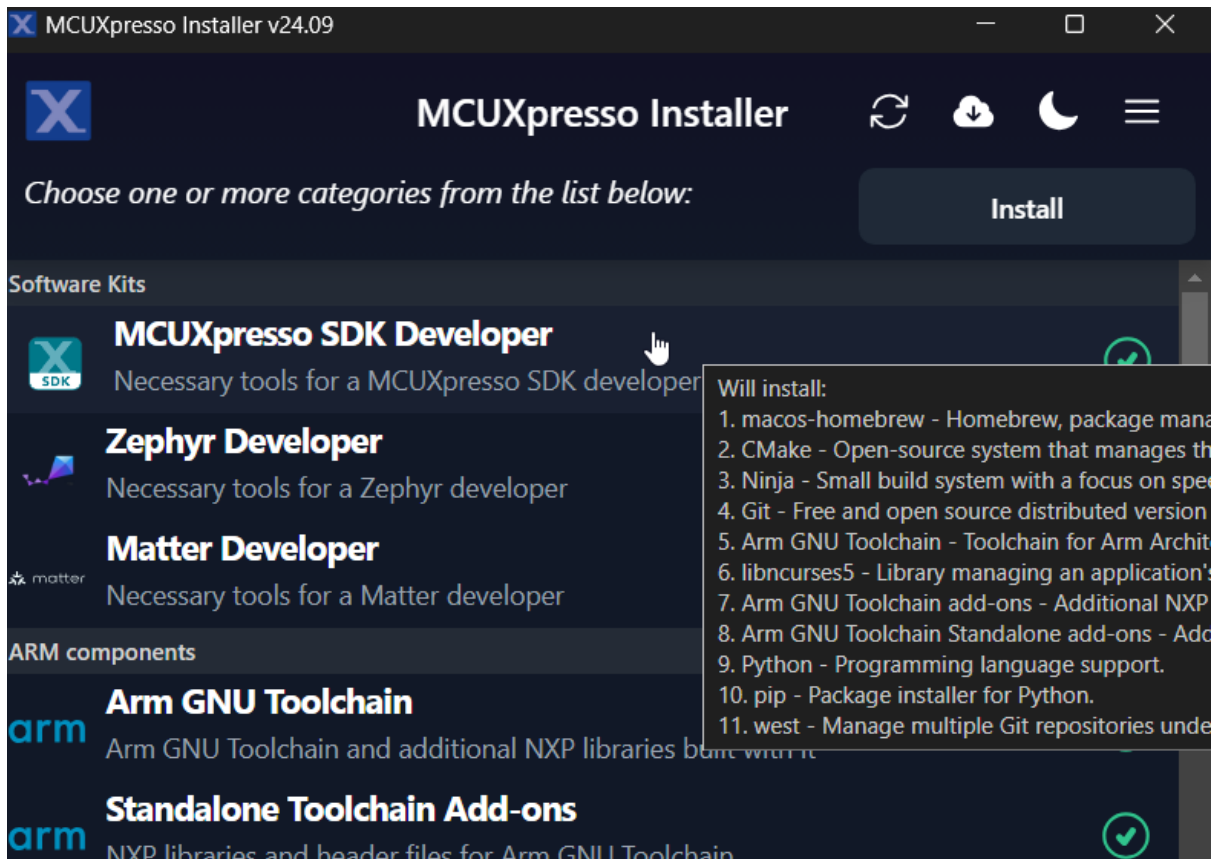
1.3.1 Getting Started with MCUXpresso SDK Repository

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.

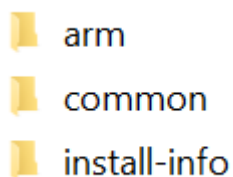
Toolchain MCUXpresso SDK supports all mainstream toolchains for embedded development. You can install your used or interested toolchains following the guides.

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	AR-MGCC_DIR	C:\armgcc for windows/usr for Linux. Typically arm-none-eabi-* is installed under /usr/bin	– toolchain armgcc
IAR	IAR_DIR	C:\iar\ewarm-9.60.3 for Windows/opt/iarsystems/bxarm-9.60.3 for Linux	– toolchain iar
MDK	MDK_DIR	C:\Keil_v5 for Windows.MDK IDE is not officially supported with Linux.	– toolchain mdk
Armclang	ARM-CLANG_DIR	C:\ArmCompilerforEmbedded6.22 for Windows/opt/ArmCompilerforEmbedded6.21 for Linux	– toolchain mdk
Zephyr	ZEPHYR_SDK	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	RISCV-LVM_DIR	C:\riscv-llvm-win32_b298_b298_2024.08.12 for Windows/opt/riscv-llvm-Linux-x64_b298_b298_2024.08.12 for Linux	– toolchain riscv-llvm

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



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

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

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

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

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

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

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

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

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

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

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

Get MCUXpresso SDK Repo

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

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

# Update the west projects
cd mcuxpresso-sdk
west update

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

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

1. Navigate to the core directory:

```
cd mcuxsdk
```

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

```
python -m venv .venv

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

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

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

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

3. Install the required Python packages:

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

Explore Contents

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

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

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

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

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

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

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

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

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

Run a demo using MCUXpresso for VS Code

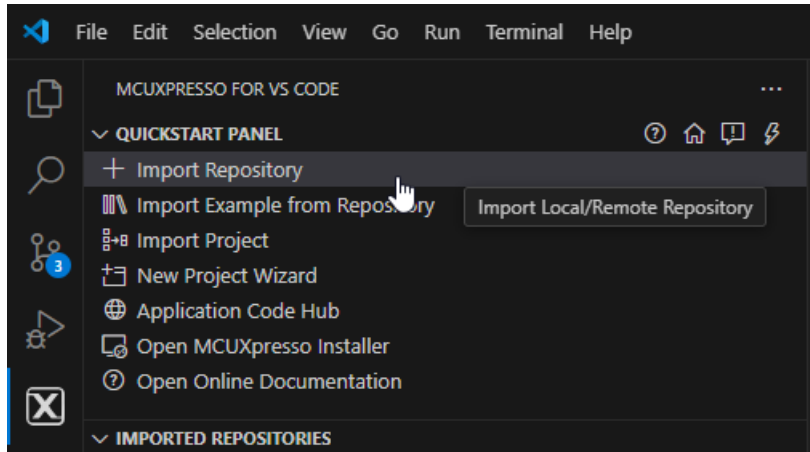
This section explains how to configure MCUXpresso for VS Code to build, run, and debug example applications. This guide uses the `hello_world` demo application as an example. However, these

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

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

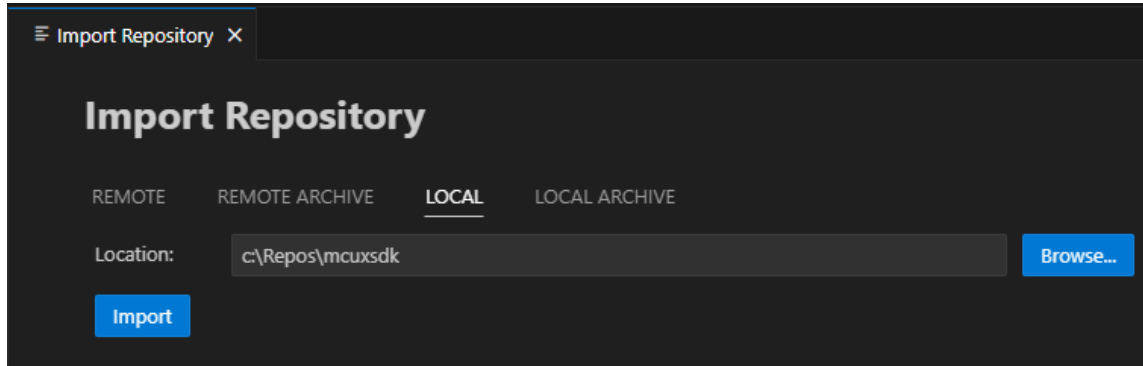
To build an example application:

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

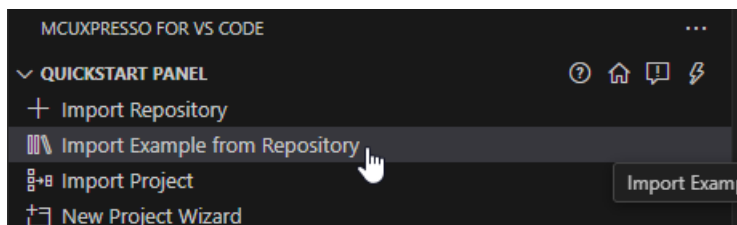


Note: You can import the SDK in several ways. Refer to [MCUXpresso for VS Code Wiki](#) for details.

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



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



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

Import Example from Repository

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

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

Board: FRDM-MCXC444 | v

 FRDM-MCXC444

Template: demo_apps/hello_world | v

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

App type: Freestanding application | v

Name: frdmmxc444_hello_world

Location: c:\nxp_examples [Browse...](#)

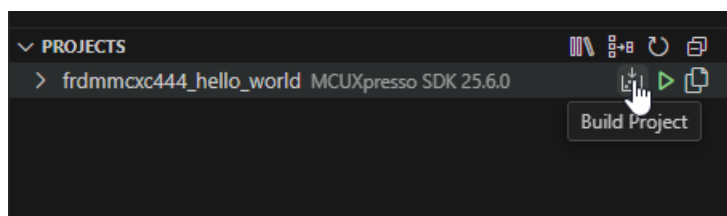
Note: Path doesn't exist. Folder(s) will be created.

☐ Open readme file after project is imported

[Import](#)

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

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



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

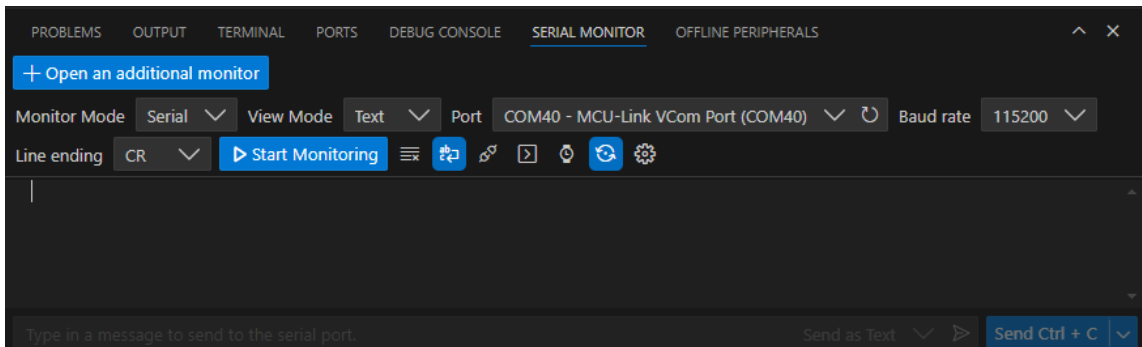
```

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

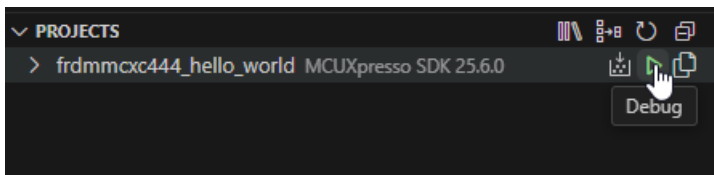
```

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

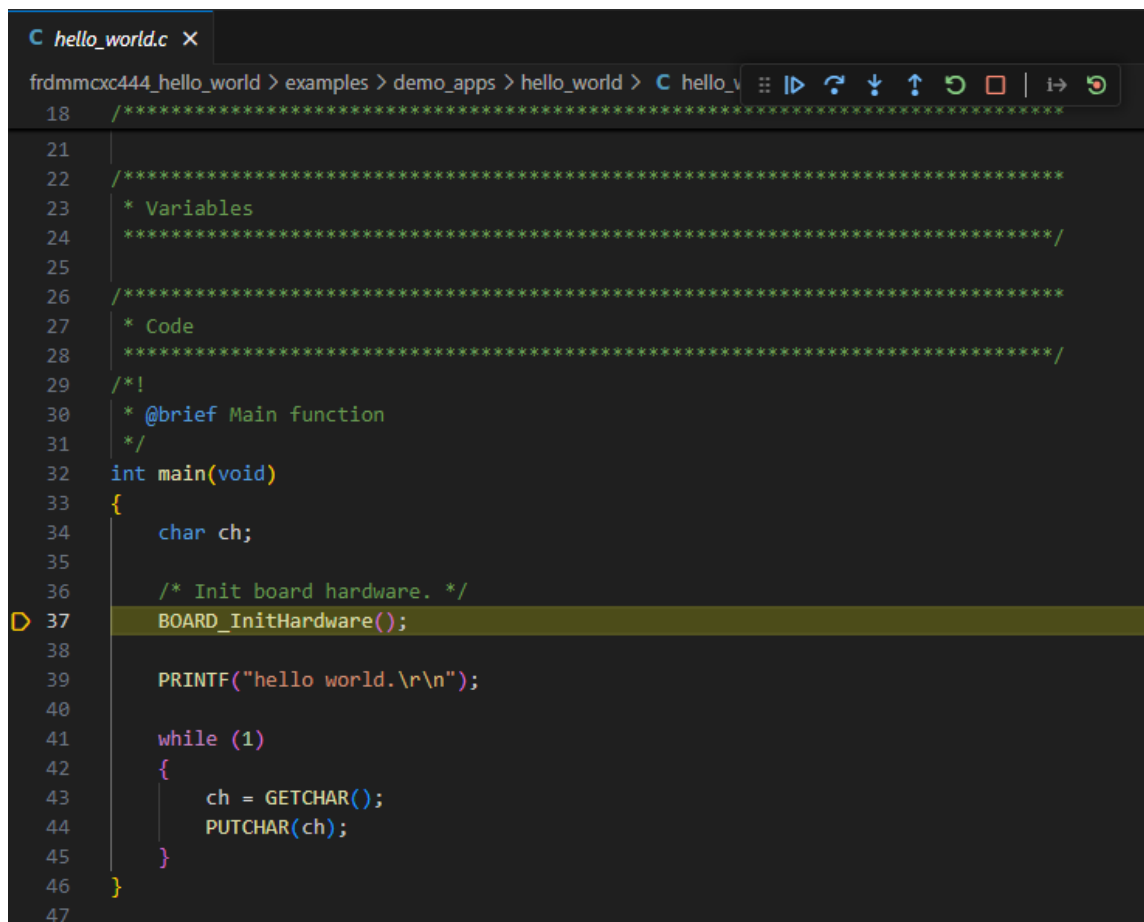
1. Open the **Serial Monitor** from the VS Code's integrated terminal. Select the VCom Port for your device and set the baud rate to 115200.



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



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

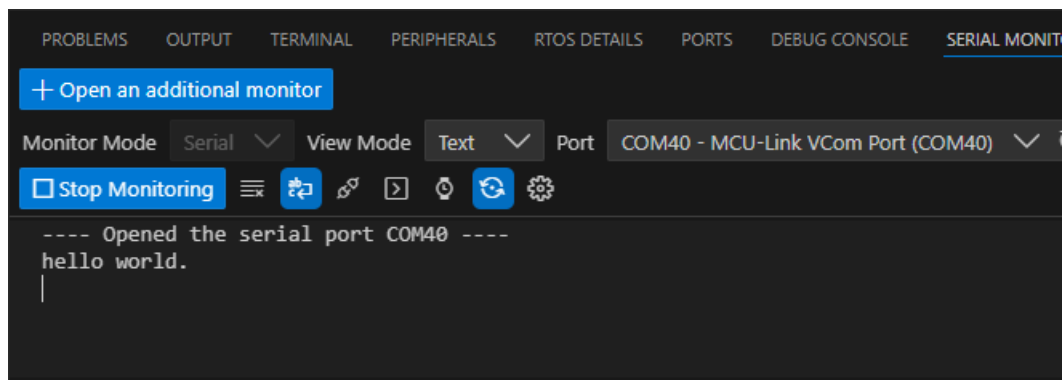


```

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

```

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



```

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

```

Running a demo using ARMGCC CLI/IAR/MDK

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

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

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

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

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

(continues on next page)

(continued from previous page)

```

↪ evkbbmimxrt1060]
INFO: [ 4][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b ↪
↪ evkbbmimxrt1170 -Dcore_id=cm4]
INFO: [ 5][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b ↪
↪ evkbbmimxrt1170 -Dcore_id=cm7]
INFO: [ 6][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b ↪
↪ evkcmimxrt1060]
INFO: [ 7][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b ↪
↪ evkmcimx7ulp]
...

```

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

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

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

Here are some typical usages for generating a SDK example:

```

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

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

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

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

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

```

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

```

west build -b evkbbmimxrt1170 examples/demo_apps/hello_world --toolchain iar -Dcore_id=cm7 --config ↪
↪ flexspi_nor_debug

```

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

```

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

```

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

```

west build -b evkbbmimxrt1170 --sysbuild ./examples/multicore_examples/hello_world/primary -Dcore_ ↪
↪ id=cm7 --config flexspi_nor_debug --toolchain=armgcc -p always

```

For more details, please refer to System build.

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

1. Run cmake configuration

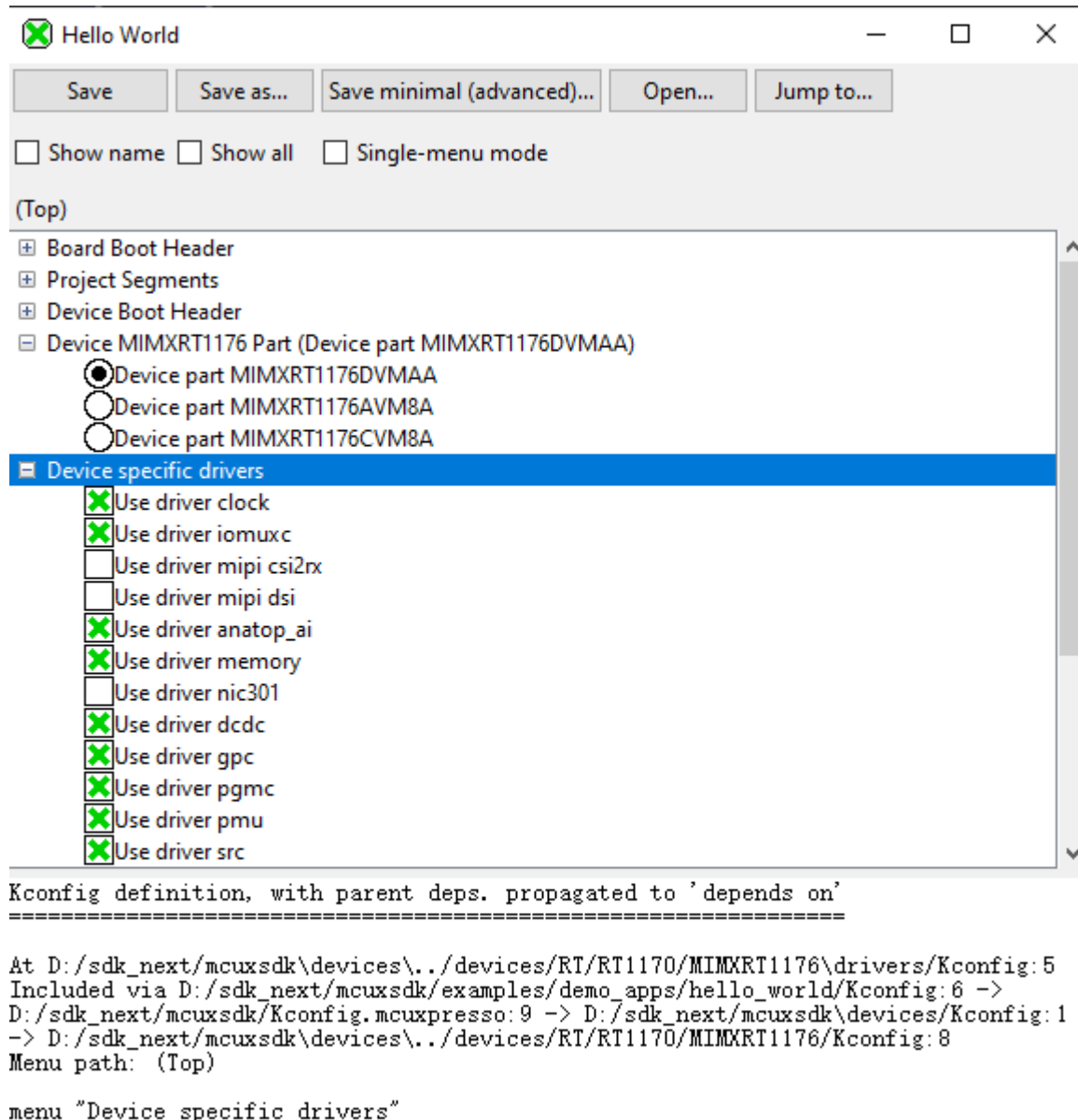
```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

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

2. Run guiconfig target

```
west build -t guiconfig
```

Then you will get the Kconfig GUI launched, like



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

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

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

Flash the hello_world example:

```
west flash -r linkserver
```

Debug Start a gdb interface by following command:

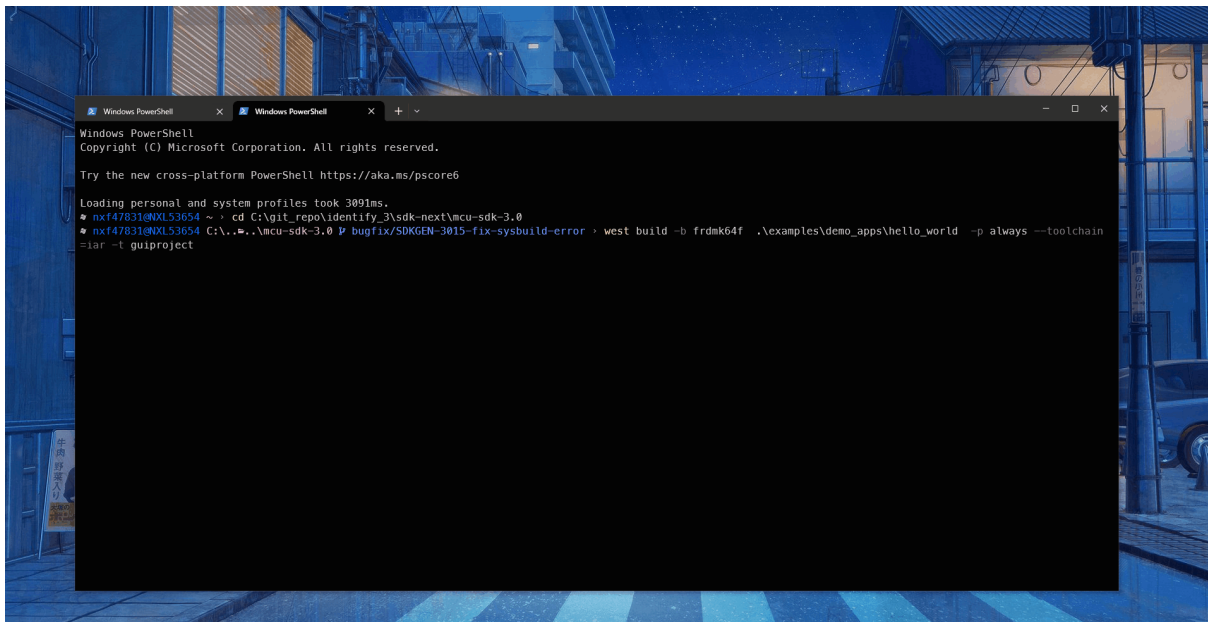
```
west debug -r linkserver
```

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

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

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

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



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

1.4 Release Notes

1.4.1 MCUXpresso SDK Release Notes

Overview

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

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

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

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

MCUXpresso SDK

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

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

Development tools

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

- MCUXpresso IDE, Rev. 25.06.xx
- IAR Embedded Workbench for Arm, version is 9.60.4
- Keil MDK, version is 5.42
- MCUXpresso for VS Code v25.09
- GCC Arm Embedded Toolchain 14.2.x

Supported development systems

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

Devel- opment boards	MCU devices		
MIMXRT1160-EVK	MIMXRT1165CVM5A, MIMXRT1166CVM5A,	MIMXRT1165DVM6A, MIMXRT1166DVM6A ,	MIMXRT1165XVM5A, MIMXRT1166XVM5A

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.

MCU Boot MCU Boot (formerly KBOOT) NXP/Freescale proprietary loader

coreHTTP coreHTTP

openvg OpenVG library for devices with graphics acceleration hardware

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

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

VG-Lite GPU Library VGLite library for devices with VGLite graphics hardware acceleration engine

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

Simple Open EtherCAT Master Simple Open EtherCAT Master (SOEM) is an open source EtherCAT master stack that is used to write custom EtherCAT Master applications. For more information on how to use SOEM, see the *Getting Started with MCUXpresso SDK for SOEM* document.

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

PNG decoder An 'embedded-friendly' PNG image decoding 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".

Openh264 H.264 Codec Library

Multicore Multicore Software Development Kit

MMCAU The NXP Memory-Mapped Cryptographic Acceleration Unit

MCU Boot Open source MCU Bootloader.

mbedTLS mbedtls SSL/TLS library v3.x

mbedTLS mbedtls SSL/TLS library v2.x

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

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

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

Maestro Audio Framework for MCU Maestro Audio Framework library for MCU

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

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

Audio Voice components Audio Voice components for MCU

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

eIQ machine learning SDK containing:

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

LVGL LVGL Open Source Graphics Library

llhttp HTTP parser llhttp

LittleFS LittleFS filesystem stack

JPEG library JPEG library

FreeMASTER FreeMASTER communication driver for 32-bit platforms.

File systemFatfs The FatFs file system is integrated with the MCUXpresso SDK and can be used to access either the SD card or the USB memory stick when the SD card driver or the USB Mass Storage Device class implementation is used.

emWin The MCUXpresso SDK is pre-integrated with the SEGGER emWin GUI middleware. The AppWizard provides developers and designers with a flexible tool to create stunning user interface applications, without writing any code.

NAND Flash Management Stack NAND Flash Management Stack

cJSON Ultralightweight JSON parser in ANSI C

NXP PSA CRYPTO DRIVER PSA crypto driver for crypto library integration via driver wrappers

Release contents

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

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

Known issues

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

New Project Wizard compile failure

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

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

Note: xxx means core variants, such as, cm0plus, cm33, cm4, cm33_nodsp.

****Components:****issdk_mag3110, issdk_host, systick, gpio_kinetis, gpio_lpc, issdk_mpl3115, sensor_fusion_agm01, sensor_fusion_agm01_lpc, issdk_mma845x, issdk_mma8491q, issdk_mma865x, issdk_mma9553, and CMSIS_RTOS2.CMSIS_RTOS2, and components which include cache driver, such as enet_qos.

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

For example, if there are two types of timer adapters, gpt_adapter and pit_adapter, only one can be selected as timer adapter

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

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

CMSIS-PACK svd issue

CMSIS-PACK DFP installation takes a while. When installing cmsis-pack DFP, Keil MDK processes the MCU SVD file. The large size of SVD file takes considerable time to finish this conversion. During the installation, the progress appears stalled. However, it finishes after approximately 20 minutes.

CMSIS PACK new project compile failure

The generated configuration cannot be applied globally. The components, `serial_manager_usb_cdc_virtual` and `serial_manager_usb_cdc_virtual_xxx` (xxx means core variants like `cm0plus`, `cm33`, `cm4`, and `cm33_nodsp`) are unsupported for new project wizard of CMSIS pack and will lead to compile failure if selected while creating new project(s).

MCUXpresso IDE limitation

- **Cannot debug cm4 sdram related demos with CMSIS-DAP**

MCUXpresso IDE does not support initialization of sdram when debugging.

IAR debug limitation

CM4 flash target demos cannot be debugged on IAR with JLINK.

Extra option required when using CMSIS-DAP to debug

When using CMSIS-DAP to debug CM4 sdram related target in IAR, such as `flexspi_nor_sdram` and `sdram_txt`, an extra option must be specified in the debugger settings.

`aws_httpscli_corehttp` example for `evkmimxrt1160` issue in MCUXpressoIDE release target

The `aws_httpscli_corehttp` example for `evkmimxrt1160` does not work correctly in MCUXpressoIDE release target. Use the debug target only in this IDE.

`aws_httpscli_corehttp` example for `evkmimxrt1160` issue in MCUXpressoIDE release target

The `aws_httpscli_corehttp` example for `evkmimxrt1160` does not work correctly in MCUXpressoIDE release target. Use the debug target only in this IDE.

The `cmsis_lpi2c_edma_b2b_transfer` examples don't work correctly on CM4 core.

Boards cannot transfer data successfully.

Affected toolchains: mcux **Affected platforms:** `evkmimxrt1160`, `evkbmimxrt1170`

Modify dummy cycles value for external qspi flash

More NXP SOCs now support executed code in external flash. Projects require higher QSPI speed. According to QSPI flash device datasheet descriptions, higher QSPI speed operates stably only when you pair it with the appropriate dummy cycle value.

Note that some board XIP files directly modify the dummy cycle value in external flash through ROM using the volatile method. Such modifications may not work with released toolchain versions. Upgrade the current toolchain to the latest version. NXP also optimizes the corresponding flashloader.

When users modify dummy cycle value in non-volatile register of external flash, the NXP flashloader becomes invalid. Users need to create their own flashloader to adapt to the external flash dummy cycle requirements.

Affected platforms: mimxrt1020-evk, mimxrt1060-evkb, mimxrt1160-evk, mimxrt1170-evkb

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
-

ACMP

[2.4.0]

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

[2.3.0]

- Improvements
 - Expose C0 register FILTER_CNT bitfield and FPR bitfield to the user.

[2.2.0]

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

[2.1.0]

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

[2.0.6]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

- Other Changes
 - Changed coding style of peripheral base address from "s_acmpBases" to "s_acmpBase".

[2.0.1]

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

ADC_ETC

[2.3.2]

- Improvements
 - Corrected that FSL_FEATURE_ADC_ETC_HAS_NO_TSC1_TRIG should be used instead of FSL_FEATURE_ADC_ETC_HAS_NO_TSC0_TRIG in some places.
 - For ADC_ETC without TSC trigger source, CTRL [bit 30] shall be cleared explicitly.

[2.3.1]

- Improvements
 - Change ADC_ETC default DMA Mode to kADC_ETC_TrigDMAWithPulsedSignal. Generally speaking, DMA transfer requests should only be cleared by DMA ACK, and the CPU should not clear the request source. If some users use option kADC_ETC_TrigDMAWithLatchedSignal, changing the mode to kADC_ETC_TrigDMAWithPulsedSignal also meet their requirements.

[2.3.0]

- Improvements
 - Added blocking way to implement SW trigger.

[2.2.1]

- Improvements
 - Modified macro “ADC_ETC_DONE2_ERR_IRQ_TRIG0_DONE2_MASK” to “ADC_ETC_DONE2_3_ERR_IRQ_TRIG0_DONE2_MASK” based on the updates of header file.

[2.2.0]

- Improvements
 - Defined two macros to support some devices that do not equipped with TSC trigger.

[2.1.1]

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

[2.1.0]

- New Features
 - Supported independent IRQ enable bit in ADC-ETC chain configuration registers.
 - Supported trigger n DONE3 interrupt operations.
- Bug Fixes
 - Fixed the violation of MISRA-2012 rules:
 - * Rule 10.1 10.3 10.7 15.5 16.1 16.3 16.4 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.
-

ANATOP_AI**[2.0.0]**

- initial version.
-

AOI**[2.0.2]**

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

[2.0.1]

- Bug Fixes
 - MISRA C-2012 issue fixed: rule 10.8, 2.2.

[2.0.0]

- Initial version.
-

ASRC**[2.1.3]**

- Bug Fixes
 - Fixed function did not match the specified channel pair issue.

[2.1.2]

- Improvements
 - Correct feature name in source file by changing FSL_FEATURE_ASRC_PARAMETER_REGISTER_NAME_ to FSL_FEATURE_ASRC_PARAMETER_REGISTER_NAME_ASRPM.
 - Removed the asrc_clock_source_t from driver header file, as SOC header file will provide detail definition.
- Bug Fixes
 - Fixed the ASRC_SetChannelPairConfig/ASRC_ChannelPairEnable functions missing functionality when using channel pair B/C.
 - Fixed violations of the MISRA C-2012 rules 10.7.

[2.1.1]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 10.1–10.4, 12.2.

[2.1.0]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 10.3, 10.4, 14.4, 10.1, 17.7, 11.9, 8.6, 12.2, 11.6.

[2.0.1]

- Improvements
 - Added feature macro FSL_FEATURE_ASRC_PARAMETER_REGISTER_NAME_ASRPM for ASRC parameter register.
- Bug Fixes
 - Fixed the unused build warning in asrc edma driver.

[2.0.0]

- Initial version.
-

ASRC EDMA Driver

[2.2.0]

- Bug Fixes
 - Fixed the “watermark” and “channel” was defined in struct asrc_p2p_edma_config_t but never used issue.

[2.1.0]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 10.3, 10.4, 14.4, 10.1, 17.7, 11.9, 8.6, 12.2, 11.6.

[2.0.1]

- Improvements
 - Added feature macro FSL_FEATURE_ASRC_PARAMETER_REGISTER_NAME_ASPRM for ASRC parameter register.
- Bug Fixes
 - Fixed the unused build warning in asrc edma driver.

[2.0.0]

- Initial version.
-

CAAM**[2.4.0]**

- Add new APIs for native asymmetric operations (RSA, ECC) instead of only accelerating mathematical primitives and support for black keys and blobs for both symmetric and asymmetric operations.

[2.3.2]

- Fix MISRA-2012 issues.

[2.3.1]

- Modified function caam_aes_ccm_check_input_args() to allow payload be empty as is specified in NIST800-38C Section 5.3..

[2.3.0]

- Add support for SHA HMAC.

[2.2.4]

- Fix issue where the outputSize parameter of CAAM_HASH_Finish() has impact on hash calculation.

[2.2.3]

- Fix DCACHE invalidation in CAAM_HASH_Finish().

[2.2.2]

- Modify RNG to not reseed with each request.

[2.2.1]

- Fixed AES-CCM decrypt failing with TAG length bigger than 8 byte.

[2.2.0]

- Added API for Blob functions and CRC

[2.1.6]

- Improve DCACHE handling. Requires CAAM used and cached memory set in write-through mode.

[2.1.5]

- Support EXTENDED data size for all AES, HASH and RNG operations.
- Support multiple De-Initialization/Initialization of CAAM driver within one POR event.

[2.1.4]

- Fix MISRA-2012 issues.

[2.1.3]

- Fix MISRA-2012 issues.

[2.1.2]

- Add data offset feature to provide support for mirrored (high-speed) memory.

[2.1.1]

- Add DCACHE support.

[2.1.0]

- Add return codes check and handling.

[2.0.3]

- Use MACRO instead of numbers in descriptor.
- Correct descriptor size mask.

[2.0.2]

- Add Data and Instruction Synchronization Barrier in `caam_input_ring_set_jobs_added()` to make sure that the descriptor will be loaded into CAAM correctly.

[2.0.1]

- Add Job Ring 2 and 3.

[2.0.0]

- Initial version.

CACHE LMEM

[2.1.0]

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

[2.1.0]

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

[2.0.6]

- Bug Fixes
 - Fixed doxygen issue.

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

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.
-

CDOG

[2.1.3]

- Re-design multiple instance IRQs and Clocks
- Add fix for RESTART command errata

[2.1.2]

- Support multiple IRQs
- Fix default CONTROL values

[2.1.1]

- Remove bit CONTROL[CONTROL_CTRL].

[2.1.0]

- Rename CWT to CDOG.

[2.0.2]

- Fix MISRA-2012 issues.

[2.0.1]

- Fix doxygen issues.

[2.0.0]

- Initial version.
-

CLOCK**[2.2.0]**

- New Features
 - Added APIs to set/get CLK01/O2.

[2.1.6]

- Bug Fixes
 - Fix an issue in CLOCK_InitArmPll() of wrong bitmask used

[2.1.5]

- Bug Fixes
 - Fix clock_pll_post_div_t value.

[2.1.4]

- Improvements
 - Move s_clockSourceName array to c from header.

[2.1.3]

- Improvements
 - Toggle hold_ring_off during arm pll initialization.

[2.1.2]

- Bug Fixes
 - Fixed bug in XBARA_CLOCKS macro define.
 - Fixed bug in CLOCK_InitSysPll1() function.

[2.1.1]

- Bug Fixes
 - Fixed bug in CLOCK_InitArmPll() function.
 - Fixed bug clock root divider set to cut off at 255.

[2.1.0]

- New Features
 - Added CLOCK_DeinitPfd() function.
- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.4.
 - Fixed bug in XBARA_CLOCKS macro define.

[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.
-

CSI

[2.2.0]

- Improvements
 - Update driver to invoke callback whenever there is a full frame received.

[2.1.5]

- Improvements
 - Updated for new CSI register and macro names.

[2.1.4]

- Improvements
 - Added memory address conversion to support buffers which could only be accessed using alias address by non-core masters.

[2.1.3]

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

[2.1.2]

- Improvements
 - Supported new CSI_Type register naming.

[2.1.1]

- Bug Fixes
 - Fixed IAR build warning Pa082.
 - Fixed violations of the MISRA C-2012 rules 8.4, 10.1, 10.3, 10.4, 10.6, 11.6, 14.4, 17.7.

[2.1.0]

- New Features
 - Added 16-bit and 24-bit data bus support.
- Bug Fixes:
 - Fixed the bug that CSI writes to wrong buffer when empty buffer not submitted in time.

[2.0.3]

- Bug Fixes
 - Fixed wrong circular queue delta calculation.
 - Fixed double buffering capture issue where, when the transfer is ongoing and the device has empty buffer slot, the function CSI_TransferSubmitEmptyBuffer sets the empty buffer to CSI device.

[2.0.2]

- New Features
 - Added fragment mode support.

[2.0.1]

- Improvements
 - Switched DMA output buffer at the first data after each VSYNC. It originally happened when the DMA transfer was done.

[2.0.0]

- Initial version.
-

DAC12**[2.1.2]**

- Bug Fixes
 - Fixed CERT INT31-C issue.

[2.1.1]

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

[2.1.0]

- Improvements
 - Defined the macro “FSL_FEATURE_HAS_NO_ITRM_REGISTER” to distinguish different scenes that ITRM register may not equipped one some devices.

[2.0.1]

- Bug Fixes
 - Fixed the violations of MISRA C-2012 rules:
 - * Rule 10.8, 17.7.

[2.0.0]

- Initial version.
-

DCDC**[2.1.2]**

- Improvements
 - The DCDC_GetInstance() function is only available when FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL is set to 0.

[2.1.1]

- Bug Fixes
 - Fixed Doxygen warnings.

[2.1.0]

- Improvements
 - Updated DCDC_BootIntoDCM() function.
 - Based on the updates of header file, updated dcdc driver.
- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.3, rule 10.7, and rule 12.2.

[2.0.0]

- Initial version.
-

DCIC

[2.0.2]

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

[2.0.1]

- Bug Fixes
 - Fixed the violations of MISRA 2012 rules: 10.1, 10.4.

[2.0.0]

- Initial version.
-

DMAMUX

[2.1.3]

- Improvements
 - Wrap DMAMUX_GetInstance into FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL to avoid build issues.

[2.1.2]

- Bug Fixes
 - Add macro FSL_DMAMUX_CHANNEL_NUM to calculate correct DMAMUX channel number when input EDAM channel number.

[2.1.1]

- Improvements
 - Add macro `FSL_FEATURE_DMAMUX_CHANNEL_NEEDS_ENDIAN_CONVERT` and `DMAMUX_CHANNEL_ENDIAN_CONVERTn` to do channel endian convert.

[2.1.0]

- Improvements
 - Modify the type of parameter source from `uint32_t` to `int32_t` in the `DMA_MUX_SetSource`.

[2.0.5]

- Improvements
 - Added feature `FSL_FEATURE_DMAMUX_CHCFG_REGISTER_WIDTH` for the difference of CHCFG register width.

[2.0.4]

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

[2.0.3]

- Bug Fixes
 - Fixed the issue for MISRA-2012 check.
 - * Fixed rule 10.4 and rule 10.3.

[2.0.2]

- New Features
 - Added an always-on enable feature to a DMA channel for ULP1 DMAMUX support.

[2.0.1]

- Bug Fixes
 - Fixed the build warning issue by changing the type of parameter source from `uint8_t` to `uint32_t` when setting DMA request source in `DMAMUX_SetSourceChange`.

[2.0.0]

- Initial version.
-

EDMA**[2.4.7]**

- Bug Fixes
 - Fixed coverity MSG issues with CERT INT31-C compliance.

[2.4.6]

- Bug Fixes
 - Fixed the EDMA header index retrieval error caused by done bit calculation mistake issue.

[2.4.5]

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

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

[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.
-

ELCDIF**[2.1.0]**

- New Features
 - Added API ELCDIF_SetPixelComponentOrder to support configure pixel component order.

[2.0.7]

- Bug Fixes
 - Fixed faulty operation of CTRL1 in ELCDIF_RgbModeSetPixelFormat.

[2.0.6]

- Bug Fixes
 - Fixed bug in ELCDIF_RgbModeStop that the API shall return until RUN bit is cleared, so that the RGB mode is properly stopped.

[2.0.5]

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

[2.0.4]

- Improvements
 - Increase outstanding transactions for better performance.
 - Added memory address conversion to support buffers which could only be accessed using alias address by non-core masters.

[2.0.3]

- Improvements
 - Supported the platforms which don't have PXP handshake feature.
- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 17.7.

[2.0.2]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 3.1, 8.4, 10.1, 10.6, 10.7, 10.8, 14.4, 17.7
 - Removed hardcode delay in function ELCDIF_Reset.

[2.0.1]

- Improvements
 - Added the function ELCDIF_RgbModeSetPixelFormat.

[2.0.0]

- Initial version.
-

ENC

[2.2.1]

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

[2.2.0]

- New Features
 - Supported input filter prescaler.

[2.1.0]

- Improvements
 - Supported period measurement function.

[2.0.2]

- Improvements
 - Added feature macro for CTRL2[SABIE] and CTRL2[SABIRQ] bits.

[2.0.1]

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

[2.0.0]

- Initial version.
-

ENET

[2.11.0]

- New Features
 - Added function ENET_Ptp1588JumpTimer which adjusts the ENET PTP 1588 timer by jumping a relative time difference. Compared to ENET_Ptp1588SetTimer, this function yields more accurate results when the relative time difference between the PTP clock and the target clock is known.

[2.10.1]

- Bug Fixes
 - Fixed WAKEUP interrupt not being handled.

[2.10.0]

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

[2.9.3]

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

[2.9.2]

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

[2.9.1]

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

[2.9.0]

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

[2.8.0]

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

[2.7.1]

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

[2.7.0]

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

[2.6.3]

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

[2.6.2]

- Improvements
 - Changed ENET1_MAC0_Rx_Tx_Done0_DriverIRQHandler/ENET1_MAC0_Rx_Tx_Done1_DriverIRQHandler to ENET1_MAC0_Rx_Tx_Done1_DriverIRQHandler/ENET1_MAC0_Rx_Tx_Done2_DriverIRQHandler which represent ring 1 and ring 2.

[2.6.1]

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

[2.6.0]

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

[2.5.4]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

- Bug Fixes
 - Fixed document typo.

[2.5.0]

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

[2.4.3]

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

[2.4.2]

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

[2.4.1]

- Bug Fixes
 - Fixed the PTP time capture issue.

[2.4.0]

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

[2.3.4]

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

[2.3.3]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

- Bug Fixes
 - Fixed the issue that clause 45 MDIO read/write API doesn't check the transmission over status between two transmissions.
 - Fixed violations of the MISRA C-2012 rules 2.2,10.3,10.4,10.7,11.6,11.8,13.5,14.4,15.7,17.7.
- New Features
 - Added APIs to support send/receive frame with Zero-Copy.

- Improvements
 - Separated the clock configuration from module configuration when init and deinit.
 - Added functions to set second level interrupt handler.
 - Provided new function to get 1588 timer count without disabling interrupt.
 - Improved timestamp controlling, deleted all old timestamp management APIs and data structures.
 - Merged the single/multiple ring(s) APIs, now these APIs can handle both.
 - Used base and index to control buffer descriptor, aligned with qos and lpc enet driver.

[2.2.6]

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

[2.2.5]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.4, 10.6, 10.7, 11.6, 11.9, 13.5, 14.4, 16.4, 17.7, 21.15, 3.1, 8.4.
 - Changed to use ARRAY_SIZE(s_enetBases) as the array size for s_ENETHandle, fixed the hardfault issue for using some ENET instance when ARRAY_SIZE(s_enetBases) is not same as FSL_FEATURE_SOC_ENET_COUNT.

[2.2.4]

- Improvements
 - Added call to Data Synchronization Barrier instruction before activating Tx/Rx buffer descriptor to ensure previous data update is completed.
 - Improved ENET_TransmitIRQHandler to store timestamps for multiple transmit buffer descriptors.
- Bug Fixes
 - Fixed the issue that ENET_Ptp1588GetTimer did not handle the timer wrap situation.

[2.2.3]

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

[2.2.2]

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

[2.2.1]

- Improvements
 - Changed the input data pointer attribute to const in ENET_SendFrame().

[2.1.1]

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

[2.0.1]

- Bug Fixes
 - Used direct transmit busy check when doing data transmit.
- Miscellaneous Changes
 - Updated IRQ handler work flow.
 - Changed the TX/RX interrupt macro from kENET_RxByteInterrupt to kENET_RxBufferInterrupt, from kENET_TxByteInterrupt to kENET_TxBufferInterrupt.
 - Deleted unnecessary parameters in ENET handler.

[2.0.0]

- Initial version.
-

EWM**[2.0.4]**

- Bug Fixes
 - Fixed CERT INT31-C violations.

[2.0.3]

- Bug Fixes
 - Fixed violation of MISRA C-2012 rules: 10.1, 10.3.

[2.0.2]

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

[2.0.1]

- Bug Fixes
 - Fixed the hard fault in EWM_Deinit.

[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 ERFCR[NFE] bits to the correct value.
 - Fixed the issue that FLEXCAN_ReceiveFifoEDMACallback() can't differentiate between Rx fifo and enhanced rx fifo.
 - Fixed the issue that FLEXCAN_TransferHandleIRQ() can't report Legacy Rx FIFO warning status.

[2.9.0]

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

[2.8.7]

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

[2.8.6]

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

[2.8.5]

- Improvements
 - Make driver aarch64 compatible.

[2.8.4]

- Bug Fixes
 - Fixed FlexCan_Errata_6032 to disable all interrupts.

[2.8.3]

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

[2.8.2]

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

[2.8.1]

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

[2.8.0]

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

[2.7.1]

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

[2.7.0]

- Improvements
 - Update config configuration.
 - * Added enablePretendedNetworking member to support enable/disable Pretended Networking feature.
 - * Added enableTransceiverDelayMeasure member to support enable/disable Transceiver Delay 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 evkmimxrt1064.
- Fixed the issue of setting CANFD Bit Rate Switch.
- Fixed the issue of operating unknown pointer risk.
 - * when used the pointer “handle->mbFrameBuf[mbIdx]” to update the timestamp in a short-live TX frame, the frame pointer became as unknown, the action of operating it would result in program stack destroyed.
- Added assert to check current CAN clock source affected by other clock gates in current device.
 - * In some chips, CAN clock sources could be selected by CCM. But for some clock sources affected by other clock gates, if user insisted on using that clock source, they had to open these gates at the same time. However, they should take into consideration the power consumption issue at system level. In RT10xx chips, CAN clock source 2 was affected by the clock gate of lpuart1. ERRATA ID: (ERR050235 in CCM).
- Improvements
 - Implementation for new FLEXCAN with ECC feature able to exit Freeze mode.
 - Optimized the function of interruption handler.
 - Added two APIs for FLEXCAN EDMA driver.
 - * FLEXCAN_PrepareTransfConfiguration
 - * FLEXCAN_StartTransferDatafromRxFIFO
 - Added new API for FLEXCAN driver.
 - * FLEXCAN_GetTimeStamp
 - For TX non-blocking API, we wrote the frame into mailbox only, so no need to register TX frame address to the pointer, and the timestamp could be updated into the new global variable handle->timestamp[mbIdx], the FLEXCAN driver provided a new API for user to get it by handle and index number after TX DONE Success.
 - * FLEXCAN_EnterFreezeMode
 - * FLEXCAN_ExitFreezeMode
 - Added new configuration for user.
 - * disableSelfReception
 - * enableListenOnlyMode
 - Renamed the two clock source enum macros based on CLKSRC bit field value directly.
 - * The CLKSRC bit value had no property about Oscillator or Peripheral type in lots of devices, it acted as two different clock input source only, but the legacy enum macros name contained such property, that misled user to select incorrect CAN clock source.
 - Created two new enum macros for the FLEXCAN driver.
 - * kFLEXCAN_ClkSrc0
 - * kFLEXCAN_ClkSrc1
 - Deprecated two legacy enum macros for the FLEXCAN driver.
 - * kFLEXCAN_ClkSrcOsc

- * kFLEXCAN_ClkSrcPeri
- Changed the process flow for Remote request frame response..
 - * Created a new enum macro for the FLEXCAN driver.
 - kStatus_FLEXCAN_RxRemote
- Changed the process flow for kFLEXCAN_StateRxRemote state in the interrupt handler.
 - * Should the TX frame not register to the pointer of frame handle, interrupt handler would not be able to read the remote response frame from the mail box to ram, so user should read the frame by manual from mail box after a complete remote frame transfer.

[2.4.0]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.3]

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

[2.2.2]

- Improvements
 - Added a time stamp feature and enable it in the interrupt_transfer example.

[2.2.1]

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

[2.2.0]

- Improvements
 - Added FSL_FEATURE_FLEXCAN_HAS_SUPPORT_ENGINE_CLK_SEL_REMOVE feature to support SoCs without CAN Engine Clock selection in FlexCAN module.
 - Added FlexCAN Serial Clock Operation to support i.MX SoCs.

[2.1.0]

- Bug Fixes
 - Corrected the spelling error in the function name FLEXCAN_XXX().
 - Moved Freeze Enable/Disable setting from FLEXCAN_Enter/ExitFreezeMode() to FLEXCAN_Init().
 - Corrected wrong helper macro values.
- Improvements
 - Hid FLEXCAN_Reset() from user.
 - Used NDEBUB 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
-

FLEXRAM

[2.3.0]

- New Features
 - Supported platforms which have ECC but no ECC error injection.

[2.2.0]

- New Features
 - Supported flexram ECC error injection function.

[2.1.0]

- New Features
 - Supported flexram ECC function.

[2.0.7]

- Bug Fixes
 - Fixed doxygen issue.

[2.0.6]

- New Features
 - Updated bank configuration and TCM size with GPR16/GPR17/GPR18 into SOC level for different SOC.

[2.0.5]

- New Features
 - Added the magic address feature for OCRAM, DTCM and ITCM.

[2.0.4]

- Bug Fixes
 - Fixed FlexRAM driver's missing extern C around functions in header file.
 - Removed magic address feature from driver.

[2.0.3]

- Bug Fixes
 - Fixed the issue that TCM size configuration was wrong when TCM bank number was not a value power of 2.

[2.0.2]

- Bug Fixes
 - Updated driver due to Reference Manual update.

[2.0.1]

- Bug Fixes
 - Fixed MISRA issue.

[2.0.0]

- Initial version.
-

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_COMBINATION.

[2.3.3]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

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

[2.1.0]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

FLEXSPI EDMA Driver

[2.3.3]

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

[2.3.2]

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

[2.0.2]

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

[2.0.0]

- Initial version.
-

GPC

[2.5.0]

- Improvements
 - Set GPC_CM_ConfigCpuModeTransitionStep(), GPC_SP_ConfigSetPointTransitionStep(), GPC_STBY_ConfigStandbyTransitionStep() as deprecated.
 - Added GPC_CM_EnableCpuModeTransitionStep(), GPC_CM_DisableCpuModeTransitionStep().
 - Added GPC_SP_EnableSetPointTransitionStep(), GPC_SP_DisableSetPointTransitionStep().
 - Added GPC_STBY_EnableStandbyTransitionStep(), GPC_STBY_DisableStandbyTransitionStep().
 - Added GPC_STBY_SetPmicOutStepCountMode() to set count mode of PMIC_OUT step.

[2.4.0]

- Improvements
 - Deleted cnt_mode and step count of gpc_tran_step_config_t structure to aligned with updates of header file.

[2.3.1]

- Bug Fixes
 - Fixed the violation of MISRA C-2012 rule 5.8.

[2.3.0]

- Bug Fixes
 - Fixed wrong offset value of DCDC_UP_CTRL register.
- New Features
 - Added GPC_STBY_ForceCoreRequestStandbyMode() function to force core to enter standby mode.

[2.2.0]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.8.
 - Fixed violations of MISRA C-2012 rule 8.6 by removing the declaration of GPC_SP_GetResponseCount() function.

[2.1.1]

- Bug Fixes
 - Fixed Doxygen warnings.

[2.1.0]

- Improvements
 - Removed status related APIs based on the updates of header file.

[2.0.0]

- Initial version.
-

GPIO**[2.0.7]**

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

[2.0.6]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

GPT

[2.0.6]

- Bug Fixes
 - Fix CERT INT30-C issues.

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

IEE

[2.1.1]

- Fixed MISRA issues.

[2.1.0]

- Add region lock function IEE_LockRegionConfig() and driver clock control.

[2.0.0]

- Initial version.
-

IEE_APC

[2.0.2]

- Updated to newer version of implementation in HW.

[2.0.1]

- Fixed MISRA issues.

[2.0.0]

- Initial version.
-

IOMUXC

[2.0.1]

- Doxygen improvement.

[2.0.0]

- initial version.
-

KEYMGR

[2.0.2]

- Fix MISRA-2012 issues.

[2.0.1]

- Fix MISRA-2012 issues.

[2.0.0]

- Initial version.
-

KPP

[2.1.1]

- Bug Fixes
 - Fixed coverity MSG issues with CERT INT30-C, CERT ARR30-C compliance.

[2.1.0]

- Improvements
 - Optimize rowData debounce method to adapt to multi-key detection
 - Modify the KPP_keyPressScanning type to status_t.

[2.0.1]

- Bug Fixes
 - Fixed the violations of MISRA 2012 rules:
 - * Rule 10.3 10.4 10.6 14.4 17.7

[2.0.0]

- Initial version.
-

LCDIFv2**[2.3.3]**

- Other Changes
 - Removed PDI_PARA register operation due to IP change.

[2.3.2]

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

[2.3.1]

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

[2.3.0]

- New Features:
 - Added API to calculate global alpha based on desired blended alpha.

[2.2.3]

- Improvements
 - Added memory address conversion to support buffers which could only be accessed using alias address by non-core masters.

[2.2.2]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 10.1, 10.2, 10.4, 10.6, 12.2.

[2.2.1]

- Improvements
 - Updated for the new LCDIFV2_Type structure.

[2.2.0]

- Bug Fixes
 - Fixed LCDIFV2_GetPorterDuffConfig issue that does not set color mode correctly.
- Other Changes
 - Removed the store functions.

[2.1.1]

- Bug Fixes
 - Fixed the issue that LCDIFV2_SetLut could not access the last index.

[2.1.0]

- New Features:
 - Added function to get Porter Duff configuration.

[2.0.1]

- Bug Fixes
 - Fixed the issue that register value not reset by LCDIFV2_Deinit and LCDIFV2_Reset.

[2.0.0]

- Initial version.
-

LPADC

[2.9.4]

- Improvements
 - Update LPADC_GetDefaultConfig, change default conversionAverageMode value to: kLPADC_ConversionAverage128 for 3 bit width. kLPADC_ConversionAverage1024 for 4 bit width.

[2.9.3]

- Improvements
 - Add timeout for while loop code.

[2.9.2]

- Improvements
 - Fixed CERT-C issues.

[2.9.1]

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

[2.9.0]

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

[2.8.4]

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

[2.8.3]

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

[2.8.2]

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

[2.8.1]

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

[2.8.0]

- Improvements
 - Release peripheral from reset if necessary in init function.
- Bug Fixes
 - Fixed function LPADC_GetConvResult() issue.
 - Fixed function LPADC_SetConvCommandConfig() bugs.

[2.7.2]

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

[2.7.1]

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

[2.7.0]

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

[2.6.2]

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

[2.6.1]

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

[2.6.0]

- Improvements
 - Added the API LPADC_SetOffset12BitValue() to configure 12bit ADC conversion offset trim value manually.
 - Added the API LPADC_SetOffset16BitValue() to configure 16bit ADC conversion offset trim value manually.
 - Added API to set offset calibration mode.
 - Added configuration of alternate channel.
 - Updated auto calibration API and added calibration value conversion API.
- New feature
 - Added API LPADC_EnableHardwareTriggerCommandSelection() to enable trigger commands controlled by ADC_ETC.
 - Updated LPADC_DoAutoCalibration() to allow doing something else before the ADC initialization to be totally complete. Enhance initialization duration time of the ADC.
 - Added two new APIs to get/set calibration value.

[2.5.2]

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

[2.5.1]

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

[2.5.0]

- Improvements
 - Added missing items to enable trigger interrupts.

[2.4.0]

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

[2.3.0]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

- Improvements
 - Updated the gain calibration formula.
 - Used feature to segregate the new item kLPADC_TriggerPriorityPreemptSubsequently.

[2.1.0]

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

[2.0.2]

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

[2.0.1]

- Bug Fixes
 - Ensured the API LPADC_SetConvCommandConfig configure related registers correctly.

[2.0.0]

- Initial version.
-

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
-

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_LPSP_I_HAS_NO_PCSCFG and FSL_FEATURE_LPSP_I_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 LPSPIIn_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.
-

LPUART**[2.10.0]**

- New Feature
 - Added support to configure RTS watermark.

[2.9.4]

- Improvements
 - Merged duplicate code.

[2.9.3]

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

[2.9.2]

- Bug Fixes
 - Fixed coverity issues.

[2.9.1]

- Bug Fixes
 - Fixed coverity issues.

[2.9.0]

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

[2.8.3]

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

[2.8.2]

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

[2.8.1]

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

[2.8.0]

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

[2.7.7]

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

[2.7.6]

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

[2.7.5]

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

[2.7.4]

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

[2.7.3]

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

[2.7.2]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

- Improvements
 - Use separate data for TX and RX in lpuart_transfer_t.
- Bug Fixes

- Fixed bug that when ring buffer is used, if some data is received in ring buffer first before calling `LPUART_TransferReceiveNonBlocking`, the received data count returned by `LPUART_TransferGetReceiveCount` is wrong.

[2.5.0]

- Bug Fixes
 - Added missing interrupt enable masks `kLPUART_Match1InterruptEnable` and `kLPUART_Match2InterruptEnable`.
 - Fixed bug in `LPUART_EnableInterrupts`, `LPUART_DisableInterrupts` and `LPUART_GetEnabledInterrupts` that the `BAUD[LBKDIE]` bit field should be soc specific.
 - Fixed bug in `LPUART_TransferHandleIRQ` that idle line interrupt should be disabled when rx data size is zero.
 - Deleted unused status flags `kLPUART_NoiseErrorInRxDataRegFlag` and `kLPUART_ParityErrorInRxDataRegFlag`, since firstly their function are the same as `kLPUART_NoiseErrorFlag` and `kLPUART_ParityErrorFlag`, secondly to obtain them one data word must be read out thus interfering with the receiving process.
 - Fixed bug in `LPUART_GetStatusFlags` that the `STAT[LBKDIF]`, `STAT[MA1F]` and `STAT[MA2F]` should be soc specific.
 - Fixed bug in `LPUART_ClearStatusFlags` that tx/rx FIFO is reset by mistake when clearing flags.
 - Fixed bug in `LPUART_TransferHandleIRQ` that while clearing idle line flag the other bits should be masked in case other status bits be cleared by accident.
 - Fixed bug of race condition during LPUART transfer using transactional APIs, by disabling and re-enabling the global interrupt before and after critical operations on interrupt enable register.
 - Fixed DMA/eDMA transfer blocking issue by enabling tx idle interrupt after DMA/eDMA transmission finishes.
- New Features
 - Added APIs `LPUART_GetRxFifoCount/LPUART_GetTxFifoCount` to get rx/tx FIFO data count.
 - Added APIs `LPUART_SetRxFifoWatermark/LPUART_SetTxFifoWatermark` to set rx/tx FIFO water mark.

[2.4.1]

- Bug Fixes
 - Fixed MISRA advisory 17.7 issues.

[2.4.0]

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

[2.3.1]

- Bug Fixes
 - Fixed MISRA advisory 15.5 issues.

[2.3.0]

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

[2.2.8]

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

[2.2.7]

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

[2.2.6]

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

[2.2.5]

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

[2.2.4]

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

[2.2.3]

- Improvements
 - Changed parameter type in LPUART_RTOS_Init struct from `rtos_lpuart_config` to `lpuart_rtos_config_t`.
- Bug Fixes
 - Disabled LPUART receive interrupt instead of all NVICs when reading data from ring buffer. Otherwise when the ring buffer is used, receive nonblocking method will disable all NVICs to protect the ring buffer. This may 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.
-

MECC**[2.1.1]**

- Bug fixes:
 - Add volatile to variable counter to fix armgcc 13.2.1 -Os optimization issue.

[2.1.0]

- Bug fixes:
 - Removed Ocram1StartAddress, Ocram1EndAddress, Ocram2StartAddress, Ocram2EndAddress in mecc_config_t structure. Use startAddress and endAddress as instead.
 - Removed static function MECC_GetInstance().
- New Features:
 - Added new function MECC_GetPendingFlags().
 - Added new members: enableReadDataWait, enableReadAddrPipeline, enableWriteDataPipeline, enableWriteAddrPipeline in mecc_config_t structure to support pipeline features.

[2.0.2]

- Bug fixes:
 - Fixed MISRA 2012 issue: 10.3, 10.4.

[2.0.1]

- Bug fixes:
 - Fixed MISRA 2012 issue: 10.1, 10.3, 10.4, 10.6.

[2.0.0]

- Initial version.
-

MIPI CSI2RX

[2.0.4]

- Improvements
 - Updated for new format MIPI_CSI2RX_Type definition.

[2.0.3]

- Bug Fixes
 - Fixed the violations of MISRA 2012 rules: 3.1, 10.3, 10.4, 10.8, 17.7.

[2.0.2]

- Improvements
 - Updated to support MIMX8QX C0 header file.

[2.0.1]

- Improvements
 - Updated to support platforms that don't have dedicated MIPI CSI2RX CSR.
- Bug Fixes
 - Fixed the issue that the register bit PRG_RXHS_SETTLE set to wrong value.

[2.0.0]

- Initial version.
-

MIPI_DSI

[2.3.0]

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

[2.2.5]

- Bug Fixes
 - Fixed issue that VACTIVE setting shall equal to the number of active lines (height), no need to minus 1.

[2.2.4]

- Bug Fixes
 - Updated the DPI setting to use float for coefficient value for more accurate calculation.

[2.2.3]

- Bug Fixes
 - Fixed the DSI_TransferNonBlocking no interrupt issue.
 - Fixed the violations of MISRA 2012 advisory rules.

[2.2.2]

- Bug Fixes
 - Fixed the DPI horizontal timing setting issue.
 - Fixed MISRA issue

[2.2.1]

- Bug Fixes
 - Fixed the bug that runs to hardfault when sending long packet with 4-byte unaligned address.

[2.2.1]

- Improvements
 - Supported long package read.

[2.2.0]

- Improvements
 - Change parameter MIPI_DSI_Type pointer to const type.

[2.1.0]

- Initial version.
-

MU

[2.3.1]

- Bug Fixes
 - Fixed FSL_FEATURE_MU_HAS_RESET_DEASSERT_INT macro use.

[2.3.0]

- New Features
 - Added MU_BUSY_POLL_COUNT parameter to prevent infinite polling loops in MU operations.
 - Added timeout mechanism to all polling loops in MU driver code.
 - Added new function MU_ReceiveMsgTimeout() to include timeout mechanism.
- Improvements
 - Updated function signatures to return status codes for better error handling:
 - * Changed MU_ResetBothSides to return status_t instead of void
 - * Updated MU_SendMsg to return status_t for timeout indication
 - * Updated MU_ReceiveMsg to use MU_TIMEOUT_VALUE (0xFFFFFFFF) as a special return value to indicate timeout
 - Enhanced documentation across all functions to clarify timeout behavior and return values.

[2.2.0]

- New Features
 - Added API MU_GetRxStatusFlags.

[2.1.3]

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

[2.1.2]

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

[2.1.1]

- Bug Fixes
 - Fixed general interrupt comment typo.

[2.1.0]

- Improvements
 - Added new enum `mu_msg_reg_index_t`.

[2.0.7]

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

[2.0.6]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

- Improvements
 - Added support for MIMX8MQx.

[2.0.1]

- Improvements
 - Added support for MCIMX7Ux_M4.

[2.0.0]

- Initial version.
-

NIC301

[2.0.1]

- Bug Fixes.
 - Fixed the repeat of offset addition in this file API.

[2.0.0]

- Initial version.
-

OCOTP

[2.1.4]

- Bug fixes
 - Fixed the bug that OCOTP_ReadFuseShadowRegisterExt can't read more than one word.

[2.1.3]

- Bug fixes
 - Fixed MISRA 2012 issue: 8.4, 10.3, 10.4, 14.3.
 - Fixed doxygen warning.

[2.1.2]

- Improvements
 - Updated for new MIMXRT117X header file.

[2.1.1]

- Improvements
 - Updated OCOTP_ReloadShadowRegister to return error status.
 - Added functions OCOTP_ReadFuseShadowRegisterExt and OCOTP_WriteFuseShadowRegisterWithLock.
- Bug fixes
 - Fixed MISRA 2012 rule 10.3 issue.

[2.0.1]

- Bug Fixes
 - Fixed doxygen issues.

[2.0.0]

- Initial version.
-

OTFAD

[2.1.4]

- Bug fixes
 - Fixed MISRA 2012 issue: 10.1.

[2.1.3]

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

[2.1.2]

- Bug fixes
 - Fixed MISRA 2012 issue: 10.4.

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.
-

PDM

[2.9.3]

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

[2.9.2]

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

[2.9.1]

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

[2.9.0]

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

[2.8.1]

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

[2.8.0]

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

[2.7.4]

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

[2.7.3]

- Improvements
- Added feature FSL_FEATURE_PDM_HAS_NO_VADEF to remove the support of VADEF bit-field in VAD0_STAT register.

[2.7.2]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

- Improvements
 - Added new features FSL_FEATURE_PDM_HAS_STATUS_LOW_FREQ/FSL_FEATURE_PDM_HAS_DC_OUT

[2.5.0]

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

[2.4.1]

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

[2.4.0]

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

[2.3.0]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

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

[2.1.0]

- Improvements
 - Added software buffer queue for transactional API.

[2.0.1]

- Improvements
 - Improved HWVAD feature.

[2.0.0]

- Initial version.
-

PDM_EDMA

[2.6.5]

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

[2.6.4]

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

[2.6.3]

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

[2.6.2]

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

[2.6.1]

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

[2.6.0]

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

[2.5.0]

- Refer PDM driver change log 2.1.0 to 2.5.0
-

PGMC**[2.1.2]**

- Bug Fixes
 - Fixed bug in PGMC_PPC_TriggerPMICStandbySoftMode() function.

[2.1.1]

- Bug Fixes
 - Fixed Doxygen warnings.

[2.1.0]

- Improvements
 - Updated PGMC driver based on the updates of header file.

[2.0.0]

- Initial version.
-

PIT**[2.2.0]**

- Bug Fixes
 - According to ERR050763, PIT_LDVAL_STAT register is not reliable in dynamic load mode, so remove the status check in PIT_SetRtiTimerPeriod which added since 2.1.1.
 - Removed not used bit PIT_RTI_TCTRL_CHN_MASK.
- Improvements
 - Added more guide about get RTI load status in PIT_SetRtiTimerPeriod's API comment.

- Change PIT_RTI_Deinit to inline API.
- Ensure PIT peripheral clock enabled in PIT_RTI_Init.
- New Features
 - Added PIT_ClearRtiSyncStatus API to clear the RTI_LDVAL_STAT register.

[2.1.1]

- Bug Fixes
 - Enable PIT when using RTI to ensure RTI can work properly in debug mode.
- Improvements
 - Added status check in PIT_SetRtiTimerPeriod to ensure the load value is synchronized into the RTI clock domain.
 - Added note for PIT_RTI_Init to remind users wait RTI sync.

[2.1.0]

- New Features
 - Support RTI (Real Time Interrupt) timer.

[2.0.5]

- Improvements
 - Support workaround for ERR007914. This workaround guarantee the write to MCR register is not ignored.

[2.0.4]

- Bug Fixes
 - Fixed PIT_SetTimerPeriod implementation, the load value trigger should be PIT clock cycles minus 1.

[2.0.3]

- Bug Fixes
 - Clear all status bits for all channels to make sure the status of all TCTRL registers is clean.

[2.0.2]

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

[2.0.1]

- Bug Fixes
 - Cleared timer enable bit for all channels in function PIT_Init() to make sure all channels stay in disable status before setting other configurations.
 - Fixed MISRA-2012 rules.
 - * Rule 14.4, rule 10.4.

[2.0.0]

- Initial version.
-

PMU**[2.1.2]**

- Bug Fixes
 - Updated PMU_StaticEnablePllLdo() with disabling LDO current limit after LDO is stable to minimize ARM PLL jitter in cold temperature.

[2.1.1]

- Bug Fixes
 - Fixed bugs in FBB configuration.
 - Updated delay value from 1us to 100us in PMU_StaticEnablePllLdo() function.

[2.1.0]

- Improvements
 - Updated the PMU driver based on the new header file.
 - Defined the macro to separate different scenes that some devices may do not support FBB.
 - Fixed Doxygen warnings.
 - Fixed violations of MISRA C-2012 rule 14.3.
- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 13.1, rule 10.1, rule 10.4, and rule 14.3.

[2.0.0]

- Initial version.
-

PUF**[2.2.0]**

- Add support for kPUF_KeySlot4.
- Add new PUF_ClearKey() function, that clears a desired PUF internal HW key register.

[2.1.6]

- Changed wait time in PUF_Init(), when initialization fails it will try PUF_Powercycle() with shorter time. If this shorter time will also fail, initialization will be tried with worst case time as before.

[2.1.5]

- Use common SDK delay in puf_wait_usec().

[2.1.4]

- Replace register uint32_t ticksCount with volatile uint32_t ticksCount in puf_wait_usec() to prevent optimization out delay loop.

[2.1.3]

- Fix MISRA C-2012 issue.

[2.1.2]

- Update: Add automatic big to little endian swap for user (pre-shared) keys destined to secret hardware bus (PUF key index 0).

[2.1.1]

- Fix ARMGCC build warning .

[2.1.0]

- Align driver with PUF SRAM controller registers on LPCXpresso55s16.
- Update initialization logic .

[2.0.3]

- Fix MISRA C-2012 issue.

[2.0.2]

- New feature:
 - Add PUF configuration structure and support for PUF SRAM controller.
- Improvements:
 - Remove magic constants.

[2.0.1]

- Bug Fixes:
 - Fixed puf_wait_usec function optimization issue.

[2.0.0]

- Initial version.
-

PWM**[2.9.1]**

- Improvements
 - Add new API `PWM_SetupFaultsExt` and `PWM_SetupFaultInputFilterExt` to support Flex-PWM which has more than one fault input channels.
 - Support fault 4-7 interrupt and its flag.
- Bug Fixes
 - Fixed violations of the CERT INT31-C.

[2.9.0]

- Improvements
 - Support PWMX channel output for edge aligned PWM.
 - Forbid submodule 0 counter initialize with master sync and master reload mode.
 - Clarify `kPWM_BusClock` meaning.
 - Verify `pulseCnt` within 65535 when update period register.

[2.8.4]

- Improvements
 - Support workaround for ERR051989. This function helps realize no phase delay between submodule 0 and other submodule.

[2.8.3]

- Bug Fixes
 - Fixed MISRA C-2012 Rule 15.7

[2.8.2]

- Bug Fixes
 - Fixed warning conversion from 'int' to 'uint16_t' on API `PWM_Init`.
 - Fixed warning unused variable 'reg' on API `PWM_SetPwmForceOutputToZero`.

[2.8.1]

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

[2.8.0]

- Improvements
 - Added API PWM_UpdatePwmPeriodAndDutycycle to update the PWM signal's period and dutycycle for a PWM submodule.
 - Added API PWM_SetPeriodRegister and PWM_SetDutycycleRegister to merge duplicate code in API PWM_SetupPwm, PWM_UpdatePwmDutycycleHighAccuracy and PWM_UpdatePwmPeriodAndDutycycle

[2.7.1]

- Improvements
 - Supported UPDATE_MASK bit in MASK register.

[2.7.0]

- Improvements
 - Supported platforms which don't have Capture feature with channel A and B.
 - Supported platforms which don't have Submodule 3.
 - Added assert function in API PWM_SetPhaseDelay to prevent wrong argument.

[2.6.1]

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

[2.6.0]

- Improvements
 - Added API PWM_SetPhaseDelay to set the phase delay from the master sync signal of submodule 0.
 - Added API PWM_SetFilterSampleCount to set number of consecutive samples that must agree prior to the input filter.
 - Added API PWM_SetFilterSamplePeriod to set the sampling period of the fault pin input filter.

[2.5.1]

- Bug Fixes
 - Fixed MISRA C-2012 rules: 10.1, 10.3, 10.4 , 10.6 and 10.8.
 - Fixed the issue that PWM_UpdatePwmDutycycle() can't update duty cycle status value correct.

[2.5.0]

- Improvements
 - Added API PWM_SetOutputToIdle to set pwm channel output to idle.
 - Added API PWM_GetPwmChannelState to get the pwm channel output duty cycle value.

- Added API PWM_SetPwmForceOutputToZero to set the pwm channel output to zero logic.
- Added API PWM_SetChannelOutput to set the pwm channel output state.
- Added API PWM_SetClockMode to set the value of the clock prescaler.
- Added API PWM_SetupPwmPhaseShift to set PWM which a special phase shift and 50% duty cycle.
- Added API PWM_SetVALxValue/PWM_GetVALxValue to set/get PWM VALs registers values directly.

[2.4.0]

- Improvements
 - Supported the PWM which can't work in wait mode.

[2.3.0]

- Improvements
 - Add PWM output enable&disbale API for SDK.
- Bug Fixes
 - Fixed changing channel B configuration when parameter is kPWM_PWMX and PWMX configuration is not supported yet.

[2.2.1]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rules: 10.3, 10.4.
- Bug Fixes
 - Fixed the issue that PWM drivers computed VAL1 improperly.
- Improvements
 - Updated calculation accuracy of reloadValue in dutyCycleToReloadValue function.

[2.2.0]

- Improvements
 - Added new enumeration and two APIs to support enabling and disabling one or more PWM output triggers.
 - Added a new function to make the most of 16-bit resolution PWM.
 - Added one API to support updating fault status of PWM output.
 - Added one API to support PWM DMA write request.
 - Added three APIs to support PWM DMA capture read request.
 - Added one API to support get default fault config of PWM.
 - Added one API to support setting PWM fault disable mapping.

[2.1.0]

- Improvements
 - Moved the configuration of fault input filter into a new API to avoid be initialized multiple times.
- Bug Fixes
 - MISRA C-2012 issue fixed.
 - * Fix rules, containing: rule-10.2, rule-10.3, rule-10.4, rule-10.7, rule-10.8, rule-14.4, rule-16.4.

[2.0.1]

- Bug Fixes
 - Fixed the issue that PWM submodule may be initialized twice in function PWM_SetupPwm().

[2.0.0]

- Initial version.
-

PXP

[2.7.0]

- New Features
 - Added the PS_LRC setting for V4.
 - Added the PXP_SetPath setting for V4.
 - Fixed the code logic, V4 do not support DATA_PATH_CTRL1.

[2.6.1]

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

[2.6.0]

- Bug Fixes
 - Added missing configuration option for fetch engine background value.
 - Fixed bug in PXP_SetStoreEngineConfig that the address increment for store mask is not linear.
 - Added channel arbitration configuration for fetch engine, channel combine for store engine.
 - Fixed wrong method of obtaining the store mask address.
 - Fixed wrong method of configuring flag shift mask/width which can only be written in word boundary.
 - Fixed wrong configurations of block store and pitch in PXP_SetStoreEngineConfig.
 - Fixed wrong method of obtaining cfaValue address and calculating word count.

- Fixed the channel word order cannot be updated when configuring the second channel.
- Fixed bugs in PXP_SetHistogramConfig of wrong method to obtain the store mask address and wrong access of 32-bit registers.

[2.5.0]

- New Features
 - Added new API PXP_GetPorterDuffConfigExt for flexible Porter-Duff configuration.
 - Added enumerations for new AS/PS pixel formats for certain SoCs.

[2.4.1]

- New Features
 - Added API PXP_ResetControl to reset the PXP and the control register to initialized state.

[2.4.0]

- New Features
 - Added the API PXP_BuildRect of building a solid rectangle of given pixel value.
 - Added the interrupt enable/disable and status mask for V3.
 - Added API PXP_EnableProcessEngine to enable/disable process engines for V3.
 - Added API PXP_SetHistogramSize to re-configure the histogram size for each update.
 - Updated PXP_WfeaInit and PXP_SetWfeaConfig according to header file's update of WFE related registers.
 - Updated PXP_WfeaInit to support handshake with upstream dither store engine and added API PXP_WfeaEnableDitherHandshake to enable/disable the feature.
 - Added API PXP_GetLutUsage to get the occupied LUT list.
 - Updated APIs to support alpha blending engine1.
 - Added the API PXP_MemCopy to support all memory size copy.
- Bug Fixes
 - Fixed wrong naming for mux16.
 - Fixed wrong naming for enumerations in pxp_scanline_burst_t.
 - Fixed bug in PXP_GetHistogramMatchResult since there are 2 histograms engines rather than 1.
 - Fixed bug in PXP_SetFetchEngineConfig that the fetch size should not be minus one coding.

[2.3.0]

- New Features
 - Added the configuration of fetch engine, store engine, pre-dither engine and histogram block.

[2.2.2]

- Improvements
 - Disable alpha surface (AS) in PXP_Init.

[2.2.1]

- Improvements
 - Added memory address conversion to support buffers which could only be accessed using alias address by non-core masters.

[2.2.0]

- Bug Fixes
 - Fixed Porter Duff configuration error.

[2.1.0]

- New Features
 - Added Porter Duff support.
 - Added APIs PXP_StartMemCopy and PXP_StartPictureCopy.
 - Added API PXP_SetProcessSurfaceYUVFormat.

[2.0.2]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 3.1, 10.8, 11.6, 12.2.

[2.0.1]

- Bug Fixes
 - Fixed the rotate function issue for i.MX 6ULL.

[2.0.0]

- Initial version.
-

QTMR

[2.3.0]

- Improvements
 - Support for platforms which QTMR registers are 32-bit.

[2.2.2]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rules: 10.1, 10.8.

[2.2.1]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rules: 10.1, 10.8.

[2.2.0]

- Improvements
 - Added API QTMR_SetPwmOutputToIdle to set the generated pwm signal to the configured idle value.
 - Added API QTMR_GetPwmOutputStatus to return the output status of the generated pwm signal.
 - Added API QTMR_GetPwmChannelStatus to return the channel dutycycle value.
 - Added API QTMR_SetPwmClockMode to set clock mode change peripheral clock frequency.
- Bug Fixes
 - Fixed the issue that pwm duty cycle could not be 0 and 100.

[2.1.0]

- Bug Fixes
 - Fixed the issue QTMR_SetTimerPeriod needs to decrement down count by 1, and added new APIs to configure the LOAD register, COMP register.

[2.0.2]

- Bug Fixes
 - Fixed the issue introduced by previous code correction for improving the output signal accuracy.

[2.0.1]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rules: 10.1, 10.3, 11.5, 11.9.
- Improvements
 - Improved the output signal accuracy.

[2.0.0]

- Initial version.
-

RDC**[2.2.0]**

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

[2.1.1]

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

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

RDC_SEMA42

[2.0.5]

- Bug Fixes
 - Fixed CERT INT30-C issues.

[2.0.4]

- Improvements
 - Changed to implement RDC_SEMAPHORE_Lock base on RDC_SEMAPHORE_TryLock.

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

ROMAPI

[1.1.2]

- New features
 - Support new silicon Rev

[1.1.1]

- Improvements
 - Update the comments of “clear cache” function.

[1.1.0]

- New features
 - Support B0 silicon

[1.0.0]

- initial version.
-

RTWDOG

[2.1.4]

- Bug Fixes
 - Fixed CERT INT30-C, INT31-C issue.
 - Make API RTWDOG_CountToMesec return 0 if result overflow.

[2.1.3]

- Improvements
 - Waited the over status after CS register operation in case next CS operation causes problem.

[2.1.2]

- Bug Fixes
 - Fixed doxygen issue.
-

[2.1.1]

- Bug Fixes
 - MISRA C-2012 issue fixed.
 - * Fixed rules, containing: rule-10.3, rule-10.8, rule-11.9, rule-14.4, rule-15.5.

[2.1.0]

- Improvements
 - Added an API to enable or disable the window mode.
 - Added an API to convert a raw count value to millisecond.
 - Used AT_QUICKACCESS_SECTION_CODE macro to decorate RTWDOG_Init, and copied this function from flash to QUICKACCESS section.

[2.0.1]

- Bug Fixes
 - Fixed bug in the RTWDOG_Init; added check for register's unlock status when configuring the RTWDOG in RTWDOG_init.

[2.0.0]

- Initial version.
-

SAI

[2.4.10]

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

[2.4.9]

- Added Errata ERR051421 workaround.

[2.4.8]

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

[2.4.7]

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

[2.4.6]

- Bug Fixes
 - Fixed the IAR build warning.

[2.4.5]

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

[2.4.4]

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

[2.4.3]

- Bug Fixes
 - Fixed enumeration `sai_fifo_combine_t` value configuration issue.

[2.4.2]

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

[2.4.1]

- Bug Fixes
 - Fixed bitWidth incorrectly assigned issue.

[2.4.0]

- Improvements
 - Removed deprecated APIs.

[2.3.8]

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

[2.3.7]

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

[2.3.6]

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

[2.3.5]

- Improvements
 - Make driver to be aarch64 compatible.

[2.3.4]

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

[2.3.3]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

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

[2.1.9]

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

[2.1.8]

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

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

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

[2.0.0]

- Initial version.
-

SAI_EDMA

[2.7.3]

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

[2.7.2]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

- Improvements
 - Removed deprecated APIs.

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.2]

- Refer SAI driver change log 2.1.0 to 2.3.2
-

SEMA4**[2.2.2]**

- Improvements
 - Updated SEMA4_TryLock function to avoid unsigned integer operations wrap issue.

[2.2.1]

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

[2.2.0]

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

[2.1.0]

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

[2.0.3]

- Improvements
 - Changed to implement SEMA4_Lock base on SEMA4_TryLock.

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

SEMC

[2.7.1]

- Bug Fixes
 - Fixed the wrong write operation to INTR register. The INTR register is a W1C register, so the right write operation is write directly to it to clear.

[2.7.0]

- Improvements
 - Add new autofreshTimes parameter in semc_sdram_config_t.
- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.4.

[2.6.0]

- Bug Fixes
 - Fixed the SEMC SRAM function bug that some configuration options can't be set.
 - Correct legacy SEMC SRAM function feature macros.
- Improvements
 - Add new SEMC SRAM function feature macros.

[2.5.1]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 Rule 14.3.
 - Fixed SEMC_ConfigureDBI bug that RDX not set correctly.

[2.5.0]

- Bug Fixes
 - Fixed definitions of bitfields of BMCR0 and BMCR1 - wrong field order and incorrect semantical naming
 - The fix alters the driver API regarding configuration of AXI bus queue reordering

[2.4.3]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 Rule 5.6.

[2.4.2]

- Improvements
 - Deleted meaningless parameter in memory size conversion function.

[2.4.1]

- Bug Fixes
 - Fixed PSRAM A8 configuration issue, which should be 0x06U for PSRAM while pix mux bit width is 0x04U, based on different pix mux bit width.

[2.4.0]

- Improvements
 - Improved nor and sram timing configuration on sync mode.

[2.3.1]

- Bug Fixes
 - Updated refresh timer period(RT) timing setting, which updated into (RT+1)*(Prescaler period) for SDRAM.
 - Supported new DBI control register 2 to configure CSX interval time(CEITV).
 - Fixed violations of the MISRA C-2012 Rule 10.8.
 - Fixed doxygen warning.

[2.3.0]

- New Features
 - Limited burst length as 1 according to ERR050577, Auto-refresh command may possibly fail to be triggered during long time back-to-back write (or read) when SDRAM controller's burst length is greater than 1.
 - Supported 8 bits column address for SDRAM.

[2.2.1]

- New Features
 - Added queue weight control, which can control queue a/b is working or not.
 - Updated NAND FLASH configuration API which disables and enables SEMC between configure control registers.
 - Added ONFI parameter Integrity CRC check for SEMC flash component.

[2.2.0]

- New Features
 - Supported up to 4 PSRAM CS.
 - Added programmable delay line for DQS.
 - Added ready/wait feature for SRAM in asynchronous mode.

[2.1.0]

- Bug Fixes
 - MISRA C-2012 issue fixed: rule 10.3, 10.4, and 14.4.
 - Updated parameter type from uint16_t into uint32_t for send IP command API.

[2.0.4]

- Bug Fixes
 - Fixed the SEMC queueA and queueB weight configuration issue.
 - Fixed the wrong configuration of DBICR1 register in SEMC_ConfigureDBI.

[2.0.3]

- Bug Fixes
 - Added feature macro to control WDS&WDH bit setting for NOR synchronous transfer.

[2.0.2]

- Bug Fixes
 - Changed SEMC NAND configuration structure and verify SEMC NAND related APIs.
 - Added extended SEMC clock enable.

[2.0.1]

- Bug Fixes
 - Fixed data size mask configure in SEMC_ConfigureIPCommand API.
 - Updated the command mode in IP command type.

[2.0.0]

- Initial version.
-

SMARTCARD**[2.3.0]**

- New features:
 - Added support for USIM

[2.2.2]

- Bug fix:
 - Fixed MISRA C-2012 rule 10.4.

[2.2.1]

- Bug fix:
 - Fixed IAR warnings Pa082 in smartcard_emvsim
 - Fixed MISRA issues
 - Fixed rules 10.1, 10.3, 10.4, 10.6, 10.7, 10.8, 14.4, 16.1, 16.3, 16.4, 17.7

[2.2.0]

- New features:
 - Updated to use RX/TX FIFO

[2.1.2]

- Provided time delay function which works in microseconds.
- Bug fix:
 - Changed event to semaphore in RTOS driver (KPSDK-11634).
 - Added check if de-initialized variables are not null iSMARTCARD_RTOS_Deinit() (KPSDK-8788).
 - Changed deactivation sequence iSMARTCARD_PHY_TDA8035_Deactivate() to properly stop the clockPOSCR-35).
 - Fixed timing issue with VSEL0/1 signals in smartcard TDA803driver (KPSDK-10160)

[2.1.1]

- New features:
 - Added default phy interface selection into smartcard RTOS drivers (KPSDK-9063).
 - Replaced smartcard_phy_ncn8025 driver by smartcard_phy_tda8035.
- Bug fix:
 - Fixed protocol timers activation sequences in smartcard_emvsim and smartcard_phy_tda8035 drivers during emv11 pre-certification tests (KPSDK-9170, KPSDK-9556).

[2.1.0]

- Initial version.
-

SNVS_HP

[2.3.2]

- Make SNVS_HP_RTC_Init()/SNVS_HP_RTC_Deinit more transparent. Use function SNVS_HP_Init()/SNVS_HP_Deinit() instead of copy of this code in SNVS_HP_RTC_XXX() function.

[2.3.1]

- Fixed problem in SNVS_HP_RTC_Init(), which is clearing bits that should stay intact.

[2.3.0]

- Re-map Security Violation for RT11xx specific violations.

[2.2.0]

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

[2.1.4]

- Fix MISRA issues.

[2.1.3]

- Fixed IAR Pa082 warnings.

[2.1.2]

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

[2.1.1]

- Added APIs for HP security violation status flags.

[2.1.0]

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

[2.0.0]

- Initial version.
-

SNVS_LP**[2.4.6]**

- Fix a bug in SNVS_LP_EnableRxActiveTamper() where assignments to base->LPATRC2R were done wrongly to LPATRC1R.

[2.4.5]

- Fix a bug in SNVS_LP_EnableRxActiveTamper() where assignments to base->LPATRC1R would overwrite previously set bits.

[2.4.4]

- Make SNVS_LP_SRTC_Init()/SNVS_LP_SRTC_Deinit more transparent. Use function SNVS_LP_Init()/SNVS_LP_Deinit() instead of copy of this code in SNVS_LP_SRTC_XXX() function.

[2.4.3]

- Fixed problem in SNVS_LP_SRTC_Init(), which is clearing bits that should stay intact.

[2.4.2]

- Updated driver to match with new device header files.

[2.4.1]

- Fixed MISRA issues.

[2.4.0]

- Fix backward compatibility with version 2.2.x.

[2.3.0]

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

[2.2.0]

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

[2.1.2]

- Fix MISRA issues.

[2.1.1]

- Fix IAR Pa082 warning.

[2.1.0]

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

[2.0.0]

- Initial version.
-

SOC_MIPI_CSI2RX

[2.0.2]

- Updated for new header file.

[2.0.1]

- Bug Fixes
 - Fixed MISRA-C 2012 10.8 issue.

[2.0.0]

- initial version.
-

SPDIF

[2.0.7]

- Improvements
 - Add feature macro FSL_FEATURE_SPDIF_HAS_NO_SIC_REGISTER to handle non-existent SIC register.

[2.0.6]

- Bug Fixes
 - Fixed the Q/U channel interrupt enabled unexpectedly while Q/U transfer pointer is NULL.

[2.0.5]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 11.3.

[2.0.4]

- Bug Fixes
 - Added udata/qdata buffer address validation in driver IRQ handler to ensure that NULL pointer dereferences do not occur.

[2.0.3]

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

[2.0.2]

- Bug Fixes
 - Corrected operator used for size value assertion in SPDIF_ReadBlocking/SPDIF_WriteBlocking.

[2.0.1]

- Bug Fixes
 - Corrected the feature macro name used to define s_edmaPrivateHandle.

[2.0.0]

- Initial version.
-

SPDIF DMA Driver**[2.0.8]**

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

[2.0.7]

- Bug Fixes
 - Fixed the incompatibility issue with edma4 driver.

[2.0.6]

- Bug Fixes
 - Add feature macro to determine whether to use the API MEMORY_ConvertMemoryMapAddress to translate TCD addresses for DLAST_SGA.

[2.0.5]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 11.3.

[2.0.4]

- Bug Fixes
 - Added udata/qdata buffer address validation in driver IRQ handler to ensure that NULL pointer dereferences do not occur.

[2.0.3]

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

[2.0.2]

- Bug Fixes
 - Corrected operator used for size value assertion in SPDIF_ReadBlocking/SPDIF_WriteBlocking.

[2.0.1]

- Bug Fixes
 - Corrected the feature macro name used to define s_edmaPrivateHandle.

[2.0.0]

- Initial version.
-

SSARC

[2.1.0]

- Improvements
 - Updated the structure ssarc_descriptor_config_t, make it more friendly to users.

[2.0.0]

- Initial version.
-

TEMPSENSOR

[2.1.2]

- Bug Fixes
 - Fixed the bug of incorrect default value of temperature sensor registers in initialization state.

[2.1.1]

- Improvements
 - CTRL0 register fields are not needed for customer, they are trim registers for the IP that are determined during calibration.

[2.1.0]

- Improvements
 - Supported directly access to TEMPSENSOR registers.

[2.0.3]

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

[2.0.2]

- Bug Fixes
 - Fixed bug that FINISH flag not cleared after temperature read out.

[2.0.1]

- Improvements
 - Updated temperature calculation formula, to get more accurate result with high or low temperature..

[2.0.0]

- Initial version.
-

USDHC

[2.8.6]

- Bug Fixes
 - Invalidate cache after blocking read.

[2.8.5]

- Improvements
 - Enable the driver to be AARCH64 compatible.

[2.8.4]

- Improvements
 - Add feature macro FSL_FEATURE_USDHC_HAS_NO_VS18.

[2.8.3]

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

[2.8.2]

- Improvements
 - Added feature macro `FSL_FEATURE_USDHC_HAS_NO_VOLTAGE_SELECT`.

[2.8.1]

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

[2.8.0]

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

[2.7.0]

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

[2.6.0]

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

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.0]

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

[2.2.8]

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

[2.2.7]

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

[2.2.6]

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

[2.2.5]

- Bug Fixes
 - Fixed MDK 66-D warning.

[2.2.4]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

- Improvements
 - Improved USDHC to support MMC boot feature.

[2.1.3]

- Bug Fixes
 - Fixed MISRA issue.

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.
-

WDOG**[2.2.0]**

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

[2.1.1]

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

[2.1.0]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

XBARA

[2.0.6]

- Bug Fixes
 - Fixed typo in kXBARA_RequestInterruptEnalbe item.

[2.0.5]

- Bug Fixes
 - Fixed IAR build warning Pa082.
 - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.4, 10.6, 10.7, 10.8, 12.1, 18.1, 20.7.

[2.0.4]

- Improvements
 - Optimized XBARA_SetOutputSignalConfig.

[2.0.3]

- Bug Fixes
 - Corrected configuration for function XBAR_SetOutputSignalConfig.

[2.0.2]

- Other Changes
 - Changed array clock name.

[2.0.1]

- Bug Fixes
 - Fixed w1c bits for XBARA_SetOutputSignalConfig function.

[2.0.0]

- Initial version.
-

XBARB

[2.0.2]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 12.2, 10.7

[2.0.1]

- Bug Fixes
 - Corrected XBARB_SetSignalsConnection function.
- Other Changes
 - Changed array clock name.

[2.0.0]

- Initial version.
-

XECC

[2.0.0]

- Initial version.
-

XRDC2

[2.0.3]

- Bug Fixes
 - Fixed the bug that domain access policy is set to the incorrect domain ID.

[2.0.2]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.4, 10.8, 12.2, 14.1.

[2.0.1]

- Improvements
 - Updated for new header file.

[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.

[MIMXRT1166](#)

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 VG-Lite GPU Library

[vglite](#)

1.7.2 Multicore

[multicore](#)

1.7.3 MCU Boot

[mcuboot_opensource](#)

1.7.4 Audio Voice components

[components](#)

1.7.5 Maestro Audio Framework for MCU

[Maestro Audio Framework](#)

1.7.6 eIQ

eIQ

1.7.7 FreeMASTER

freemaster

1.7.8 AWS IoT

aws_iot

1.7.9 NXP Wi-Fi

wifi-bluetooth-802.15.4

1.7.10 FreeRTOS

FreeRTOS

1.7.11 lwIP

lwip

1.7.12 File systemFatfs

fatfs

Chapter 2

MIMXRT1166

2.1 ACMP: Analog Comparator Driver

`void ACMP_Init(CMP_Type *base, const acmp_config_t *config)`

Initializes the ACMP.

The default configuration can be got by calling `ACMP_GetDefaultConfig()`.

Parameters

- `base` – ACMP peripheral base address.
- `config` – Pointer to ACMP configuration structure.

`void ACMP_Deinit(CMP_Type *base)`

Deinitializes the ACMP.

Parameters

- `base` – ACMP peripheral base address.

`void ACMP_GetDefaultConfig(acmp_config_t *config)`

Gets the default configuration for ACMP.

This function initializes the user configuration structure to default value. The default value are:

Example:

```
config->enableHighSpeed = false;
config->enableInvertOutput = false;
config->useUnfilteredOutput = false;
config->enablePinOut = false;
config->enableHysteresisBothDirections = false;
config->hysteresisMode = kACMP_hysteresisMode0;
```

Parameters

- `config` – Pointer to ACMP configuration structure.

`void ACMP_Enable(CMP_Type *base, bool enable)`

Enables or disables the ACMP.

Parameters

- `base` – ACMP peripheral base address.
- `enable` – True to enable the ACMP.

void ACMP_EnableLinkToDAC(CMP_Type *base, bool enable)

Enables the link from CMP to DAC enable.

When this bit is set, the DAC enable/disable is controlled by the bit CMP_C0[EN] instead of CMP_C1[DACEN].

Parameters

- base – ACMP peripheral base address.
- enable – Enable the feature or not.

void ACMP_SetChannelConfig(CMP_Type *base, const *acmp_channel_config_t* *config)

Sets the channel configuration.

Note that the plus/minus mux's setting is only valid when the positive/negative port's input isn't from DAC but from channel mux.

Example:

```
acmp_channel_config_t configStruct = {0};
configStruct.positivePortInput = kACMP_PortInputFromDAC;
configStruct.negativePortInput = kACMP_PortInputFromMux;
configStruct.minusMuxInput = 1U;
ACMP_SetChannelConfig(CMP0, &configStruct);
```

Parameters

- base – ACMP peripheral base address.
- config – Pointer to channel configuration structure.

void ACMP_EnableDMA(CMP_Type *base, bool enable)

Enables or disables DMA.

Parameters

- base – ACMP peripheral base address.
- enable – True to enable DMA.

void ACMP_EnableWindowMode(CMP_Type *base, bool enable)

Enables or disables window mode.

Parameters

- base – ACMP peripheral base address.
- enable – True to enable window mode.

void ACMP_SetFilterConfig(CMP_Type *base, const *acmp_filter_config_t* *config)

Configures the filter.

The filter can be enabled when the filter count is bigger than 1, the filter period is greater than 0 and the sample clock is from divided bus clock or the filter is bigger than 1 and the sample clock is from external clock. Detailed usage can be got from the reference manual.

Example:

```
acmp_filter_config_t configStruct = {0};
configStruct.filterCount = 5U;
configStruct.filterPeriod = 200U;
configStruct.enableSample = false;
ACMP_SetFilterConfig(CMP0, &configStruct);
```

Parameters

- base – ACMP peripheral base address.

- `config` – Pointer to filter configuration structure.

`void ACMP_SetDACConfig(CMP_Type *base, const acmp_dac_config_t *config)`

Configures the internal DAC.

Example:

```
acmp_dac_config_t configStruct = {0};
configStruct.referenceVoltageSource = kACMP_VrefSourceVin1;
configStruct.DACValue = 20U;
configStruct.enableOutput = false;
configStruct.workMode = kACMP_DACWorkLowSpeedMode;
ACMP_SetDACConfig(CMP0, &configStruct);
```

Parameters

- `base` – ACMP peripheral base address.
- `config` – Pointer to DAC configuration structure. “NULL” is for disabling the feature.

`void ACMP_SetRoundRobinConfig(CMP_Type *base, const acmp_round_robin_config_t *config)`

Configures the round robin mode.

Example:

```
acmp_round_robin_config_t configStruct = {0};
configStruct.fixedPort = kACMP_FixedPlusPort;
configStruct.fixedChannelNumber = 3U;
configStruct.checkerChannelMask = 0xF7U;
configStruct.sampleClockCount = 0U;
configStruct.delayModulus = 0U;
ACMP_SetRoundRobinConfig(CMP0, &configStruct);
```

Parameters

- `base` – ACMP peripheral base address.
- `config` – Pointer to round robin mode configuration structure. “NULL” is for disabling the feature.

`void ACMP_SetRoundRobinPreState(CMP_Type *base, uint32_t mask)`

Defines the pre-set state of channels in round robin mode.

Note: The pre-state has different circuit with get-round-robin-result in the SOC even though they are same bits. So get-round-robin-result can't return the same value as the value are set by pre-state.

Parameters

- `base` – ACMP peripheral base address.
- `mask` – Mask of round robin channel index. Available range is channel0:0x01 to channel7:0x80.

`static inline uint32_t ACMP_GetRoundRobinStatusFlags(CMP_Type *base)`

Gets the channel input changed flags in round robin mode.

Parameters

- `base` – ACMP peripheral base address.

Returns

Mask of channel input changed asserted flags. Available range is channel0:0x01 to channel7:0x80.

`void ACMP_ClearRoundRobinStatusFlags(CMP_Type *base, uint32_t mask)`

Clears the channel input changed flags in round robin mode.

Parameters

- `base` – ACMP peripheral base address.
- `mask` – Mask of channel index. Available range is channel0:0x01 to channel7:0x80.

`static inline uint32_t ACMP_GetRoundRobinResult(CMP_Type *base)`

Gets the round robin result.

Note that the set-pre-state has different circuit with get-round-robin-result in the SOC even though they are same bits. So `ACMP_GetRoundRobinResult()` can't return the same value as the value are set by `ACMP_SetRoundRobinPreState`.

Parameters

- `base` – ACMP peripheral base address.

Returns

Mask of round robin channel result. Available range is channel0:0x01 to channel7:0x80.

`void ACMP_EnableInterrupts(CMP_Type *base, uint32_t mask)`

Enables interrupts.

Parameters

- `base` – ACMP peripheral base address.
- `mask` – Interrupts mask. See “_acmp_interrupt_enable”.

`void ACMP_DisableInterrupts(CMP_Type *base, uint32_t mask)`

Disables interrupts.

Parameters

- `base` – ACMP peripheral base address.
- `mask` – Interrupts mask. See “_acmp_interrupt_enable”.

`uint32_t ACMP_GetStatusFlags(CMP_Type *base)`

Gets status flags.

Parameters

- `base` – ACMP peripheral base address.

Returns

Status flags asserted mask. See “_acmp_status_flags”.

`void ACMP_ClearStatusFlags(CMP_Type *base, uint32_t mask)`

Clears status flags.

Parameters

- `base` – ACMP peripheral base address.
- `mask` – Status flags mask. See “_acmp_status_flags”.

`void ACMP_SetDiscreteModeConfig(CMP_Type *base, const acmp_discrete_mode_config_t *config)`

Configure the discrete mode.

Configure the discrete mode when supporting 3V domain with 1.8V core.

Parameters

- base – ACMP peripheral base address.
- config – Pointer to configuration structure. See “acmp_discrete_mode_config_t”.

void ACMP_GetDefaultDiscreteModeConfig(*acmp_discrete_mode_config_t* *config)

Get the default configuration for discrete mode setting.

Parameters

- config – Pointer to configuration structure to be restored with the setting values.

FSL_ACMP_DRIVER_VERSION

ACMP driver version 2.4.0.

enum _acmp_interrupt_enable

Interrupt enable/disable mask.

Values:

enumerator kACMP_OutputRisingInterruptEnable

Enable the interrupt when comparator outputs rising.

enumerator kACMP_OutputFallingInterruptEnable

Enable the interrupt when comparator outputs falling.

enumerator kACMP_RoundRobinInterruptEnable

Enable the Round-Robin interrupt.

enum _acmp_status_flags

Status flag mask.

Values:

enumerator kACMP_OutputRisingEventFlag

Rising-edge on compare output has occurred.

enumerator kACMP_OutputFallingEventFlag

Falling-edge on compare output has occurred.

enumerator kACMP_OutputAssertEventFlag

Return the current value of the analog comparator output.

enum _acmp_offset_mode

Comparator hard block offset control.

If OFFSET level is 1, then there is no hysteresis in the case of positive port input crossing negative port input in the positive direction (or negative port input crossing positive port input in the negative direction). Hysteresis still exists for positive port input crossing negative port input in the falling direction. If OFFSET level is 0, then the hysteresis selected by *acmp_hysteresis_mode_t* is valid for both directions.

Values:

enumerator kACMP_OffsetLevel0

The comparator hard block output has level 0 offset internally.

enumerator kACMP_OffsetLevel1

The comparator hard block output has level 1 offset internally.

enum _acmp_hysteresis_mode

Comparator hard block hysteresis control.

See chip data sheet to get the actual hysteresis value with each level.

Values:

enumerator kACMP_HysteresisLevel0

Offset is level 0 and Hysteresis is level 0.

enumerator kACMP_HysteresisLevel1

Offset is level 0 and Hysteresis is level 1.

enumerator kACMP_HysteresisLevel2

Offset is level 0 and Hysteresis is level 2.

enumerator kACMP_HysteresisLevel3

Offset is level 0 and Hysteresis is level 3.

enum _acmp_reference_voltage_source

CMP Voltage Reference source.

Values:

enumerator kACMP_VrefSourceVin1

Vin1 is selected as resistor ladder network supply reference Vin.

enumerator kACMP_VrefSourceVin2

Vin2 is selected as resistor ladder network supply reference Vin.

enum _acmp_port_input

Port input source.

Values:

enumerator kACMP_PortInputFromDAC

Port input from the 8-bit DAC output.

enumerator kACMP_PortInputFromMux

Port input from the analog 8-1 mux.

enum _acmp_fixed_port

Fixed mux port.

Values:

enumerator kACMP_FixedPlusPort

Only the inputs to the Minus port are swept in each round.

enumerator kACMP_FixedMinusPort

Only the inputs to the Plus port are swept in each round.

enum _acmp_dac_work_mode

Internal DAC's work mode.

Values:

enumerator kACMP_DACWorkLowSpeedMode

DAC is selected to work in low speed and low power mode.

enumerator kACMP_DACWorkHighSpeedMode

DAC is selected to work in high speed high power mode.

typedef enum _acmp_offset_mode acmp_offset_mode_t

Comparator hard block offset control.

If OFFSET level is 1, then there is no hysteresis in the case of positive port input crossing negative port input in the positive direction (or negative port input crossing positive port input in the negative direction). Hysteresis still exists for positive port input crossing negative port input in the falling direction. If OFFSET level is 0, then the hysteresis selected by acmp_hysteresis_mode_t is valid for both directions.

```
typedef enum _acmp_hysteresis_mode acmp_hysteresis_mode_t
```

Comparator hard block hysteresis control.

See chip data sheet to get the actual hysteresis value with each level.

```
typedef enum _acmp_reference_voltage_source acmp_reference_voltage_source_t
```

CMP Voltage Reference source.

```
typedef enum _acmp_port_input acmp_port_input_t
```

Port input source.

```
typedef enum _acmp_fixed_port acmp_fixed_port_t
```

Fixed mux port.

```
typedef enum _acmp_dac_work_mode acmp_dac_work_mode_t
```

Internal DAC's work mode.

```
typedef struct _acmp_config acmp_config_t
```

Configuration for ACMP.

```
typedef struct _acmp_channel_config acmp_channel_config_t
```

Configuration for channel.

The comparator's port can be input from channel mux or DAC. If port input is from channel mux, detailed channel number for the mux should be configured.

```
typedef struct _acmp_filter_config acmp_filter_config_t
```

Configuration for filter.

```
typedef struct _acmp_dac_config acmp_dac_config_t
```

Configuration for DAC.

```
typedef struct _acmp_round_robin_config acmp_round_robin_config_t
```

Configuration for round robin mode.

```
typedef struct _acmp_discrete_mode_config acmp_discrete_mode_config_t
```

Configuration for discrete mode.

```
CMP_C0_CFX_MASK
```

The mask of status flags cleared by writing 1.

```
CMP_C1_CHNn_MASK
```

```
CMP_C2_CHnF_MASK
```

```
struct _acmp_config
```

#include <fsl_acmp.h> Configuration for ACMP.

Public Members

```
acmp_offset_mode_t offsetMode
```

Offset mode.

```
acmp_hysteresis_mode_t hysteresisMode
```

Hysteresis mode.

```
bool enableHighSpeed
```

Enable High Speed (HS) comparison mode.

```
bool enableInvertOutput
```

Enable inverted comparator output.

bool useUnfilteredOutput

Set compare output(COUT) to equal COUTA(true) or COUT(false).

bool enablePinOut

The comparator output is available on the associated pin.

struct __acmp_channel_config

#include <fsl_acmp.h> Configuration for channel.

The comparator's port can be input from channel mux or DAC. If port input is from channel mux, detailed channel number for the mux should be configured.

Public Members

acmp_port_input_t positivePortInput

Input source of the comparator's positive port.

uint32_t plusMuxInput

Plus mux input channel(0~7).

acmp_port_input_t negativePortInput

Input source of the comparator's negative port.

uint32_t minusMuxInput

Minus mux input channel(0~7).

struct __acmp_filter_config

#include <fsl_acmp.h> Configuration for filter.

Public Members

bool enableSample

Using external SAMPLE as sampling clock input, or using divided bus clock.

uint32_t filterCount

Filter Sample Count. Available range is 1-7, 0 would cause the filter disabled.

uint32_t filterPeriod

Filter Sample Period. The divider to bus clock. Available range is 0-255.

struct __acmp_dac_config

#include <fsl_acmp.h> Configuration for DAC.

Public Members

acmp_reference_voltage_source_t referenceVoltageSource

Supply voltage reference source.

uint32_t DACValue

Value for DAC Output Voltage. Available range is 0-255.

bool enableOutput

Enable the DAC output.

struct __acmp_round_robin_config

#include <fsl_acmp.h> Configuration for round robin mode.

Public Members

acmp_fixed_port_t fixedPort

Fixed mux port.

uint32_t fixedChannelNumber

Indicates which channel is fixed in the fixed mux port.

uint32_t checkerChannelMask

Mask of checker channel index. Available range is channel0:0x01 to channel7:0x80 for round-robin checker.

uint32_t sampleClockCount

Specifies how many round-robin clock cycles(0~3) later the sample takes place.

uint32_t delayModulus

Comparator and DAC initialization delay modulus.

struct _acmp_discrete_mode_config

#include <fsl_acmp.h> Configuration for discrete mode.

Public Members

bool enablePositiveChannelDiscreteMode

Positive Channel Continuous Mode Enable. By default, the continuous mode is used.

bool enableNegativeChannelDiscreteMode

Negative Channel Continuous Mode Enable. By default, the continuous mode is used.

2.2 ADC_ETC: ADC External Trigger Control

void ADC_ETC_Init(ADC_ETC_Type *base, const *adc_etc_config_t* *config)

Initialize the ADC_ETC module.

Parameters

- base – ADC_ETC peripheral base address.
- config – Pointer to “adc_etc_config_t” structure.

void ADC_ETC_Deinit(ADC_ETC_Type *base)

De-Initialize the ADC_ETC module.

Parameters

- base – ADC_ETC peripheral base address.

void ADC_ETC_GetDefaultConfig(*adc_etc_config_t* *config)

Gets an available pre-defined settings for the ADC_ETC’s configuration. This function initializes the ADC_ETC’s configuration structure with available settings. The default values are:

```
config->enableTSCBypass = true;
config->enableTSC0Trigger = false;
config->enableTSC1Trigger = false;
config->TSC0triggerPriority = 0U;
config->TSC1triggerPriority = 0U;
config->clockPreDivider = 0U;
config->XBARtriggerMask = 0U;
```

Parameters

- config – Pointer to “adc_etc_config_t” structure.

```
void ADC_ETC_SetTriggerConfig(ADC_ETC_Type *base, uint32_t triggerGroup, const  
                             adc_etc_trigger_config_t *config)
```

Set the external XBAR trigger configuration.

Parameters

- base – ADC_ETC peripheral base address.
- triggerGroup – Trigger group index.
- config – Pointer to “adc_etc_trigger_config_t” structure.

```
void ADC_ETC_SetTriggerChainConfig(ADC_ETC_Type *base, uint32_t triggerGroup, uint32_t  
                                   chainGroup, const adc_etc_trigger_chain_config_t  
                                   *config)
```

Set the external XBAR trigger chain configuration. For example, if triggerGroup is set to 0U and chainGroup is set to 1U, which means Trigger0 source's chain1 would be configured.

Parameters

- base – ADC_ETC peripheral base address.
- triggerGroup – Trigger group index. Available number is 0~7.
- chainGroup – Trigger chain group index. Available number is 0~7.
- config – Pointer to “adc_etc_trigger_chain_config_t” structure.

```
uint32_t ADC_ETC_GetInterruptStatusFlags(ADC_ETC_Type *base,  
                                          adc_etc_external_trigger_source_t sourceIndex)
```

Gets the interrupt status flags of external XBAR and TSC triggers.

Parameters

- base – ADC_ETC peripheral base address.
- sourceIndex – trigger source index.

Returns

Status flags mask of trigger. Refer to “_adc_etc_status_flag_mask”.

```
void ADC_ETC_ClearInterruptStatusFlags(ADC_ETC_Type *base,  
                                       adc_etc_external_trigger_source_t sourceIndex,  
                                       uint32_t mask)
```

Clears the ADC_ETC's interrupt status falgs.

Parameters

- base – ADC_ETC peripheral base address.
- sourceIndex – trigger source index.
- mask – Status flags mask of trigger. Refer to “_adc_etc_status_flag_mask”.

```
static inline void ADC_ETC_EnableDMA(ADC_ETC_Type *base, uint32_t triggerGroup)
```

Enable the DMA corresponding to each trigger source.

Parameters

- base – ADC_ETC peripheral base address.
- triggerGroup – Trigger group index. Available number is 0~7.

```
static inline void ADC_ETC_DisableDMA(ADC_ETC_Type *base, uint32_t triggerGroup)
```

Disable the DMA corresponding to each trigger sources.

Parameters

- base – ADC_ETC peripheral base address.
- triggerGroup – Trigger group index. Available number is 0~7.

```
static inline uint32_t ADC_ETC_GetDMAStatusFlags(ADC_ETC_Type *base)
```

Get the DMA request status falgs. Only external XBAR sources support DMA request.

Parameters

- base – ADC_ETC peripheral base address.

Returns

Mask of external XBAR tirgger's DMA request asserted flags. Available range is trigger0:0x01 to trigger7:0x80.

```
static inline void ADC_ETC_ClearDMAStatusFlags(ADC_ETC_Type *base, uint32_t mask)
```

Clear the DMA request status falgs. Only external XBAR sources support DMA request.

Parameters

- base – ADC_ETC peripheral base address.
- mask – Mask of external XBAR tirgger's DMA request asserted flags. Available range is trigger0:0x01 to trigger7:0x80.

```
static inline void ADC_ETC_DoSoftwareReset(ADC_ETC_Type *base, bool enable)
```

When enable, all logical will be reset.

Parameters

- base – ADC_ETC peripheral base address.
- enable – Enable/Disable the software reset.

```
static inline void ADC_ETC_DoSoftwareTrigger(ADC_ETC_Type *base, uint32_t triggerGroup)
```

Do software trigger corresponding to each XBAR trigger sources. Each XBAR trigger sources can be configured as HW or SW trigger mode. In hardware trigger mode, trigger source is from XBAR. In software mode, trigger source is from software tigger. TSC trigger sources can only work in hardware trigger mode.

Parameters

- base – ADC_ETC peripheral base address.
- triggerGroup – Trigger group index. Available number is 0~7.

```
static inline void ADC_ETC_DoSoftwareTriggerBlocking(ADC_ETC_Type *base, uint32_t triggerGroup)
```

Do software trigger corresponding to each XBAR trigger sources.

Note: This function provides a workaround implementation for ERR052412 by using blocking way to implement SW trigger.

Parameters

- base – ADC_ETC peripheral base address.
- triggerGroup – Trigger group index. Available number is 0~7.

```
uint32_t ADC_ETC_GetADCConversionValue(ADC_ETC_Type *base, uint32_t triggerGroup,  
                                         uint32_t chainGroup)
```

Get ADC conversion result from external XBAR sources. For example, if triggerGroup is set to 0U and chainGroup is set to 1U, which means the API would return Trigger0 source's chain1 conversion result.

Parameters

- base – ADC_ETC peripheral base address.
- triggerGroup – Trigger group index. Available number is 0~7.
- chainGroup – Trigger chain group index. Available number is 0~7.

Returns

ADC conversion result value.

```
enum _adc_etc_status_flag_mask
```

ADC_ETC customized status flags mask.

Values:

enumerator kADC_ETC_Done0StatusFlagMask

enumerator kADC_ETC_Done1StatusFlagMask

enumerator kADC_ETC_Done2StatusFlagMask

enumerator kADC_ETC_Done3StatusFlagMask

enumerator kADC_ETC_ErrorStatusFlagMask

```
enum _adc_etc_external_trigger_source
```

External triggers sources.

Values:

enumerator kADC_ETC_Trg0TriggerSource

enumerator kADC_ETC_Trg1TriggerSource

enumerator kADC_ETC_Trg2TriggerSource

enumerator kADC_ETC_Trg3TriggerSource

enumerator kADC_ETC_Trg4TriggerSource

enumerator kADC_ETC_Trg5TriggerSource

enumerator kADC_ETC_Trg6TriggerSource

enumerator kADC_ETC_Trg7TriggerSource

enumerator kADC_ETC_TSC0TriggerSource

enumerator kADC_ETC_TSC1TriggerSource

```
enum _adc_etc_interrupt_enable
```

Interrupt enable/disable mask.

Values:

enumerator kADC_ETC_Done0InterruptEnable

enumerator kADC_ETC_Done1InterruptEnable

enumerator kADC_ETC_Done2InterruptEnable

enumerator kADC_ETC_Done3InterruptEnable

enum _adc_etc_dma_mode_selection
DMA mode selection.

Values:

enumerator kADC_ETC_TrigDMAWithLatchedSignal

enumerator kADC_ETC_TrigDMAWithPulsedSignal

typedef enum _adc_etc_external_trigger_source adc_etc_external_trigger_source_t
External triggers sources.

typedef enum _adc_etc_interrupt_enable adc_etc_interrupt_enable_t
Interrupt enable/disable mask.

typedef enum _adc_etc_dma_mode_selection adc_etc_dma_mode_selection_t
DMA mode selection.

typedef struct _adc_etc_config adc_etc_config_t
ADC_ETC configuration.

typedef struct _adc_etc_trigger_chain_config adc_etc_trigger_chain_config_t
ADC_ETC trigger chain configuration.

typedef struct _adc_etc_trigger_config adc_etc_trigger_config_t
ADC_ETC trigger configuration.

FSL_ADC_ETC_DRIVER_VERSION
ADC_ETC driver version.

Version 2.3.2.

ADC_ETC_DMA_CTRL_TRGn_REQ_MASK
The mask of status flags cleared by writing 1.

struct _adc_etc_config
#include <fsl_adc_etc.h> ADC_ETC configuration.

struct _adc_etc_trigger_chain_config
#include <fsl_adc_etc.h> ADC_ETC trigger chain configuration.

struct _adc_etc_trigger_config
#include <fsl_adc_etc.h> ADC_ETC trigger configuration.

2.3 AOI: Crossbar AND/OR/INVERT Driver

void AOI_Init(AOI_Type *base)
Initializes an AOI instance for operation.
This function un-gates the AOI clock.

Parameters

- base – AOI peripheral address.

void AOI_Deinit(AOI_Type *base)
Deinitializes an AOI instance for operation.
This function shutdowns AOI module.

Parameters

- base – AOI peripheral address.

void AOI_GetEventLogicConfig(AOI_Type *base, *aoi_event_t* event, *aoi_event_config_t* *config)

Gets the Boolean evaluation associated.

This function returns the Boolean evaluation associated.

Example:

```
aoi_event_config_t demoEventLogicStruct;

AOI_GetEventLogicConfig(AOI, kAOI_Event0, &demoEventLogicStruct);
```

Parameters

- base – AOI peripheral address.
- event – Index of the event which will be set of type *aoi_event_t*.
- config – Selected input configuration .

void AOI_SetEventLogicConfig(AOI_Type *base, *aoi_event_t* event, const *aoi_event_config_t* *eventConfig)

Configures an AOI event.

This function configures an AOI event according to the *aoiEventConfig* structure. This function configures all inputs (A, B, C, and D) of all product terms (0, 1, 2, and 3) of a desired event.

Example:

```
aoi_event_config_t demoEventLogicStruct;

demoEventLogicStruct.PT0AC = kAOI_InvInputSignal;
demoEventLogicStruct.PT0BC = kAOI_InputSignal;
demoEventLogicStruct.PT0CC = kAOI_LogicOne;
demoEventLogicStruct.PT0DC = kAOI_LogicOne;

demoEventLogicStruct.PT1AC = kAOI_LogicZero;
demoEventLogicStruct.PT1BC = kAOI_LogicOne;
demoEventLogicStruct.PT1CC = kAOI_LogicOne;
demoEventLogicStruct.PT1DC = kAOI_LogicOne;

demoEventLogicStruct.PT2AC = kAOI_LogicZero;
demoEventLogicStruct.PT2BC = kAOI_LogicOne;
demoEventLogicStruct.PT2CC = kAOI_LogicOne;
demoEventLogicStruct.PT2DC = kAOI_LogicOne;

demoEventLogicStruct.PT3AC = kAOI_LogicZero;
demoEventLogicStruct.PT3BC = kAOI_LogicOne;
demoEventLogicStruct.PT3CC = kAOI_LogicOne;
demoEventLogicStruct.PT3DC = kAOI_LogicOne;

AOI_SetEventLogicConfig(AOI, kAOI_Event0, demoEventLogicStruct);
```

Parameters

- base – AOI peripheral address.
- event – Event which will be configured of type *aoi_event_t*.
- eventConfig – Pointer to type *aoi_event_config_t* structure. The user is responsible for filling out the members of this structure and passing the pointer to this function.

FSL_AOI_DRIVER_VERSION

Version 2.0.2.

enum _aoi_input_config

AOI input configurations.

The selection item represents the Boolean evaluations.

Values:

enumerator kAOI_LogicZero

Forces the input to logical zero.

enumerator kAOI_InputSignal

Passes the input signal.

enumerator kAOI_InvInputSignal

Inverts the input signal.

enumerator kAOI_LogicOne

Forces the input to logical one.

enum _aoi_event

AOI event indexes, where an event is the collection of the four product terms (0, 1, 2, and 3) and the four signal inputs (A, B, C, and D).

Values:

enumerator kAOI_Event0

Event 0 index

enumerator kAOI_Event1

Event 1 index

enumerator kAOI_Event2

Event 2 index

enumerator kAOI_Event3

Event 3 index

typedef enum _aoi_input_config aoi_input_config_t

AOI input configurations.

The selection item represents the Boolean evaluations.

typedef enum _aoi_event aoi_event_t

AOI event indexes, where an event is the collection of the four product terms (0, 1, 2, and 3) and the four signal inputs (A, B, C, and D).

typedef struct _aoi_event_config aoi_event_config_t

AOI event configuration structure.

Defines structure _aoi_event_config and use the AOI_SetEventLogicConfig() function to make whole event configuration.

AOI

AOI peripheral address

struct _aoi_event_config

#include <fsl_aoi.h> AOI event configuration structure.

Defines structure _aoi_event_config and use the AOI_SetEventLogicConfig() function to make whole event configuration.

Public Members

aoi_input_config_t PT0AC
Product term 0 input A

aoi_input_config_t PT0BC
Product term 0 input B

aoi_input_config_t PT0CC
Product term 0 input C

aoi_input_config_t PT0DC
Product term 0 input D

aoi_input_config_t PT1AC
Product term 1 input A

aoi_input_config_t PT1BC
Product term 1 input B

aoi_input_config_t PT1CC
Product term 1 input C

aoi_input_config_t PT1DC
Product term 1 input D

aoi_input_config_t PT2AC
Product term 2 input A

aoi_input_config_t PT2BC
Product term 2 input B

aoi_input_config_t PT2CC
Product term 2 input C

aoi_input_config_t PT2DC
Product term 2 input D

aoi_input_config_t PT3AC
Product term 3 input A

aoi_input_config_t PT3BC
Product term 3 input B

aoi_input_config_t PT3CC
Product term 3 input C

aoi_input_config_t PT3DC
Product term 3 input D

2.4 ASRC: Asynchronous sample rate converter

2.5 ASRC Driver

uint32_t ASRC_GetInstance(ASRC_Type *base)
Get instance number of the ASRC peripheral.

Parameters

- base – ASRC base pointer.

void ASRC_Init(ASRC_Type *base, uint32_t asrcPeripheralClock_Hz)

brief Initializes the asrc peripheral.

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

param base asrc base pointer. param asrcPeripheralClock_Hz peripheral clock of ASRC.

void ASRC_Deinit(ASRC_Type *base)

De-initializes the ASRC peripheral.

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

Parameters

- base – ASRC base pointer.

void ASRC_SoftwareReset(ASRC_Type *base)

Do software reset .

This software reset bit is self-clear bit, it will generate a software reset signal inside ASRC. After 9 cycles of the ASRC processing clock, this reset process will stop and this bit will cleared automatically.

Parameters

- base – ASRC base pointer

status_t ASRC_SetChannelPairConfig(ASRC_Type *base, asrc_channel_pair_t channelPair,
asrc_channel_pair_config_t *config, uint32_t
inputSampleRate, uint32_t outputSampleRate)

ASRC configure channel pair.

Parameters

- base – ASRC base pointer.
- channelPair – index of channel pair, reference _asrc_channel_pair.
- config – ASRC channel pair configuration pointer.
- inputSampleRate – input audio data sample rate.
- outputSampleRate – output audio data sample rate.

uint32_t ASRC_GetOutSamplesSize(ASRC_Type *base, asrc_channel_pair_t channelPair, uint32_t
inSampleRate, uint32_t outSampleRate, uint32_t
inSamplesize)

Get output sample buffer size.

Note: This API is depends on the ASRC output configuration, should be called after the ASRC_SetChannelPairConfig.

Parameters

- base – asrc base pointer.
- channelPair – ASRC channel pair number.
- inSampleRate – input sample rate.
- outSampleRate – output sample rate.
- inSamplesize – input sampleS size.

Return values

output – buffer size in byte.

```
uint32_t ASRC_MapSamplesWidth(ASRC_Type *base, asrc_channel_pair_t channelPair, uint32_t
                               *inWidth, uint32_t *outWidth)
```

Map register sample width to real sample width.

Note: This API is depends on the ASRC configuration, should be called after the ASRC_SetChannelPairConfig.

Parameters

- base – asrc base pointer.
- channelPair – asrc channel pair index.
- inWidth – ASRC channel pair number.
- outWidth – input sample rate.

Return values

input – sample mask value.

```
uint32_t ASRC_GetRemainFifoSamples(ASRC_Type *base, asrc_channel_pair_t channelPair,
                                   uint32_t *buffer, uint32_t outSampleWidth, uint32_t
                                   remainSamples)
```

Get left samples in fifo.

Parameters

- base – asrc base pointer.
- channelPair – ASRC channel pair number.
- buffer – input sample numbers.
- outSampleWidth – output sample width.
- remainSamples – output sample rate.

Return values

remain – samples number.

```
static inline void ASRC_ModuleEnable(ASRC_Type *base, bool enable)
```

ASRC module enable.

Parameters

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

```
static inline void ASRC_ChannelPairEnable(ASRC_Type *base, asrc_channel_pair_t channelPair,
                                           bool enable)
```

ASRC enable channel pair.

Parameters

- base – ASRC base pointer.
- channelPair – channel pair mask value, reference _asrc_channel_pair_mask.
- enable – true is enable, false is disable.

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

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

Parameters

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

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

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

Parameters

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

static inline uint32_t ASRC_GetStatus(ASRC_Type *base)

Gets the ASRC status flag state.

Parameters

- base – ASRC base pointer

Returns

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

static inline bool ASRC_GetChannelPairInitialStatus(ASRC_Type *base, *asrc_channel_pair_t* channel)

Gets the ASRC channel pair initialization state.

Parameters

- base – ASRC base pointer
- channel – ASRC channel pair.

Returns

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

static inline uint32_t ASRC_GetChannelPairFifoStatus(ASRC_Type *base, *asrc_channel_pair_t* channelPair)

Gets the ASRC channel A fifo a status flag state.

Parameters

- base – ASRC base pointer
- channelPair – ASRC channel pair.

Returns

ASRC channel pair a fifo status flag value. Use the Status Mask to get the status value needed.

static inline void ASRC_ChannelPairWriteData(ASRC_Type *base, *asrc_channel_pair_t* channelPair, uint32_t data)

Writes data into ASRC channel pair FIFO. Note: ASRC fifo width is 24bit.

Parameters

- base – ASRC base pointer.
- channelPair – ASRC channel pair.
- data – Data needs to be written.

```
static inline uint32_t ASRC_ChannelPairReadData(ASRC_Type *base, asrc_channel_pair_t
                                              channelPair)
```

Read data from ASRC channel pair FIFO. Note: ASRC fifo width is 24bit.

Parameters

- base – ASRC base pointer.
- channelPair – ASRC channel pair.

Return values

value – read from fifo.

```
static inline uint32_t ASRC_GetInputDataRegisterAddress(ASRC_Type *base, asrc_channel_pair_t
                                                       channelPair)
```

Get input data fifo address. Note: ASRC fifo width is 24bit.

Parameters

- base – ASRC base pointer.
- channelPair – ASRC channel pair.

```
static inline uint32_t ASRC_GetOutputDataRegisterAddress(ASRC_Type *base,
                                                         asrc_channel_pair_t channelPair)
```

Get output data fifo address. Note: ASRC fifo width is 24bit.

Parameters

- base – ASRC base pointer.
- channelPair – ASRC channel pair.

```
status_t ASRC_SetIdealRatioConfig(ASRC_Type *base, asrc_channel_pair_t channelPair, uint32_t
                                   inputSampleRate, uint32_t outputSampleRate)
```

ASRC configure ideal ratio. The ideal ratio should be used when input clock source is not available.

Parameters

- base – ASRC base pointer.
- channelPair – ASRC channel pair.
- inputSampleRate – input audio data sample rate.
- outputSampleRate – output audio data sample rate.

```
status_t ASRC_TransferSetChannelPairConfig(ASRC_Type *base, asrc_handle_t *handle,
                                             asrc_channel_pair_config_t *config, uint32_t
                                             inputSampleRate, uint32_t outputSampleRate)
```

ASRC configure channel pair.

Parameters

- base – ASRC base pointer.
- handle – ASRC transactional handle pointer.
- config – ASRC channel pair configuration pointer.
- inputSampleRate – input audio data sample rate.
- outputSampleRate – output audio data sample rate.

```
void ASRC_TransferCreateHandle(ASRC_Type *base, asrc_handle_t *handle, asrc_channel_pair_t
                               channelPair, asrc_transfer_callback_t inCallback,
                               asrc_transfer_callback_t outCallback, void *userData)
```

Initializes the ASRC handle.

This function initializes the handle for the ASRC transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – ASRC base pointer
- handle – ASRC handle pointer.
- channelPair – ASRC channel pair.
- inCallback – Pointer to the user callback function.
- outCallback – Pointer to the user callback function.
- userData – User parameter passed to the callback function

status_t ASRC_TransferNonBlocking(*ASRC_Type* *base, *asrc_handle_t* *handle, *asrc_transfer_t* *xfer)

Performs an interrupt non-blocking convert on asrc.

Note: This API returns immediately after the transfer initiates, application should check the wait and check the callback status.

Parameters

- base – asrc base pointer.
- handle – Pointer to the *asrc_handle_t* structure which stores the transfer state.
- xfer – Pointer to the *ASRC_transfer_t* structure.

Return values

- *kStatus_Success* – Successfully started the data receive.
- *kStatus_ASRCBusy* – Previous receive still not finished.

status_t ASRC_TransferBlocking(*ASRC_Type* *base, *asrc_channel_pair_t* channelPair, *asrc_transfer_t* *xfer)

Performs an blocking convert on asrc.

Note: This API returns immediately after the convert finished.

Parameters

- base – asrc base pointer.
- channelPair – channel pair index.
- xfer – Pointer to the *ASRC_transfer_t* structure.

Return values

kStatus_Success – Successfully started the data receive.

status_t ASRC_TransferGetConvertedCount(*ASRC_Type* *base, *asrc_handle_t* *handle, *size_t* *count)

Get converted byte count.

Parameters

- base – ASRC base pointer.

- `handle` – Pointer to the `asrc_handle_t` structure which stores the transfer state.
- `count` – Bytes count sent.

Return values

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

`void ASRC_TransferAbortConvert(ASRC_Type *base, asrc_handle_t *handle)`

Aborts the current convert.

Note: This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- `base` – ASRC base pointer.
- `handle` – Pointer to the `asrc_handle_t` structure which stores the transfer state.

`void ASRC_TransferTerminateConvert(ASRC_Type *base, asrc_handle_t *handle)`

Terminate all ASRC convert.

This function will clear all transfer slots buffered in the asrc queue. If users only want to abort the current transfer slot, please call `ASRC_TransferAbortConvert`.

Parameters

- `base` – ASRC base pointer.
- `handle` – ASRC eDMA handle pointer.

`void ASRC_TransferHandleIRQ(ASRC_Type *base, asrc_handle_t *handle)`

ASRC convert interrupt handler.

Parameters

- `base` – ASRC base pointer.
- `handle` – Pointer to the `asrc_handle_t` structure.

`FSL_ASRC_DRIVER_VERSION`

Version 2.1.3

ASRC return status .

Values:

enumerator `kStatus_ASRCIdle`

ASRC is idle.

enumerator `kStatus_ASRCInIdle`

ASRC in is idle.

enumerator `kStatus_ASRCOutIdle`

ASRC out is idle.

enumerator `kStatus_ASRCBusy`

ASRC is busy.

enumerator kStatus_ASRCInvalidArgument

ASRC invalid argument.

enumerator kStatus_ASRCClockConfigureFailed

ASRC clock configure failed

enumerator kStatus_ASRCChannelPairConfigureFailed

ASRC clock configure failed

enumerator kStatus_ASRCConvertError

ASRC clock configure failed

enumerator kStatus_ASRCNotSupport

ASRC not support

enumerator kStatus_ASRCQueueFull

ASRC queue is full

enumerator kStatus_ASRCOutQueueIdle

ASRC out queue is idle

enumerator kStatus_ASRCInQueueIdle

ASRC in queue is idle

enum _asrc_channel_pair

ASRC channel pair mask.

Values:

enumerator kASRC_ChannelPairA

channel pair A value

enumerator kASRC_ChannelPairB

channel pair B value

enumerator kASRC_ChannelPairC

channel pair C value

ASRC support sample rate .

Values:

enumerator kASRC_SampleRate_8000HZ

asrc sample rate 8KHZ

enumerator kASRC_SampleRate_11025HZ

asrc sample rate 11.025KHZ

enumerator kASRC_SampleRate_12000HZ

asrc sample rate 12KHZ

enumerator kASRC_SampleRate_16000HZ

asrc sample rate 16KHZ

enumerator kASRC_SampleRate_22050HZ

asrc sample rate 22.05KHZ

enumerator kASRC_SampleRate_24000HZ

asrc sample rate 24KHZ

enumerator kASRC_SampleRate_30000HZ

asrc sample rate 30KHZ

enumerator kASRC_SampleRate_32000HZ
asrc sample rate 32KHZ

enumerator kASRC_SampleRate_44100HZ
asrc sample rate 44.1KHZ

enumerator kASRC_SampleRate_48000HZ
asrc sample rate 48KHZ

enumerator kASRC_SampleRate_64000HZ
asrc sample rate 64KHZ

enumerator kASRC_SampleRate_88200HZ
asrc sample rate 88.2KHZ

enumerator kASRC_SampleRate_96000HZ
asrc sample rate 96KHZ

enumerator kASRC_SampleRate_128000HZ
asrc sample rate 128KHZ

enumerator kASRC_SampleRate_176400HZ
asrc sample rate 176.4KHZ

enumerator kASRC_SampleRate_192000HZ
asrc sample rate 192KHZ

The ASRC interrupt enable flag .

Values:

enumerator kASRC_FPInWaitStateInterruptEnable
FP in wait state mask

enumerator kASRC_OverLoadInterruptMask
overload interrupt mask

enumerator kASRC_DataOutputCInterruptMask
data output c interrupt mask

enumerator kASRC_DataOutputBInterruptMask
data output b interrupt mask

enumerator kASRC_DataOutputAInterruptMask
data output a interrupt mask

enumerator kASRC_DataInputCInterruptMask
data input c interrupt mask

enumerator kASRC_DataInputBInterruptMask
data input b interrupt mask

enumerator kASRC_DataInputAInterruptMask
data input a interrupt mask

The ASRC interrupt status .

Values:

enumerator kASRC_StatusDSLCounterReady
DSL counter

enumerator kASRC__StatusTaskQueueOverLoad
task queue overload

enumerator kASRC__StatusPairCOutputOverLoad
pair c output overload

enumerator kASRC__StatusPairBOutputOverLoad
pair b output overload

enumerator kASRC__StatusPairAOutputOverLoad
pair a output overload

enumerator kASRC__StatusPairCInputOverLoad
pair c input overload

enumerator kASRC__StatusPairBInputOverLoad
pair b input overload

enumerator kASRC__StatusPairAInputOverLoad
pair a input overload

enumerator kASRC__StatusPairCOutputOverflow
pair c output overflow

enumerator kASRC__StatusPairBOutputOverflow
pair b output overflow

enumerator kASRC__StatusPairAOutputOverflow
pair a output overflow

enumerator kASRC__StatusPairCInputUnderflow
pair c input underflow

enumerator kASRC__StatusPairBInputUnderflow
pair b input under flow

enumerator kASRC__StatusPairAInputUnderflow
pair a input underflow

enumerator kASRC__StatusFPInWaitState
FP in wait state

enumerator kASRC__StatusOverloadError
overload error

enumerator kASRC__StatusInputError
input error status

enumerator kASRC__StatusOutputError
output error status

enumerator kASRC__StatusPairCOutputReady
pair c output ready

enumerator kASRC__StatusPairBOutputReady
pair b output ready

enumerator kASRC__StatusPairAOutputReady
pair a output ready

enumerator kASRC__StatusPairCInputReady
pair c input ready

enumerator kASRC__StatusPairBInputReady
pair b input ready

enumerator kASRC__StatusPairAInputReady
pair a input ready

enumerator kASRC__StatusPairAInterrupt
pair A interrupt

enumerator kASRC__StatusPairBInterrupt
pair B interrupt

enumerator kASRC__StatusPairCInterrupt
pair C interrupt

ASRC channel pair status .

Values:

enumerator kASRC__OutputFifoNearFull
channel pair output fifo near full

enumerator kASRC__InputFifoNearEmpty
channel pair input fifo near empty

enum _asrc_ratio

ASRC ideal ratio.

Values:

enumerator kASRC__RatioNotUsed
ideal ratio not used

enumerator kASRC__RatioUseInternalMeasured
ideal ratio use internal measure ratio, can be used for real time streaming audio

enumerator kASRC__RatioUseIdealRatio
ideal ratio use manual configure ratio, can be used for the non-real time streaming audio

enum _asrc_audio_channel

Number of channels in audio data.

Values:

enumerator kASRC__ChannelsNumber1
channel number is 1

enumerator kASRC__ChannelsNumber2
channel number is 2

enumerator kASRC__ChannelsNumber3
channel number is 3

enumerator kASRC__ChannelsNumber4
channel number is 4

enumerator kASRC__ChannelsNumber5
channel number is 5

enumerator kASRC__ChannelsNumber6
channel number is 6

enumerator kASRC_ChannelsNumber7
channel number is 7

enumerator kASRC_ChannelsNumber8
channel number is 8

enumerator kASRC_ChannelsNumber9
channel number is 9

enumerator kASRC_ChannelsNumber10
channel number is 10

enum _asrc_data_width
data width

Values:

enumerator kASRC_DataWidth24Bit
data width 24bit

enumerator kASRC_DataWidth16Bit
data width 16bit

enumerator kASRC_DataWidth8Bit
data width 8bit

enum _asrc_data_align
data alignment

Values:

enumerator kASRC_DataAlignMSB
data alignment MSB

enumerator kASRC_DataAlignLSB
data alignment LSB

enum _asrc_sign_extension
sign extension

Values:

enumerator kASRC_NoSignExtension
no sign extension

enumerator kASRC_SignExtension
sign extension

typedef enum _asrc_channel_pair asrc_channel_pair_t
ASRC channel pair mask.

typedef enum _asrc_ratio asrc_ratio_t
ASRC ideal ratio.

typedef enum _asrc_audio_channel asrc_audio_channel_t
Number of channels in audio data.

typedef enum _asrc_data_width asrc_data_width_t
data width

typedef enum _asrc_data_align asrc_data_align_t
data alignment

```
typedef enum _asrc_sign_extension asrc_sign_extension_t
    sign extension

typedef struct _asrc_channel_pair_config asrc_channel_pair_config_t
    asrc channel pair configuration

typedef struct _asrc_transfer asrc_transfer_t
    SAI transfer structure.

typedef struct _asrc_handle asrc_handle_t
    asrc handler

typedef void (*asrc_transfer_callback_t)(ASRC_Type *base, asrc_handle_t *handle, status_t
status, void *userData)
    ASRC transfer callback prototype.

typedef struct _asrc_in_handle asrc_in_handle_t
    asrc in handler

typedef struct _asrc_out_handle asrc_out_handle_t
    output handler

ASRC_XFER_QUEUE_SIZE
    ASRC transfer queue size, user can refine it according to use case.

FSL_ASRC_CHANNEL_PAIR_COUNT
    ASRC channel pair count.

FSL_ASRC_CHANNEL_PAIR_FIFO_DEPTH
    ASRC FIFO depth.

ASRC_ASRCCTR_AT_MASK(index)
    ASRC register access macro.

ASRC_ASRCCTR_RATIO_MASK(index)

ASRC_ASRCCTR_RATIO(ratio, index)

ASRC_ASRIER_INPUT_INTERRUPT_MASK(index)

ASRC_ASRIER_OUTPUT_INTERRUPT_MASK(index)

ASRC_ASRCNCR_CHANNEL_COUNTER_MASK(index)

ASRC_ASRCNCR_CHANNEL_COUNTER(counter, index)

ASRC_ASRCFG_PRE_MODE_MASK(index)

ASRC_ASRCFG_PRE_MODE(mode, index)

ASRC_ASRCFG_POST_MODE_MASK(index)

ASRC_ASRCFG_POST_MODE(mode, index)

ASRC_ASRCFG_INIT_DONE_MASK(index)

ASRC_ASRC_SR_INPUT_CLOCK_SOURCE_MASK(index)

ASRC_ASRC_SR_INPUT_CLOCK_SOURCE(source, index)

ASRC_ASRC_SR_OUTPUT_CLOCK_SOURCE_MASK(index)

ASRC_ASRC_SR_OUTPUT_CLOCK_SOURCE(source, index)
```

```
ASRC_ASRCDR_INPUT_PRESCALER_MASK(index)
ASRC_ASRCDR_INPUT_PRESCALER(prescaler, index)
ASRC_ASRCDR_INPUT_DIVIDER_MASK(index)
ASRC_ASRCDR_INPUT_DIVIDER(divider, index)
ASRC_ASRCDR_OUTPUT_PRESCALER_MASK(index)
ASRC_ASRCDR_OUTPUT_PRESCALER(prescaler, index)
ASRC_ASRCDR_OUTPUT_DIVIDER_MASK(index)
ASRC_ASRCDR_OUTPUT_DIVIDER(divider, index)
ASRC_ASRCDR_OUTPUT_CLOCK_DIVIDER_PRESCALER(value, index)
ASRC_ASRCDR_INPUT_CLOCK_DIVIDER_PRESCALER(value, index)
ASRC_IDEAL_RATIO_HIGH(base, index)
ASRC_IDEAL_RATIO_LOW(base, index)
ASRC_ASRMCR(base, index)
ASRC_ASRMCR1(base, index)
ASRC_ASRDI(base, index)
ASRC_ASRDO(base, index)
ASRC_ASRDI_ADDR(base, index)
ASRC_ASRDO_ADDR(base, index)
ASRC_ASRFST_ADDR(base, index)
ASRC_GET_CHANNEL_COUNTER(base, index)
struct _asrc_channel_pair_config
    #include <fsl_asrc.h> asrc channel pair configuration
```

Public Members

```
asrc_audio_channel_t audioDataChannels
    audio data channel numbers
asrc_clock_source_t inClockSource
    input clock source, reference the clock source definition in SOC header file
uint32_t inSourceClock_Hz
    input source clock frequency
asrc_clock_source_t outClockSource
    output clock source, reference the clock source definition in SOC header file
uint32_t outSourceClock_Hz
    output source clock frequency
asrc_ratio_t sampleRateRatio
    sample rate ratio type
```

asrc_data_width_t inDataWidth
input data width

asrc_data_align_t inDataAlign
input data alignment

asrc_data_width_t outDataWidth
output data width

asrc_data_align_t outDataAlign
output data alignment

asrc_sign_extension_t outSignExtension
output extension

uint8_t outFifoThreshold
output fifo threshold

uint8_t inFifoThreshold
input fifo threshold

bool bufStallWhenFifoEmptyFull
stall Pair A conversion in case of Buffer near empty full condition

struct __asrc_transfer
#include <fsl_asrc.h> SAI transfer structure.

Public Members

void *inData
Data address to convert.

size_t inDataSize
input data size.

void *outData
Data address to store converted data

size_t outDataSize
output data size.

struct __asrc_in_handle
#include <fsl_asrc.h> asrc in handler

Public Members

asrc_transfer_callback_t callback
Callback function called at convert complete

uint32_t sampleWidth
data width

uint32_t sampleMask
data mask

uint32_t fifoThreshold
fifo threshold

uint8_t *asrcQueue[(4U)]
Transfer queue storing queued transfer

```
size_t transferSamples[(4U)]
    Data bytes need to convert
volatile uint8_t queueUser
    Index for user to queue transfer
volatile uint8_t queueDriver
    Index for driver to get the transfer data and size
struct _asrc_out_handle
    #include <fsl_asrc.h> output handler
```

Public Members

```
asrc_transfer_callback_t callback
    Callback function called at convert complete
uint32_t sampleWidth
    data width
uint32_t fifoThreshold
    fifo threshold
uint8_t *asrcQueue[(4U)]
    Transfer queue storing queued transfer
size_t transferSamples[(4U)]
    Data bytes need to convert
volatile uint8_t queueUser
    Index for user to queue transfer
volatile uint8_t queueDriver
    Index for driver to get the transfer data and size
struct _asrc_handle
    #include <fsl_asrc.h> ASRC handle structure.
```

Public Members

```
ASRC_Type *base
    base address
uint32_t state
    Transfer status
void *userData
    Callback parameter passed to callback function
asrc_audio_channel_t audioDataChannels
    audio channel number
asrc_channel_pair_t channelPair
    channel pair mask
asrc_in_handle_t in
    asrc input handler
asrc_out_handle_t out
    asrc output handler
```


2.6 ASRC EDMA Driver

```
void ASRC__TransferInCreateHandleEDMA(ASRC_Type *base, asrc_edma_handle_t *handle,  
                                     asrc_channel_pair_t channelPair,  
                                     asrc_edma_callback_t callback, edma_handle_t  
                                     *inDmaHandle, const asrc_p2p_edma_config_t  
                                     *periphConfig, void *userData)
```

Initializes the ASRC IN eDMA handle.

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

Parameters

- base – ASRC base pointer.
- channelPair – ASRC channel pair
- handle – ASRC eDMA handle pointer.
- callback – Pointer to user callback function.
- inDmaHandle – DMA handler for ASRC in.
- periphConfig – peripheral configuration.
- userData – User parameter passed to the callback function.

```
void ASRC__TransferOutCreateHandleEDMA(ASRC_Type *base, asrc_edma_handle_t *handle,  
                                       asrc_channel_pair_t channelPair,  
                                       asrc_edma_callback_t callback, edma_handle_t  
                                       *outDmaHandle, const asrc_p2p_edma_config_t  
                                       *periphConfig, void *userData)
```

Initializes the ASRC OUT eDMA handle.

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

Parameters

- base – ASRC base pointer.
- channelPair – ASRC channel pair
- handle – ASRC eDMA handle pointer.
- callback – Pointer to user callback function.
- outDmaHandle – DMA handler for ASRC out.
- periphConfig – peripheral configuration.
- userData – User parameter passed to the callback function.

```
status_t ASRC__TransferSetChannelPairConfigEDMA(ASRC_Type *base, asrc_edma_handle_t  
                                                *handle, asrc_channel_pair_config_t  
                                                *asrcConfig, uint32_t inSampleRate,  
                                                uint32_t outSampleRate)
```

Configures the ASRC P2P channel pair.

Parameters

- base – ASRC base pointer.
- handle – ASRC eDMA handle pointer.
- asrcConfig – asrc configurations.

- inSampleRate – ASRC input sample rate.
- outSampleRate – ASRC output sample rate.

```
uint32_t ASRC_GetOutSamplesSizeEDMA(ASRC_Type *base, asrc_edma_handle_t *handle,  
                                     uint32_t inSampleRate, uint32_t outSampleRate,  
                                     uint32_t inSamplesize)
```

Get output sample buffer size can be transferred by edma.

Note: This API is depends on the ASRC output configuration, should be called after the ASRC_TransferSetChannelPairConfigEDMA.

Parameters

- base – asrc base pointer.
- handle – ASRC channel pair edma handle.
- inSampleRate – input sample rate.
- outSampleRate – output sample rate.
- inSamplesize – input sampleS size.

Return values

output – buffer size in byte.

```
status_t ASRC_TransferEDMA(ASRC_Type *base, asrc_edma_handle_t *handle, asrc_transfer_t  
                           *xfer)
```

Performs a non-blocking ASRC m2m convert using EDMA.

Note: This interface returns immediately after the transfer initiates.

Parameters

- base – ASRC base pointer.
- handle – ASRC eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure.

Return values

- kStatus_Success – Start a ASRC eDMA send successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_ASRCQueueFull – ASRC EDMA driver queue is full.

```
void ASRC_TransferInAbortEDMA(ASRC_Type *base, asrc_edma_handle_t *handle)
```

Aborts a ASRC IN 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 ASRC_TransferTerminalP2PEDMA.

Parameters

- base – ASRC base pointer.
- handle – ASRC eDMA handle pointer.

void ASRC_TransferOutAbortEDMA(ASRC_Type *base, *asrc_edma_handle_t* *handle)

Aborts a ASRC OUT 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 ASRC_TransferTerminalP2PEDMA.

Parameters

- base – ASRC base pointer.
- handle – ASRC eDMA handle pointer.

void ASRC_TransferInTerminalEDMA(ASRC_Type *base, *asrc_edma_handle_t* *handle)

Terminate In ASRC Convert.

This function will clear all transfer slots buffered in the asrc queue. If users only want to abort the current transfer slot, please call ASRC_TransferAbortPP2PEDMA.

Parameters

- base – ASRC base pointer.
- handle – ASRC eDMA handle pointer.

void ASRC_TransferOutTerminalEDMA(ASRC_Type *base, *asrc_edma_handle_t* *handle)

Terminate Out ASRC Convert.

This function will clear all transfer slots buffered in the asrc queue. If users only want to abort the current transfer slot, please call ASRC_TransferAbortPP2PEDMA.

Parameters

- base – ASRC base pointer.
- handle – ASRC eDMA handle pointer.

FSL_ASRC_EDMA_DRIVER_VERSION

Version 2.2.0

typedef struct *_asrc_edma_handle* *asrc_edma_handle_t*

typedef void (**asrc_edma_callback_t*)(ASRC_Type *base, *asrc_edma_handle_t* *handle, *status_t* status, void *userData)

ASRC eDMA transfer callback function for finish and error.

typedef void (**asrc_start_peripheral_t*)(bool start)

ASRC trigger peripheral function pointer.

typedef struct *_asrc_p2p_edma_config* *asrc_p2p_edma_config_t*

destination peripheral configuration

typedef struct *_asrc_in_edma_handle* *asrc_in_edma_handle_t*

@ brief asrc in edma handler

typedef struct *_asrc_out_edma_handle* *asrc_out_edma_handle_t*

@ brief asrc out edma handler

ASRC_XFER_IN_QUEUE_SIZE

ASRC IN edma QUEUE size.

<

ASRC_XFER_OUT_QUEUE_SIZE

struct *_asrc_p2p_edma_config*

#include <*fsl_asrc_edma.h*> destination peripheral configuration

Public Members

asrc_start_peripheral_t startPeripheral
trigger peripheral start

struct *_asrc_in_edma_handle*
#include <fsl_asrc_edma.h> @ brief asrc in edma handler

Public Members

edma_handle_t *inDmaHandle
DMA handler for ASRC in

uint8_t tcd[(4U + 1U) * sizeof(*edma_tcd_t*)]
TCD pool for eDMA send.

uint32_t sampleWidth
input data width

uint32_t fifoThreshold
ASRC input fifo threshold

uint32_t *asrcQueue[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 state
Internal state for ASRC eDMA transfer

const *asrc_p2p_edma_config_t* *peripheralConfig
peripheral configuration pointer

struct *_asrc_out_edma_handle*
#include <fsl_asrc_edma.h> @ brief asrc out edma handler

Public Members

edma_handle_t *outDmaHandle
DMA handler for ASRC out

uint8_t tcd[(((4U) * 2U) + 1U) * sizeof(*edma_tcd_t*)]
TCD pool for eDMA send.

uint32_t sampleWidth
output data width

uint32_t fifoThreshold
ASRC output fifo threshold

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

```
size_t transferSize[((4U) * 2U)]  
    Data bytes need to transfer  
volatile uint8_t queueUser  
    Index for user to queue transfer.  
volatile uint8_t queueDriver  
    Index for driver to get the transfer data and size  
uint32_t state  
    Internal state for ASRC eDMA transfer  
const asrc_p2p_edma_config_t *peripheralConfig  
    peripheral configuration pointer  
struct _asrc_edma_handle  
    #include <fsl_asrc_edma.h> ASRC DMA transfer handle.
```

Public Members

```
asrc_in_edma_handle_t in  
    asrc in handler  
asrc_out_edma_handle_t out  
    asrc out handler  
asrc_channel_pair_t channelPair  
    channel pair  
void *userData  
    User callback parameter  
asrc_edma_callback_t callback  
    Callback for users while transfer finish or error occurs
```

2.7 CAAM: Cryptographic Acceleration and Assurance Module

FSL_CAAM_DRIVER_VERSION

CAAM driver version.

Current version: 2.4.0

Change log:

- Version 2.0.0
 - Initial version
- Version 2.0.1
 - Add Job Ring 2 and 3.
- Version 2.0.2
 - Add Data and Instruction Synchronization Barrier in `caam_input_ring_set_jobs_added()` to make sure that the descriptor will be loaded into CAAM correctly.
- Version 2.0.3
 - Use MACRO instead of numbers in descriptor.
 - Correct descriptor size mask.

- Version 2.1.0
 - Add return codes check and handling.
- Version 2.1.1
 - Add DCACHE support.
- Version 2.1.2
 - Add data offset feature to provide support for mirrored (high-speed) memory.
- Version 2.1.3
 - Fix MISRA-2012 issues.
- Version 2.1.4
 - Fix MISRA-2012 issues.
- Version 2.1.5
 - Support EXTENDED data size for all AES, HASH and RNG operations.
 - Support multiple De-Initialization/Initialization of CAAM driver within one POR event.
- Version 2.1.6
 - Improve DCACHE handling. Requires CAAM used and cached memory set in write-through mode.
- Version 2.2.0
 - Added API for Blob functions and CRC
- Version 2.2.1
 - Fixed AES-CCM decrypt failing with TAG length bigger than 8 byte.
- Version 2.2.2
 - Modify RNG to not reseed with each request.
- Version 2.2.3
 - Fix DCACHE invalidation in CAAM_HASH_Finish().
- Version 2.2.4
 - Fix issue where the outputSize parameter of CAAM_HASH_Finish() has impact on hash calculation.
- Version 2.3.0
 - Add support for SHA HMAC.
- Version 2.3.1
 - Modified function caam_aes_ccm_check_input_args() to allow payload be empty as is specified in NIST800-38C Section 5.3.
- Version 2.3.2
 - Fix MISRA-2012 issues.
- Version 2.4.0
 - Add new APIs for native asymmetric operations (RSA, ECC) instead of only accelerating mathematical primitives and support for black keys and blobs for both symmetric and asymmetric operations.

CAAM status return codes.

Values:

enumerator kStatus_CAAM_Again

Non-blocking function shall be called again.

enumerator kStatus_CAAM_DataOverflow

Input data too big.

enum _caam_job_ring_t

CAAM job ring selection.

Values:

enumerator kCAAM_JobRing0

CAAM Job ring 0

enumerator kCAAM_JobRing1

CAAM Job ring 1

enumerator kCAAM_JobRing2

CAAM Job ring 2

enumerator kCAAM_JobRing3

CAAM Job ring 3

enum _caam_wait_mode

CAAM driver wait mechanism.

Values:

enumerator kCAAM_Blocking

CAAM_Wait blocking mode

enumerator kCAAM_Nonblocking

CAAM Wait non-blocking mode

enum _caam_rng_sample_mode

CAAM RNG sample mode. Used by caam_config_t.

Values:

enumerator kCAAM_RNG_SampleModeVonNeumann

Use von Neumann data in both Entropy shifter and Statistical Checker.

enumerator kCAAM_RNG_SampleModeRaw

Use raw data into both Entropy shifter and Statistical Checker.

enumerator kCAAM_RNG_SampleModeVonNeumannRaw

Use von Neumann data in Entropy shifter. Use raw data into Statistical Checker.

enum _caam_rng_ring_osc_div

CAAM RNG ring oscillator divide. Used by caam_config_t.

Values:

enumerator kCAAM_RNG_RingOscDiv0

Ring oscillator with no divide

enumerator kCAAM_RNG_RingOscDiv2

Ring oscillator divided-by-2.

enumerator kCAAM_RNG_RingOscDiv4

Ring oscillator divided-by-4.

enumerator kCAAM_RNG_RingOscDiv8

Ring oscillator divided-by-8.

enum _caam_priblob

CAAM Private Blob. Used by caam_config_t.

Values:

enumerator kCAAM_PrivSecureBootBlobs

Private secure boot software blobs.

enumerator kCAAM_PrivProvisioningBlobsType1

Private Provisioning Type 1 blobs.

enumerator kCAAM_PrivProvisioningBlobsType2

Private Provisioning Type 2 blobs.

enumerator kCAAM_NormalOperationBlobs

Normal operation blobs.

enum _caam_ext_key_xfr_source

CAAM External Key Transfer command SRC (The source from which the key will be obtained)

Values:

enumerator kCAAM_ExtKeyXfr_KeyRegisterClass1

The Class 1 Key Register is the source.

enumerator kCAAM_ExtKeyXfr_KeyRegisterClass2

The Class 2 Key Register is the source.

enumerator kCAAM_ExtKeyXfr_PkhaRamE

The PKHA E RAM is the source.

enum _caam_ecc_ecdsel

Values:

enumerator kCAAM_ECDSEL_P_192

enumerator kCAAM_ECDSEL_P_224

enumerator kCAAM_ECDSEL_P_256

enumerator kCAAM_ECDSEL_P_384

enumerator kCAAM_ECDSEL_P_521

enumerator kCAAM_ECDSEL_brainpoolP160r1

enumerator kCAAM_ECDSEL_brainpoolP160t1

enumerator kCAAM_ECDSEL_brainpoolP192r1

enumerator kCAAM_ECDSEL_brainpoolP192t1

enumerator kCAAM_ECDSEL_brainpoolP224r1

enumerator kCAAM_ECDSEL_brainpoolP224t1

enumerator kCAAM_ECDSEL_brainpoolP256r1

enumerator kCAAM_ECDSEL_brainpoolP256t1
enumerator kCAAM_ECDSEL_brainpoolP320r1
enumerator kCAAM_ECDSEL_brainpoolP320t1
enumerator kCAAM_ECDSEL_brainpoolP384r1
enumerator kCAAM_ECDSEL_brainpoolP384t1
enumerator kCAAM_ECDSEL_brainpoolP512r1
enumerator kCAAM_ECDSEL_brainpoolP512t1
enumerator kCAAM_ECDSEL_prime192v2
enumerator kCAAM_ECDSEL_prime192v3
enumerator kCAAM_ECDSEL_prime239v1
enumerator kCAAM_ECDSEL_prime239v2
enumerator kCAAM_ECDSEL_prime239v3
enumerator kCAAM_ECDSEL_secp112r1
enumerator kCAAM_ECDSEL_wtls8
enumerator kCAAM_ECDSEL_wtls9
enumerator kCAAM_ECDSEL_secp160k1
enumerator kCAAM_ECDSEL_secp160r1
enumerator kCAAM_ECDSEL_secp160r2
enumerator kCAAM_ECDSEL_secp192k1
enumerator kCAAM_ECDSEL_secp224k1
enumerator kCAAM_ECDSEL_secp256k1
enumerator kCAAM_ECDSEL_B_163
enumerator kCAAM_ECDSEL_B_233
enumerator kCAAM_ECDSEL_B_283
enumerator kCAAM_ECDSEL_B_409
enumerator kCAAM_ECDSEL_B_571
enumerator kCAAM_ECDSEL_K_163
enumerator kCAAM_ECDSEL_K_233
enumerator kCAAM_ECDSEL_K_283
enumerator kCAAM_ECDSEL_K_409
enumerator kCAAM_ECDSEL_K_571
enumerator kCAAM_ECDSEL_wtls1
enumerator kCAAM_ECDSEL_sect113r1

enumerator kCAAM_ECDSEL_c2pnb163v1

enumerator kCAAM_ECDSEL_c2pnb163v2

enumerator kCAAM_ECDSEL_c2pnb163v3

enumerator kCAAM_ECDSEL_sect163r1

enumerator kCAAM_ECDSEL_sect193r1

enumerator kCAAM_ECDSEL_sect193r2

enumerator kCAAM_ECDSEL_sect239k1

typedef struct *caam_job_callback* caam_job_callback_t
CAAM callback function.

typedef enum *caam_job_ring_t* caam_job_ring_t
CAAM job ring selection.

typedef struct *caam_handle_t* caam_handle_t
CAAM handle Specifies jobRing and optionally the user callback function. The user callback functions is invoked only if jobRing interrupt has been enabled by the user. By default the jobRing interrupt is disabled (default job complete test is polling CAAM output ring).

typedef enum *caam_wait_mode* caam_wait_mode_t
CAAM driver wait mechanism.

typedef uint32_t caam_desc_aes_ecb_t[64]
Memory buffer to hold CAAM descriptor for AESA ECB job.

typedef uint32_t caam_desc_aes_cbc_t[64]
Memory buffer to hold CAAM descriptor for AESA CBC job.

typedef uint32_t caam_desc_aes_ctr_t[64]
Memory buffer to hold CAAM descriptor for AESA CTR job.

typedef uint32_t caam_desc_aes_ccm_t[64]
Memory buffer to hold CAAM descriptor for AESA CCM job.

typedef uint32_t caam_desc_aes_gcm_t[64]
Memory buffer to hold CAAM descriptor for AESA GCM job.

typedef uint32_t caam_desc_hash_t[64]
Memory buffer to hold CAAM descriptor for MDHA job or AESA CMAC job.

typedef uint32_t caam_desc_rng_t[64]
Memory buffer to hold CAAM descriptor for RNG jobs.

typedef uint32_t caam_desc_cipher_des_t[64]
Memory buffer to hold CAAM descriptor for DESA jobs.

typedef uint32_t caam_desc_pkha_t[64]
Memory buffer to hold CAAM descriptor for PKHA jobs.

typedef uint32_t caam_desc_pkha_ecc_t[64]
Memory buffer to hold CAAM descriptor for PKHA ECC jobs.

typedef uint32_t caam_desc_key_black_t[64]
Memory buffer to hold CAAM descriptor for performing key blackening jobs.

typedef uint32_t caam_desc_gen_enc_blob_t[64]
Memory buffer to hold CAAM descriptor for performing generating dek blob jobs.

typedef uint32_t caam_desc_gen_dep_blob_t[64]

typedef uint32_t caam_desc_rsa_t[64]

Memory buffer to hold CAAM descriptor for performing generating dek blob jobs.

typedef uint32_t caam_desc_ecc_t[64]

Memory buffer to hold CAAM descriptor for performing generating dek blob jobs.

typedef struct *_caam_job_ring_interface* caam_job_ring_interface_t

typedef enum *_caam_rng_sample_mode* caam_rng_sample_mode_t

CAAM RNG sample mode. Used by caam_config_t.

typedef enum *_caam_rng_ring_osc_div* caam_rng_ring_osc_div_t

CAAM RNG ring oscillator divide. Used by caam_config_t.

typedef enum *_caam_priblob* caam_priblob_t

CAAM Private Blob. Used by caam_config_t.

typedef struct *_caam_config* caam_config_t

CAAM configuration structure.

typedef enum *_caam_ext_key_xfr_source* caam_ext_key_xfr_source_t

CAAM External Key Transfer command SRC (The source from which the key will be obtained)

typedef enum *_caam_ecc_ecdsel* caam_ecc_ecdsel_t

status_t CAAM_Init(CAAM_Type *base, const *caam_config_t* *config)

Initializes the CAAM driver.

This function initializes the CAAM driver, including CAAM's internal RNG.

Parameters

- base – CAAM peripheral base address
- config – Pointer to configuration structure.

Returns

kStatus_Success the CAAM Init has completed with zero termination status word

Returns

kStatus_Fail the CAAM Init has completed with non-zero termination status word

status_t CAAM_Deinit(CAAM_Type *base)

Deinitializes the CAAM driver. This function deinitializes the CAAM driver.

Parameters

- base – CAAM peripheral base address

Returns

kStatus_Success the CAAM Deinit has completed with zero termination status word

Returns

kStatus_Fail the CAAM Deinit has completed with non-zero termination status word

```
void CAAM_GetDefaultConfig(caam_config_t *config)
```

Gets the default configuration structure.

This function initializes the CAAM configuration structure to a default value. The default values are as follows. `caamConfig->rngSampleMode = kCAAM_RNG_SampleModeVonNeumann;` `caamConfig->rngRingOscDiv = kCAAM_RNG_RingOscDiv4;`

Parameters

- `config` – **[out]** Pointer to configuration structure.

```
status_t CAAM_Wait(CAAM_Type *base, caam_handle_t *handle, uint32_t *descriptor,
                  caam_wait_mode_t mode)
```

Wait for a CAAM job to complete.

This function polls CAAM output ring for a specific job.

The CAAM job ring is specified by the `jobRing` field in the `caam_handle_t` structure. The job to be waited is specified by its descriptor address.

This function has two modes, determined by the `mode` argument. In blocking mode, the function polls the specified `jobRing` until the descriptor is available in the CAAM output job ring. In non-blocking mode, it polls the output ring once and returns status immediately.

The function can be called from multiple threads or interrupt service routines, as internally it uses global critical section (global interrupt disable enable) to protect its operation against concurrent accesses. The global interrupt is disabled only when the descriptor is found in the output ring, for a very short time, to remove the descriptor from the output ring safely.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Data structure with CAAM `jobRing` used for this request
- `descriptor` –
- `mode` – Blocking and non-blocking mode. Zero is blocking. Non-zero is non-blocking.

Returns

`kStatus_Success` the CAAM job has completed with zero job termination status word

Returns

`kStatus_Fail` the CAAM job has completed with non-zero job termination status word

Returns

`kStatus_Again` In non-blocking mode, the job is not ready in the CAAM Output Ring

```
status_t CAAM_ExternalKeyTransfer(CAAM_Type *base, caam_handle_t *handle,
                                caam_ext_key_xfr_source_t keySource, size_t keySize)
```

External Key Transfer.

This function loads the given key source to an CAAM external destination via a private interface, such as Inline Encryption Engine IEE Private Key bus.

The CAAM job ring is specified by the `jobRing` field in the `caam_handle_t` structure.

This function is blocking.

Parameters

- `base` – CAAM peripheral base address

- `handle` – Data structure with CAAM jobRing used for this request.
- `keySource` – The source from which the key will be obtained.
- `keySize` – Size of the key in bytes.

Returns

`kStatus_Success` the CAAM job has completed with zero job termination status word

Returns

`kStatus_Fail` the CAAM job has completed with non-zero job termination status word

```
struct __caam_job_callback
#include <fsl_caam.h> CAAM callback function.
```

Public Members

```
void (*JobCompleted)(void *userData)
    CAAM Job complete callback
```

```
struct __caam_handle_t
#include <fsl_caam.h> CAAM handle Specifies jobRing and optionally the user callback function. The user callback functions is invoked only if jobRing interrupt has been enabled by the user. By default the jobRing interrupt is disabled (default job complete test is polling CAAM output ring).
```

Public Members

```
caam_job_callback_t callback
    Callback function
```

```
void *userData
    Parameter for CAAM job complete callback
```

```
struct __caam_job_ring_interface
#include <fsl_caam.h>
```

```
struct __caam_config
#include <fsl_caam.h> CAAM configuration structure.
```

Public Members

```
caam_rng_sample_mode_t rngSampleMode
    RTMCTL Sample Mode.
```

```
caam_rng_ring_osc_div_t rngRingOscDiv
    RTMCTL Oscillator Divide.
```

```
bool scfgrLockTrngProgramMode
    SCFGR Lock TRNG Program Mode.
```

```
bool scfgrEnableRandomDataBuffer
    SCFGR Enable random data buffer.
```

```
bool scfgrRandomRngStateHandle0
    SCFGR Random Number Generator State Handle 0.
```

`bool scfgrRandomDpaResistance`

SCFGR Random Differential Power Analysis Resistance.

`caam_priblob_t scfgrPriblob`

SCFGR Private Blob.

2.8 CAAM AES driver

`status_t CAAM_AES_EncryptEcb(CAAM_Type *base, caam_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t *key, size_t keySize)`

Encrypts AES using the ECB block mode.

Encrypts AES using the ECB block mode.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies jobRing.
- `plaintext` – Input plain text to encrypt
- `ciphertext` – **[out]** Output cipher text
- `size` – Size of input and output data in bytes. Must be multiple of 16 bytes.
- `key` – Input key to use for encryption
- `keySize` – Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from encrypt operation

`status_t CAAM_AES_DecryptEcb(CAAM_Type *base, caam_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t *key, size_t keySize)`

Decrypts AES using ECB block mode.

Decrypts AES using ECB block mode.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies jobRing.
- `ciphertext` – Input cipher text to decrypt
- `plaintext` – **[out]** Output plain text
- `size` – Size of input and output data in bytes. Must be multiple of 16 bytes.
- `key` – Input key.
- `keySize` – Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from decrypt operation

`status_t CAAM_AES_EncryptEcbExtended(CAAM_Type *base, caam_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t *key, size_t keySize, caam_key_type_t blackKeyType)`

Encrypts AES using the ECB block mode using black key.

Encrypts AES using the ECB block mode.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- blackKeyType – Type of black key

Returns

Status from encrypt operation

```
status_t CAAM_AES_DecryptEcbExtended(CAAM_Type *base, caam_handle_t *handle, const
                                     uint8_t *ciphertext, uint8_t *plaintext, size_t size,
                                     const uint8_t *key, size_t keySize, caam_key_type_t
                                     blackKeyType)
```

Decrypts AES using ECB block mode using black key.

Decrypts AES using ECB block mode.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- key – Input key.
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- blackKeyType – Type of black key

Returns

Status from decrypt operation

```
status_t CAAM_AES_EncryptCbc(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                              *plaintext, uint8_t *ciphertext, size_t size, const uint8_t
                              iv[16], const uint8_t *key, size_t keySize)
```

Encrypts AES using CBC block mode.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from encrypt operation

```
status_t CAAM_AES_DecryptCbc(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                             *ciphertext, uint8_t *plaintext, size_t size, const uint8_t
                             iv[16], const uint8_t *key, size_t keySize)
```

Decrypts AES using CBC block mode.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.
- key – Input key to use for decryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from decrypt operation

```
status_t CAAM_AES_EncryptCbcExtended(CAAM_Type *base, caam_handle_t *handle, const
                                     uint8_t *plaintext, uint8_t *ciphertext, size_t size,
                                     const uint8_t iv[16], const uint8_t *key, size_t
                                     keySize, caam_key_type_t blackKeyType)
```

Encrypts AES using CBC block mode using black key.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- blackKeyType – Type of black key

Returns

Status from encrypt operation

```
status_t CAAM_AES_DecryptCbcExtended(CAAM_Type *base, caam_handle_t *handle, const
                                     uint8_t *ciphertext, uint8_t *plaintext, size_t size,
                                     const uint8_t iv[16], const uint8_t *key, size_t
                                     keySize, caam_key_type_t blackKeyType)
```

Decrypts AES using CBC block mode using black key.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- ciphertext – Input cipher text to decrypt

- plaintext – **[out]** Output plain text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.
- key – Input key to use for decryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- blackKeyType – Type of black key

Returns

Status from decrypt operation

```
status_t CAAM_AES_CryptCtr(CAAM_Type *base, caam_handle_t *handle, const uint8_t *input,
                           uint8_t *output, size_t size, uint8_t counter[16], const uint8_t
                           *key, size_t keySize, uint8_t counterlast[16], size_t *szLeft)
```

Encrypts or decrypts AES using CTR block mode.

Encrypts or decrypts AES using CTR block mode. AES CTR mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- input – Input data for CTR block mode
- output – **[out]** Output data for CTR block mode
- size – Size of input and output data in bytes
- counter – **[inout]** Input counter (updates on return)
- key – Input key to use for forward AES cipher
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- counterlast – **[out]** Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
- szLeft – **[out]** Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used.

Returns

Status from encrypt operation

```
status_t CAAM_AES_CryptCtrExtended(CAAM_Type *base, caam_handle_t *handle, const
                                   uint8_t *input, uint8_t *output, size_t size, uint8_t
                                   counter[16], const uint8_t *key, size_t keySize, uint8_t
                                   counterlast[16], size_t *szLeft, caam_key_type_t
                                   blackKeyType)
```

Encrypts or decrypts AES using CTR block mode using black key.

Encrypts or decrypts AES using CTR block mode. AES CTR mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

Parameters

- base – CAAM peripheral base address

- `handle` – Handle used for this request. Specifies `jobRing`.
- `input` – Input data for CTR block mode
- `output` – **[out]** Output data for CTR block mode
- `size` – Size of input and output data in bytes
- `counter` – **[inout]** Input counter (updates on return)
- `key` – Input key to use for forward AES cipher
- `keySize` – Size of the input key, in bytes. Must be 16, 24, or 32.
- `counterlast` – **[out]** Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
- `szLeft` – **[out]** Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used.
- `blackKeyType` – Type of black key

Returns

Status from encrypt operation

```
status_t CAAM_AES_EncryptTagCcm(CAAM_Type *base, caam_handle_t *handle, const uint8_t
    *plaintext, uint8_t *ciphertext, size_t size, const uint8_t
    *iv, size_t ivSize, const uint8_t *aad, size_t aadSize, const
    uint8_t *key, size_t keySize, uint8_t *tag, size_t tagSize)
```

Encrypts AES and tags using CCM block mode.

Encrypts AES and optionally tags using CCM block mode.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies `jobRing`.
- `plaintext` – Input plain text to encrypt
- `ciphertext` – **[out]** Output cipher text.
- `size` – Size of input and output data in bytes. Zero means authentication only.
- `iv` – Nonce
- `ivSize` – Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
- `aad` – Input additional authentication data. Can be NULL if `aadSize` is zero.
- `aadSize` – Input size in bytes of AAD. Zero means data mode only (authentication skipped).
- `key` – Input key to use for encryption
- `keySize` – Size of the input key, in bytes. Must be 16, 24, or 32.
- `tag` – **[out]** Generated output tag. Set to NULL to skip tag processing.
- `tagSize` – Input size of the tag to generate, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.

Returns

Status from encrypt operation

```
status_t CAAM_AES_DecryptTagCcm(CAAM_Type *base, caam_handle_t *handle, const uint8_t
    *ciphertext, uint8_t *plaintext, size_t size, const uint8_t
    *iv, size_t ivSize, const uint8_t *aad, size_t aadSize, const
    uint8_t *key, size_t keySize, const uint8_t *tag, size_t
    tagSize)
```

Decrypts AES and authenticates using CCM block mode.

Decrypts AES and optionally authenticates using CCM block mode.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies jobRing.
- `ciphertext` – Input cipher text to decrypt
- `plaintext` – **[out]** Output plain text.
- `size` – Size of input and output data in bytes. Zero means authentication data only.
- `iv` – Nonce
- `ivSize` – Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
- `aad` – Input additional authentication data. Can be NULL if `aadSize` is zero.
- `aadSize` – Input size in bytes of AAD. Zero means data mode only (authentication data skipped).
- `key` – Input key to use for decryption
- `keySize` – Size of the input key, in bytes. Must be 16, 24, or 32.
- `tag` – Received tag. Set to NULL to skip tag processing.
- `tagSize` – Input size of the received tag to compare with the computed tag, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.

Returns

Status from decrypt operation

```
status_t CAAM_AES_EncryptTagCcmExtended(CAAM_Type *base, caam_handle_t *handle,  
                                         const uint8_t *plaintext, uint8_t *ciphertext,  
                                         size_t size, const uint8_t *iv, size_t ivSize, const  
                                         uint8_t *aad, size_t aadSize, const uint8_t *key,  
                                         size_t keySize, uint8_t *tag, size_t tagSize,  
                                         caam_key_type_t blackKeyType)
```

Encrypts AES and tags using CCM block mode using black key.

Encrypts AES and optionally tags using CCM block mode.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies jobRing.
- `plaintext` – Input plain text to encrypt
- `ciphertext` – **[out]** Output cipher text.
- `size` – Size of input and output data in bytes. Zero means authentication only.
- `iv` – Nonce
- `ivSize` – Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
- `aad` – Input additional authentication data. Can be NULL if `aadSize` is zero.
- `aadSize` – Input size in bytes of AAD. Zero means data mode only (authentication skipped).
- `key` – Input key to use for encryption

- `keySize` – Size of the input key, in bytes. Must be 16, 24, or 32.
- `tag` – **[out]** Generated output tag. Set to NULL to skip tag processing.
- `tagSize` – Input size of the tag to generate, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.
- `blackKeyType` – Type of black key

Returns

Status from encrypt operation

```
status_t CAAM_AES_DecryptTagCcmExtended(CAAM_Type *base, caam_handle_t *handle,
                                         const uint8_t *ciphertext, uint8_t *plaintext,
                                         size_t size, const uint8_t *iv, size_t ivSize, const
                                         uint8_t *aad, size_t aadSize, const uint8_t *key,
                                         size_t keySize, const uint8_t *tag, size_t tagSize,
                                         caam_key_type_t blackKeyType)
```

Decrypts AES and authenticates using CCM block mode using black key.

Decrypts AES and optionally authenticates using CCM block mode.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies jobRing.
- `ciphertext` – Input cipher text to decrypt
- `plaintext` – **[out]** Output plain text.
- `size` – Size of input and output data in bytes. Zero means authentication data only.
- `iv` – Nonce
- `ivSize` – Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
- `aad` – Input additional authentication data. Can be NULL if `aadSize` is zero.
- `aadSize` – Input size in bytes of AAD. Zero means data mode only (authentication data skipped).
- `key` – Input key to use for decryption
- `keySize` – Size of the input key, in bytes. Must be 16, 24, or 32.
- `tag` – Received tag. Set to NULL to skip tag processing.
- `tagSize` – Input size of the received tag to compare with the computed tag, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.
- `blackKeyType` – Type of black key

Returns

Status from decrypt operation

```
status_t CAAM_AES_EncryptTagGcm(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                                *plaintext, uint8_t *ciphertext, size_t size, const uint8_t
                                *iv, size_t ivSize, const uint8_t *aad, size_t aadSize, const
                                uint8_t *key, size_t keySize, uint8_t *tag, size_t tagSize)
```

Encrypts AES and tags using GCM block mode.

Encrypts AES and optionally tags using GCM block mode. If plaintext is NULL, only the GHASH is calculated and output in the 'tag' field.

Parameters

- `base` – CAAM peripheral base address

- handle – Handle used for this request. Specifies jobRing.
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text.
- size – Size of input and output data in bytes
- iv – Input initial vector
- ivSize – Size of the IV
- aad – Input additional authentication data
- aadSize – Input size in bytes of AAD
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – **[out]** Output hash tag. Set to NULL to skip tag processing.
- tagSize – Input size of the tag to generate, in bytes. Must be 4,8,12,13,14,15 or 16.

Returns

Status from encrypt operation

```
status_t CAAM_AES_DecryptTagGcm(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                                *ciphertext, uint8_t *plaintext, size_t size, const uint8_t
                                *iv, size_t ivSize, const uint8_t *aad, size_t aadSize, const
                                uint8_t *key, size_t keySize, const uint8_t *tag, size_t
                                tagSize)
```

Decrypts AES and authenticates using GCM block mode.

Decrypts AES and optionally authenticates using GCM block mode. If ciphertext is NULL, only the GHASH is calculated and compared with the received GHASH in 'tag' field.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text.
- size – Size of input and output data in bytes
- iv – Input initial vector
- ivSize – Size of the IV
- aad – Input additional authentication data
- aadSize – Input size in bytes of AAD
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – Input hash tag to compare. Set to NULL to skip tag processing.
- tagSize – Input size of the tag, in bytes. Must be 4, 8, 12, 13, 14, 15, or 16.

Returns

Status from decrypt operation

```
status_t CAAM_AES_EncryptTagGcmExtended(CAAM_Type *base, caam_handle_t *handle,
                                         const uint8_t *plaintext, uint8_t *ciphertext,
                                         size_t size, const uint8_t *iv, size_t ivSize, const
                                         uint8_t *aad, size_t aadSize, const uint8_t *key,
                                         size_t keySize, uint8_t *tag, size_t tagSize,
                                         caam_key_type_t blackKeyType)
```

Encrypts AES and tags using GCM block mode using black key.

Encrypts AES and optionally tags using GCM block mode. If plaintext is NULL, only the GHASH is calculated and output in the 'tag' field. Uses black key.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text.
- size – Size of input and output data in bytes
- iv – Input initial vector
- ivSize – Size of the IV
- aad – Input additional authentication data
- aadSize – Input size in bytes of AAD
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – **[out]** Output hash tag. Set to NULL to skip tag processing.
- tagSize – Input size of the tag to generate, in bytes. Must be 4,8,12,13,14,15 or 16.
- blackenKeyType – Type of black key

Returns

Status from encrypt operation

```
status_t CAAM_AES_DecryptTagGcmExtended(CAAM_Type *base, caam_handle_t *handle,
                                         const uint8_t *ciphertext, uint8_t *plaintext,
                                         size_t size, const uint8_t *iv, size_t ivSize, const
                                         uint8_t *aad, size_t aadSize, const uint8_t *key,
                                         size_t keySize, const uint8_t *tag, size_t tagSize,
                                         caam_key_type_t blackKeyType)
```

Decrypts AES and authenticates using GCM block mode using black key.

Decrypts AES and optionally authenticates using GCM block mode. If ciphertext is NULL, only the GHASH is calculated and compared with the received GHASH in 'tag' field. Uses black key.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text.
- size – Size of input and output data in bytes
- iv – Input initial vector

- `ivSize` – Size of the IV
- `aad` – Input additional authentication data
- `aadSize` – Input size in bytes of AAD
- `key` – Input key to use for encryption
- `keySize` – Size of the input key, in bytes. Must be 16, 24, or 32.
- `tag` – Input hash tag to compare. Set to NULL to skip tag processing.
- `tagSize` – Input size of the tag, in bytes. Must be 4, 8, 12, 13, 14, 15, or 16.
- `blackenKeyType` – Type of black key

Returns

Status from decrypt operation

`CAAM_AES_BLOCK_SIZE`

AES block size in bytes

2.9 CAAM Key Blankening driver

`size_t CAAM_BLACK_KeyBlackenSize(caam_fifost_type_t fifostType, size_t dataSize)`

Return size of blacken key based on encryption type and data to encrypt size.

Parameters

- `fifostType` – Type of AES-CBC or AEC-CCM to encrypt plaintext
- `dataSize` – Size of data to be encrypted

Returns

`size_t` Size of blacken key.

`status_t CAAM_BLACK_GetKeyBlacken(CAAM_Type *base, caam_handle_t *handle, const uint8_t *data, size_t dataSize, caam_fifost_type_t fifostType, uint8_t *blackdata)`

Construct a black key.

This function constructs a job descriptor capable of performing a key blackening operation on a plaintext secure memory resident object.

Parameters

- `base` – CAAM peripheral base address
- `handle` – jobRing used for this request
- `data` – Pointer address uses to pointed the plaintext.
- `dataSize` – Size of the buffer pointed by the data parameter
- `fifostType` – Type of AES-CBC or AEC-CCM to encrypt plaintext
- `blackdata` – **[out]** Pointer address uses to pointed the black key

Returns

Status of the request

2.10 CAAM Blob driver

enum _caam_fifost_type
CAAM FIFOST types.

Values:

enumerator kCAAM_FIFOST_Type_Ecb_Jkek

Key Register, encrypted using AES-ECB with the job descriptor key encryption key.

enumerator kCAAM_FIFOST_Type_Ecb_Tkek

Key Register, encrypted using AES-ECB with the trusted descriptor key encryption key.

enumerator kCAAM_FIFOST_Type_Ccm_Jkek

Key Register, encrypted using AES-CCM with the job descriptor key encryption key.

enumerator kCAAM_FIFOST_Type_Ccm_Tkek

Key register, encrypted using AES-CCM with the trusted descriptor key encryption key.

enum _caam_desc_type
CAAM descriptor types.

Values:

enumerator kCAAM_Descriptor_Type_Ecb_Jkek

Key Register, encrypted using AES-ECB with the job descriptor key encryption key.

enumerator kCAAM_Descriptor_Type_Ecb_Tkek

Key Register, encrypted using AES-ECB with the trusted descriptor key encryption key.

enumerator kCAAM_Descriptor_Type_Ccm_Jkek

Key Register, encrypted using AES-CCM with the job descriptor key encryption key.

enumerator kCAAM_Descriptor_Type_Ccm_Tkek

Key register, encrypted using AES-CCM with the trusted descriptor key encryption key.

enum _caam_key_type
CAAM key types.

Values:

enumerator kCAAM_Key_Type_None

enumerator kCAAM_Key_Type_Ecb_Jkek

Key Register, encrypted using AES-ECB with the job descriptor key encryption key.

enumerator kCAAM_Key_Type_Ecb_Tkek

Key Register, encrypted using AES-ECB with the trusted descriptor key encryption key.

enumerator kCAAM_Key_Type_Ccm_Jkek

Key Register, encrypted using AES-CCM with the job descriptor key encryption key.

enumerator kCAAM_Key_Type_Ccm_Tkek

Key register, encrypted using AES-CCM with the trusted descriptor key encryption key.

enum _caam_ecc_encryption_type
CAAM ecc encryption types.

Values:

enumerator kCAAM_Ecc_Encryption_Type_None

enumerator kCAAM_Ecc_Encryption_Type_Ecb_Jkek

Key Register, encrypted using AES-ECB with the job descriptor key encryption key.

enumerator kCAAM_Ecc_Encryption_Type_Ccm_Jkek

Key Register, encrypted using AES-CCM with the job descriptor key encryption key.

enum *_caam_rsa_key_type*

CAAM rsa key encryption types.

Values:

enumerator *kCAAM_Rsa_Key_Type_None*

enumerator *kCAAM_Rsa_Key_Type_Ecb_Jkek*

Key Register, encrypted using AES-ECB with the job descriptor key encryption key.

enumerator *kCAAM_Rsa_Key_Type_Ccm_Jkek*

Key Register, encrypted using AES-CCM with the job descriptor key encryption key.

enum *_caam_rsa_encryption_type*

CAAM rsa encryption types.

Values:

enumerator *kCAAM_Rsa_Encryption_Type_None*

enumerator *kCAAM_Rsa_Encryption_Type_Ecb_Jkek*

Key Register, encrypted using AES-ECB with the job descriptor key encryption key.

enumerator *kCAAM_Rsa_Encryption_Type_Ecb_Tkek*

Key Register, encrypted using AES-ECB with the trusted descriptor key encryption key.

enumerator *kCAAM_Rsa_Encryption_Type_Ccm_Jkek*

Key Register, encrypted using AES-CCM with the job descriptor key encryption key.

enumerator *kCAAM_Rsa_Encryption_Type_Ccm_Tkek*

Key register, encrypted using AES-CCM with the trusted descriptor key encryption key.

enum *_caam_rsa_format_type*

Values:

enumerator *kCAAM_Rsa_Format_Type_None*

No formatting

enumerator *kCAAM_Rsa_Format_Type_PKCS1*

EME-PKCS1-v1_5 encryption decoding function

typedef enum *_caam_fifost_type* *caam_fifost_type_t*

CAAM FIFOST types.

typedef enum *_caam_desc_type* *caam_desc_type_t*

CAAM descriptor types.

typedef enum *_caam_key_type* *caam_key_type_t*

CAAM key types.

typedef enum *_caam_ecc_encryption_type* *caam_ecc_encryption_type_t*

CAAM ecc encryption types.

typedef enum *_caam_rsa_key_type* *caam_rsa_key_type_t*

CAAM rsa key encryption types.

typedef enum *_caam_rsa_encryption_type* *caam_rsa_encryption_type_t*

CAAM rsa encryption types.

typedef enum *_caam_rsa_format_type* *caam_rsa_format_type_t*

```
status_t CAAM_RedBlob_Encapsule(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                                *keyModifier, size_t keyModifierSize, const uint8_t *data,
                                size_t dataSize, uint8_t *blob_data)
```

Construct a encrypted Red Blob.

This function constructs a job descriptor capable of performing a encrypted blob operation on a plaintext object.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- keyModifier – Address of the random key modifier generated by RNG
- keyModifierSize – Size of keyModifier buffer in bytes
- data – Data address
- dataSize – Size of the buffer pointed by the data parameter
- blob_data – **[out]** Output blob data adress

Returns

Status of the request

```
status_t CAAM_RedBlob_Decapsule(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                                *keyModifier, size_t keyModifierSize, const uint8_t
                                *blob_data, uint8_t *data, size_t dataSize)
```

Decrypt red blob.

This function constructs a job descriptor capable of performing decrypting red blob .

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- keyModifier – Address of the random key modifier generated by RNG
- keyModifierSize – Size of keyModifier buffer in bytes
- blob_data – Address of blob data
- data – **[out]** Output data adress.
- dataSize – Size of the buffer pointed by the data parameter in bytes

Returns

Status of the request

```
status_t CAAM_BlackBlob_Encapsule(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                                   *keyModifier, size_t keyModifierSize, const uint8_t *data,
                                   size_t dataSize, uint8_t *blob_data, caam_desc_type_t
                                   blackKeyType)
```

Construct a encrypted Black Blob.

This function constructs a job descriptor capable of performing a encrypted blob operation on a plaintext object.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- keyModifier – Address of the random key modifier generated by RNG
- keyModifierSize – Size of keyModifier buffer in bytes

- data – Data address
- dataSize – Size of the buffer pointed by the data parameter
- blob_data – **[out]** Output blob data adress
- blackKeyType – Type of black key see enum caam_desc_type_t for more info

Returns

Status of the request

```
status_t CAAM_BlackBlob_Decapsule(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                                *keyModifier, size_t keyModifierSize, const uint8_t
                                *blob_data, uint8_t *data, size_t dataSize,
                                caam_desc_type_t blackKeyType)
```

Construct a decrypted black blob.

This function constructs a job descriptor capable of performing decrypting black blob.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- keyModifier – Address of the random key modifier generated by RNG
- keyModifierSize – Size of keyModifier buffer in bytes
- blob_data – Address of blob data
- data – **[out]** Output data adress.
- dataSize – Size of the buffer pointed by the data parameter in bytes
- blackKeyType – Type of black key see enum caam_desc_type_t for more info

Returns

Status of the request

2.11 CAAM CRC driver

```
status_t CAAM_CRC_Init(CAAM_Type *base, caam_handle_t *handle, caam_hash_ctx_t *ctx,
                       caam_crc_algo_t algo, const uint8_t *polynomial, size_t
                       polynomialSize, caam_aai_crc_alg_t mode)
```

Initialize CRC context.

This function initializes the CRC context. polynomial shall be supplied if the underlaying algorithm is kCAAM_CrcCUSTPOLY. polynomial shall be NULL if the underlaying algorithm is kCAAM_CrcIEEE or kCAAM_CrciSCSI.

This functions is used to initialize the context for CAAM_CRC API

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request.
- ctx – **[out]** Output crc context
- algo – Underlaying algorithm to use for CRC computation
- polynomial – CRC polynomial (NULL if underlaying algorithm is kCAAM_CrcIEEE or kCAAM_CrciSCSI)
- polynomialSize – Size of polynomial in bytes (0u if underlaying algorithm is kCAAM_CrcIEEE or kCAAM_CrciSCSI)

- `mode` – Specify how CRC engine manipulates its input and output data

Returns

Status of initialization

`status_t` CAAM_CRC_Update(*caam_hash_ctx_t* *ctx, const `uint8_t` *input, `size_t` inputSize)

Add data to current CRC.

Add data to current CRC. This can be called repeatedly. The functions blocks. If it returns `kStatus_Success`, the running CRC has been updated (CAAM has processed the input data), so the memory at input pointer can be released back to system. The context is updated with the running CRC and with all necessary information to support possible context switch.

Parameters

- `ctx` – **[inout]** CRC context
- `input` – Input data
- `inputSize` – Size of input data in bytes

Returns

Status of the crc update operation

`status_t` CAAM_CRC_Finish(*caam_hash_ctx_t* *ctx, `uint8_t` *output, `size_t` *outputSize)

Finalize CRC.

Outputs the final CRC (computed by CAAM_CRC_Update()) and erases the context.

Parameters

- `ctx` – **[inout]** Input crc context
- `output` – **[out]** Output crc data
- `outputSize` – **[out]** Output parameter storing the size of the output crc in bytes

Returns

Status of the crc finish operation

`status_t` CAAM_CRC(CAAM_Type *base, *caam_handle_t* *handle, `caam_crc_algo_t` algo, `caam_aai_crc_alg_t` mode, const `uint8_t` *input, `size_t` inputSize, const `uint8_t` *polynomial, `size_t` polynomialSize, `uint8_t` *output, `size_t` *outputSize)

Create CRC on given data.

Perform CRC in one function call.

Polynomial shall be supplied if underlaying algorithm is `kCAAM_CrcCUSTPOLY`. Polynomial shall be `NULL` if underlaying algorithm is `kCAAM_CrcIEEE` or `kCAAM_CrciSCSI`.

The function is blocking.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request.
- `algo` – Underlaying algorithm to use for crc computation.
- `mode` – Specify how CRC engine manipulates its input and output data.
- `input` – Input data
- `inputSize` – Size of input data in bytes
- `polynomial` – CRC polynomial (`NULL` if underlaying algorithm is `kCAAM_CrcIEEE` or `kCAAM_CrciSCSI`)

- polynomialSize – Size of input polynomial in bytes (0U if underlaying algorithm is kCAAM_CrcIEEE or kCAAM_CrciSCSI)
- output – **[out]** Output crc data
- outputSize – **[out]** Output parameter storing the size of the output crc in bytes

Returns

Status of the one call crc operation.

```
status_t CAAM_CRC_NonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                             caam_desc_hash_t descriptor, caam_crc_algo_t algo,  
                             caam_aai_crc_alg_t mode, const uint8_t *input, size_t  
                             inputSize, const uint8_t *polynomial, size_t polynomialSize,  
                             uint8_t *output, size_t *outputSize)
```

Create CRC on given data.

Perform CRC in one function call.

Polynomial shall be supplied if underlaying algorithm is kCAAM_CrcCUSTPOLY. Polynomial shall be NULL if underlaying algorithm is kCAAM_CrcIEEE or kCAAM_CrciSCSI.

The function is non-blocking. The request is scheduled at CAAM.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request.
- descriptor – **[out]** Memory for the CAAM descriptor.
- algo – Underlaying algorithm to use for crc computation.
- mode – Specify how CRC engine manipulates its input and output data.
- input – Input data
- inputSize – Size of input data in bytes
- polynomial – CRC polynomial (NULL if underlaying algorithm is kCAAM_CrcIEEE or kCAAM_CrciSCSI)
- polynomialSize – Size of input polynomial in bytes (0U if underlaying algorithm is kCAAM_CrcIEEE or kCAAM_CrciSCSI)
- output – **[out]** Output crc data
- outputSize – **[out]** Output parameter storing the size of the output crc in bytes

Returns

Status of the one call crc operation.

2.12 CAAM DES driver

```
status_t CAAM_DES_EncryptEcb(CAAM_Type *base, caam_handle_t *handle, const uint8_t  
                             *plaintext, uint8_t *ciphertext, size_t size, const uint8_t  
                             key[8U])
```

Encrypts DES using ECB block mode.

Encrypts DES using ECB block mode.

Parameters

- base – CAAM peripheral base address

- handle – Handle used for this request. Specifies jobRing.
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key – Input key to use for encryption

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES_DecryptEcb(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                             *ciphertext, uint8_t *plaintext, size_t size, const uint8_t
                             key[8U])
```

Decrypts DES using ECB block mode.

Decrypts DES using ECB block mode.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key – Input key to use for decryption

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES_EncryptCbc(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                              *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[8],
                              const uint8_t key[8U])
```

Encrypts DES using CBC block mode.

Encrypts DES using CBC block mode.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key – Input key to use for encryption

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES_DecryptCbc(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                              *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[8],
                              const uint8_t key[8U])
```

Decrypts DES using CBC block mode.

Decrypts DES using CBC block mode.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies jobRing.
- `ciphertext` – Input ciphertext to decrypt
- `plaintext` – **[out]** Output plaintext
- `size` – Size of input data in bytes
- `iv` – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- `key` – Input key to use for decryption

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES_EncryptCfb(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                             *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[8],
                             const uint8_t key[8U])
```

Encrypts DES using CFB block mode.

Encrypts DES using CFB block mode.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies jobRing.
- `plaintext` – Input plaintext to encrypt
- `size` – Size of input data in bytes
- `iv` – Input initial block.
- `key` – Input key to use for encryption
- `ciphertext` – **[out]** Output ciphertext

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES_DecryptCfb(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                              *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[8],
                              const uint8_t key[8U])
```

Decrypts DES using CFB block mode.

Decrypts DES using CFB block mode.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies jobRing.
- `ciphertext` – Input ciphertext to decrypt
- `plaintext` – **[out]** Output plaintext
- `size` – Size of input and output data in bytes
- `iv` – Input initial block.
- `key` – Input key to use for decryption

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES_EncryptOfb(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                             *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[8],
                             const uint8_t key[8U])
```

Encrypts DES using OFB block mode.

Encrypts DES using OFB block mode.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key – Input key to use for encryption

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES_DecryptOfb(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                              *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[8],
                              const uint8_t key[8U])
```

Decrypts DES using OFB block mode.

Decrypts DES using OFB block mode.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key – Input key to use for decryption

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES2_EncryptEcb(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                              *plaintext, uint8_t *ciphertext, size_t size, const uint8_t
                              key1[8U], const uint8_t key2[8U])
```

Encrypts triple DES using ECB block mode with two keys.

Encrypts triple DES using ECB block mode with two keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.

- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

status_t CAAM_DES2_DecryptEcb(CAAM_Type *base, *caam_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t key1[8U], const uint8_t key2[8U])

Decrypts triple DES using ECB block mode with two keys.

Decrypts triple DES using ECB block mode with two keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

status_t CAAM_DES2_EncryptCbc(CAAM_Type *base, *caam_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[8], const uint8_t key1[8U], const uint8_t key2[8U])

Encrypts triple DES using CBC block mode with two keys.

Encrypts triple DES using CBC block mode with two keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

status_t CAAM_DES2_DecryptCbc(CAAM_Type *base, *caam_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[8], const uint8_t key1[8U], const uint8_t key2[8U])

Decrypts triple DES using CBC block mode with two keys.

Decrypts triple DES using CBC block mode with two keys.

Parameters

- base – CAAM peripheral base address

- `handle` – Handle used for this request. Specifies `jobRing`.
- `ciphertext` – Input ciphertext to decrypt
- `plaintext` – **[out]** Output plaintext
- `size` – Size of input and output data in bytes
- `iv` – Input initial vector to combine with the first plaintext block. The `iv` does not need to be secret, but it must be unpredictable.
- `key1` – First input key for key bundle
- `key2` – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES2_EncryptCfb(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                               *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[8],
                               const uint8_t key1[8U], const uint8_t key2[8U])
```

Encrypts triple DES using CFB block mode with two keys.

Encrypts triple DES using CFB block mode with two keys.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies `jobRing`.
- `plaintext` – Input plaintext to encrypt
- `ciphertext` – **[out]** Output ciphertext
- `size` – Size of input and output data in bytes
- `iv` – Input initial block.
- `key1` – First input key for key bundle
- `key2` – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES2_DecryptCfb(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                               *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[8],
                               const uint8_t key1[8U], const uint8_t key2[8U])
```

Decrypts triple DES using CFB block mode with two keys.

Decrypts triple DES using CFB block mode with two keys.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies `jobRing`.
- `ciphertext` – Input ciphertext to decrypt
- `plaintext` – **[out]** Output plaintext
- `size` – Size of input and output data in bytes
- `iv` – Input initial block.
- `key1` – First input key for key bundle
- `key2` – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES2_EncryptOfb(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                               *plaintext, uint8_t *ciphertext, size_t size, const uint8_t
                               iv[8], const uint8_t key1[8U], const uint8_t key2[8U])
```

Encrypts triple DES using OFB block mode with two keys.

Encrypts triple DES using OFB block mode with two keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES2_DecryptOfb(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                               *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[8],
                               const uint8_t key1[8U], const uint8_t key2[8U])
```

Decrypts triple DES using OFB block mode with two keys.

Decrypts triple DES using OFB block mode with two keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES3_EncryptEcb(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                               *plaintext, uint8_t *ciphertext, size_t size, const uint8_t
                               key1[8U], const uint8_t key2[8U], const uint8_t key3[8U])
```

Encrypts triple DES using ECB block mode with three keys.

Encrypts triple DES using ECB block mode with three keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- plaintext – Input plaintext to encrypt

- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES3_DecryptEcb(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                             *ciphertext, uint8_t *plaintext, size_t size, const uint8_t
                             key1[8U], const uint8_t key2[8U], const uint8_t key3[8U])
```

Decrypts triple DES using ECB block mode with three keys.

Decrypts triple DES using ECB block mode with three keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES3_EncryptCbc(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                             *plaintext, uint8_t *ciphertext, size_t size, const uint8_t
                             iv[8], const uint8_t key1[8U], const uint8_t key2[8U], const
                             uint8_t key3[8U])
```

Encrypts triple DES using CBC block mode with three keys.

Encrypts triple DES using CBC block mode with three keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES3_DecryptCbc(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                             *ciphertext, uint8_t *plaintext, size_t size, const uint8_t
                             iv[8], const uint8_t key1[8U], const uint8_t key2[8U], const
                             uint8_t key3[8U])
```

Decrypts triple DES using CBC block mode with three keys.

Decrypts triple DES using CBC block mode with three keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES3_EncryptCfb(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                              *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[8],
                              const uint8_t key1[8U], const uint8_t key2[8U], const uint8_t
                              key3[8U])
```

Encrypts triple DES using CFB block mode with three keys.

Encrypts triple DES using CFB block mode with three keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES3_DecryptCfb(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                              *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[8],
                              const uint8_t key1[8U], const uint8_t key2[8U], const uint8_t
                              key3[8U])
```

Decrypts triple DES using CFB block mode with three keys.

Decrypts triple DES using CFB block mode with three keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES3_EncryptOfb(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                               *plaintext, uint8_t *ciphertext, size_t size, const uint8_t
                               iv[8], const uint8_t key1[8U], const uint8_t key2[8U], const
                               uint8_t key3[8U])
```

Encrypts triple DES using OFB block mode with three keys.

Encrypts triple DES using OFB block mode with three keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t CAAM_DES3_DecryptOfb(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                               *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[8],
                               const uint8_t key1[8U], const uint8_t key2[8U], const uint8_t
                               key3[8U])
```

Decrypts triple DES using OFB block mode with three keys.

Decrypts triple DES using OFB block mode with three keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext

- `size` – Size of input and output data in bytes
- `iv` – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- `key1` – First input key for key bundle
- `key2` – Second input key for key bundle
- `key3` – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

`CAAM_DES_KEY_SIZE`

CAAM DES key size - 64 bits.

`CAAM_DES_IV_SIZE`

CAAM DES IV size - 8 bytes.

`CAAM_BLACKEN_ECB_SIZE(x)`

CAAM blacken key size for ECB encryption.

`CAAM_BLACKEN_CCM_SIZE(x)`

CAAM blacken key size for CCM encryption.

`CAAM_DSA_PUBLIC_KEY_LENGTH(domain)`

CAAM DSA public key length for EC domain.

`CAAM_ECC_PUBLIC_KEY_LENGTH(domain)`

CAAM ECC public key length for EC domain.

`CAAM_ECC_PRIVATE_KEY_LENGTH(domain)`

CAAM ECC private key length for EC domain.

`CAAM_ECC_SECOND_SIGN_BUFFER_SIZE(domain)`

CAAM blacken key size for ECB encryption.

The second part of key size and buffer length needed for computation may differ.

2.13 Caam_driver_ecc

`size_t CAAM_ECC_PrivateKeySize(caam_ecc_encryption_type_t encryptKeyType,
caam_ecc_ecdsel_t ecdsel)`

Return size of private key based on encryption type and elliptic curve domain.

Parameters

- `encryptKeyType` – Type of key encryption.
- `ecdsel` – Elliptic curve domain selection

Returns

`size_t` Size of private key.

`status_t CAAM_ECC_KeyPair(CAAM_Type *base, caam_handle_t *handle, caam_ecc_ecdsel_t
ecdsel, caam_ecc_encryption_type_t encryptKeyType, uint8_t
*privKey, uint8_t *pubKey)`

Generates public and private key for ECC.

The buffer size of `privKey` can be determined using `CAAM_ECC_PRIVATE_KEY_LENGTH`. The buffer size of `pubKey` can be determined using `CAAM_ECC_PUBLIC_KEY_LENGTH`. For encrypted `privKey`, the buffer size can be determined using `CAAM_BLACKEN_ECB_SIZE` or `CAAM_BLACKEN_CCM_SIZE` macros.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- ecdsel – Elliptic curve domain selection
- encryptKeyType – Type of key encryption
- privKey – **[out]** Private key
- pubKey – **[out]** Public key

Returns

Operation status.

```
status_t CAAM_ECC_Sign(CAAM_Type *base, caam_handle_t *handle, const uint8_t *privKey,  
                      const uint8_t *data, size_t dataSize, caam_ecc_ecdsel_t ecdsel,  
                      caam_ecc_encryption_type_t encryptKeyType, uint8_t *signFirst,  
                      uint8_t *signSecond)
```

Generates signature using ECC.

The buffer size of signFirst can be determined using CAAM_ECC_PRIVATE_KEY_LENGTH.
! The buffer size of signSecond can be determined using CAAM_ECC_SECOND_SIGN_BUFFER_SIZE.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- privKey – Private key
- data – Hashed data
- dataSize – Hashed data length
- ecdsel – Elliptic curve domain selection
- encryptKeyType – Type of key encryption
- signFirst – **[out]** First part of the signature
- signSecond – **[out]** Second part of the signature

Returns

Operation status.

```
status_t CAAM_ECC_VerifyPublicKey(CAAM_Type *base, caam_handle_t *handle, const uint8_t  
                                *pubKey, const uint8_t *data, size_t dataSize, const  
                                uint8_t *signFirst, const uint8_t *signSecond,  
                                caam_ecc_ecdsel_t ecdsel, uint8_t *tmp)
```

Verify ECC signature using public key.

The buffer size of tmp can be determined using CAAM_ECC_PUBLIC_KEY_LENGTH.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- pubKey – Public key
- data – Hashed data
- dataSize – Hashed data length
- signFirst – First part of the signature

- signSecond – Second part of the signature
- ecdsel – Elliptic curve domain selection
- tmp – **[inout]** Temporary storage for intermediate results

Returns

Operation status.

```
status_t CAAM_ECC_VerifyPrivateKey(CAAM_Type *base, caam_handle_t *handle, const
uint8_t *privKey, const uint8_t *data, size_t dataSize,
const uint8_t *signFirst, const uint8_t *signSecond,
caam_ecc_ecdsel_t ecdsel, caam_ecc_encryption_type_t
encryptKeyType)
```

Verify ECC signature using private key.

Faster than verifying using public key.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- privKey – Private key
- data – Hashed data
- dataSize – Hashed data length
- signFirst – First part of the signature
- signSecond – Second part of the signature
- ecdsel – Elliptic curve domain selection
- encryptKeyType – Type of key encryption

Returns

Operation status.

2.14 CAAM HASH driver

```
enum _caam_hash_algo_t
```

Supported cryptographic block cipher functions for HASH creation.

Values:

```
enumerator kCAAM_XcbcMac
```

XCBC-MAC (AES engine)

```
enumerator kCAAM_Cmac
```

CMAC (AES engine)

```
enumerator kCAAM_Sha1
```

SHA_1 (MDHA engine)

```
enumerator kCAAM_Sha224
```

SHA_224 (MDHA engine)

```
enumerator kCAAM_Sha256
```

SHA_256 (MDHA engine)

```
enumerator kCAAM_Sha384
```

SHA_384 (MDHA engine)

```

enumerator kCAAM_Sha512
    SHA_512 (MDHA engine)
enumerator kCAAM_HmacSha1
    HMAC_SHA_1 (MDHA engine)
enumerator kCAAM_HmacSha224
    HMAC_SHA_224 (MDHA engine)
enumerator kCAAM_HmacSha256
    HMAC_SHA_256 (MDHA engine)
enumerator kCAAM_HmacSha384
    HMAC_SHA_384 (MDHA engine)
enumerator kCAAM_HmacSha512
    HMAC_SHA_512 (MDHA engine)

```

```
typedef enum caam_hash_algo_t caam_hash_algo_t
```

Supported cryptographic block cipher functions for HASH creation.

```
typedef uint32_t caam_hash_ctx_t[83]
```

Storage type used to save hash context.

```
status_t CAAM_HASH_Init(CAAM_Type *base, caam_handle_t *handle, caam_hash_ctx_t *ctx,
    caam_hash_algo_t algo, const uint8_t *key, size_t keySize)
```

Initialize HASH context.

This function initializes the HASH. Key shall be supplied if the underlaying algorithm is AES XCBC-MAC or CMAC. Key shall be NULL if the underlaying algorithm is SHA.

For XCBC-MAC, the key length must be 16. For CMAC, the key length can be the AES key lengths supported by AES engine. For MDHA the key length argument is ignored.

This functions is used to initialize the context for both blocking and non-blocking CAAM_HASH API. For blocking CAAM HASH API, the HASH context contains all information required for context switch, such as running hash or MAC. For non-blocking CAAM HASH API, the HASH context is used to hold SGT. Therefore, the HASH context cannot be shared between blocking and non-blocking HASH API. With one HASH context, either use only blocking HASH API or only non-blocking HASH API.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request.
- ctx – **[out]** Output hash context
- algo – Underlaying algorithm to use for hash computation.
- key – Input key (NULL if underlaying algorithm is SHA)
- keySize – Size of input key in bytes

Returns

Status of initialization

```
status_t CAAM_HASH_Update(caam_hash_ctx_t *ctx, const uint8_t *input, size_t inputSize)
```

Add data to current HASH.

Add data to current HASH. This can be called repeatedly with an arbitrary amount of data to be hashed. The functions blocks. If it returns kStatus_Success, the running hash or mac has been updated (CAAM has processed the input data), so the memory at input pointer can be released back to system. The context is updated with the running hash or mac and with all necessary information to support possible context switch.

Parameters

- `ctx` – **[inout]** HASH context
- `input` – Input data
- `inputSize` – Size of input data in bytes

Returns

Status of the hash update operation

`status_t` CAAM_HASH_Finish(*caam_hash_ctx_t* *ctx, *uint8_t* *output, *size_t* *outputSize)

Finalize hashing.

Outputs the final hash (computed by CAAM_HASH_Update()) and erases the context.

Parameters

- `ctx` – **[inout]** Input hash context
- `output` – **[out]** Output hash data
- `outputSize` – **[out]** Output parameter storing the size of the output hash in bytes

Returns

Status of the hash finish operation

`status_t` CAAM_HASH(CAAM_Type *base, *caam_handle_t* *handle, *caam_hash_algo_t* algo, *const uint8_t* *input, *size_t* inputSize, *const uint8_t* *key, *size_t* keySize, *uint8_t* *output, *size_t* *outputSize)

Create HASH on given data.

Perform the full keyed XCBC-MAC/CMAC or SHA in one function call.

Key shall be supplied if the underlaying algorithm is AES XCBC-MAC or CMAC. Key shall be NULL if the underlaying algorithm is SHA.

For XCBC-MAC, the key length must be 16. For CMAC, the key length can be the AES key lengths supported by AES engine. For MDHA the key length argument is ignored.

The function is blocking.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request.
- `algo` – Underlaying algorithm to use for hash computation.
- `input` – Input data
- `inputSize` – Size of input data in bytes
- `key` – Input key (NULL if underlaying algorithm is SHA)
- `keySize` – Size of input key in bytes
- `output` – **[out]** Output hash data
- `outputSize` – **[out]** Output parameter storing the size of the output hash in bytes

Returns

Status of the one call hash operation.

CAAM_SHA_BLOCK_SIZE

CAAM HASH Context size.

up to SHA-512 block size

CAAM_HASH_BLOCK_SIZE

CAAM hash block size

CAAM_HASH_CTX_SIZE

CAAM HASH Context size.

2.15 Caam_driver_hmac

status_t CAAM_HMAC_Init(CAAM_Type *base, *caam_handle_t* *handle, *caam_hash_ctx_t* *ctx, *caam_hash_algo_t* algo, const uint8_t *key, size_t keySize)

Initialize HMAC context.

This function initializes the HMAC.

For XCBC-MAC, the key length must be 16. For CMAC, the key length can be the AES key lengths supported by AES engine. For MDHA the key length argument is ignored.

This functions is used to initialize the context for both blocking and non-blocking CAAM_HMAC API.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request.
- ctx – **[out]** Output HMAC context
- algo – Underlying algorithm to use for HMAC computation.
- key – Input key
- keySize – Size of input key in bytes

Returns

Status of initialization

status_t CAAM_HMAC(CAAM_Type *base, *caam_handle_t* *handle, *caam_hash_algo_t* algo, const uint8_t *input, size_t inputSize, const uint8_t *key, size_t keySize, uint8_t *output, size_t *outputSize)

Create Message Authentication Code (MAC) on given data.

Perform the full keyed XCBC-MAC/CMAC, or HMAC-SHA in one function call.

Key shall be supplied if the underlying algorithm is AES XCBC-MAC, CMAC, or SHA HMAC.

For XCBC-MAC, the key length must be 16. For CMAC, the key length can be the AES key lengths supported by AES engine. For HMAC, the key can have any size.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request.
- algo – Underlying algorithm to use for MAC computation.
- input – Input data
- inputSize – Size of input data in bytes
- key – Input key
- keySize – Size of input key in bytes
- output – **[out]** Output MAC data

- outputSize – **[out]** Output parameter storing the size of the output MAC in bytes

Returns

Status of the one call hash operation.

2.16 CAAM PKHA driver

enum `_caam_pkha_timing_t`

Use of timing equalized version of a PKHA function.

Values:

enumerator `kCAAM_PKHA_NoTimingEqualized`

Normal version of a PKHA operation

enumerator `kCAAM_PKHA_TimingEqualized`

Timing-equalized version of a PKHA operation

enum `_caam_pkha_f2m_t`

Integer vs binary polynomial arithmetic selection.

Values:

enumerator `kCAAM_PKHA_IntegerArith`

Use integer arithmetic

enumerator `kCAAM_PKHA_F2mArith`

Use binary polynomial arithmetic

enum `_caam_pkha_montgomery_form_t`

Montgomery or normal PKHA input format.

Values:

enumerator `kCAAM_PKHA_NormalValue`

PKHA number is normal integer

enumerator `kCAAM_PKHA_MontgomeryFormat`

PKHA number is in montgomery format

typedef struct `_caam_pkha_ecc_point_t` `caam_pkha_ecc_point_t`

PKHA ECC point structure

typedef enum `_caam_pkha_timing_t` `caam_pkha_timing_t`

Use of timing equalized version of a PKHA function.

typedef enum `_caam_pkha_f2m_t` `caam_pkha_f2m_t`

Integer vs binary polynomial arithmetic selection.

typedef enum `_caam_pkha_montgomery_form_t` `caam_pkha_montgomery_form_t`

Montgomery or normal PKHA input format.

int `CAAM_PKHA_CompareBigNum(const uint8_t *a, size_t sizeA, const uint8_t *b, size_t sizeB)`

`status_t` `CAAM_PKHA_NormalToMontgomery(CAAM_Type *base, caam_handle_t *handle, const uint8_t *N, size_t sizeN, uint8_t *A, size_t *sizeA, uint8_t *B, size_t *sizeB, uint8_t *R2, size_t *sizeR2, caam_pkha_timing_t equalTime, caam_pkha_f2m_t arithType)`

Converts from integer to Montgomery format.

This function computes $R2 \bmod N$ and optionally converts A or B into Montgomery format of A or B.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- N – modulus
- sizeN – size of N in bytes
- A – **[inout]** The first input in non-Montgomery format. Output Montgomery format of the first input.
- sizeA – **[inout]** pointer to size variable. On input it holds size of input A in bytes. On output it holds size of Montgomery format of A in bytes.
- B – **[inout]** Second input in non-Montgomery format. Output Montgomery format of the second input.
- sizeB – **[inout]** pointer to size variable. On input it holds size of input B in bytes. On output it holds size of Montgomery format of B in bytes.
- R2 – **[out]** Output Montgomery factor $R2 \bmod N$.
- sizeR2 – **[out]** pointer to size variable. On output it holds size of Montgomery factor $R2 \bmod N$ in bytes.
- equalTime – Run the function time equalized or no timing equalization.
- arithType – Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

```
status_t CAAM_PKHA_MontgomeryToNormal(CAAM_Type *base, caam_handle_t *handle, const
                                         uint8_t *N, size_t sizeN, uint8_t *A, size_t *sizeA,
                                         uint8_t *B, size_t *sizeB, caam_pkha_timing_t
                                         equalTime, caam_pkha_f2m_t arithType)
```

Converts from Montgomery format to int.

This function converts Montgomery format of A or B into int A or B.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- N – modulus.
- sizeN – size of N modulus in bytes.
- A – **[inout]** Input first number in Montgomery format. Output is non-Montgomery format.
- sizeA – **[inout]** pointer to size variable. On input it holds size of the input A in bytes. On output it holds size of non-Montgomery A in bytes.
- B – **[inout]** Input first number in Montgomery format. Output is non-Montgomery format.
- sizeB – **[inout]** pointer to size variable. On input it holds size of the input B in bytes. On output it holds size of non-Montgomery B in bytes.
- equalTime – Run the function time equalized or no timing equalization.

- `arithType` – Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

```
status_t CAAM_PKHA_ModAdd(CAAM_Type *base, caam_handle_t *handle, const uint8_t *A,
                           size_t sizeA, const uint8_t *B, size_t sizeB, const uint8_t *N,
                           size_t sizeN, uint8_t *result, size_t *resultSize,
                           caam_pkha_f2m_t arithType)
```

Performs modular addition - $(A + B) \bmod N$.

This function performs modular addition of $(A + B) \bmod N$, with either integer or binary polynomial (F2m) inputs. In the F2m form, this function is equivalent to a bitwise XOR and it is functionally the same as subtraction.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies jobRing.
- `A` – first addend (integer or binary polynomial)
- `sizeA` – Size of A in bytes
- `B` – second addend (integer or binary polynomial)
- `sizeB` – Size of B in bytes
- `N` – modulus.
- `sizeN` – Size of N in bytes.
- `result` – **[out]** Output array to store result of operation
- `resultSize` – **[out]** Output size of operation in bytes
- `arithType` – Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

```
status_t CAAM_PKHA_ModSub1(CAAM_Type *base, caam_handle_t *handle, const uint8_t *A,
                           size_t sizeA, const uint8_t *B, size_t sizeB, const uint8_t *N,
                           size_t sizeN, uint8_t *result, size_t *resultSize)
```

Performs modular subtraction - $(A - B) \bmod N$.

This function performs modular subtraction of $(A - B) \bmod N$ with integer inputs.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies jobRing.
- `A` – first addend (integer or binary polynomial)
- `sizeA` – Size of A in bytes
- `B` – second addend (integer or binary polynomial)
- `sizeB` – Size of B in bytes
- `N` – modulus
- `sizeN` – Size of N in bytes
- `result` – **[out]** Output array to store result of operation
- `resultSize` – **[out]** Output size of operation in bytes

Returns

Operation status.

```
status_t CAAM_PKHA_ModSub2(CAAM_Type *base, caam_handle_t *handle, const uint8_t *A,
                             size_t sizeA, const uint8_t *B, size_t sizeB, const uint8_t *N,
                             size_t sizeN, uint8_t *result, size_t *resultSize)
```

Performs modular subtraction - $(B - A) \bmod N$.

This function performs modular subtraction of $(B - A) \bmod N$, with integer inputs.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- B – second addend (integer or binary polynomial)
- sizeB – Size of B in bytes
- N – modulus
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes

Returns

Operation status.

```
status_t CAAM_PKHA_ModMul(CAAM_Type *base, caam_handle_t *handle, const uint8_t *A,
                             size_t sizeA, const uint8_t *B, size_t sizeB, const uint8_t *N,
                             size_t sizeN, uint8_t *result, size_t *resultSize,
                             caam_pkha_f2m_t arithType, caam_pkha_montgomery_form_t
                             montIn, caam_pkha_montgomery_form_t montOut,
                             caam_pkha_timing_t equalTime)
```

Performs modular multiplication - $(A \times B) \bmod N$.

This function performs modular multiplication with either integer or binary polynomial (F2m) inputs. It can optionally specify whether inputs and/or outputs will be in Montgomery form or not.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- B – second addend (integer or binary polynomial)
- sizeB – Size of B in bytes
- N – modulus.
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)
- montIn – Format of inputs

- `montOut` – Format of output
- `equalTime` – Run the function time equalized or no timing equalization. This argument is ignored for F2m modular multiplication.

Returns

Operation status.

```
status_t CAAM_PKHA_ModExp(CAAM_Type *base, caam_handle_t *handle, const uint8_t *A,  
                           size_t sizeA, const uint8_t *N, size_t sizeN, const uint8_t *E,  
                           size_t sizeE, uint8_t *result, size_t *resultSize,  
                           caam_pkha_f2m_t arithType, caam_pkha_montgomery_form_t  
                           montIn, caam_pkha_timing_t equalTime)
```

Performs modular exponentiation - $(A^E) \bmod N$.

This function performs modular exponentiation with either integer or binary polynomial (F2m) inputs.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies `jobRing`.
- `A` – first addend (integer or binary polynomial)
- `sizeA` – Size of A in bytes
- `N` – modulus
- `sizeN` – Size of N in bytes
- `E` – exponent
- `sizeE` – Size of E in bytes
- `result` – **[out]** Output array to store result of operation
- `resultSize` – **[out]** Output size of operation in bytes
- `montIn` – Format of A input (normal or Montgomery)
- `arithType` – Type of arithmetic to perform (integer or F2m)
- `equalTime` – Run the function time equalized or no timing equalization.

Returns

Operation status.

```
status_t CAAM_PKHA_ModRed(CAAM_Type *base, caam_handle_t *handle, const uint8_t *A,  
                           size_t sizeA, const uint8_t *N, size_t sizeN, uint8_t *result,  
                           size_t *resultSize, caam_pkha_f2m_t arithType)
```

Performs modular reduction - $(A) \bmod N$.

This function performs modular reduction with either integer or binary polynomial (F2m) inputs.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies `jobRing`.
- `A` – first addend (integer or binary polynomial)
- `sizeA` – Size of A in bytes
- `N` – modulus
- `sizeN` – Size of N in bytes
- `result` – **[out]** Output array to store result of operation

- `resultSize` – **[out]** Output size of operation in bytes
- `arithType` – Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

`status_t` CAAM_PKHA_ModInv(CAAM_Type *base, *caam_handle_t* *handle, const uint8_t *A, size_t sizeA, const uint8_t *N, size_t sizeN, uint8_t *result, size_t *resultSize, *caam_pkha_f2m_t* arithType)

Performs modular inversion - $(A^{-1}) \bmod N$.

This function performs modular inversion with either integer or binary polynomial (F2m) inputs.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies jobRing.
- `A` – first addend (integer or binary polynomial)
- `sizeA` – Size of A in bytes
- `N` – modulus
- `sizeN` – Size of N in bytes
- `result` – **[out]** Output array to store result of operation
- `resultSize` – **[out]** Output size of operation in bytes
- `arithType` – Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

`status_t` CAAM_PKHA_ModR2(CAAM_Type *base, *caam_handle_t* *handle, const uint8_t *N, size_t sizeN, uint8_t *result, size_t *resultSize, *caam_pkha_f2m_t* arithType)

Computes integer Montgomery factor $R^2 \bmod N$.

This function computes a constant to assist in converting operands into the Montgomery residue system representation.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies jobRing.
- `N` – modulus
- `sizeN` – Size of N in bytes
- `result` – **[out]** Output array to store result of operation
- `resultSize` – **[out]** Output size of operation in bytes
- `arithType` – Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

`status_t` CAAM_PKHA_ModGcd(CAAM_Type *base, *caam_handle_t* *handle, const uint8_t *A, size_t sizeA, const uint8_t *N, size_t sizeN, uint8_t *result, size_t *resultSize, *caam_pkha_f2m_t* arithType)

Calculates the greatest common divisor - GCD (A, N).

This function calculates the greatest common divisor of two inputs with either integer or binary polynomial (F2m) inputs.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- A – first value (must be smaller than or equal to N)
- sizeA – Size of A in bytes
- N – second value (must be non-zero)
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

```
status_t CAAM_PKHA_PrimalityTest(CAAM_Type *base, caam_handle_t *handle, const uint8_t
                                *A, size_t sizeA, const uint8_t *B, size_t sizeB, const
                                uint8_t *N, size_t sizeN, bool *res)
```

Executes Miller-Rabin primality test.

This function calculates whether or not a candidate prime number is likely to be a prime.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- A – initial random seed
- sizeA – Size of A in bytes
- B – number of trial runs
- sizeB – Size of B in bytes
- N – candidate prime integer
- sizeN – Size of N in bytes
- res – **[out]** True if the value is likely prime or false otherwise

Returns

Operation status.

```
status_t CAAM_PKHA_ECC_PointAdd(CAAM_Type *base, caam_handle_t *handle, const
                                caam_pkha_ecc_point_t *A, const
                                caam_pkha_ecc_point_t *B, const uint8_t *N, const
                                uint8_t *R2modN, const uint8_t *aCurveParam, const
                                uint8_t *bCurveParam, size_t size, caam_pkha_f2m_t
                                arithType, caam_pkha_ecc_point_t *result)
```

Adds elliptic curve points - A + B.

This function performs ECC point addition over a prime field (Fp) or binary field (F2m) using affine coordinates.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.

- A – Left-hand point
- B – Right-hand point
- N – Prime modulus of the field
- R2modN – NULL (the function computes R2modN internally) or pointer to pre-computed R2modN (obtained from CAAM_PKHA_ModR2() function).
- aCurveParam – A parameter from curve equation
- bCurveParam – B parameter from curve equation (constant)
- size – Size in bytes of curve points and parameters
- arithType – Type of arithmetic to perform (integer or F2m)
- result – **[out]** Result point

Returns

Operation status.

```
status_t CAAM_PKHA_ECC_PointDouble(CAAM_Type *base, caam_handle_t *handle, const
                                     caam_pkha_ecc_point_t *B, const uint8_t *N, const
                                     uint8_t *aCurveParam, const uint8_t *bCurveParam,
                                     size_t size, caam_pkha_f2m_t arithType,
                                     caam_pkha_ecc_point_t *result)
```

Doubles elliptic curve points - B + B.

This function performs ECC point doubling over a prime field (Fp) or binary field (F2m) using affine coordinates.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- B – Point to double
- N – Prime modulus of the field
- aCurveParam – A parameter from curve equation
- bCurveParam – B parameter from curve equation (constant)
- size – Size in bytes of curve points and parameters
- arithType – Type of arithmetic to perform (integer or F2m)
- result – **[out]** Result point

Returns

Operation status.

```
status_t CAAM_PKHA_ECC_PointMul(CAAM_Type *base, caam_handle_t *handle, const
                                  caam_pkha_ecc_point_t *A, const uint8_t *E, size_t sizeE,
                                  const uint8_t *N, const uint8_t *R2modN, const uint8_t
                                  *aCurveParam, const uint8_t *bCurveParam, size_t size,
                                  caam_pkha_timing_t equalTime, caam_pkha_f2m_t
                                  arithType, caam_pkha_ecc_point_t *result)
```

Multiplies an elliptic curve point by a scalar - E x (A0, A1).

This function performs ECC point multiplication to multiply an ECC point by a scalar integer multiplier over a prime field (Fp) or a binary field (F2m).

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.

- A – Point as multiplicand
- E – Scalar multiple
- sizeE – The size of E, in bytes
- N – Modulus, a prime number for the Fp field or Irreducible polynomial for F2m field.
- R2modN – NULL (the function computes R2modN internally) or pointer to pre-computed R2modN (obtained from CAAM_PKHA_ModR2() function).
- aCurveParam – A parameter from curve equation
- bCurveParam – B parameter from curve equation (C parameter for operation over F2m).
- size – Size in bytes of curve points and parameters
- equalTime – Run the function time equalized or no timing equalization.
- arithType – Type of arithmetic to perform (integer or F2m)
- result – **[out]** Result point

Returns

Operation status.

```
struct __caam_pkha_ecc_point_t  
#include <fsl_caam.h> PKHA ECC point structure
```

Public Members

```
uint8_t *X  
    X coordinate (affine)  
uint8_t *Y  
    Y coordinate (affine)
```

2.17 CAAM RNG driver

```
enum __caam_rng_state_handle  
    CAAM RNG state handle.
```

Values:

```
enumerator kCAAM_RngStateHandle0  
    CAAM RNG state handle 0  
enumerator kCAAM_RngStateHandle1  
    CAAM RNG state handle 1
```

```
enum __caam_rng_random_type  
    Type of random data to generate.
```

Values:

```
enumerator kCAAM_RngDataAny  
    CAAM RNG any random data bytes  
enumerator kCAAM_RngDataOddParity  
    CAAM RNG odd parity random data bytes
```

enumerator kCAAM_RngDataNonZero

CAAM RNG non zero random data bytes

typedef enum *caam_rng_state_handle* caam_rng_state_handle_t
CAAM RNG state handle.

typedef enum *caam_rng_random_type* caam_rng_random_type_t
Type of random data to generate.

typedef uint32_t caam_rng_generic256_t[256 / sizeof(uint32_t)]
256-bit value used as optional additional entropy input

typedef struct *caam_rng_user_config* caam_rng_config_t
CAAM RNG configuration.

status_t CAAM_RNG_GetDefaultConfig(*caam_rng_config_t* *config)
Initializes user configuration structure to default.

This function initializes the configure structure to default value. the default value are:

```
config->autoReseedInterval = 0;
config->personalString = NULL;
```

Parameters

- config – User configuration structure.

Returns

status of the request

status_t CAAM_RNG_Init(CAAM_Type *base, *caam_handle_t* *handle, *caam_rng_state_handle_t* stateHandle, const *caam_rng_config_t* *config)

Instantiate the CAAM RNG state handle.

This function instantiates CAAM RNG state handle. The function is blocking and returns after CAAM has processed the request.

Parameters

- base – CAAM peripheral base address
- handle – CAAM jobRing used for this request
- stateHandle – RNG state handle to instantiate
- config – Pointer to configuration structure.

Returns

Status of the request

status_t CAAM_RNG_Deinit(CAAM_Type *base, *caam_handle_t* *handle, *caam_rng_state_handle_t* stateHandle)

Uninstantiate the CAAM RNG state handle.

This function uninstantiates CAAM RNG state handle. The function is blocking and returns after CAAM has processed the request.

Parameters

- base – CAAM peripheral base address
- handle – jobRing used for this request.
- stateHandle – RNG state handle to uninstantiate

Returns

Status of the request

```
status_t CAAM_RNG_GenerateSecureKey(CAAM_Type *base, caam_handle_t *handle,  
                                     caam_rng_generic256_t additionalEntropy)
```

Generate Secure Key.

This function generates random data writes it to Secure Key registers. The function is blocking and returns after CAAM has processed the request. RNG state handle 0 is always used.

Parameters

- base – CAAM peripheral base address
- handle – jobRing used for this request
- additionalEntropy – NULL or Pointer to optional 256-bit additional entropy.

Returns

Status of the request

```
status_t CAAM_RNG_Reseed(CAAM_Type *base, caam_handle_t *handle,  
                          caam_rng_state_handle_t stateHandle, caam_rng_generic256_t  
                          additionalEntropy)
```

Reseed the CAAM RNG state handle.

This function reseeds the CAAM RNG state handle. For a state handle in nondeterministic mode, the DRNG is seeded with 384 bits of entropy from the TRNG and an optional 256-bit additional input from the descriptor via the Class 1 Context Register.

The function is blocking and returns after CAAM has processed the request.

Parameters

- base – CAAM peripheral base address
- handle – jobRing used for this request
- stateHandle – RNG state handle to reseed
- additionalEntropy – NULL or Pointer to optional 256-bit additional entropy.

Returns

Status of the request

```
status_t CAAM_RNG_GetRandomData(CAAM_Type *base, caam_handle_t *handle,  
                                caam_rng_state_handle_t stateHandle, uint8_t *data,  
                                size_t dataSize, caam_rng_random_type_t dataType,  
                                caam_rng_generic256_t additionalEntropy)
```

Get random data.

This function gets random data from CAAM RNG.

The function is blocking and returns after CAAM has generated the requested data or an error occurred.

Parameters

- base – CAAM peripheral base address
- handle – jobRing used for this request
- stateHandle – RNG state handle used to generate random data
- data – **[out]** Pointer address used to store random data
- dataSize – Size of the buffer pointed by the data parameter
- dataType – Type of random data to be generated
- additionalEntropy – NULL or Pointer to optional 256-bit additional entropy.

Returns

Status of the request

```
struct _caam_rng_user_config
#include <fsl_caam.h> CAAM RNG configuration.
```

Public Members

```
uint32_t autoReseedInterval
    Automatic reseed internal. If set to zero, CAAM RNG will use hardware default interval
    of 10.000.000 generate requests.

caam_rng_generic256_t *personalString
    NULL or pointer to optional personalization string
```

2.18 Caam_driver_rsa

```
size_t CAAM_RSA_PrivateExponentSize(caam_rsa_key_type_t prvKeyType, uint32_t
    privExponentSize)
```

Return size for private key buffer based on encryption type and ecliptic curve domain.

Parameters

- `prvKeyType` – Type of private key
- `privExponentSize` – Expected length of private exponent without encryption padding.

Returns

`size_t` Size for private key buffer.

```
status_t CAAM_RSA_KeyPair(CAAM_Type *base, caam_handle_t *handle, const uint8_t
    *primeP, const uint8_t *primeQ, uint32_t primesSize, const
    uint8_t *pubExponent, uint32_t pubExponentSize,
    caam_rsa_key_type_t prvKeyType, uint8_t *modulus, uint32_t
    modulusSize, uint8_t *privExponent, size_t *privExponentSize)
```

Generates RSA key.

Generates modulus N and private exponent D give prime numbers P and Q and public exponent E. Public key is {E,N}. Private key is {D,N}.

! `privExponentSize` value may differ for different P abd Q with same bit length. For encrypted `privExponent`, the buffer size can be determined using `CAAM_BLACKEN_ECB_SIZE` or `CAAM_BLACKEN_CCM_SIZE` macros.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies jobRing.
- `primeP` – Prime number P
- `primeQ` – Prime number Q
- `primesSize` – Byte length of primeP or primeQ (primeP and primeQ must have the same byte length)
- `pubExponent` – Public exponent E
- `pubExponentSize` – Byte length of pubExponent
- `prvKeyType` – Type of private key
- `modulus` – **[out]** Buffer for calculated modulus N
- `modulusSize` – Byte length of modulus buffer

- `privExponent` – **[out]** Buffer for calculated private exponent D
- `privExponentSize` – **[out]** Byte length of calculated private exponent.

Returns

Operation status.

```
status_t CAAM_RSA_Encrypt(CAAM_Type *base, caam_handle_t *handle, const uint8_t
    *plainText, uint32_t plainTextSize, const uint8_t *modulus,
    uint32_t modulusSize, const uint8_t *pubExponent, uint32_t
    pubExponentSize, caam_rsa_encryption_type_t dataOutType,
    caam_rsa_format_type_t format, uint8_t *cipherText)
```

Performs the RSA public key primitive.

Performs the RSA public key primitive which can be used when encrypting a secret or verifying a signature.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies jobRing.
- `plainText` – Input data
- `plainTextSize` – Byte length of the input data
- `modulus` – Modulus N
- `modulusSize` – Byte length of modulus
- `pubExponent` – Public exponent E
- `pubExponentSize` – Byte length of pubExponent
- `dataOutType` – Type of encryption of output data
- `cipherText` – **[out]** Buffer for RSA encrypted data

Returns

Operation status

```
status_t CAAM_RSA_Decrypt(CAAM_Type *base, caam_handle_t *handle, const uint8_t
    *cipherText, const uint8_t *modulus, uint32_t modulusSize,
    const uint8_t *privExponent, uint32_t privExponentSize,
    caam_rsa_encryption_type_t prvKeyType,
    caam_rsa_encryption_type_t dataOutType,
    caam_rsa_format_type_t format, uint8_t *plainText, size_t
    *rsaDecSize)
```

Performs the RSA private key primitive.

Performs the RSA private key primitive which can be used when decrypting a secret or creating a signature.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies jobRing.
- `cipherText` – Input data
- `modulus` – Modulus N
- `modulusSize` – Byte length of modulus
- `privExponent` – Private exponent D
- `privExponentSize` – Byte length of privExponent
- `prvKeyType` – Type of private key encryption

- dataOutType – Type of encryption of output data
- plainText – **[out]** Buffer for RSA encrypted data
- rsaDecSize – **[out]** Returned output size

Returns

Operation status

2.19 CAAM Blocking APIs

2.20 CAAM Non-blocking APIs

2.21 CAAM Non-blocking AES driver

```
status_t CAAM_AES_EncryptEcbNonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                                         caam_desc_aes_ecb_t descriptor, const uint8_t  
                                         *plaintext, uint8_t *ciphertext, size_t size, const  
                                         uint8_t *key, size_t keySize)
```

Encrypts AES using the ECB block mode.

Puts AES ECB encrypt descriptor to CAAM input job ring.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- plaintext – Input plain text to encrypt
- descriptor – **[out]** Memory for the CAAM descriptor.
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from job descriptor push

```
status_t CAAM_AES_DecryptEcbNonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                                         caam_desc_aes_ecb_t descriptor, const uint8_t  
                                         *ciphertext, uint8_t *plaintext, size_t size, const  
                                         uint8_t *key, size_t keySize)
```

Decrypts AES using ECB block mode.

Puts AES ECB decrypt descriptor to CAAM input job ring.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** Memory for the CAAM descriptor.
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.

- key – Input key.
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from job descriptor push

status_t CAAM_AES_EncryptEcbNonBlockingExtended(CAAM_Type *base, caam_handle_t *handle, caam_desc_aes_ecb_t descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t *key, size_t keySize, caam_key_type_t blackKeyType)

Encrypts AES using the ECB block mode using black key.

Puts AES ECB encrypt descriptor to CAAM input job ring.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- plaintext – Input plain text to encrypt
- descriptor – **[out]** Memory for the CAAM descriptor.
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- blackKeyType – Type of black key

Returns

Status from job descriptor push

status_t CAAM_AES_DecryptEcbNonBlockingExtended(CAAM_Type *base, caam_handle_t *handle, caam_desc_aes_ecb_t descriptor, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t *key, size_t keySize, caam_key_type_t blackKeyType)

Decrypts AES using ECB block mode using black key.

Puts AES ECB decrypt descriptor to CAAM input job ring.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** Memory for the CAAM descriptor.
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- key – Input key.
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- blackKeyType – Type of black key

Returns

Status from job descriptor push

status_t CAAM_AES_EncryptCbcNonBlocking(CAAM_Type *base, *caam_handle_t* *handle, *caam_desc_aes_cbc_t* descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t *iv, const uint8_t *key, size_t keySize)

Encrypts AES using CBC block mode.

Puts AES CBC encrypt descriptor to CAAM input job ring.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** Memory for the CAAM descriptor.
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from job descriptor push

status_t CAAM_AES_DecryptCbcNonBlocking(CAAM_Type *base, *caam_handle_t* *handle, *caam_desc_aes_cbc_t* descriptor, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t *iv, const uint8_t *key, size_t keySize)

Decrypts AES using CBC block mode.

Puts AES CBC decrypt descriptor to CAAM input job ring.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** Memory for the CAAM descriptor.
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.
- key – Input key to use for decryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from job descriptor push

status_t CAAM_AES_EncryptCbcNonBlockingExtended(CAAM_Type *base, *caam_handle_t* *handle, *caam_desc_aes_cbc_t* descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t *iv, const uint8_t *key, size_t keySize, *caam_key_type_t* blackKeyType)

Encrypts AES using CBC block mode using black key.

Puts AES CBC encrypt descriptor to CAAM input job ring.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** Memory for the CAAM descriptor.
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- blackKeyType – Type of black key

Returns

Status from job descriptor push

```
status_t CAAM_AES_DecryptCbcNonBlockingExtended(CAAM_Type *base, caam_handle_t
                                                    *handle, caam_desc_aes_cbc_t
                                                    descriptor, const uint8_t *ciphertext,
                                                    uint8_t *plaintext, size_t size, const
                                                    uint8_t *iv, const uint8_t *key, size_t
                                                    keySize, caam_key_type_t blackKeyType)
```

Decrypts AES using CBC block mode using black key.

Puts AES CBC decrypt descriptor to CAAM input job ring.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** Memory for the CAAM descriptor.
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.
- key – Input key to use for decryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- blackKeyType – Type of black key

Returns

Status from job descriptor push

```
status_t CAAM_AES_CryptCtrNonBlocking(CAAM_Type *base, caam_handle_t *handle,
                                         caam_desc_aes_ctr_t descriptor, const uint8_t
                                         *input, uint8_t *output, size_t size, uint8_t *counter,
                                         const uint8_t *key, size_t keySize, uint8_t
                                         *counterlast, size_t *szLeft)
```

Encrypts or decrypts AES using CTR block mode.

Encrypts or decrypts AES using CTR block mode. AES CTR mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

Puts AES CTR crypt descriptor to CAAM input job ring.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** Memory for the CAAM descriptor.
- input – Input data for CTR block mode
- output – **[out]** Output data for CTR block mode
- size – Size of input and output data in bytes
- counter – **[inout]** Input counter (updates on return)
- key – Input key to use for forward AES cipher
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- counterlast – **[out]** Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
- szLeft – **[out]** Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used.

Returns

Status from job descriptor push

```
status_t CAAM_AES_CryptCtrNonBlockingExtended(CAAM_Type *base, caam_handle_t *handle,  
                                              caam_desc_aes_ctr_t descriptor, const  
                                              uint8_t *input, uint8_t *output, size_t size,  
                                              uint8_t *counter, const uint8_t *key, size_t  
                                              keySize, uint8_t *counterlast, size_t *szLeft,  
                                              caam_key_type_t blackKeyType)
```

Encrypts or decrypts AES using CTR block mode using black key.

Encrypts or decrypts AES using CTR block mode. AES CTR mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

Puts AES CTR crypt descriptor to CAAM input job ring.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** Memory for the CAAM descriptor.
- input – Input data for CTR block mode
- output – **[out]** Output data for CTR block mode
- size – Size of input and output data in bytes
- counter – **[inout]** Input counter (updates on return)

- `key` – Input key to use for forward AES cipher
- `keySize` – Size of the input key, in bytes. Must be 16, 24, or 32.
- `counterlast` – **[out]** Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
- `szLeft` – **[out]** Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used.
- `blackKeyType` – Type of black key

Returns

Status from job descriptor push

```
status_t CAAM_AES_EncryptTagCcmNonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                                             caam_desc_aes_ccm_t descriptor, const  
                                             uint8_t *plaintext, uint8_t *ciphertext, size_t  
                                             size, const uint8_t *iv, size_t ivSize, const  
                                             uint8_t *aad, size_t aadSize, const uint8_t  
                                             *key, size_t keySize, uint8_t *tag, size_t  
                                             tagSize)
```

Encrypts AES and tags using CCM block mode.

Puts AES CCM encrypt and tag descriptor to CAAM input job ring.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies jobRing.
- `descriptor` – **[out]** Memory for the CAAM descriptor.
- `plaintext` – Input plain text to encrypt
- `ciphertext` – **[out]** Output cipher text.
- `size` – Size of input and output data in bytes. Zero means authentication only.
- `iv` – Nonce
- `ivSize` – Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
- `aad` – Input additional authentication data. Can be NULL if aadSize is zero.
- `aadSize` – Input size in bytes of AAD. Zero means data mode only (authentication skipped).
- `key` – Input key to use for encryption
- `keySize` – Size of the input key, in bytes. Must be 16, 24, or 32.
- `tag` – **[out]** Generated output tag. Set to NULL to skip tag processing.
- `tagSize` – Input size of the tag to generate, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.

Returns

Status from job descriptor push

```
status_t CAAM_AES_DecryptTagCcmNonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                                             caam_desc_aes_ccm_t descriptor, const  
                                             uint8_t *ciphertext, uint8_t *plaintext, size_t  
                                             size, const uint8_t *iv, size_t ivSize, const  
                                             uint8_t *aad, size_t aadSize, const uint8_t  
                                             *key, size_t keySize, const uint8_t *tag, size_t  
                                             tagSize)
```

Decrypts AES and authenticates using CCM block mode.

Puts AES CCM decrypt and check tag descriptor to CAAM input job ring.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** Memory for the CAAM descriptor.
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text.
- size – Size of input and output data in bytes. Zero means authentication data only.
- iv – Nonce
- ivSize – Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
- aad – Input additional authentication data. Can be NULL if aadSize is zero.
- aadSize – Input size in bytes of AAD. Zero means data mode only (authentication data skipped).
- key – Input key to use for decryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – Received tag. Set to NULL to skip tag processing.
- tagSize – Input size of the received tag to compare with the computed tag, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.

Returns

Status from job descriptor push

```
status_t CAAM_AES_EncryptTagCcmNonBlockingExtended(CAAM_Type *base, caam_handle_t
                                                    *handle, caam_desc_aes_ccm_t
                                                    descriptor, const uint8_t *plaintext,
                                                    uint8_t *ciphertext, size_t size, const
                                                    uint8_t *iv, size_t ivSize, const
                                                    uint8_t *aad, size_t aadSize, const
                                                    uint8_t *key, size_t keySize, uint8_t
                                                    *tag, size_t tagSize, caam_key_type_t
                                                    blackKeyType)
```

Encrypts AES and tags using CCM block mode using black key.

Puts AES CCM encrypt and tag descriptor to CAAM input job ring.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** Memory for the CAAM descriptor.
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text.
- size – Size of input and output data in bytes. Zero means authentication only.
- iv – Nonce
- ivSize – Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.

- `aad` – Input additional authentication data. Can be NULL if `aadSize` is zero.
- `aadSize` – Input size in bytes of AAD. Zero means data mode only (authentication skipped).
- `key` – Input key to use for encryption
- `keySize` – Size of the input key, in bytes. Must be 16, 24, or 32.
- `tag` – **[out]** Generated output tag. Set to NULL to skip tag processing.
- `tagSize` – Input size of the tag to generate, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.
- `blackKeyType` – Type of black key

Returns

Status from job descriptor push

```
status_t CAAM_AES_DecryptTagCcmNonBlockingExtended(CAAM_Type *base, caam_handle_t
                                                    *handle, caam_desc_aes_ccm_t
                                                    descriptor, const uint8_t *ciphertext,
                                                    uint8_t *plaintext, size_t size, const
                                                    uint8_t *iv, size_t ivSize, const
                                                    uint8_t *aad, size_t aadSize, const
                                                    uint8_t *key, size_t keySize, const
                                                    uint8_t *tag, size_t tagSize,
                                                    caam_key_type_t blackKeyType)
```

Decrypts AES and authenticates using CCM block mode using black key.

Puts AES CCM decrypt and check tag descriptor to CAAM input job ring.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies jobRing.
- `descriptor` – **[out]** Memory for the CAAM descriptor.
- `ciphertext` – Input cipher text to decrypt
- `plaintext` – **[out]** Output plain text.
- `size` – Size of input and output data in bytes. Zero means authentication data only.
- `iv` – Nonce
- `ivSize` – Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
- `aad` – Input additional authentication data. Can be NULL if `aadSize` is zero.
- `aadSize` – Input size in bytes of AAD. Zero means data mode only (authentication data skipped).
- `key` – Input key to use for decryption
- `keySize` – Size of the input key, in bytes. Must be 16, 24, or 32.
- `tag` – Received tag. Set to NULL to skip tag processing.
- `tagSize` – Input size of the received tag to compare with the computed tag, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.
- `blackKeyType` – Type of black key

Returns

Status from job descriptor push

```
status_t CAAM_AES_EncryptTagGcmNonBlocking(CAAM_Type *base, caam_handle_t *handle,
                                             caam_desc_aes_gcm_t descriptor, const
                                             uint8_t *plaintext, uint8_t *ciphertext, size_t
                                             size, const uint8_t *iv, size_t ivSize, const
                                             uint8_t *aad, size_t aadSize, const uint8_t
                                             *key, size_t keySize, uint8_t *tag, size_t
                                             tagSize)
```

Encrypts AES and tags using GCM block mode.

Encrypts AES and optionally tags using GCM block mode. If plaintext is NULL, only the GHASH is calculated and output in the 'tag' field. Puts AES GCM encrypt and tag descriptor to CAAM input job ring.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** Memory for the CAAM descriptor.
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text.
- size – Size of input and output data in bytes
- iv – Input initial vector
- ivSize – Size of the IV
- aad – Input additional authentication data
- aadSize – Input size in bytes of AAD
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – **[out]** Output hash tag. Set to NULL to skip tag processing.
- tagSize – Input size of the tag to generate, in bytes. Must be 4,8,12,13,14,15 or 16.

Returns

Status from job descriptor push

```
status_t CAAM_AES_DecryptTagGcmNonBlocking(CAAM_Type *base, caam_handle_t *handle,
                                             caam_desc_aes_gcm_t descriptor, const
                                             uint8_t *ciphertext, uint8_t *plaintext, size_t
                                             size, const uint8_t *iv, size_t ivSize, const
                                             uint8_t *aad, size_t aadSize, const uint8_t
                                             *key, size_t keySize, const uint8_t *tag, size_t
                                             tagSize)
```

Decrypts AES and authenticates using GCM block mode.

Decrypts AES and optionally authenticates using GCM block mode. If ciphertext is NULL, only the GHASH is calculated and compared with the received GHASH in 'tag' field. Puts AES GCM decrypt and check tag descriptor to CAAM input job ring.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** Memory for the CAAM descriptor.
- ciphertext – Input cipher text to decrypt

- plaintext – **[out]** Output plain text.
- size – Size of input and output data in bytes
- iv – Input initial vector
- ivSize – Size of the IV
- aad – Input additional authentication data
- aadSize – Input size in bytes of AAD
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – Input hash tag to compare. Set to NULL to skip tag processing.
- tagSize – Input size of the tag, in bytes. Must be 4, 8, 12, 13, 14, 15, or 16.

Returns

Status from job descriptor push

```
status_t CAAM_AES_EncryptTagGcmNonBlockingExtended(CAAM_Type *base, caam_handle_t
                                                    *handle, caam_desc_aes_gcm_t
                                                    descriptor, const uint8_t *plaintext,
                                                    uint8_t *ciphertext, size_t size, const
                                                    uint8_t *iv, size_t ivSize, const
                                                    uint8_t *aad, size_t aadSize, const
                                                    uint8_t *key, size_t keySize, uint8_t
                                                    *tag, size_t tagSize, caam_key_type_t
                                                    blackKeyType)
```

Encrypts AES and tags using GCM block mode using black key.

Encrypts AES and optionally tags using GCM block mode. If plaintext is NULL, only the GHASH is calculated and output in the 'tag' field. Puts AES GCM encrypt and tag descriptor to CAAM input job ring. Uses black key.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** Memory for the CAAM descriptor.
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text.
- size – Size of input and output data in bytes
- iv – Input initial vector
- ivSize – Size of the IV
- aad – Input additional authentication data
- aadSize – Input size in bytes of AAD
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – **[out]** Output hash tag. Set to NULL to skip tag processing.
- tagSize – Input size of the tag to generate, in bytes. Must be 4,8,12,13,14,15 or 16.
- blackenKeyType – Type of black key

Returns

Status from job descriptor push

```
status_t CAAM_AES_DecryptTagGcmNonBlockingExtended(CAAM_Type *base, caam_handle_t
                                                    *handle, caam_desc_aes_gcm_t
                                                    descriptor, const uint8_t *ciphertext,
                                                    uint8_t *plaintext, size_t size, const
                                                    uint8_t *iv, size_t ivSize, const
                                                    uint8_t *aad, size_t aadSize, const
                                                    uint8_t *key, size_t keySize, const
                                                    uint8_t *tag, size_t tagSize,
                                                    caam_key_type_t blackKeyType)
```

Decrypts AES and authenticates using GCM block mode using black key.

Decrypts AES and optionally authenticates using GCM block mode. If ciphertext is NULL, only the GHASH is calculated and compared with the received GHASH in 'tag' field. Puts AES GCM decrypt and check tag descriptor to CAAM input job ring. Uses black key.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** Memory for the CAAM descriptor.
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text.
- size – Size of input and output data in bytes
- iv – Input initial vector
- ivSize – Size of the IV
- aad – Input additional authentication data
- aadSize – Input size in bytes of AAD
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – Input hash tag to compare. Set to NULL to skip tag processing.
- tagSize – Input size of the tag, in bytes. Must be 4, 8, 12, 13, 14, 15, or 16.
- blackenKeyType – Type of black key

Returns

Status from job descriptor push

2.22 CAAM Non-blocking DES driver

```
status_t CAAM_DES_EncryptEcbNonBlocking(CAAM_Type *base, caam_handle_t *handle,
                                          caam_desc_cipher_des_t descriptor, const uint8_t
                                          *plaintext, uint8_t *ciphertext, size_t size, const
                                          uint8_t key[8U])
```

Encrypts DES using ECB block mode.

Encrypts DES using ECB block mode.

Parameters

- base – CAAM peripheral base address

- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** memory for CAAM commands
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key – Input key to use for encryption

Returns

Status from descriptor push

```
status_t CAAM_DES_DecryptEcbNonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                                         caam_desc_cipher_des_t descriptor, const uint8_t  
                                         *ciphertext, uint8_t *plaintext, size_t size, const  
                                         uint8_t key[8U])
```

Decrypts DES using ECB block mode.

Decrypts DES using ECB block mode.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** memory for CAAM commands
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key – Input key to use for decryption

Returns

Status from descriptor push

```
status_t CAAM_DES_EncryptCbcNonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                                         caam_desc_cipher_des_t descriptor, const uint8_t  
                                         *plaintext, uint8_t *ciphertext, size_t size, const  
                                         uint8_t iv[8], const uint8_t key[8U])
```

Encrypts DES using CBC block mode.

Encrypts DES using CBC block mode.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** memory for CAAM commands
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key – Input key to use for encryption

Returns

Status from descriptor push

```
status_t CAAM_DES_DecryptCbcNonBlocking(CAAM_Type *base, caam_handle_t *handle,
                                         caam_desc_cipher_des_t descriptor, const uint8_t
                                         *ciphertext, uint8_t *plaintext, size_t size, const
                                         uint8_t iv[8], const uint8_t key[8U])
```

Decrypts DES using CBC block mode.

Decrypts DES using CBC block mode.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** memory for CAAM commands
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key – Input key to use for decryption

Returns

Status from descriptor push

```
status_t CAAM_DES_EncryptCfbNonBlocking(CAAM_Type *base, caam_handle_t *handle,
                                         caam_desc_cipher_des_t descriptor, const uint8_t
                                         *plaintext, uint8_t *ciphertext, size_t size, const
                                         uint8_t iv[8], const uint8_t key[8U])
```

Encrypts DES using CFB block mode.

Encrypts DES using CFB block mode.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** memory for CAAM commands
- plaintext – Input plaintext to encrypt
- size – Size of input data in bytes
- iv – Input initial block.
- key – Input key to use for encryption
- ciphertext – **[out]** Output ciphertext

Returns

Status from descriptor push

```
status_t CAAM_DES_DecryptCfbNonBlocking(CAAM_Type *base, caam_handle_t *handle,
                                         caam_desc_cipher_des_t descriptor, const uint8_t
                                         *ciphertext, uint8_t *plaintext, size_t size, const
                                         uint8_t iv[8], const uint8_t key[8U])
```

Decrypts DES using CFB block mode.

Decrypts DES using CFB block mode.

Parameters

- base – CAAM peripheral base address

- `handle` – Handle used for this request. Specifies `jobRing`.
- `descriptor` – **[out]** memory for CAAM commands
- `ciphertext` – Input ciphertext to decrypt
- `plaintext` – **[out]** Output plaintext
- `size` – Size of input and output data in bytes
- `iv` – Input initial block.
- `key` – Input key to use for decryption

Returns

Status from descriptor push

```
status_t CAAM_DES_EncryptOfbNonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                                         caam_desc_cipher_des_t descriptor, const uint8_t  
                                         *plaintext, uint8_t *ciphertext, size_t size, const  
                                         uint8_t iv[8], const uint8_t key[8U])
```

Encrypts DES using OFB block mode.

Encrypts DES using OFB block mode.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies `jobRing`.
- `descriptor` – **[out]** memory for CAAM commands
- `plaintext` – Input plaintext to encrypt
- `ciphertext` – **[out]** Output ciphertext
- `size` – Size of input and output data in bytes
- `iv` – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- `key` – Input key to use for encryption

Returns

Status from descriptor push

```
status_t CAAM_DES_DecryptOfbNonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                                         caam_desc_cipher_des_t descriptor, const uint8_t  
                                         *ciphertext, uint8_t *plaintext, size_t size, const  
                                         uint8_t iv[8], const uint8_t key[8U])
```

Decrypts DES using OFB block mode.

Decrypts DES using OFB block mode.

Parameters

- `base` – CAAM peripheral base address
- `handle` – Handle used for this request. Specifies `jobRing`.
- `descriptor` – **[out]** memory for CAAM commands
- `ciphertext` – Input ciphertext to decrypt
- `plaintext` – **[out]** Output plaintext
- `size` – Size of input and output data in bytes. Must be multiple of 8 bytes.
- `iv` – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.

- key – Input key to use for decryption

Returns

Status from descriptor push

status_t CAAM_DES2_EncryptEcbNonBlocking(CAAM_Type *base, caam_handle_t *handle, caam_desc_cipher_des_t descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t key1[8U], const uint8_t key2[8U])

Encrypts triple DES using ECB block mode with two keys.

Encrypts triple DES using ECB block mode with two keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** memory for CAAM commands
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from descriptor push

status_t CAAM_DES2_DecryptEcbNonBlocking(CAAM_Type *base, caam_handle_t *handle, caam_desc_cipher_des_t descriptor, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t key1[8U], const uint8_t key2[8U])

Decrypts triple DES using ECB block mode with two keys.

Decrypts triple DES using ECB block mode with two keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** memory for CAAM commands
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from descriptor push

status_t CAAM_DES2_EncryptCbcNonBlocking(CAAM_Type *base, caam_handle_t *handle, caam_desc_cipher_des_t descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[8], const uint8_t key1[8U], const uint8_t key2[8U])

Encrypts triple DES using CBC block mode with two keys.

Encrypts triple DES using CBC block mode with two keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** memory for CAAM commands
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from descriptor push

```
status_t CAAM_DES2_DecryptCbcNonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                                           caam_desc_cipher_des_t descriptor, const uint8_t  
                                           *ciphertext, uint8_t *plaintext, size_t size, const  
                                           uint8_t iv[8], const uint8_t key1[8U], const  
                                           uint8_t key2[8U])
```

Decrypts triple DES using CBC block mode with two keys.

Decrypts triple DES using CBC block mode with two keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** memory for CAAM commands
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from descriptor push

```
status_t CAAM_DES2_EncryptCfbNonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                                           caam_desc_cipher_des_t descriptor, const uint8_t  
                                           *plaintext, uint8_t *ciphertext, size_t size, const  
                                           uint8_t iv[8], const uint8_t key1[8U], const  
                                           uint8_t key2[8U])
```

Encrypts triple DES using CFB block mode with two keys.

Encrypts triple DES using CFB block mode with two keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.

- descriptor – **[out]** memory for CAAM commands
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from descriptor push

```
status_t CAAM_DES2_DecryptCfbNonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                                         caam_desc_cipher_des_t descriptor, const uint8_t  
                                         *ciphertext, uint8_t *plaintext, size_t size, const  
                                         uint8_t iv[8], const uint8_t key1[8U], const  
                                         uint8_t key2[8U])
```

Decrypts triple DES using CFB block mode with two keys.

Decrypts triple DES using CFB block mode with two keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** memory for CAAM commands
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from descriptor push

```
status_t CAAM_DES2_EncryptOfbNonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                                         caam_desc_cipher_des_t descriptor, const uint8_t  
                                         *plaintext, uint8_t *ciphertext, size_t size, const  
                                         uint8_t iv[8], const uint8_t key1[8U], const  
                                         uint8_t key2[8U])
```

Encrypts triple DES using OFB block mode with two keys.

Encrypts triple DES using OFB block mode with two keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** memory for CAAM commands
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes

- *iv* – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- *key1* – First input key for key bundle
- *key2* – Second input key for key bundle

Returns

Status from descriptor push

```
status_t CAAM_DES2_DecryptOfbNonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                                           caam_desc_cipher_des_t descriptor, const uint8_t  
                                           *ciphertext, uint8_t *plaintext, size_t size, const  
                                           uint8_t iv[8], const uint8_t key1[8U], const  
                                           uint8_t key2[8U])
```

Decrypts triple DES using OFB block mode with two keys.

Decrypts triple DES using OFB block mode with two keys.

Parameters

- *base* – CAAM peripheral base address
- *handle* – Handle used for this request. Specifies jobRing.
- *descriptor* – **[out]** memory for CAAM commands
- *ciphertext* – Input ciphertext to decrypt
- *plaintext* – **[out]** Output plaintext
- *size* – Size of input and output data in bytes
- *iv* – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- *key1* – First input key for key bundle
- *key2* – Second input key for key bundle

Returns

Status from descriptor push

```
status_t CAAM_DES3_EncryptEcbNonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                                           caam_desc_cipher_des_t descriptor, const uint8_t  
                                           *plaintext, uint8_t *ciphertext, size_t size, const  
                                           uint8_t key1[8U], const uint8_t key2[8U], const  
                                           uint8_t key3[8U])
```

Encrypts triple DES using ECB block mode with three keys.

Encrypts triple DES using ECB block mode with three keys.

Parameters

- *base* – CAAM peripheral base address
- *handle* – Handle used for this request. Specifies jobRing.
- *descriptor* – **[out]** memory for CAAM commands
- *plaintext* – Input plaintext to encrypt
- *ciphertext* – **[out]** Output ciphertext
- *size* – Size of input and output data in bytes. Must be multiple of 8 bytes.
- *key1* – First input key for key bundle
- *key2* – Second input key for key bundle
- *key3* – Third input key for key bundle

Returns

Status from descriptor push

status_t CAAM_DES3_DecryptEcbNonBlocking(CAAM_Type *base, *caam_handle_t* *handle, *caam_desc_cipher_des_t* descriptor, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t key1[8U], const uint8_t key2[8U], const uint8_t key3[8U])

Decrypts triple DES using ECB block mode with three keys.

Decrypts triple DES using ECB block mode with three keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** memory for CAAM commands
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from descriptor push

status_t CAAM_DES3_EncryptCbcNonBlocking(CAAM_Type *base, *caam_handle_t* *handle, *caam_desc_cipher_des_t* descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[8], const uint8_t key1[8U], const uint8_t key2[8U], const uint8_t key3[8U])

Encrypts triple DES using CBC block mode with three keys.

Encrypts triple DES using CBC block mode with three keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** memory for CAAM commands
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from descriptor push

```
status_t CAAM_DES3_DecryptCbcNonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                                           caam_desc_cipher_des_t descriptor, const uint8_t  
                                           *ciphertext, uint8_t *plaintext, size_t size, const  
                                           uint8_t iv[8], const uint8_t key1[8U], const  
                                           uint8_t key2[8U], const uint8_t key3[8U])
```

Decrypts triple DES using CBC block mode with three keys.

Decrypts triple DES using CBC block mode with three keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** memory for CAAM commands
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from descriptor push

```
status_t CAAM_DES3_EncryptCfbNonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                                           caam_desc_cipher_des_t descriptor, const uint8_t  
                                           *plaintext, uint8_t *ciphertext, size_t size, const  
                                           uint8_t iv[8], const uint8_t key1[8U], const  
                                           uint8_t key2[8U], const uint8_t key3[8U])
```

Encrypts triple DES using CFB block mode with three keys.

Encrypts triple DES using CFB block mode with three keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** memory for CAAM commands
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from descriptor push

```
status_t CAAM_DES3_DecryptCfbNonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                                         caam_desc_cipher_des_t descriptor, const uint8_t  
                                         *ciphertext, uint8_t *plaintext, size_t size, const  
                                         uint8_t iv[8], const uint8_t key1[8U], const  
                                         uint8_t key2[8U], const uint8_t key3[8U])
```

Decrypts triple DES using CFB block mode with three keys.

Decrypts triple DES using CFB block mode with three keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** memory for CAAM commands
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from descriptor push

```
status_t CAAM_DES3_EncryptOfbNonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                                         caam_desc_cipher_des_t descriptor, const uint8_t  
                                         *plaintext, uint8_t *ciphertext, size_t size, const  
                                         uint8_t iv[8], const uint8_t key1[8U], const  
                                         uint8_t key2[8U], const uint8_t key3[8U])
```

Encrypts triple DES using OFB block mode with three keys.

Encrypts triple DES using OFB block mode with three keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** memory for CAAM commands
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from descriptor push

```
status_t CAAM_DES3_DecryptOfbNonBlocking(CAAM_Type *base, caam_handle_t *handle,
                                         caam_desc_cipher_des_t descriptor, const uint8_t
                                         *ciphertext, uint8_t *plaintext, size_t size, const
                                         uint8_t iv[8], const uint8_t key1[8U], const
                                         uint8_t key2[8U], const uint8_t key3[8U])
```

Decrypts triple DES using OFB block mode with three keys.

Decrypts triple DES using OFB block mode with three keys.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- descriptor – **[out]** memory for CAAM commands
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from descriptor push

2.23 CAAM Non-blocking HASH driver

```
status_t CAAM_HASH_UpdateNonBlocking(caam_hash_ctx_t *ctx, const uint8_t *input, size_t
                                     inputSize)
```

Add input address and size to input data table.

Add data input pointer to a table maintained internally in the context. Each call of this function creates one entry in the table. The entry consists of the input pointer and inputSize. All entries created by one or multiple calls of this function can be processed in one call to CAAM_HASH_FinishNonBlocking() function. Individual entries can point to non-continuous data in the memory. The processing will occur in the order in which the CAAM_HASH_UpdateNonBlocking() have been called.

Memory pointers will be later accessed by CAAM (at time of CAAM_HASH_FinishNonBlocking()), so the memory must stay valid until CAAM_HASH_FinishNonBlocking() has been called and CAAM completes the processing.

Parameters

- ctx – **[inout]** HASH context
- input – Input data
- inputSize – Size of input data in bytes

Returns

Status of the hash update operation

status_t CAAM_HASH_FinishNonBlocking(*caam_hash_ctx_t* *ctx, *caam_desc_hash_t* descriptor, *uint8_t* *output, *size_t* *outputSize)

Finalize hashing.

The actual algorithm is computed with all input data, the memory pointers are accessed by CAAM after the function returns. The input data chunks have been specified by prior calls to CAAM_HASH_UpdateNonBlocking(). The function schedules the request at CAAM, then returns. After a while, when the CAAM completes processing of the input data chunks, the result is written to the output[] array, outputSize is written and the context is cleared.

Parameters

- ctx – **[inout]** Input hash context
- descriptor – **[out]** Memory for the CAAM descriptor.
- output – **[out]** Output hash data
- outputSize – **[out]** Output parameter storing the size of the output hash in bytes

Returns

Status of the hash finish operation

status_t CAAM_HASH_NonBlocking(*CAAM_Type* *base, *caam_handle_t* *handle, *caam_desc_hash_t* descriptor, *caam_hash_algo_t* algo, const *uint8_t* *input, *size_t* inputSize, const *uint8_t* *key, *size_t* keySize, *uint8_t* *output, *size_t* *outputSize)

Create HASH on given data.

Perform the full keyed XCBC-MAC/CMAC or SHA in one function call.

Key shall be supplied if the underlaying algorithm is AES XCBC-MAC or CMAC. Key shall be NULL if the underlaying algorithm is SHA.

For XCBC-MAC, the key length must be 16. For CMAC, the key length can be the AES key lengths supported by AES engine. For MDHA the key length argument is ignored.

The function is non-blocking. The request is scheduled at CAAM.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request.
- descriptor – **[out]** Memory for the CAAM descriptor.
- algo – Underlaying algorithm to use for hash computation.
- input – Input data
- inputSize – Size of input data in bytes
- key – Input key (NULL if underlaying algorithm is SHA)
- keySize – Size of input key in bytes
- output – **[out]** Output hash data
- outputSize – **[out]** Output parameter storing the size of the output hash in bytes

Returns

Status of the one call hash operation.

2.24 Caam_nonblocking_driver_hmac

```
status_t CAAM_HMAC_NonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                               caam_desc_hash_t descriptor, caam_hash_algo_t algo,  
                               const uint8_t *input, size_t inputSize, const uint8_t *key,  
                               size_t keySize, uint8_t *output, size_t *outputSize)
```

Create Message Authentication Code (MAC) on given data.

Perform the full keyed XCBC-MAC/CMAC, or HMAC-SHA in one function call.

Key shall be supplied if the underlaying algorithm is AES XCBC-MAC, CMAC, or SHA HMAC.

For XCBC-MAC, the key length must be 16. For CMAC, the key length can be the AES key lengths supported by AES engine. For HMAC, the key can have any size, however the function will block if the supplied key is bigger than the block size of the underlying hashing algorithm (e.g. >64 bytes for SHA256).

The function is not blocking with the exception of supplying large key sizes. In that case the function will block until the large key is hashed down with the supplied hashing algorithm (as per FIPS 198-1), after which operation is resumed to calling non-blocking HMAC.

Parameters

- base – CAAM peripheral base address
- handle – Handle used for this request.
- descriptor – **[out]** Memory for the CAAM descriptor.
- algo – Underlaying algorithm to use for MAC computation.
- input – Input data
- inputSize – Size of input data in bytes
- key – Input key
- keySize – Size of input key in bytes
- output – **[out]** Output MAC data
- outputSize – **[out]** Output parameter storing the size of the output MAC in bytes

Returns

Status of the one call hash operation.

2.25 CAAM Non-blocking RNG driver

```
status_t CAAM_RNG_GetRandomDataNonBlocking(CAAM_Type *base, caam_handle_t *handle,  
                                             caam_rng_state_handle_t stateHandle,  
                                             caam_desc_rng_t descriptor, void *data,  
                                             size_t dataSize, caam_rng_random_type_t  
                                             dataType, caam_rng_generic256_t  
                                             additionalEntropy)
```

Request random data.

This function schedules the request for random data from CAAM RNG. Memory at memory pointers will be accessed by CAAM shortly after this function returns, according to actual CAAM schedule.

Parameters

- base – CAAM peripheral base address

- handle – RNG handle used for this request
- stateHandle – RNG state handle used to generate random data
- descriptor – **[out]** memory for CAAM commands
- data – **[out]** Pointer address used to store random data
- dataSize – Size of the buffer pointed by the data parameter, in bytes.
- dataType – Type of random data to be generated.
- additionalEntropy – NULL or Pointer to optional 256-bit additional entropy.

Returns

status of the request

2.26 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.

2.27 CACHE: LMEM CACHE Memory Controller

`void L1CACHE_EnableCodeCache(void)`

Enables the processor code bus cache.

`void L1CACHE_DisableCodeCache(void)`

Disables the processor code bus cache.

`void L1CACHE_InvalidateCodeCache(void)`

Invalidates the processor code bus cache.

`void L1CACHE_InvalidateCodeCacheByRange(uint32_t address, uint32_t size_byte)`

Invalidates processor code bus cache by range.

Note: Address and size should be aligned to “L1CODCACHE_LINESIZE_BYTE”. The startAddr here will be forced to align to L1CODEBUSCACHE_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.

`void L1CACHE_CleanCodeCache(void)`

Cleans the processor code bus cache.

`void L1CACHE_CleanCodeCacheByRange(uint32_t address, uint32_t size_byte)`

Cleans processor code bus cache by range.

Note: Address and size should be aligned to “L1CODEBUSCACHE_LINESIZE_BYTE”. The startAddr here will be forced to align to L1CODEBUSCACHE_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.

`void L1CACHE_CleanInvalidateCodeCache(void)`

Cleans and invalidates the processor code bus cache.

`void L1CACHE_CleanInvalidateCodeCacheByRange(uint32_t address, uint32_t size_byte)`

Cleans and invalidate processor code bus cache by range.

Note: Address and size should be aligned to “L1CODEBUSCACHE_LINESIZE_BYTE”. The startAddr here will be forced to align to L1CODEBUSCACHE_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.

static inline void L1CACHE_EnableCodeCacheWriteBuffer(bool enable)

Enables/disables the processor code bus write buffer.

Parameters

- enable – The enable or disable flag. true - enable the code bus write buffer.
false - disable the code bus write buffer.

void L1CACHE_EnableSystemCache(void)

Enables the processor system bus cache.

void L1CACHE_DisableSystemCache(void)

Disables the processor system bus cache.

void L1CACHE_InvalidateSystemCache(void)

Invalidates the processor system bus cache.

void L1CACHE_InvalidateSystemCacheByRange(uint32_t address, uint32_t size_byte)

Invalidates processor system bus cache by range.

Note: Address and size should be aligned to “L1SYSTEMBUSCACHE_LINESIZE_BYTE”. The startAddr here will be forced to align to L1SYSTEMBUSCACHE_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.

void L1CACHE_CleanSystemCache(void)

Cleans the processor system bus cache.

void L1CACHE_CleanSystemCacheByRange(uint32_t address, uint32_t size_byte)

Cleans processor system bus cache by range.

Note: Address and size should be aligned to “L1SYSTEMBUSCACHE_LINESIZE_BYTE”. The startAddr here will be forced to align to L1SYSTEMBUSCACHE_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.

void L1CACHE_CleanInvalidateSystemCache(void)

Cleans and invalidates the processor system bus cache.

void L1CACHE_CleanInvalidateSystemCacheByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates processor system bus cache by range.

Note: Address and size should be aligned to “L1SYSTEMBUSCACHE_LINESIZE_BYTE”. The startAddr here will be forced to align to L1SYSTEMBUSCACHE_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 Clean and Invalidated.

```
static inline void L1CACHE_EnableSystemCacheWriteBuffer(bool enable)
```

Enables/disables the processor system bus write buffer.

Parameters

- enable – The enable or disable flag. true - enable the code bus write buffer.
false - disable the code bus write buffer.

```
void L1CACHE_InvalidateICacheByRange(uint32_t address, uint32_t size_byte)
```

Invalidates cortex-m4 L1 instrument cache by range.

Note: The start address and size_byte should be 16-Byte(FSL_FEATURE_L1ICACHE_LINESIZE_BYTE) aligned.

Parameters

- address – The start address of the memory to be invalidated.
- size_byte – The memory size.

```
static inline void L1CACHE_InvalidateDCacheByRange(uint32_t address, uint32_t size_byte)
```

Invalidates cortex-m4 L1 data cache by range.

Note: The start address and size_byte should be 16-Byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned.

Parameters

- address – The start address of the memory to be invalidated.
- size_byte – The memory size.

```
void L1CACHE_CleanDCacheByRange(uint32_t address, uint32_t size_byte)
```

Cleans cortex-m4 L1 data cache by range.

Note: The start address and size_byte should be 16-Byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned.

Parameters

- address – The start address of the memory to be cleaned.
- size_byte – The memory size.

```
void L1CACHE_CleanInvalidateDCacheByRange(uint32_t address, uint32_t size_byte)
```

Cleans and Invalidates cortex-m4 L1 data cache by range.

Note: The start address and size_byte should be 16-Byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned.

Parameters

- address – The start address of the memory to be clean and invalidated.

- size_byte – The memory 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 16-Byte due to the cache operation unit FSL_FEATURE_L1ICACHE_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

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

static inline void DCACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates data cache by range.

Note: Address and size should be aligned to 16-Byte due to the cache operation unit FSL_FEATURE_L1DCACHE_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

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

static inline void DCACHE_CleanByRange(uint32_t address, uint32_t size_byte)

Clean data cache by range.

Note: Address and size should be aligned to 16-Byte due to the cache operation unit FSL_FEATURE_L1DCACHE_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

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

static inline void DCACHE_CleanInvalidateByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates data cache by range.

Note: Address and size should be aligned to 16-Byte due to the cache operation unit FSL_FEATURE_L1DCACHE_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be Cleaned and Invalidated.

FSL_CACHE_DRIVER_VERSION

cache driver version.

L1CODEBUSCACHE_LINESIZE_BYTE

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

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

L1SYSTEMBUSCACHE_LINESIZE_BYTE

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

2.28 CDOG

status_t CDOG_Init(CDOG_Type *base, *cdog_config_t* *conf)

Initialize CDOG.

This function initializes CDOG block and setting.

Parameters

- base – CDOG peripheral base address
- conf – CDOG configuration structure

Returns

Status of the init operation

void CDOG_Deinit(CDOG_Type *base)

Deinitialize CDOG.

This function deinitializes CDOG secure counter.

Parameters

- base – CDOG peripheral base address

void CDOG_GetDefaultConfig(*cdog_config_t* *conf)

Sets the default configuration of CDOG.

This function initialize CDOG config structure to default values.

Parameters

- conf – CDOG configuration structure

void CDOG_Stop(CDOG_Type *base, uint32_t stop)

Stops secure counter and instruction timer.

This function stops instruction timer and secure counter. This also change state of CDOG to IDLE.

Parameters

- base – CDOG peripheral base address
- stop – expected value which will be compared with value of secure counter

void CDOG_Start(CDOG_Type *base, uint32_t reload, uint32_t start)

Sets secure counter and instruction timer values.

This function sets value in RELOAD and START registers for instruction timer and secure counter

Parameters

- base – CDOG peripheral base address

- reload – reload value
- start – start value

void CDOG_Check(CDOG_Type *base, uint32_t check)

Checks secure counter.

This function compares stop value in handler with secure counter value by writing to RELOAD register.

Parameters

- base – CDOG peripheral base address
- check – expected (stop) value

void CDOG_Set(CDOG_Type *base, uint32_t stop, uint32_t reload, uint32_t start)

Sets secure counter and instruction timer values.

This function sets value in STOP, RELOAD and START registers for instruction timer and secure counter.

Parameters

- base – CDOG peripheral base address
- stop – expected value which will be compared with value of secure counter
- reload – reload value for instruction timer
- start – start value for secure timer

void CDOG_Add(CDOG_Type *base, uint32_t add)

Add value to secure counter.

This function add specified value to secure counter.

Parameters

- base – CDOG peripheral base address.
- add – Value to be added.

void CDOG_Add1(CDOG_Type *base)

Add 1 to secure counter.

This function add 1 to secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Add16(CDOG_Type *base)

Add 16 to secure counter.

This function add 16 to secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Add256(CDOG_Type *base)

Add 256 to secure counter.

This function add 256 to secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Sub(CDOG_Type *base, uint32_t sub)

brief Subtract value to secure counter

This function subtract specified value to secure counter.

param base CDOG peripheral base address. param sub Value to be subtracted.

void CDOG_Sub1(CDOG_Type *base)

Subtract 1 from secure counter.

This function subtract specified 1 from secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Sub16(CDOG_Type *base)

Subtract 16 from secure counter.

This function subtract specified 16 from secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Sub256(CDOG_Type *base)

Subtract 256 from secure counter.

This function subtract specified 256 from secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_WritePersistent(CDOG_Type *base, uint32_t value)

Set the CDOG persistent word.

Parameters

- base – CDOG peripheral base address.
- value – The value to be written.

uint32_t CDOG_ReadPersistent(CDOG_Type *base)

Get the CDOG persistent word.

Parameters

- base – CDOG peripheral base address.

Returns

The persistent word.

FSL_CDOG_DRIVER_VERSION

Defines CDOG driver version 2.1.3.

Change log:

- Version 2.1.3
 - Re-design multiple instance IRQs and Clocks
 - Add fix for RESTART command errata
- Version 2.1.2
 - Support multiple IRQs
 - Fix default CONTROL values
- Version 2.1.1

- Remove bit CONTROL[CONTROL_CTRL]
- Version 2.1.0
 - Rename CWT to CDOG
- Version 2.0.2
 - Fix MISRA-2012 issues
- Version 2.0.1
 - Fix doxygen issues
- Version 2.0.0
 - initial version

enum __cdog_debug_Action_ctrl_enum

Values:

enumerator kCDOG_DebugHaltCtrl_Run

enumerator kCDOG_DebugHaltCtrl_Pause

enum __cdog_irq_pause_ctrl_enum

Values:

enumerator kCDOG_IrqPauseCtrl_Run

enumerator kCDOG_IrqPauseCtrl_Pause

enum __cdog_fault_ctrl_enum

Values:

enumerator kCDOG_FaultCtrl_EnableReset

enumerator kCDOG_FaultCtrl_EnableInterrupt

enumerator kCDOG_FaultCtrl_NoAction

enum __code_lock_ctrl_enum

Values:

enumerator kCDOG_LockCtrl_Lock

enumerator kCDOG_LockCtrl_Unlock

typedef uint32_t secure_counter_t

SC_ADD(add)

SC_ADD1

SC_ADD16

SC_ADD256

SC_SUB(sub)

SC_SUB1

SC_SUB16

SC_SUB256

SC_CHECK(val)

struct cdog_config_t

#include <fsl_cdog.h>

2.29 Clock Driver

enum _clock_lpcg

Clock LPCG index.

Values:

enumerator kCLOCK_M7

Clock LPCG M7.

enumerator kCLOCK_M4

Clock LPCG M4.

enumerator kCLOCK_Sim_M7

Clock LPCG SIM M7.

enumerator kCLOCK_Sim_M

Clock LPCG SIM M4.

enumerator kCLOCK_Sim_Disb

Clock LPCG SIM DISP.

enumerator kCLOCK_Sim_Per

Clock LPCG SIM PER.

enumerator kCLOCK_Sim_Lpsr

Clock LPCG SIM LPSR.

enumerator kCLOCK_Anadig

Clock LPCG Anadig.

enumerator kCLOCK_Dcdc

Clock LPCG DCDC.

enumerator kCLOCK_Src

Clock LPCG SRC.

enumerator kCLOCK_Ccm

Clock LPCG CCM.

enumerator kCLOCK_Gpc

Clock LPCG GPC.

enumerator kCLOCK_Ssarc

Clock LPCG SSARC.

enumerator kCLOCK_Sim_R

Clock LPCG SIM_R.

enumerator kCLOCK_Wdog1

Clock LPCG WDOG1.

enumerator kCLOCK_Wdog2

Clock LPCG WDOG2.

enumerator kCLOCK_Wdog3

Clock LPCG WDOG3.

enumerator kCLOCK_Wdog4

Clock LPCG WDOG4.

enumerator kCLOCK_Ewm0

Clock LPCG EWM0.

enumerator kCLOCK_Sema
Clock LPCG SEMA.

enumerator kCLOCK_Mu_A
Clock LPCG MU_A.

enumerator kCLOCK_Mu_B
Clock LPCG MU_B.

enumerator kCLOCK_Edma
Clock LPCG EDMA.

enumerator kCLOCK_Edma_Lpsr
Clock LPCG EDMA_LPSR.

enumerator kCLOCK_Romcp
Clock LPCG ROMCP.

enumerator kCLOCK_Ocram
Clock LPCG OCRAM.

enumerator kCLOCK_Flexram
Clock LPCG FLEXRAM.

enumerator kCLOCK_Lmem
Clock LPCG Lmem.

enumerator kCLOCK_Flexspi1
Clock LPCG Flexspi1.

enumerator kCLOCK_Flexspi2
Clock LPCG Flexspi2.

enumerator kCLOCK_Rdc
Clock LPCG RDC.

enumerator kCLOCK_M7_Xrdc
Clock LPCG M7 XRDC.

enumerator kCLOCK_M4_Xrdc
Clock LPCG M4 XRDC.

enumerator kCLOCK_Semc
Clock LPCG SEMC.

enumerator kCLOCK_Xecc
Clock LPCG XECC.

enumerator kCLOCK_Iee
Clock LPCG IEE.

enumerator kCLOCK_Key_Manager
Clock LPCG KEY_MANAGER.

enumerator kCLOCK_Puf
Clock LPCG PUF.

enumerator kCLOCK_Ocotp
Clock LPCG OSOTP.

enumerator kCLOCK_Snvs_Hp
Clock LPCG SNVS_HP.

enumerator kCLOCK_Snvs
Clock LPCG SNVS.

enumerator kCLOCK_Caam
Clock LPCG Caam.

enumerator kCLOCK_Jtag_Mux
Clock LPCG JTAG_MUX.

enumerator kCLOCK_Cstrace
Clock LPCG CSTRACE.

enumerator kCLOCK_Xbar1
Clock LPCG XBAR1.

enumerator kCLOCK_Xbar2
Clock LPCG XBAR2.

enumerator kCLOCK_Xbar3
Clock LPCG XBAR3.

enumerator kCLOCK_Aoi1
Clock LPCG AOI1.

enumerator kCLOCK_Aoi2
Clock LPCG AOI2.

enumerator kCLOCK_Adc_Etc
Clock LPCG ADC_ETC.

enumerator kCLOCK_Iomuxc
Clock LPCG IOMUXC.

enumerator kCLOCK_Iomuxc_Lpsr
Clock LPCG IOMUXC_LPSR.

enumerator kCLOCK_Gpio
Clock LPCG GPIO.

enumerator kCLOCK_Kpp
Clock LPCG KPP.

enumerator kCLOCK_Flexio1
Clock LPCG FLEXIO1.

enumerator kCLOCK_Flexio2
Clock LPCG FLEXIO2.

enumerator kCLOCK_Lpadc1
Clock LPCG LPADC1.

enumerator kCLOCK_Lpadc2
Clock LPCG LPADC2.

enumerator kCLOCK_Dac
Clock LPCG DAC.

enumerator kCLOCK_Acmp1
Clock LPCG ACMP1.

enumerator kCLOCK_Acmp2
Clock LPCG ACMP2.

enumerator kCLOCK_Acmp3
Clock LPCG ACMP3.

enumerator kCLOCK_Acmp4
Clock LPCG ACMP4.

enumerator kCLOCK_Pit1
Clock LPCG PIT1.

enumerator kCLOCK_Pit2
Clock LPCG PIT2.

enumerator kCLOCK_Gpt1
Clock LPCG GPT1.

enumerator kCLOCK_Gpt2
Clock LPCG GPT2.

enumerator kCLOCK_Gpt3
Clock LPCG GPT3.

enumerator kCLOCK_Gpt4
Clock LPCG GPT4.

enumerator kCLOCK_Gpt5
Clock LPCG GPT5.

enumerator kCLOCK_Gpt6
Clock LPCG GPT6.

enumerator kCLOCK_Qtimer1
Clock LPCG QTIMER1.

enumerator kCLOCK_Qtimer2
Clock LPCG QTIMER2.

enumerator kCLOCK_Qtimer3
Clock LPCG QTIMER3.

enumerator kCLOCK_Qtimer4
Clock LPCG QTIMER4.

enumerator kCLOCK_Enc1
Clock LPCG Enc1.

enumerator kCLOCK_Enc2
Clock LPCG Enc2.

enumerator kCLOCK_Enc3
Clock LPCG Enc3.

enumerator kCLOCK_Enc4
Clock LPCG Enc4.

enumerator kCLOCK_Hrtimer
Clock LPCG Hrtimer.

enumerator kCLOCK_Pwm1
Clock LPCG PWM1.

enumerator kCLOCK_Pwm2
Clock LPCG PWM2.

enumerator kCLOCK_Pwm3
Clock LPCG PWM3.

enumerator kCLOCK_Pwm4
Clock LPCG PWM4.

enumerator kCLOCK_Can1
Clock LPCG CAN1.

enumerator kCLOCK_Can2
Clock LPCG CAN2.

enumerator kCLOCK_Can3
Clock LPCG CAN3.

enumerator kCLOCK_Lpuart1
Clock LPCG LPUART1.

enumerator kCLOCK_Lpuart2
Clock LPCG LPUART2.

enumerator kCLOCK_Lpuart3
Clock LPCG LPUART3.

enumerator kCLOCK_Lpuart4
Clock LPCG LPUART4.

enumerator kCLOCK_Lpuart5
Clock LPCG LPUART5.

enumerator kCLOCK_Lpuart6
Clock LPCG LPUART6.

enumerator kCLOCK_Lpuart7
Clock LPCG LPUART7.

enumerator kCLOCK_Lpuart8
Clock LPCG LPUART8.

enumerator kCLOCK_Lpuart9
Clock LPCG LPUART9.

enumerator kCLOCK_Lpuart10
Clock LPCG LPUART10.

enumerator kCLOCK_Lpuart11
Clock LPCG LPUART11.

enumerator kCLOCK_Lpuart12
Clock LPCG LPUART12.

enumerator kCLOCK_Lpi2c1
Clock LPCG LPI2C1.

enumerator kCLOCK_Lpi2c2
Clock LPCG LPI2C2.

enumerator kCLOCK_Lpi2c3
Clock LPCG LPI2C3.

enumerator kCLOCK_Lpi2c4
Clock LPCG LPI2C4.

enumerator kCLOCK_Lpi2c5
Clock LPCG LPI2C5.

enumerator kCLOCK_Lpi2c6
Clock LPCG LPI2C6.

enumerator kCLOCK_Lpspi1
Clock LPCG LPSPI1.

enumerator kCLOCK_Lpspi2
Clock LPCG LPSPI2.

enumerator kCLOCK_Lpspi3
Clock LPCG LPSPI3.

enumerator kCLOCK_Lpspi4
Clock LPCG LPSPI4.

enumerator kCLOCK_Lpspi5
Clock LPCG LPSPI5.

enumerator kCLOCK_Lpspi6
Clock LPCG LPSPI6.

enumerator kCLOCK_Sim1
Clock LPCG SIM1.

enumerator kCLOCK_Sim2
Clock LPCG SIM2.

enumerator kCLOCK_Enet
Clock LPCG ENET.

enumerator kCLOCK_Enet_1g
Clock LPCG ENET 1G.

enumerator kCLOCK_Usb
Clock LPCG USB.

enumerator kCLOCK_Cdog
Clock LPCG CDOG.

enumerator kCLOCK_Usdhc1
Clock LPCG USDHC1.

enumerator kCLOCK_Usdhc2
Clock LPCG USDHC2.

enumerator kCLOCK_Asrc
Clock LPCG ASRC.

enumerator kCLOCK_Mqs
Clock LPCG MQS.

enumerator kCLOCK_Pdm
Clock LPCG PDM.

enumerator kCLOCK_Spdif
Clock LPCG SPDIF.

enumerator kCLOCK_Sai1
Clock LPCG SAI1.

enumerator kCLOCK_Sai2
Clock LPCG SAI2.

enumerator kCLOCK_Sai3
Clock LPCG SAI3.

enumerator kCLOCK_Sai4
Clock LPCG SAI4.

enumerator kCLOCK_Pxp
Clock LPCG PXP.

enumerator kCLOCK_Gpu2d
Clock LPCG GPU2D.

enumerator kCLOCK_Lcdif
Clock LPCG LCDIF.

enumerator kCLOCK_Lcdifv2
Clock LPCG LCDIFV2.

enumerator kCLOCK_Mipi_Dsi
Clock LPCG MIPI DSI.

enumerator kCLOCK_Mipi_Csi
Clock LPCG MIPI CSI.

enumerator kCLOCK_Csi
Clock LPCG CSI.

enumerator kCLOCK_Dcic_Mipi
Clock LPCG DCIC MIPI.

enumerator kCLOCK_Dcic_Lcd
Clock LPCG DCIC LCD.

enumerator kCLOCK_Video_Mux
Clock LPCG VIDEO MUX.

enumerator kCLOCK_Uniq_Edt_I
Clock LPCG Uniq_Edt_I.

enumerator kCLOCK_IpInvalid
Invalid value.

enum _clock_name

Clock name.

Values:

enumerator kCLOCK_OscRc16M
16MHz RC Oscillator.

enumerator kCLOCK_OscRc48M
48MHz RC Oscillator.

enumerator kCLOCK_OscRc48MDiv2
48MHz RC Oscillator Div2.

enumerator kCLOCK_OscRc400M
400MHz RC Oscillator.

enumerator kCLOCK_Osc24M
24MHz Oscillator.

enumerator kCLOCK_Osc24MOut
48MHz Oscillator Out.

enumerator kCLOCK_ArmPll
ARM PLL.

enumerator kCLOCK_ArmPllOut
ARM PLL Out.

enumerator kCLOCK_SysPll2
SYS PLL2.

enumerator kCLOCK_SysPll2Out
SYS PLL2 OUT.

enumerator kCLOCK_SysPll2Pfd0
SYS PLL2 PFD0.

enumerator kCLOCK_SysPll2Pfd1
SYS PLL2 PFD1.

enumerator kCLOCK_SysPll2Pfd2
SYS PLL2 PFD2.

enumerator kCLOCK_SysPll2Pfd3
SYS PLL2 PFD3.

enumerator kCLOCK_SysPll3
SYS PLL3.

enumerator kCLOCK_SysPll3Out
SYS PLL3 OUT.

enumerator kCLOCK_SysPll3Div2
SYS PLL3 DIV2

enumerator kCLOCK_SysPll3Pfd0
SYS PLL3 PFD0.

enumerator kCLOCK_SysPll3Pfd1
SYS PLL3 PFD1

enumerator kCLOCK_SysPll3Pfd2
SYS PLL3 PFD2

enumerator kCLOCK_SysPll3Pfd3
SYS PLL3 PFD3

enumerator kCLOCK_SysPll1
SYS PLL1.

enumerator kCLOCK_SysPll1Out
SYS PLL1 OUT.

enumerator kCLOCK_SysPll1Div2
SYS PLL1 DIV2.

enumerator kCLOCK_SysPll1Div5
SYS PLL1 DIV5.

enumerator kCLOCK_AudioPll
SYS AUDIO PLL.

enumerator kCLOCK_AudioPllOut
SYS AUDIO PLL OUT.

enumerator kCLOCK_VideoPll
SYS VIDEO PLL.

enumerator kCLOCK_VideoPllOut
SYS VIDEO PLL OUT.

enumerator kCLOCK_CpuClk
SYS CPU CLK.

enumerator kCLOCK_CoreSysClk
SYS CORE SYS CLK.

enumerator kCLOCK_Reserved
Reserved.

enum _clock_root
Root clock index.

Values:

enumerator kCLOCK_Root_M7
CLOCK Root M7.

enumerator kCLOCK_Root_M4
CLOCK Root M4.

enumerator kCLOCK_Root_Bus
CLOCK Root Bus.

enumerator kCLOCK_Root_Bus_Lpsr
CLOCK Root Bus Lpsr.

enumerator kCLOCK_Root_Semc
CLOCK Root Semc.

enumerator kCLOCK_Root_Cssys
CLOCK Root Cssys.

enumerator kCLOCK_Root_Cstrace
CLOCK Root Cstrace.

enumerator kCLOCK_Root_M4_Systick
CLOCK Root M4 Systick.

enumerator kCLOCK_Root_M7_Systick
CLOCK Root M7 Systick.

enumerator kCLOCK_Root_Adc1
CLOCK Root Adc1.

enumerator kCLOCK_Root_Adc2
CLOCK Root Adc2.

enumerator kCLOCK_Root_Acmp
CLOCK Root Acmp.

enumerator kCLOCK_Root_Flexio1
CLOCK Root Flexio1.

enumerator kCLOCK_Root_Flexio2
CLOCK Root Flexio2.

enumerator kCLOCK_Root_Gpt1
CLOCK Root Gpt1.

enumerator kCLOCK_Root_Gpt2
CLOCK Root Gpt2.

enumerator kCLOCK_Root_Gpt3
CLOCK Root Gpt3.

enumerator kCLOCK_Root_Gpt4
CLOCK Root Gpt4.

enumerator kCLOCK_Root_Gpt5
CLOCK Root Gpt5.

enumerator kCLOCK_Root_Gpt6
CLOCK Root Gpt6.

enumerator kCLOCK_Root_Flexspi1
CLOCK Root Flexspi1.

enumerator kCLOCK_Root_Flexspi2
CLOCK Root Flexspi2.

enumerator kCLOCK_Root_Can1
CLOCK Root Can1.

enumerator kCLOCK_Root_Can2
CLOCK Root Can2.

enumerator kCLOCK_Root_Can3
CLOCK Root Can3.

enumerator kCLOCK_Root_Lpuart1
CLOCK Root Lpuart1.

enumerator kCLOCK_Root_Lpuart2
CLOCK Root Lpuart2.

enumerator kCLOCK_Root_Lpuart3
CLOCK Root Lpuart3.

enumerator kCLOCK_Root_Lpuart4
CLOCK Root Lpuart4.

enumerator kCLOCK_Root_Lpuart5
CLOCK Root Lpuart5.

enumerator kCLOCK_Root_Lpuart6
CLOCK Root Lpuart6.

enumerator kCLOCK_Root_Lpuart7
CLOCK Root Lpuart7.

enumerator kCLOCK_Root_Lpuart8
CLOCK Root Lpuart8.

enumerator kCLOCK_Root_Lpuart9
CLOCK Root Lpuart9.

enumerator kCLOCK_Root_Lpuart10
CLOCK Root Lpuart10.

enumerator kCLOCK_Root_Lpuart11
CLOCK Root Lpuart11.

enumerator kCLOCK_Root_Lpuart12
CLOCK Root Lpuart12.

enumerator kCLOCK_Root_Lpi2c1
CLOCK Root Lpi2c1.

enumerator kCLOCK_Root_Lpi2c2
CLOCK Root Lpi2c2.

enumerator kCLOCK_Root_Lpi2c3
CLOCK Root Lpi2c3.

enumerator kCLOCK_Root_Lpi2c4
CLOCK Root Lpi2c4.

enumerator kCLOCK_Root_Lpi2c5
CLOCK Root Lpi2c5.

enumerator kCLOCK_Root_Lpi2c6
CLOCK Root Lpi2c6.

enumerator kCLOCK_Root_Lpspi1
CLOCK Root Lpspi1.

enumerator kCLOCK_Root_Lpspi2
CLOCK Root Lpspi2.

enumerator kCLOCK_Root_Lpspi3
CLOCK Root Lpspi3.

enumerator kCLOCK_Root_Lpspi4
CLOCK Root Lpspi4.

enumerator kCLOCK_Root_Lpspi5
CLOCK Root Lpspi5.

enumerator kCLOCK_Root_Lpspi6
CLOCK Root Lpspi6.

enumerator kCLOCK_Root_Emv1
CLOCK Root Emv1.

enumerator kCLOCK_Root_Emv2
CLOCK Root Emv2.

enumerator kCLOCK_Root_Enet1
CLOCK Root Enet1.

enumerator kCLOCK_Root_Enet2
CLOCK Root Enet2.

enumerator kCLOCK_Root_Enet_25m
CLOCK Root Enet 25M.

enumerator kCLOCK_Root_Enet_Timer1
CLOCK Root Enet Timer1.

enumerator kCLOCK_Root_Enet_Timer2
CLOCK Root Enet Timer2.

enumerator kCLOCK_Root_Usdhc1
CLOCK Root Usdhc1.

enumerator kCLOCK_Root_Usdhc2
CLOCK Root Usdhc2.

enumerator kCLOCK_Root_Asrc
CLOCK Root Asrc.

enumerator kCLOCK_Root_Mqs
CLOCK Root Mqs.

enumerator kCLOCK_Root_Mic
CLOCK Root MIC.

enumerator kCLOCK_Root_Spdif
CLOCK Root Spdif

enumerator kCLOCK_Root_Sai1
CLOCK Root Sai1.

enumerator kCLOCK_Root_Sai2
CLOCK Root Sai2.

enumerator kCLOCK_Root_Sai3
CLOCK Root Sai3.

enumerator kCLOCK_Root_Sai4
CLOCK Root Sai4.

enumerator kCLOCK_Root_Gc355
CLOCK Root Gc355.

enumerator kCLOCK_Root_Lcdif
CLOCK Root Lcdif.

enumerator kCLOCK_Root_Lcdifv2
CLOCK Root Lcdifv2.

enumerator kCLOCK_Root_Mipi_Ref
CLOCK Root Mipi Ref.

enumerator kCLOCK_Root_Mipi_Esc
CLOCK Root Mipi Esc.

enumerator kCLOCK_Root_Csi2
CLOCK Root Csi2.

enumerator kCLOCK_Root_Csi2_Esc
CLOCK Root Csi2 Esc.

enumerator kCLOCK_Root_Csi2_Ui
CLOCK Root Csi2 Ui.

enumerator kCLOCK_Root_Csi
CLOCK Root Csi.

enumerator kCLOCK_Root_Cko1
CLOCK Root CKo1.

enumerator kCLOCK_Root_Cko2
CLOCK Root CKo2.

enum _clock_root_mux_source

The enumerator of clock roots' clock source mux value.

Values:

enumerator kCLOCK_M7_ClockRoot_MuxOscRc48MDiv2
M7 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_M7_ClockRoot_MuxOsc24MOut
M7 mux from MuxOsc24MOut.

enumerator kCLOCK_M7_ClockRoot_MuxOscRc400M
M7 mux from MuxOscRc400M.

enumerator kCLOCK_M7_ClockRoot_MuxOscRc16M
M7 mux from MuxOscRc16M.

enumerator kCLOCK_M7_ClockRoot_MuxArmPllOut
M7 mux from MuxArmPllOut.

enumerator kCLOCK_M7_ClockRoot_MuxSysPll3Out
M7 mux from MuxSysPll3Out.

enumerator kCLOCK_M4_ClockRoot_MuxOscRc48MDiv2
M4 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_M4_ClockRoot_MuxOsc24MOut
M4 mux from MuxOsc24MOut.

enumerator kCLOCK_M4_ClockRoot_MuxOscRc400M
M4 mux from MuxOscRc400M.

enumerator kCLOCK_M4_ClockRoot_MuxOscRc16M
M4 mux from MuxOscRc16M.

enumerator kCLOCK_M4_ClockRoot_MuxSysPll3Pfd3
M4 mux from MuxSysPll3Pfd3.

enumerator kCLOCK_M4_ClockRoot_MuxSysPll3Out
M4 mux from MuxSysPll3Out.

enumerator kCLOCK_M4_ClockRoot_MuxSysPll2Out
M4 mux from MuxSysPll2Out.

enumerator kCLOCK_M4_ClockRoot_MuxSysPll1Div5
M4 mux from MuxSysPll1Div5.

enumerator kCLOCK_BUS_ClockRoot_MuxOscRc48MDiv2
BUS mux from MuxOscRc48MDiv2.

enumerator kCLOCK_BUS_ClockRoot_MuxOsc24MOut
BUS mux from MuxOsc24MOut.

enumerator kCLOCK_BUS_ClockRoot_MuxOscRc400M
BUS mux from MuxOscRc400M.

enumerator kCLOCK_BUS_ClockRoot_MuxOscRc16M
BUS mux from MuxOscRc16M.

enumerator kCLOCK_BUS_ClockRoot_MuxSysPll3Out
BUS mux from MuxSysPll3Out.

enumerator kCLOCK_BUS_ClockRoot_MuxSysPll1Div5
BUS mux from MuxSysPll1Div5.

enumerator kCLOCK_BUS_ClockRoot_MuxSysPll2Out
BUS mux from MuxSysPll2Out.

enumerator kCLOCK_BUS_ClockRoot_MuxSysPll2Pfd3
BUS mux from MuxSysPll2Pfd3.

enumerator kCLOCK_BUS_LPSR_ClockRoot_MuxOscRc48MDiv2
BUS_LPSR mux from MuxOscRc48MDiv2.

enumerator kCLOCK_BUS_LPSR_ClockRoot_MuxOsc24MOut
BUS_LPSR mux from MuxOsc24MOut.

enumerator kCLOCK_BUS_LPSR_ClockRoot_MuxOscRc400M
BUS_LPSR mux from MuxOscRc400M.

enumerator kCLOCK_BUS_LPSR_ClockRoot_MuxOscRc16M
BUS_LPSR mux from MuxOscRc16M.

enumerator kCLOCK_BUS_LPSR_ClockRoot_MuxSysPll3Pfd3
BUS_LPSR mux from MuxSysPll3Pfd3.

enumerator kCLOCK_BUS_LPSR_ClockRoot_MuxSysPll3Out
BUS_LPSR mux from MuxSysPll3Out.

enumerator kCLOCK_BUS_LPSR_ClockRoot_MuxSysPll2Out
BUS_LPSR mux from MuxSysPll2Out.

enumerator kCLOCK_BUS_LPSR_ClockRoot_MuxSysPll1Div5
BUS_LPSR mux from MuxSysPll1Div5.

enumerator kCLOCK_SEMC_ClockRoot_MuxOscRc48MDiv2
SEMC mux from MuxOscRc48MDiv2.

enumerator kCLOCK_SEMC_ClockRoot_MuxOsc24MOut
SEMC mux from MuxOsc24MOut.

enumerator kCLOCK_SEMC_ClockRoot_MuxOscRc400M
SEMC mux from MuxOscRc400M.

enumerator kCLOCK_SEMC_ClockRoot_MuxOscRc16M
SEMC mux from MuxOscRc16M.

enumerator kCLOCK_SEMC_ClockRoot_MuxSysPll1Div5
SEMC mux from MuxSysPll1Div5.

enumerator kCLOCK_SEMC_ClockRoot_MuxSysPll2Out
SEMC mux from MuxSysPll2Out.

enumerator kCLOCK_SEMC_ClockRoot_MuxSysPll2Pfd1
SEMC mux from MuxSysPll2Pfd1.

enumerator kCLOCK_SEMC_ClockRoot_MuxSysPll3Pfd0
SEMC mux from MuxSysPll3Pfd0.

enumerator kCLOCK_CSSYS_ClockRoot_MuxOscRc48MDiv2
CSSYS mux from MuxOscRc48MDiv2.

enumerator kCLOCK_CSSYS_ClockRoot_MuxOsc24MOut
CSSYS mux from MuxOsc24MOut.

enumerator kCLOCK_CSSYS_ClockRoot_MuxOscRc400M
CSSYS mux from MuxOscRc400M.

enumerator kCLOCK_CSSYS_ClockRoot_MuxOscRc16M
CSSYS mux from MuxOscRc16M.

enumerator kCLOCK_CSSYS_ClockRoot_MuxSysPll3Div2
CSSYS mux from MuxSysPll3Div2.

enumerator kCLOCK_CSSYS_ClockRoot_MuxSysPll1Div5
CSSYS mux from MuxSysPll1Div5.

enumerator kCLOCK_CSSYS_ClockRoot_MuxSysPll2Out
CSSYS mux from MuxSysPll2Out.

enumerator kCLOCK_CSSYS_ClockRoot_MuxSysPll2Pfd3
CSSYS mux from MuxSysPll2Pfd3.

enumerator kCLOCK_CSTRACE_ClockRoot_MuxOscRc48MDiv2
CSTRACE mux from MuxOscRc48MDiv2.

enumerator kCLOCK_CSTRACE_ClockRoot_MuxOsc24MOut
CSTRACE mux from MuxOsc24MOut.

enumerator kCLOCK_CSTRACE_ClockRoot_MuxOscRc400M
CSTRACE mux from MuxOscRc400M.

enumerator kCLOCK_CSTRACE_ClockRoot_MuxOscRc16M
CSTRACE mux from MuxOscRc16M.

enumerator kCLOCK_CSTRACE_ClockRoot_MuxSysPll3Div2
CSTRACE mux from MuxSysPll3Div2.

enumerator kCLOCK_CSTRACE_ClockRoot_MuxSysPll1Div5
CSTRACE mux from MuxSysPll1Div5.

enumerator kCLOCK_CSTRACE_ClockRoot_MuxSysPll2Pfd1
CSTRACE mux from MuxSysPll2Pfd1.

enumerator kCLOCK_CSTRACE_ClockRoot_MuxSysPll2Out
CSTRACE mux from MuxSysPll2Out.

enumerator kCLOCK_M4_SYSTICK_ClockRoot_MuxOscRc48MDiv2
M4_SYSTICK mux from MuxOscRc48MDiv2.

enumerator kCLOCK_M4_SYSTICK_ClockRoot_MuxOsc24MOut
M4_SYSTICK mux from MuxOsc24MOut.

enumerator kCLOCK_M4_SYSTICK_ClockRoot_MuxOscRc400M
M4_SYSTICK mux from MuxOscRc400M.

enumerator kCLOCK_M4_SYSTICK_ClockRoot_MuxOscRc16M
M4_SYSTICK mux from MuxOscRc16M.

enumerator kCLOCK_M4_SYSTICK_ClockRoot_MuxSysPll3Pfd3
M4_SYSTICK mux from MuxSysPll3Pfd3.

enumerator kCLOCK_M4_SYSTICK_ClockRoot_MuxSysPll3Out
M4_SYSTICK mux from MuxSysPll3Out.

enumerator kCLOCK_M4_SYSTICK_ClockRoot_MuxSysPll2Pfd0
M4_SYSTICK mux from MuxSysPll2Pfd0.

enumerator kCLOCK_M4_SYSTICK_ClockRoot_MuxSysPll1Div5
M4_SYSTICK mux from MuxSysPll1Div5.

enumerator kCLOCK_M7_SYSTICK_ClockRoot_MuxOscRc48MDiv2
M7_SYSTICK mux from MuxOscRc48MDiv2.

enumerator kCLOCK_M7_SYSTICK_ClockRoot_MuxOsc24MOut
M7_SYSTICK mux from MuxOsc24MOut.

enumerator kCLOCK_M7_SYSTICK_ClockRoot_MuxOscRc400M
M7_SYSTICK mux from MuxOscRc400M.

enumerator kCLOCK_M7_SYSTICK_ClockRoot_MuxOscRc16M
M7_SYSTICK mux from MuxOscRc16M.

enumerator kCLOCK_M7_SYSTICK_ClockRoot_MuxSysPll2Out
M7_SYSTICK mux from MuxSysPll2Out.

enumerator kCLOCK_M7_SYSTICK_ClockRoot_MuxSysPll3Div2
M7_SYSTICK mux from MuxSysPll3Div2.

enumerator kCLOCK_M7_SYSTICK_ClockRoot_MuxSysPll1Div5
M7_SYSTICK mux from MuxSysPll1Div5.

enumerator kCLOCK_M7_SYSTICK_ClockRoot_MuxSysPll2Pfd0
M7_SYSTICK mux from MuxSysPll2Pfd0.

enumerator kCLOCK_ADC1_ClockRoot_MuxOscRc48MDiv2
ADC1 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_ADC1_ClockRoot_MuxOsc24MOut
ADC1 mux from MuxOsc24MOut.

enumerator kCLOCK_ADC1_ClockRoot_MuxOscRc400M
ADC1 mux from MuxOscRc400M.

enumerator kCLOCK_ADC1_ClockRoot_MuxOscRc16M
ADC1 mux from MuxOscRc16M.

enumerator kCLOCK_ADC1_ClockRoot_MuxSysPll3Div2
ADC1 mux from MuxSysPll3Div2.

enumerator kCLOCK_ADC1_ClockRoot_MuxSysPll1Div5
ADC1 mux from MuxSysPll1Div5.

enumerator kCLOCK_ADC1_ClockRoot_MuxSysPll2Out
ADC1 mux from MuxSysPll2Out.

enumerator kCLOCK_ADC1_ClockRoot_MuxSysPll2Pfd3
ADC1 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_ADC2_ClockRoot_MuxOscRc48MDiv2
ADC2 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_ADC2_ClockRoot_MuxOsc24MOut
ADC2 mux from MuxOsc24MOut.

enumerator kCLOCK_ADC2_ClockRoot_MuxOscRc400M
ADC2 mux from MuxOscRc400M.

enumerator kCLOCK_ADC2_ClockRoot_MuxOscRc16M
ADC2 mux from MuxOscRc16M.

enumerator kCLOCK_ADC2_ClockRoot_MuxSysPll3Div2
ADC2 mux from MuxSysPll3Div2.

enumerator kCLOCK_ADC2_ClockRoot_MuxSysPll1Div5
ADC2 mux from MuxSysPll1Div5.

enumerator kCLOCK_ADC2_ClockRoot_MuxSysPll2Out
ADC2 mux from MuxSysPll2Out.

enumerator kCLOCK_ADC2_ClockRoot_MuxSysPll2Pfd3
ADC2 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_ACMP_ClockRoot_MuxOscRc48MDiv2
ACMP mux from MuxOscRc48MDiv2.

enumerator kCLOCK_ACMP_ClockRoot_MuxOsc24MOut
ACMP mux from MuxOsc24MOut.

enumerator kCLOCK_ACMP_ClockRoot_MuxOscRc400M
ACMP mux from MuxOscRc400M.

enumerator kCLOCK_ACMP_ClockRoot_MuxOscRc16M
ACMP mux from MuxOscRc16M.

enumerator kCLOCK_ACMP_ClockRoot_MuxSysPll3Out
ACMP mux from MuxSysPll3Out.

enumerator kCLOCK_ACMP_ClockRoot_MuxSysPll1Div5
ACMP mux from MuxSysPll1Div5.

enumerator kCLOCK_ACMP_ClockRoot_MuxAudioPllOut
ACMP mux from MuxAudioPllOut.

enumerator kCLOCK_ACMP_ClockRoot_MuxSysPll2Pfd3
ACMP mux from MuxSysPll2Pfd3.

enumerator kCLOCK_FLEXIO1_ClockRoot_MuxOscRc48MDiv2
FLEXIO1 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_FLEXIO1_ClockRoot_MuxOsc24MOut
FLEXIO1 mux from MuxOsc24MOut.

enumerator kCLOCK_FLEXIO1_ClockRoot_MuxOscRc400M
FLEXIO1 mux from MuxOscRc400M.

enumerator kCLOCK_FLEXIO1_ClockRoot_MuxOscRc16M
FLEXIO1 mux from MuxOscRc16M.

enumerator kCLOCK_FLEXIO1_ClockRoot_MuxSysPll3Div2
FLEXIO1 mux from MuxSysPll3Div2.

enumerator kCLOCK_FLEXIO1_ClockRoot_MuxSysPll1Div5
FLEXIO1 mux from MuxSysPll1Div5.

enumerator kCLOCK_FLEXIO1_ClockRoot_MuxSysPll2Out
FLEXIO1 mux from MuxSysPll2Out.

enumerator kCLOCK_FLEXIO1_ClockRoot_MuxSysPll2Pfd3
FLEXIO1 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_FLEXIO2_ClockRoot_MuxOscRc48MDiv2
FLEXIO2 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_FLEXIO2_ClockRoot_MuxOsc24MOut
FLEXIO2 mux from MuxOsc24MOut.

enumerator kCLOCK_FLEXIO2_ClockRoot_MuxOscRc400M
FLEXIO2 mux from MuxOscRc400M.

enumerator kCLOCK_FLEXIO2_ClockRoot_MuxOscRc16M
FLEXIO2 mux from MuxOscRc16M.

enumerator kCLOCK_FLEXIO2_ClockRoot_MuxSysPll3Div2
FLEXIO2 mux from MuxSysPll3Div2.

enumerator kCLOCK_FLEXIO2_ClockRoot_MuxSysPll1Div5
FLEXIO2 mux from MuxSysPll1Div5.

enumerator kCLOCK_FLEXIO2_ClockRoot_MuxSysPll2Out
FLEXIO2 mux from MuxSysPll2Out.

enumerator kCLOCK_FLEXIO2_ClockRoot_MuxSysPll2Pfd3
FLEXIO2 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_GPT1_ClockRoot_MuxOscRc48MDiv2
GPT1 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_GPT1_ClockRoot_MuxOsc24MOut
GPT1 mux from MuxOsc24MOut.

enumerator kCLOCK_GPT1_ClockRoot_MuxOscRc400M
GPT1 mux from MuxOscRc400M.

enumerator kCLOCK_GPT1_ClockRoot_MuxOscRc16M
GPT1 mux from MuxOscRc16M.

enumerator kCLOCK_GPT1_ClockRoot_MuxSysPll3Div2
GPT1 mux from MuxSysPll3Div2.

enumerator kCLOCK_GPT1_ClockRoot_MuxSysPll1Div5
GPT1 mux from MuxSysPll1Div5.

enumerator kCLOCK_GPT1_ClockRoot_MuxSysPll3Pfd2
GPT1 mux from MuxSysPll3Pfd2.

enumerator kCLOCK_GPT1_ClockRoot_MuxSysPll3Pfd3
GPT1 mux from MuxSysPll3Pfd3.

enumerator kCLOCK_GPT2_ClockRoot_MuxOscRc48MDiv2
GPT2 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_GPT2_ClockRoot_MuxOsc24MOut
GPT2 mux from MuxOsc24MOut.

enumerator kCLOCK_GPT2_ClockRoot_MuxOscRc400M
GPT2 mux from MuxOscRc400M.

enumerator kCLOCK_GPT2_ClockRoot_MuxOscRc16M
GPT2 mux from MuxOscRc16M.

enumerator kCLOCK_GPT2_ClockRoot_MuxSysPll3Div2
GPT2 mux from MuxSysPll3Div2.

enumerator kCLOCK_GPT2_ClockRoot_MuxSysPll1Div5
GPT2 mux from MuxSysPll1Div5.

enumerator kCLOCK_GPT2_ClockRoot_MuxAudioPllOut
GPT2 mux from MuxAudioPllOut.

enumerator kCLOCK_GPT2_ClockRoot_MuxVideoPllOut
GPT2 mux from MuxVideoPllOut.

enumerator kCLOCK_GPT3_ClockRoot_MuxOscRc48MDiv2
GPT3 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_GPT3_ClockRoot_MuxOsc24MOut
GPT3 mux from MuxOsc24MOut.

enumerator kCLOCK_GPT3_ClockRoot_MuxOscRc400M
GPT3 mux from MuxOscRc400M.

enumerator kCLOCK_GPT3_ClockRoot_MuxOscRc16M
GPT3 mux from MuxOscRc16M.

enumerator kCLOCK_GPT3_ClockRoot_MuxSysPll3Div2
GPT3 mux from MuxSysPll3Div2.

enumerator kCLOCK_GPT3_ClockRoot_MuxSysPll1Div5
GPT3 mux from MuxSysPll1Div5.

enumerator kCLOCK_GPT3_ClockRoot_MuxAudioPllOut
GPT3 mux from MuxAudioPllOut.

enumerator kCLOCK_GPT3_ClockRoot_MuxVideoPllOut
GPT3 mux from MuxVideoPllOut.

enumerator kCLOCK_GPT4_ClockRoot_MuxOscRc48MDiv2
GPT4 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_GPT4_ClockRoot_MuxOsc24MOut
GPT4 mux from MuxOsc24MOut.

enumerator kCLOCK_GPT4_ClockRoot_MuxOscRc400M
GPT4 mux from MuxOscRc400M.

enumerator kCLOCK_GPT4_ClockRoot_MuxOscRc16M
GPT4 mux from MuxOscRc16M.

enumerator kCLOCK_GPT4_ClockRoot_MuxSysPll3Div2
GPT4 mux from MuxSysPll3Div2.

enumerator kCLOCK_GPT4_ClockRoot_MuxSysPll1Div5
GPT4 mux from MuxSysPll1Div5.

enumerator kCLOCK_GPT4_ClockRoot_MuxSysPll3Pfd2
GPT4 mux from MuxSysPll3Pfd2.

enumerator kCLOCK_GPT4_ClockRoot_MuxSysPll3Pfd3
GPT4 mux from MuxSysPll3Pfd3.

enumerator kCLOCK_GPT5_ClockRoot_MuxOscRc48MDiv2
GPT5 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_GPT5_ClockRoot_MuxOsc24MOut
GPT5 mux from MuxOsc24MOut.

enumerator kCLOCK_GPT5_ClockRoot_MuxOscRc400M
GPT5 mux from MuxOscRc400M.

enumerator kCLOCK_GPT5_ClockRoot_MuxOscRc16M
GPT5 mux from MuxOscRc16M.

enumerator kCLOCK_GPT5_ClockRoot_MuxSysPll3Div2
GPT5 mux from MuxSysPll3Div2.

enumerator kCLOCK_GPT5_ClockRoot_MuxSysPll1Div5
GPT5 mux from MuxSysPll1Div5.

enumerator kCLOCK_GPT5_ClockRoot_MuxSysPll3Pfd2
GPT5 mux from MuxSysPll3Pfd2.

enumerator kCLOCK_GPT5_ClockRoot_MuxSysPll3Pfd3
GPT5 mux from MuxSysPll3Pfd3.

enumerator kCLOCK_GPT6_ClockRoot_MuxOscRc48MDiv2
GPT6 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_GPT6_ClockRoot_MuxOsc24MOut
GPT6 mux from MuxOsc24MOut.

enumerator kCLOCK_GPT6_ClockRoot_MuxOscRc400M
GPT6 mux from MuxOscRc400M.

enumerator kCLOCK_GPT6_ClockRoot_MuxOscRc16M
GPT6 mux from MuxOscRc16M.

enumerator kCLOCK_GPT6_ClockRoot_MuxSysPll3Div2
GPT6 mux from MuxSysPll3Div2.

enumerator kCLOCK_GPT6_ClockRoot_MuxSysPll1Div5
GPT6 mux from MuxSysPll1Div5.

enumerator kCLOCK_GPT6_ClockRoot_MuxSysPll3Pfd2
GPT6 mux from MuxSysPll3Pfd2.

enumerator kCLOCK_GPT6_ClockRoot_MuxSysPll3Pfd3
GPT6 mux from MuxSysPll3Pfd3.

enumerator kCLOCK_FLEXSPI1_ClockRoot_MuxOscRc48MDiv2
FLEXSPI1 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_FLEXSPI1_ClockRoot_MuxOsc24MOut
FLEXSPI1 mux from MuxOsc24MOut.

enumerator kCLOCK_FLEXSPI1_ClockRoot_MuxOscRc400M
FLEXSPI1 mux from MuxOscRc400M.

enumerator kCLOCK_FLEXSPI1_ClockRoot_MuxOscRc16M
FLEXSPI1 mux from MuxOscRc16M.

enumerator kCLOCK_FLEXSPI1_ClockRoot_MuxSysPll3Pfd0
FLEXSPI1 mux from MuxSysPll3Pfd0.

enumerator kCLOCK_FLEXSPI1_ClockRoot_MuxSysPll2Out
FLEXSPI1 mux from MuxSysPll2Out.

enumerator kCLOCK_FLEXSPI1_ClockRoot_MuxSysPll2Pfd2
FLEXSPI1 mux from MuxSysPll2Pfd2.

enumerator kCLOCK_FLEXSPI1_ClockRoot_MuxSysPll3Out
FLEXSPI1 mux from MuxSysPll3Out.

enumerator kCLOCK_FLEXSPI2_ClockRoot_MuxOscRc48MDiv2
FLEXSPI2 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_FLEXSPI2_ClockRoot_MuxOsc24MOut
FLEXSPI2 mux from MuxOsc24MOut.

enumerator kCLOCK_FLEXSPI2_ClockRoot_MuxOscRc400M
FLEXSPI2 mux from MuxOscRc400M.

enumerator kCLOCK_FLEXSPI2_ClockRoot_MuxOscRc16M
FLEXSPI2 mux from MuxOscRc16M.

enumerator kCLOCK_FLEXSPI2_ClockRoot_MuxSysPll3Pfd0
FLEXSPI2 mux from MuxSysPll3Pfd0.

enumerator kCLOCK_FLEXSPI2_ClockRoot_MuxSysPll2Out
FLEXSPI2 mux from MuxSysPll2Out.

enumerator kCLOCK_FLEXSPI2_ClockRoot_MuxSysPll2Pfd2
FLEXSPI2 mux from MuxSysPll2Pfd2.

enumerator kCLOCK_FLEXSPI2_ClockRoot_MuxSysPll3Out
FLEXSPI2 mux from MuxSysPll3Out.

enumerator kCLOCK_CAN1_ClockRoot_MuxOscRc48MDiv2
CAN1 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_CAN1_ClockRoot_MuxOsc24MOut
CAN1 mux from MuxOsc24MOut.

enumerator kCLOCK_CAN1_ClockRoot_MuxOscRc400M
CAN1 mux from MuxOscRc400M.

enumerator kCLOCK_CAN1_ClockRoot_MuxOscRc16M
CAN1 mux from MuxOscRc16M.

enumerator kCLOCK_CAN1_ClockRoot_MuxSysPll3Div2
CAN1 mux from MuxSysPll3Div2.

enumerator kCLOCK_CAN1_ClockRoot_MuxSysPll1Div5
CAN1 mux from MuxSysPll1Div5.

enumerator kCLOCK_CAN1_ClockRoot_MuxSysPll2Out
CAN1 mux from MuxSysPll2Out.

enumerator kCLOCK_CAN1_ClockRoot_MuxSysPll2Pfd3
CAN1 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_CAN2_ClockRoot_MuxOscRc48MDiv2
CAN2 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_CAN2_ClockRoot_MuxOsc24MOut
CAN2 mux from MuxOsc24MOut.

enumerator kCLOCK_CAN2_ClockRoot_MuxOscRc400M
CAN2 mux from MuxOscRc400M.

enumerator kCLOCK_CAN2_ClockRoot_MuxOscRc16M
CAN2 mux from MuxOscRc16M.

enumerator kCLOCK_CAN2_ClockRoot_MuxSysPll3Div2
CAN2 mux from MuxSysPll3Div2.

enumerator kCLOCK_CAN2_ClockRoot_MuxSysPll1Div5
CAN2 mux from MuxSysPll1Div5.

enumerator kCLOCK_CAN2_ClockRoot_MuxSysPll2Out
CAN2 mux from MuxSysPll2Out.

enumerator kCLOCK_CAN2_ClockRoot_MuxSysPll2Pfd3
CAN2 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_CAN3_ClockRoot_MuxOscRc48MDiv2
CAN3 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_CAN3_ClockRoot_MuxOsc24MOut
CAN3 mux from MuxOsc24MOut.

enumerator kCLOCK_CAN3_ClockRoot_MuxOscRc400M
CAN3 mux from MuxOscRc400M.

enumerator kCLOCK_CAN3_ClockRoot_MuxOscRc16M
CAN3 mux from MuxOscRc16M.

enumerator kCLOCK_CAN3_ClockRoot_MuxSysPll3Pfd3
CAN3 mux from MuxSysPll3Pfd3.

enumerator kCLOCK_CAN3_ClockRoot_MuxSysPll3Out
CAN3 mux from MuxSysPll3Out.

enumerator kCLOCK_CAN3_ClockRoot_MuxSysPll2Pfd3
CAN3 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_CAN3_ClockRoot_MuxSysPll1Div5
CAN3 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPUART1_ClockRoot_MuxOscRc48MDiv2
LPUART1 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPUART1_ClockRoot_MuxOsc24MOut
LPUART1 mux from MuxOsc24MOut.

enumerator kCLOCK_LPUART1_ClockRoot_MuxOscRc400M
LPUART1 mux from MuxOscRc400M.

enumerator kCLOCK_LPUART1_ClockRoot_MuxOscRc16M
LPUART1 mux from MuxOscRc16M.

enumerator kCLOCK_LPUART1_ClockRoot_MuxSysPll3Div2
LPUART1 mux from MuxSysPll3Div2.

enumerator kCLOCK_LPUART1_ClockRoot_MuxSysPll1Div5
LPUART1 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPUART1_ClockRoot_MuxSysPll2Out
LPUART1 mux from MuxSysPll2Out.

enumerator kCLOCK_LPUART1_ClockRoot_MuxSysPll2Pfd3
LPUART1 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPUART2_ClockRoot_MuxOscRc48MDiv2
LPUART2 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPUART2_ClockRoot_MuxOsc24MOut
LPUART2 mux from MuxOsc24MOut.

enumerator kCLOCK_LPUART2_ClockRoot_MuxOscRc400M
LPUART2 mux from MuxOscRc400M.

enumerator kCLOCK_LPUART2_ClockRoot_MuxOscRc16M
LPUART2 mux from MuxOscRc16M.

enumerator kCLOCK_LPUART2_ClockRoot_MuxSysPll3Div2
LPUART2 mux from MuxSysPll3Div2.

enumerator kCLOCK_LPUART2_ClockRoot_MuxSysPll1Div5
LPUART2 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPUART2_ClockRoot_MuxSysPll2Out
LPUART2 mux from MuxSysPll2Out.

enumerator kCLOCK_LPUART2_ClockRoot_MuxSysPll2Pfd3
LPUART2 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPUART3_ClockRoot_MuxOscRc48MDiv2
LPUART3 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPUART3_ClockRoot_MuxOsc24MOut
LPUART3 mux from MuxOsc24MOut.

enumerator kCLOCK_LPUART3_ClockRoot_MuxOscRc400M
LPUART3 mux from MuxOscRc400M.

enumerator kCLOCK_LPUART3_ClockRoot_MuxOscRc16M
LPUART3 mux from MuxOscRc16M.

enumerator kCLOCK_LPUART3_ClockRoot_MuxSysPll3Div2
LPUART3 mux from MuxSysPll3Div2.

enumerator kCLOCK_LPUART3_ClockRoot_MuxSysPll1Div5
LPUART3 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPUART3_ClockRoot_MuxSysPll2Out
LPUART3 mux from MuxSysPll2Out.

enumerator kCLOCK_LPUART3_ClockRoot_MuxSysPll2Pfd3
LPUART3 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPUART4_ClockRoot_MuxOscRc48MDiv2
LPUART4 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPUART4_ClockRoot_MuxOsc24MOut
LPUART4 mux from MuxOsc24MOut.

enumerator kCLOCK_LPUART4_ClockRoot_MuxOscRc400M
LPUART4 mux from MuxOscRc400M.

enumerator kCLOCK_LPUART4_ClockRoot_MuxOscRc16M
LPUART4 mux from MuxOscRc16M.

enumerator kCLOCK_LPUART4_ClockRoot_MuxSysPll3Div2
LPUART4 mux from MuxSysPll3Div2.

enumerator kCLOCK_LPUART4_ClockRoot_MuxSysPll1Div5
LPUART4 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPUART4_ClockRoot_MuxSysPll2Out
LPUART4 mux from MuxSysPll2Out.

enumerator kCLOCK_LPUART4_ClockRoot_MuxSysPll2Pfd3
LPUART4 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPUART5_ClockRoot_MuxOscRc48MDiv2
LPUART5 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPUART5_ClockRoot_MuxOsc24MOut
LPUART5 mux from MuxOsc24MOut.

enumerator kCLOCK_LPUART5_ClockRoot_MuxOscRc400M
LPUART5 mux from MuxOscRc400M.

enumerator kCLOCK_LPUART5_ClockRoot_MuxOscRc16M
LPUART5 mux from MuxOscRc16M.

enumerator kCLOCK_LPUART5_ClockRoot_MuxSysPll3Div2
LPUART5 mux from MuxSysPll3Div2.

enumerator kCLOCK_LPUART5_ClockRoot_MuxSysPll1Div5
LPUART5 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPUART5_ClockRoot_MuxSysPll2Out
LPUART5 mux from MuxSysPll2Out.

enumerator kCLOCK_LPUART5_ClockRoot_MuxSysPll2Pfd3
LPUART5 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPUART6_ClockRoot_MuxOscRc48MDiv2
LPUART6 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPUART6_ClockRoot_MuxOsc24MOut
LPUART6 mux from MuxOsc24MOut.

enumerator kCLOCK_LPUART6_ClockRoot_MuxOscRc400M
LPUART6 mux from MuxOscRc400M.

enumerator kCLOCK_LPUART6_ClockRoot_MuxOscRc16M
LPUART6 mux from MuxOscRc16M.

enumerator kCLOCK_LPUART6_ClockRoot_MuxSysPll3Div2
LPUART6 mux from MuxSysPll3Div2.

enumerator kCLOCK_LPUART6_ClockRoot_MuxSysPll1Div5
LPUART6 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPUART6_ClockRoot_MuxSysPll2Out
LPUART6 mux from MuxSysPll2Out.

enumerator kCLOCK_LPUART6_ClockRoot_MuxSysPll2Pfd3
LPUART6 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPUART7_ClockRoot_MuxOscRc48MDiv2
LPUART7 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPUART7_ClockRoot_MuxOsc24MOut
LPUART7 mux from MuxOsc24MOut.

enumerator kCLOCK_LPUART7_ClockRoot_MuxOscRc400M
LPUART7 mux from MuxOscRc400M.

enumerator kCLOCK_LPUART7_ClockRoot_MuxOscRc16M
LPUART7 mux from MuxOscRc16M.

enumerator kCLOCK_LPUART7_ClockRoot_MuxSysPll3Div2
LPUART7 mux from MuxSysPll3Div2.

enumerator kCLOCK_LPUART7_ClockRoot_MuxSysPll1Div5
LPUART7 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPUART7_ClockRoot_MuxSysPll2Out
LPUART7 mux from MuxSysPll2Out.

enumerator kCLOCK_LPUART7_ClockRoot_MuxSysPll2Pfd3
LPUART7 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPUART8_ClockRoot_MuxOscRc48MDiv2
LPUART8 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPUART8_ClockRoot_MuxOsc24MOut
LPUART8 mux from MuxOsc24MOut.

enumerator kCLOCK_LPUART8_ClockRoot_MuxOscRc400M
LPUART8 mux from MuxOscRc400M.

enumerator kCLOCK_LPUART8_ClockRoot_MuxOscRc16M
LPUART8 mux from MuxOscRc16M.

enumerator kCLOCK_LPUART8_ClockRoot_MuxSysPll3Div2
LPUART8 mux from MuxSysPll3Div2.

enumerator kCLOCK_LPUART8_ClockRoot_MuxSysPll1Div5
LPUART8 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPUART8_ClockRoot_MuxSysPll2Out
LPUART8 mux from MuxSysPll2Out.

enumerator kCLOCK_LPUART8_ClockRoot_MuxSysPll2Pfd3
LPUART8 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPUART9_ClockRoot_MuxOscRc48MDiv2
LPUART9 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPUART9_ClockRoot_MuxOsc24MOut
LPUART9 mux from MuxOsc24MOut.

enumerator kCLOCK_LPUART9_ClockRoot_MuxOscRc400M
LPUART9 mux from MuxOscRc400M.

enumerator kCLOCK_LPUART9_ClockRoot_MuxOscRc16M
LPUART9 mux from MuxOscRc16M.

enumerator kCLOCK_LPUART9_ClockRoot_MuxSysPll3Div2
LPUART9 mux from MuxSysPll3Div2.

enumerator kCLOCK_LPUART9_ClockRoot_MuxSysPll1Div5
LPUART9 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPUART9_ClockRoot_MuxSysPll2Out
LPUART9 mux from MuxSysPll2Out.

enumerator kCLOCK_LPUART9_ClockRoot_MuxSysPll2Pfd3
LPUART9 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPUART10_ClockRoot_MuxOscRc48MDiv2
LPUART10 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPUART10_ClockRoot_MuxOsc24MOut
LPUART10 mux from MuxOsc24MOut.

enumerator kCLOCK_LPUART10_ClockRoot_MuxOscRc400M
LPUART10 mux from MuxOscRc400M.

enumerator kCLOCK_LPUART10_ClockRoot_MuxOscRc16M
LPUART10 mux from MuxOscRc16M.

enumerator kCLOCK_LPUART10_ClockRoot_MuxSysPll3Div2
LPUART10 mux from MuxSysPll3Div2.

enumerator kCLOCK_LPUART10_ClockRoot_MuxSysPll1Div5
LPUART10 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPUART10_ClockRoot_MuxSysPll2Out
LPUART10 mux from MuxSysPll2Out.

enumerator kCLOCK_LPUART10_ClockRoot_MuxSysPll2Pfd3
LPUART10 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPUART11_ClockRoot_MuxOscRc48MDiv2
LPUART11 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPUART11_ClockRoot_MuxOsc24MOut
LPUART11 mux from MuxOsc24MOut.

enumerator kCLOCK_LPUART11_ClockRoot_MuxOscRc400M
LPUART11 mux from MuxOscRc400M.

enumerator kCLOCK_LPUART11_ClockRoot_MuxOscRc16M
LPUART11 mux from MuxOscRc16M.

enumerator kCLOCK_LPUART11_ClockRoot_MuxSysPll3Pfd3
LPUART11 mux from MuxSysPll3Pfd3.

enumerator kCLOCK_LPUART11_ClockRoot_MuxSysPll3Out
LPUART11 mux from MuxSysPll3Out.

enumerator kCLOCK_LPUART11_ClockRoot_MuxSysPll2Pfd3
LPUART11 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPUART11_ClockRoot_MuxSysPll1Div5
LPUART11 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPUART12_ClockRoot_MuxOscRc48MDiv2
LPUART12 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPUART12_ClockRoot_MuxOsc24MOut
LPUART12 mux from MuxOsc24MOut.

enumerator kCLOCK_LPUART12_ClockRoot_MuxOscRc400M
LPUART12 mux from MuxOscRc400M.

enumerator kCLOCK_LPUART12_ClockRoot_MuxOscRc16M
LPUART12 mux from MuxOscRc16M.

enumerator kCLOCK_LPUART12_ClockRoot_MuxSysPll3Pfd3
LPUART12 mux from MuxSysPll3Pfd3.

enumerator kCLOCK_LPUART12_ClockRoot_MuxSysPll3Out
LPUART12 mux from MuxSysPll3Out.

enumerator kCLOCK_LPUART12_ClockRoot_MuxSysPll2Pfd3
LPUART12 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPUART12_ClockRoot_MuxSysPll1Div5
LPUART12 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPI2C1_ClockRoot_MuxOscRc48MDiv2
LPI2C1 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPI2C1_ClockRoot_MuxOsc24MOut
LPI2C1 mux from MuxOsc24MOut.

enumerator kCLOCK_LPI2C1_ClockRoot_MuxOscRc400M
LPI2C1 mux from MuxOscRc400M.

enumerator kCLOCK_LPI2C1_ClockRoot_MuxOscRc16M
LPI2C1 mux from MuxOscRc16M.

enumerator kCLOCK_LPI2C1_ClockRoot_MuxSysPll3Div2
LPI2C1 mux from MuxSysPll3Div2.

enumerator kCLOCK_LPI2C1_ClockRoot_MuxSysPll1Div5
LPI2C1 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPI2C1_ClockRoot_MuxSysPll2Out
LPI2C1 mux from MuxSysPll2Out.

enumerator kCLOCK_LPI2C1_ClockRoot_MuxSysPll2Pfd3
LPI2C1 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPI2C2_ClockRoot_MuxOscRc48MDiv2
LPI2C2 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPI2C2_ClockRoot_MuxOsc24MOut
LPI2C2 mux from MuxOsc24MOut.

enumerator kCLOCK_LPI2C2_ClockRoot_MuxOscRc400M
LPI2C2 mux from MuxOscRc400M.

enumerator kCLOCK_LPI2C2_ClockRoot_MuxOscRc16M
LPI2C2 mux from MuxOscRc16M.

enumerator kCLOCK_LPI2C2_ClockRoot_MuxSysPll3Div2
LPI2C2 mux from MuxSysPll3Div2.

enumerator kCLOCK_LPI2C2_ClockRoot_MuxSysPll1Div5
LPI2C2 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPI2C2_ClockRoot_MuxSysPll2Out
LPI2C2 mux from MuxSysPll2Out.

enumerator kCLOCK_LPI2C2_ClockRoot_MuxSysPll2Pfd3
LPI2C2 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPI2C3_ClockRoot_MuxOscRc48MDiv2
LPI2C3 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPI2C3_ClockRoot_MuxOsc24MOut
LPI2C3 mux from MuxOsc24MOut.

enumerator kCLOCK_LPI2C3_ClockRoot_MuxOscRc400M
LPI2C3 mux from MuxOscRc400M.

enumerator kCLOCK_LPI2C3_ClockRoot_MuxOscRc16M
LPI2C3 mux from MuxOscRc16M.

enumerator kCLOCK_LPI2C3_ClockRoot_MuxSysPll3Div2
LPI2C3 mux from MuxSysPll3Div2.

enumerator kCLOCK_LPI2C3_ClockRoot_MuxSysPll1Div5
LPI2C3 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPI2C3_ClockRoot_MuxSysPll2Out
LPI2C3 mux from MuxSysPll2Out.

enumerator kCLOCK_LPI2C3_ClockRoot_MuxSysPll2Pfd3
LPI2C3 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPI2C4_ClockRoot_MuxOscRc48MDiv2
LPI2C4 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPI2C4_ClockRoot_MuxOsc24MOut
LPI2C4 mux from MuxOsc24MOut.

enumerator kCLOCK_LPI2C4_ClockRoot_MuxOscRc400M
LPI2C4 mux from MuxOscRc400M.

enumerator kCLOCK_LPI2C4_ClockRoot_MuxOscRc16M
LPI2C4 mux from MuxOscRc16M.

enumerator kCLOCK_LPI2C4_ClockRoot_MuxSysPll3Div2
LPI2C4 mux from MuxSysPll3Div2.

enumerator kCLOCK_LPI2C4_ClockRoot_MuxSysPll1Div5
LPI2C4 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPI2C4_ClockRoot_MuxSysPll2Out
LPI2C4 mux from MuxSysPll2Out.

enumerator kCLOCK_LPI2C4_ClockRoot_MuxSysPll2Pfd3
LPI2C4 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPI2C5_ClockRoot_MuxOscRc48MDiv2
LPI2C5 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPI2C5_ClockRoot_MuxOsc24MOut
LPI2C5 mux from MuxOsc24MOut.

enumerator kCLOCK_LPI2C5_ClockRoot_MuxOscRc400M
LPI2C5 mux from MuxOscRc400M.

enumerator kCLOCK_LPI2C5_ClockRoot_MuxOscRc16M
LPI2C5 mux from MuxOscRc16M.

enumerator kCLOCK_LPI2C5_ClockRoot_MuxSysPll3Pfd3
LPI2C5 mux from MuxSysPll3Pfd3.

enumerator kCLOCK_LPI2C5_ClockRoot_MuxSysPll3Out
LPI2C5 mux from MuxSysPll3Out.

enumerator kCLOCK_LPI2C5_ClockRoot_MuxSysPll2Pfd3
LPI2C5 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPI2C5_ClockRoot_MuxSysPll1Div5
LPI2C5 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPI2C6_ClockRoot_MuxOscRc48MDiv2
LPI2C6 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPI2C6_ClockRoot_MuxOsc24MOut
LPI2C6 mux from MuxOsc24MOut.

enumerator kCLOCK_LPI2C6_ClockRoot_MuxOscRc400M
LPI2C6 mux from MuxOscRc400M.

enumerator kCLOCK_LPI2C6_ClockRoot_MuxOscRc16M
LPI2C6 mux from MuxOscRc16M.

enumerator kCLOCK_LPI2C6_ClockRoot_MuxSysPll3Pfd3
LPI2C6 mux from MuxSysPll3Pfd3.

enumerator kCLOCK_LPI2C6_ClockRoot_MuxSysPll3Out
LPI2C6 mux from MuxSysPll3Out.

enumerator kCLOCK_LPI2C6_ClockRoot_MuxSysPll2Pfd3
LPI2C6 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPI2C6_ClockRoot_MuxSysPll1Div5
LPI2C6 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPSPi1_ClockRoot_MuxOscRc48MDiv2
LPSPi1 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPSPi1_ClockRoot_MuxOsc24MOut
LPSPi1 mux from MuxOsc24MOut.

enumerator kCLOCK_LPSPi1_ClockRoot_MuxOscRc400M
LPSPi1 mux from MuxOscRc400M.

enumerator kCLOCK_LPSPi1_ClockRoot_MuxOscRc16M
LPSPi1 mux from MuxOscRc16M.

enumerator kCLOCK_LPSPi1_ClockRoot_MuxSysPll3Pfd2
LPSPi1 mux from MuxSysPll3Pfd2.

enumerator kCLOCK_LPSPi1_ClockRoot_MuxSysPll1Div5
LPSPi1 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPSPi1_ClockRoot_MuxSysPll2Out
LPSPi1 mux from MuxSysPll2Out.

enumerator kCLOCK_LPSPi1_ClockRoot_MuxSysPll2Pfd3
LPSPi1 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPSPi2_ClockRoot_MuxOscRc48MDiv2
LPSPi2 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPSPi2_ClockRoot_MuxOsc24MOut
LPSPi2 mux from MuxOsc24MOut.

enumerator kCLOCK_LPSPi2_ClockRoot_MuxOscRc400M
LPSPi2 mux from MuxOscRc400M.

enumerator kCLOCK_LPSPi2_ClockRoot_MuxOscRc16M
LPSPi2 mux from MuxOscRc16M.

enumerator kCLOCK_LPSPi2_ClockRoot_MuxSysPll3Pfd2
LPSPi2 mux from MuxSysPll3Pfd2.

enumerator kCLOCK_LPSPi2_ClockRoot_MuxSysPll1Div5
LPSPi2 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPSPi2_ClockRoot_MuxSysPll2Out
LPSPi2 mux from MuxSysPll2Out.

enumerator kCLOCK_LPSPi2_ClockRoot_MuxSysPll2Pfd3
LPSPi2 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPSPi3_ClockRoot_MuxOscRc48MDiv2
LPSPi3 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPSPi3_ClockRoot_MuxOsc24MOut
LPSPi3 mux from MuxOsc24MOut.

enumerator kCLOCK_LPSPi3_ClockRoot_MuxOscRc400M
LPSPi3 mux from MuxOscRc400M.

enumerator kCLOCK_LPSPi3_ClockRoot_MuxOscRc16M
LPSPi3 mux from MuxOscRc16M.

enumerator kCLOCK_LPSPi3_ClockRoot_MuxSysPll3Pfd2
LPSPi3 mux from MuxSysPll3Pfd2.

enumerator kCLOCK_LPSPi3_ClockRoot_MuxSysPll1Div5
LPSPi3 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPSPi3_ClockRoot_MuxSysPll2Out
LPSPi3 mux from MuxSysPll2Out.

enumerator kCLOCK_LPSPi3_ClockRoot_MuxSysPll2Pfd3
LPSPi3 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPSPi4_ClockRoot_MuxOscRc48MDiv2
LPSPi4 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPSPi4_ClockRoot_MuxOsc24MOut
LPSPi4 mux from MuxOsc24MOut.

enumerator kCLOCK_LPSPi4_ClockRoot_MuxOscRc400M
LPSPi4 mux from MuxOscRc400M.

enumerator kCLOCK_LPSPi4_ClockRoot_MuxOscRc16M
LPSPi4 mux from MuxOscRc16M.

enumerator kCLOCK_LPSPi4_ClockRoot_MuxSysPll3Pfd2
LPSPi4 mux from MuxSysPll3Pfd2.

enumerator kCLOCK_LPSPi4_ClockRoot_MuxSysPll1Div5
LPSPi4 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPSPi4_ClockRoot_MuxSysPll2Out
LPSPi4 mux from MuxSysPll2Out.

enumerator kCLOCK_LPSPi4_ClockRoot_MuxSysPll2Pfd3
LPSPi4 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_LPSPi5_ClockRoot_MuxOscRc48MDiv2
LPSPi5 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPSPi5_ClockRoot_MuxOsc24MOut
LPSPi5 mux from MuxOsc24MOut.

enumerator kCLOCK_LPSPi5_ClockRoot_MuxOscRc400M
LPSPi5 mux from MuxOscRc400M.

enumerator kCLOCK_LPSPi5_ClockRoot_MuxOscRc16M
LPSPi5 mux from MuxOscRc16M.

enumerator kCLOCK_LPSPi5_ClockRoot_MuxSysPll3Pfd3
LPSPi5 mux from MuxSysPll3Pfd3.

enumerator kCLOCK_LPSPi5_ClockRoot_MuxSysPll3Out
LPSPi5 mux from MuxSysPll3Out.

enumerator kCLOCK_LPSPi5_ClockRoot_MuxSysPll3Pfd2
LPSPi5 mux from MuxSysPll3Pfd2.

enumerator kCLOCK_LPSPi5_ClockRoot_MuxSysPll1Div5
LPSPi5 mux from MuxSysPll1Div5.

enumerator kCLOCK_LPSPi6_ClockRoot_MuxOscRc48MDiv2
LPSPi6 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LPSPi6_ClockRoot_MuxOsc24MOut
LPSPi6 mux from MuxOsc24MOut.

enumerator kCLOCK_LPSPi6_ClockRoot_MuxOscRc400M
LPSPi6 mux from MuxOscRc400M.

enumerator kCLOCK_LPSPi6_ClockRoot_MuxOscRc16M
LPSPi6 mux from MuxOscRc16M.

enumerator kCLOCK_LPSPi6_ClockRoot_MuxSysPll3Pfd3
LPSPi6 mux from MuxSysPll3Pfd3.

enumerator kCLOCK_LPSPi6_ClockRoot_MuxSysPll3Out
LPSPi6 mux from MuxSysPll3Out.

enumerator kCLOCK_LPSPi6_ClockRoot_MuxSysPll3Pfd2
LPSPi6 mux from MuxSysPll3Pfd2.

enumerator kCLOCK_LPSPi6_ClockRoot_MuxSysPll1Div5
LPSPi6 mux from MuxSysPll1Div5.

enumerator kCLOCK_EMV1_ClockRoot_MuxOscRc48MDiv2
EMV1 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_EMV1_ClockRoot_MuxOsc24MOut
EMV1 mux from MuxOsc24MOut.

enumerator kCLOCK_EMV1_ClockRoot_MuxOscRc400M
EMV1 mux from MuxOscRc400M.

enumerator kCLOCK_EMV1_ClockRoot_MuxOscRc16M
EMV1 mux from MuxOscRc16M.

enumerator kCLOCK_EMV1_ClockRoot_MuxSysPll3Div2
EMV1 mux from MuxSysPll3Div2.

enumerator kCLOCK_EMV1_ClockRoot_MuxSysPll1Div5
EMV1 mux from MuxSysPll1Div5.

enumerator kCLOCK_EMV1_ClockRoot_MuxSysPll2Out
EMV1 mux from MuxSysPll2Out.

enumerator kCLOCK_EMV1_ClockRoot_MuxSysPll2Pfd3
EMV1 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_EMV2_ClockRoot_MuxOscRc48MDiv2
EMV2 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_EMV2_ClockRoot_MuxOsc24MOut
EMV2 mux from MuxOsc24MOut.

enumerator kCLOCK_EMV2_ClockRoot_MuxOscRc400M
EMV2 mux from MuxOscRc400M.

enumerator kCLOCK_EMV2_ClockRoot_MuxOscRc16M
EMV2 mux from MuxOscRc16M.

enumerator kCLOCK_EMV2_ClockRoot_MuxSysPll3Div2
EMV2 mux from MuxSysPll3Div2.

enumerator kCLOCK_EMV2_ClockRoot_MuxSysPll1Div5
EMV2 mux from MuxSysPll1Div5.

enumerator kCLOCK_EMV2_ClockRoot_MuxSysPll2Out
EMV2 mux from MuxSysPll2Out.

enumerator kCLOCK_EMV2_ClockRoot_MuxSysPll2Pfd3
EMV2 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_ENET1_ClockRoot_MuxOscRc48MDiv2
ENET1 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_ENET1_ClockRoot_MuxOsc24MOut
ENET1 mux from MuxOsc24MOut.

enumerator kCLOCK_ENET1_ClockRoot_MuxOscRc400M
ENET1 mux from MuxOscRc400M.

enumerator kCLOCK_ENET1_ClockRoot_MuxOscRc16M
ENET1 mux from MuxOscRc16M.

enumerator kCLOCK_ENET1_ClockRoot_MuxSysPll1Div2
ENET1 mux from MuxSysPll1Div2.

enumerator kCLOCK_ENET1_ClockRoot_MuxAudioPllOut
ENET1 mux from MuxAudioPllOut.

enumerator kCLOCK_ENET1_ClockRoot_MuxSysPll1Div5
ENET1 mux from MuxSysPll1Div5.

enumerator kCLOCK_ENET1_ClockRoot_MuxSysPll2Pfd1
ENET1 mux from MuxSysPll2Pfd1.

enumerator kCLOCK_ENET2_ClockRoot_MuxOscRc48MDiv2
ENET2 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_ENET2_ClockRoot_MuxOsc24MOut
ENET2 mux from MuxOsc24MOut.

enumerator kCLOCK_ENET2_ClockRoot_MuxOscRc400M
ENET2 mux from MuxOscRc400M.

enumerator kCLOCK_ENET2_ClockRoot_MuxOscRc16M
ENET2 mux from MuxOscRc16M.

enumerator kCLOCK_ENET2_ClockRoot_MuxSysPll1Div2
ENET2 mux from MuxSysPll1Div2.

enumerator kCLOCK_ENET2_ClockRoot_MuxAudioPllOut
ENET2 mux from MuxAudioPllOut.

enumerator kCLOCK_ENET2_ClockRoot_MuxSysPll1Div5
ENET2 mux from MuxSysPll1Div5.

enumerator kCLOCK_ENET2_ClockRoot_MuxSysPll2Pfd1
ENET2 mux from MuxSysPll2Pfd1.

enumerator kCLOCK_ENET_25M_ClockRoot_MuxOscRc48MDiv2
ENET_25M mux from MuxOscRc48MDiv2.

enumerator kCLOCK_ENET_25M_ClockRoot_MuxOsc24MOut
ENET_25M mux from MuxOsc24MOut.

enumerator kCLOCK_ENET_25M_ClockRoot_MuxOscRc400M
ENET_25M mux from MuxOscRc400M.

enumerator kCLOCK_ENET_25M_ClockRoot_MuxOscRc16M
ENET_25M mux from MuxOscRc16M.

enumerator kCLOCK_ENET_25M_ClockRoot_MuxSysPll1Div2
ENET_25M mux from MuxSysPll1Div2.

enumerator kCLOCK_ENET_25M_ClockRoot_MuxAudioPllOut
ENET_25M mux from MuxAudioPllOut.

enumerator kCLOCK_ENET_25M_ClockRoot_MuxSysPll1Div5
ENET_25M mux from MuxSysPll1Div5.

enumerator kCLOCK_ENET_25M_ClockRoot_MuxSysPll2Pfd1
ENET_25M mux from MuxSysPll2Pfd1.

enumerator kCLOCK_ENET_TIMER1_ClockRoot_MuxOscRc48MDiv2
ENET_TIMER1 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_ENET_TIMER1_ClockRoot_MuxOsc24MOut
ENET_TIMER1 mux from MuxOsc24MOut.

enumerator kCLOCK_ENET_TIMER1_ClockRoot_MuxOscRc400M
ENET_TIMER1 mux from MuxOscRc400M.

enumerator kCLOCK_ENET_TIMER1_ClockRoot_MuxOscRc16M
ENET_TIMER1 mux from MuxOscRc16M.

enumerator kCLOCK_ENET_TIMER1_ClockRoot_MuxSysPll1Div2
ENET_TIMER1 mux from MuxSysPll1Div2.

enumerator kCLOCK_ENET_TIMER1_ClockRoot_MuxAudioPllOut
ENET_TIMER1 mux from MuxAudioPllOut.

enumerator kCLOCK_ENET_TIMER1_ClockRoot_MuxSysPll1Div5
ENET_TIMER1 mux from MuxSysPll1Div5.

enumerator kCLOCK_ENET_TIMER1_ClockRoot_MuxSysPll2Pfd1
ENET_TIMER1 mux from MuxSysPll2Pfd1.

enumerator kCLOCK_ENET_TIMER2_ClockRoot_MuxOscRc48MDiv2
ENET_TIMER2 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_ENET_TIMER2_ClockRoot_MuxOsc24MOut
ENET_TIMER2 mux from MuxOsc24MOut.

enumerator kCLOCK_ENET_TIMER2_ClockRoot_MuxOscRc400M
ENET_TIMER2 mux from MuxOscRc400M.

enumerator kCLOCK_ENET_TIMER2_ClockRoot_MuxOscRc16M
ENET_TIMER2 mux from MuxOscRc16M.

enumerator kCLOCK_ENET_TIMER2_ClockRoot_MuxSysPll1Div2
ENET_TIMER2 mux from MuxSysPll1Div2.

enumerator kCLOCK_ENET_TIMER2_ClockRoot_MuxAudioPllOut
ENET_TIMER2 mux from MuxAudioPllOut.

enumerator kCLOCK_ENET_TIMER2_ClockRoot_MuxSysPll1Div5
ENET_TIMER2 mux from MuxSysPll1Div5.

enumerator kCLOCK_ENET_TIMER2_ClockRoot_MuxSysPll2Pfd1
ENET_TIMER2 mux from MuxSysPll2Pfd1.

enumerator kCLOCK_USDHC1_ClockRoot_MuxOscRc48MDiv2
USDHC1 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_USDHC1_ClockRoot_MuxOsc24MOut
USDHC1 mux from MuxOsc24MOut.

enumerator kCLOCK_USDHC1_ClockRoot_MuxOscRc400M
USDHC1 mux from MuxOscRc400M.

enumerator kCLOCK_USDHC1_ClockRoot_MuxOscRc16M
USDHC1 mux from MuxOscRc16M.

enumerator kCLOCK_USDHC1_ClockRoot_MuxSysPll2Pfd2
USDHC1 mux from MuxSysPll2Pfd2.

enumerator kCLOCK_USDHC1_ClockRoot_MuxSysPll2Pfd0
USDHC1 mux from MuxSysPll2Pfd0.

enumerator kCLOCK_USDHC1_ClockRoot_MuxSysPll1Div5
USDHC1 mux from MuxSysPll1Div5.

enumerator kCLOCK_USDHC1_ClockRoot_MuxArmPllOut
USDHC1 mux from MuxArmPllOut.

enumerator kCLOCK_USDHC2_ClockRoot_MuxOscRc48MDiv2
USDHC2 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_USDHC2_ClockRoot_MuxOsc24MOut
USDHC2 mux from MuxOsc24MOut.

enumerator kCLOCK_USDHC2_ClockRoot_MuxOscRc400M
USDHC2 mux from MuxOscRc400M.

enumerator kCLOCK_USDHC2_ClockRoot_MuxOscRc16M
USDHC2 mux from MuxOscRc16M.

enumerator kCLOCK_USDHC2_ClockRoot_MuxSysPll2Pfd2
USDHC2 mux from MuxSysPll2Pfd2.

enumerator kCLOCK_USDHC2_ClockRoot_MuxSysPll2Pfd0
USDHC2 mux from MuxSysPll2Pfd0.

enumerator kCLOCK_USDHC2_ClockRoot_MuxSysPll1Div5
USDHC2 mux from MuxSysPll1Div5.

enumerator kCLOCK_USDHC2_ClockRoot_MuxArmPllOut
USDHC2 mux from MuxArmPllOut.

enumerator kCLOCK_ASRC_ClockRoot_MuxOscRc48MDiv2
ASRC mux from MuxOscRc48MDiv2.

enumerator kCLOCK_ASRC_ClockRoot_MuxOsc24MOut
ASRC mux from MuxOsc24MOut.

enumerator kCLOCK_ASRC_ClockRoot_MuxOscRc400M
ASRC mux from MuxOscRc400M.

enumerator kCLOCK_ASRC_ClockRoot_MuxOscRc16M
ASRC mux from MuxOscRc16M.

enumerator kCLOCK_ASRC_ClockRoot_MuxSysPll1Div5
ASRC mux from MuxSysPll1Div5.

enumerator kCLOCK_ASRC_ClockRoot_MuxSysPll3Div2
ASRC mux from MuxSysPll3Div2.

enumerator kCLOCK_ASRC_ClockRoot_MuxAudioPllOut
ASRC mux from MuxAudioPllOut.

enumerator kCLOCK_ASRC_ClockRoot_MuxSysPll2Pfd3
ASRC mux from MuxSysPll2Pfd3.

enumerator kCLOCK_MQS_ClockRoot_MuxOscRc48MDiv2
MQS mux from MuxOscRc48MDiv2.

enumerator kCLOCK_MQS_ClockRoot_MuxOsc24MOut
MQS mux from MuxOsc24MOut.

enumerator kCLOCK_MQS_ClockRoot_MuxOscRc400M
MQS mux from MuxOscRc400M.

enumerator kCLOCK_MQS_ClockRoot_MuxOscRc16M
MQS mux from MuxOscRc16M.

enumerator kCLOCK_MQS_ClockRoot_MuxSysPll1Div5
MQS mux from MuxSysPll1Div5.

enumerator kCLOCK_MQS_ClockRoot_MuxSysPll3Div2
MQS mux from MuxSysPll3Div2.

enumerator kCLOCK_MQS_ClockRoot_MuxAudioPllOut
MQS mux from MuxAudioPllOut.

enumerator kCLOCK_MQS_ClockRoot_MuxSysPll2Pfd3
MQS mux from MuxSysPll2Pfd3.

enumerator kCLOCK_MIC_ClockRoot_MuxOscRc48MDiv2
MIC mux from MuxOscRc48MDiv2.

enumerator kCLOCK_MIC_ClockRoot_MuxOsc24MOut
MIC mux from MuxOsc24MOut.

enumerator kCLOCK_MIC_ClockRoot_MuxOscRc400M
MIC mux from MuxOscRc400M.

enumerator kCLOCK_MIC_ClockRoot_MuxOscRc16M
MIC mux from MuxOscRc16M.

enumerator kCLOCK_MIC_ClockRoot_MuxSysPll3Pfd3
MIC mux from MuxSysPll3Pfd3.

enumerator kCLOCK_MIC_ClockRoot_MuxSysPll3Out
MIC mux from MuxSysPll3Out.

enumerator kCLOCK_MIC_ClockRoot_MuxAudioPllOut
MIC mux from MuxAudioPllOut.

enumerator kCLOCK_MIC_ClockRoot_MuxSysPll1Div5
MIC mux from MuxSysPll1Div5.

enumerator kCLOCK_SPDIF_ClockRoot_MuxOscRc48MDiv2
SPDIF mux from MuxOscRc48MDiv2.

enumerator kCLOCK_SPDIF_ClockRoot_MuxOsc24MOut
SPDIF mux from MuxOsc24MOut.

enumerator kCLOCK_SPDIF_ClockRoot_MuxOscRc400M
SPDIF mux from MuxOscRc400M.

enumerator kCLOCK_SPDIF_ClockRoot_MuxOscRc16M
SPDIF mux from MuxOscRc16M.

enumerator kCLOCK_SPDIF_ClockRoot_MuxAudioPllOut
SPDIF mux from MuxAudioPllOut.

enumerator kCLOCK_SPDIF_ClockRoot_MuxSysPll3Out
SPDIF mux from MuxSysPll3Out.

enumerator kCLOCK_SPDIF_ClockRoot_MuxSysPll3Pfd2
SPDIF mux from MuxSysPll3Pfd2.

enumerator kCLOCK_SPDIF_ClockRoot_MuxSysPll2Pfd3
SPDIF mux from MuxSysPll2Pfd3.

enumerator kCLOCK_SAI1_ClockRoot_MuxOscRc48MDiv2
SAI1 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_SAI1_ClockRoot_MuxOsc24MOut
SAI1 mux from MuxOsc24MOut.

enumerator kCLOCK_SAI1_ClockRoot_MuxOscRc400M
SAI1 mux from MuxOscRc400M.

enumerator kCLOCK_SAI1_ClockRoot_MuxOscRc16M
SAI1 mux from MuxOscRc16M.

enumerator kCLOCK_SAI1_ClockRoot_MuxAudioPllOut
SAI1 mux from MuxAudioPllOut.

enumerator kCLOCK_SAI1_ClockRoot_MuxSysPll3Pfd2
SAI1 mux from MuxSysPll3Pfd2.


```

enumerator kCLOCK_SAI1_ClockRoot_MuxSysPll1Div5
    SAI1 mux from MuxSysPll1Div5.
enumerator kCLOCK_SAI1_ClockRoot_MuxSysPll2Pfd3
    SAI1 mux from MuxSysPll2Pfd3.
enumerator kCLOCK_SAI2_ClockRoot_MuxOscRc48MDiv2
    SAI2 mux from MuxOscRc48MDiv2.
enumerator kCLOCK_SAI2_ClockRoot_MuxOsc24MOut
    SAI2 mux from MuxOsc24MOut.
enumerator kCLOCK_SAI2_ClockRoot_MuxOscRc400M
    SAI2 mux from MuxOscRc400M.
enumerator kCLOCK_SAI2_ClockRoot_MuxOscRc16M
    SAI2 mux from MuxOscRc16M.
enumerator kCLOCK_SAI2_ClockRoot_MuxAudioPllOut
    SAI2 mux from MuxAudioPllOut.
enumerator kCLOCK_SAI2_ClockRoot_MuxSysPll3Pfd2
    SAI2 mux from MuxSysPll3Pfd2.
enumerator kCLOCK_SAI2_ClockRoot_MuxSysPll1Div5
    SAI2 mux from MuxSysPll1Div5.
enumerator kCLOCK_SAI2_ClockRoot_MuxSysPll2Pfd3
    SAI2 mux from MuxSysPll2Pfd3.
enumerator kCLOCK_SAI3_ClockRoot_MuxOscRc48MDiv2
    SAI3 mux from MuxOscRc48MDiv2.
enumerator kCLOCK_SAI3_ClockRoot_MuxOsc24MOut
    SAI3 mux from MuxOsc24MOut.
enumerator kCLOCK_SAI3_ClockRoot_MuxOscRc400M
    SAI3 mux from MuxOscRc400M.
enumerator kCLOCK_SAI3_ClockRoot_MuxOscRc16M
    SAI3 mux from MuxOscRc16M.
enumerator kCLOCK_SAI3_ClockRoot_MuxAudioPllOut
    SAI3 mux from MuxAudioPllOut.
enumerator kCLOCK_SAI3_ClockRoot_MuxSysPll3Pfd2
    SAI3 mux from MuxSysPll3Pfd2.
enumerator kCLOCK_SAI3_ClockRoot_MuxSysPll1Div5
    SAI3 mux from MuxSysPll1Div5.
enumerator kCLOCK_SAI3_ClockRoot_MuxSysPll2Pfd3
    SAI3 mux from MuxSysPll2Pfd3.
enumerator kCLOCK_SAI4_ClockRoot_MuxOscRc48MDiv2
    SAI4 mux from MuxOscRc48MDiv2.
enumerator kCLOCK_SAI4_ClockRoot_MuxOsc24MOut
    SAI4 mux from MuxOsc24MOut.
enumerator kCLOCK_SAI4_ClockRoot_MuxOscRc400M
    SAI4 mux from MuxOscRc400M.

```

enumerator kCLOCK_SAI4_ClockRoot_MuxOscRc16M
SAI4 mux from MuxOscRc16M.

enumerator kCLOCK_SAI4_ClockRoot_MuxSysPll3Pfd3
SAI4 mux from MuxSysPll3Pfd3.

enumerator kCLOCK_SAI4_ClockRoot_MuxSysPll3Out
SAI4 mux from MuxSysPll3Out.

enumerator kCLOCK_SAI4_ClockRoot_MuxAudioPllOut
SAI4 mux from MuxAudioPllOut.

enumerator kCLOCK_SAI4_ClockRoot_MuxSysPll1Div5
SAI4 mux from MuxSysPll1Div5.

enumerator kCLOCK_GC355_ClockRoot_MuxOscRc48MDiv2
GC355 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_GC355_ClockRoot_MuxOsc24MOut
GC355 mux from MuxOsc24MOut.

enumerator kCLOCK_GC355_ClockRoot_MuxOscRc400M
GC355 mux from MuxOscRc400M.

enumerator kCLOCK_GC355_ClockRoot_MuxOscRc16M
GC355 mux from MuxOscRc16M.

enumerator kCLOCK_GC355_ClockRoot_MuxSysPll2Out
GC355 mux from MuxSysPll2Out.

enumerator kCLOCK_GC355_ClockRoot_MuxSysPll2Pfd1
GC355 mux from MuxSysPll2Pfd1.

enumerator kCLOCK_GC355_ClockRoot_MuxSysPll3Out
GC355 mux from MuxSysPll3Out.

enumerator kCLOCK_GC355_ClockRoot_MuxVideoPllOut
GC355 mux from MuxVideoPllOut.

enumerator kCLOCK_LCDIF_ClockRoot_MuxOscRc48MDiv2
LCDIF mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LCDIF_ClockRoot_MuxOsc24MOut
LCDIF mux from MuxOsc24MOut.

enumerator kCLOCK_LCDIF_ClockRoot_MuxOscRc400M
LCDIF mux from MuxOscRc400M.

enumerator kCLOCK_LCDIF_ClockRoot_MuxOscRc16M
LCDIF mux from MuxOscRc16M.

enumerator kCLOCK_LCDIF_ClockRoot_MuxSysPll2Out
LCDIF mux from MuxSysPll2Out.

enumerator kCLOCK_LCDIF_ClockRoot_MuxSysPll2Pfd2
LCDIF mux from MuxSysPll2Pfd2.

enumerator kCLOCK_LCDIF_ClockRoot_MuxSysPll3Pfd0
LCDIF mux from MuxSysPll3Pfd0.

enumerator kCLOCK_LCDIF_ClockRoot_MuxVideoPllOut
LCDIF mux from MuxVideoPllOut.

enumerator kCLOCK_LCDIFV2_ClockRoot_MuxOscRc48MDiv2
LCDIFV2 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_LCDIFV2_ClockRoot_MuxOsc24MOut
LCDIFV2 mux from MuxOsc24MOut.

enumerator kCLOCK_LCDIFV2_ClockRoot_MuxOscRc400M
LCDIFV2 mux from MuxOscRc400M.

enumerator kCLOCK_LCDIFV2_ClockRoot_MuxOscRc16M
LCDIFV2 mux from MuxOscRc16M.

enumerator kCLOCK_LCDIFV2_ClockRoot_MuxSysPll2Out
LCDIFV2 mux from MuxSysPll2Out.

enumerator kCLOCK_LCDIFV2_ClockRoot_MuxSysPll2Pfd2
LCDIFV2 mux from MuxSysPll2Pfd2.

enumerator kCLOCK_LCDIFV2_ClockRoot_MuxSysPll3Pfd0
LCDIFV2 mux from MuxSysPll3Pfd0.

enumerator kCLOCK_LCDIFV2_ClockRoot_MuxVideoPllOut
LCDIFV2 mux from MuxVideoPllOut.

enumerator kCLOCK_MIPI_REF_ClockRoot_MuxOscRc48MDiv2
MIPI_REF mux from MuxOscRc48MDiv2.

enumerator kCLOCK_MIPI_REF_ClockRoot_MuxOsc24MOut
MIPI_REF mux from MuxOsc24MOut.

enumerator kCLOCK_MIPI_REF_ClockRoot_MuxOscRc400M
MIPI_REF mux from MuxOscRc400M.

enumerator kCLOCK_MIPI_REF_ClockRoot_MuxOscRc16M
MIPI_REF mux from MuxOscRc16M.

enumerator kCLOCK_MIPI_REF_ClockRoot_MuxSysPll2Out
MIPI_REF mux from MuxSysPll2Out.

enumerator kCLOCK_MIPI_REF_ClockRoot_MuxSysPll2Pfd0
MIPI_REF mux from MuxSysPll2Pfd0.

enumerator kCLOCK_MIPI_REF_ClockRoot_MuxSysPll3Pfd0
MIPI_REF mux from MuxSysPll3Pfd0.

enumerator kCLOCK_MIPI_REF_ClockRoot_MuxVideoPllOut
MIPI_REF mux from MuxVideoPllOut.

enumerator kCLOCK_MIPI_ESC_ClockRoot_MuxOscRc48MDiv2
MIPI_ESC mux from MuxOscRc48MDiv2.

enumerator kCLOCK_MIPI_ESC_ClockRoot_MuxOsc24MOut
MIPI_ESC mux from MuxOsc24MOut.

enumerator kCLOCK_MIPI_ESC_ClockRoot_MuxOscRc400M
MIPI_ESC mux from MuxOscRc400M.

enumerator kCLOCK_MIPI_ESC_ClockRoot_MuxOscRc16M
MIPI_ESC mux from MuxOscRc16M.

enumerator kCLOCK_MIPI_ESC_ClockRoot_MuxSysPll2Out
MIPI_ESC mux from MuxSysPll2Out.

enumerator kCLOCK_MIPi_ESC_ClockRoot_MuxSysPll2Pfd0
MIPi_ESC mux from MuxSysPll2Pfd0.

enumerator kCLOCK_MIPi_ESC_ClockRoot_MuxSysPll3Pfd0
MIPi_ESC mux from MuxSysPll3Pfd0.

enumerator kCLOCK_MIPi_ESC_ClockRoot_MuxVideoPllOut
MIPi_ESC mux from MuxVideoPllOut.

enumerator kCLOCK_CSI2_ClockRoot_MuxOscRc48MDiv2
CSI2 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_CSI2_ClockRoot_MuxOsc24MOut
CSI2 mux from MuxOsc24MOut.

enumerator kCLOCK_CSI2_ClockRoot_MuxOscRc400M
CSI2 mux from MuxOscRc400M.

enumerator kCLOCK_CSI2_ClockRoot_MuxOscRc16M
CSI2 mux from MuxOscRc16M.

enumerator kCLOCK_CSI2_ClockRoot_MuxSysPll2Pfd2
CSI2 mux from MuxSysPll2Pfd2.

enumerator kCLOCK_CSI2_ClockRoot_MuxSysPll3Out
CSI2 mux from MuxSysPll3Out.

enumerator kCLOCK_CSI2_ClockRoot_MuxSysPll2Pfd0
CSI2 mux from MuxSysPll2Pfd0.

enumerator kCLOCK_CSI2_ClockRoot_MuxVideoPllOut
CSI2 mux from MuxVideoPllOut.

enumerator kCLOCK_CSI2_ESC_ClockRoot_MuxOscRc48MDiv2
CSI2_ESC mux from MuxOscRc48MDiv2.

enumerator kCLOCK_CSI2_ESC_ClockRoot_MuxOsc24MOut
CSI2_ESC mux from MuxOsc24MOut.

enumerator kCLOCK_CSI2_ESC_ClockRoot_MuxOscRc400M
CSI2_ESC mux from MuxOscRc400M.

enumerator kCLOCK_CSI2_ESC_ClockRoot_MuxOscRc16M
CSI2_ESC mux from MuxOscRc16M.

enumerator kCLOCK_CSI2_ESC_ClockRoot_MuxSysPll2Pfd2
CSI2_ESC mux from MuxSysPll2Pfd2.

enumerator kCLOCK_CSI2_ESC_ClockRoot_MuxSysPll3Out
CSI2_ESC mux from MuxSysPll3Out.

enumerator kCLOCK_CSI2_ESC_ClockRoot_MuxSysPll2Pfd0
CSI2_ESC mux from MuxSysPll2Pfd0.

enumerator kCLOCK_CSI2_ESC_ClockRoot_MuxVideoPllOut
CSI2_ESC mux from MuxVideoPllOut.

enumerator kCLOCK_CSI2_UI_ClockRoot_MuxOscRc48MDiv2
CSI2_UI mux from MuxOscRc48MDiv2.

enumerator kCLOCK_CSI2_UI_ClockRoot_MuxOsc24MOut
CSI2_UI mux from MuxOsc24MOut.

enumerator kCLOCK_CSI2_UI_ClockRoot_MuxOscRc400M
CSI2_UI mux from MuxOscRc400M.

enumerator kCLOCK_CSI2_UI_ClockRoot_MuxOscRc16M
CSI2_UI mux from MuxOscRc16M.

enumerator kCLOCK_CSI2_UI_ClockRoot_MuxSysPll2Pfd2
CSI2_UI mux from MuxSysPll2Pfd2.

enumerator kCLOCK_CSI2_UI_ClockRoot_MuxSysPll3Out
CSI2_UI mux from MuxSysPll3Out.

enumerator kCLOCK_CSI2_UI_ClockRoot_MuxSysPll2Pfd0
CSI2_UI mux from MuxSysPll2Pfd0.

enumerator kCLOCK_CSI2_UI_ClockRoot_MuxVideoPllOut
CSI2_UI mux from MuxVideoPllOut.

enumerator kCLOCK_CSI_ClockRoot_MuxOscRc48MDiv2
CSI mux from MuxOscRc48MDiv2.

enumerator kCLOCK_CSI_ClockRoot_MuxOsc24MOut
CSI mux from MuxOsc24MOut.

enumerator kCLOCK_CSI_ClockRoot_MuxOscRc400M
CSI mux from MuxOscRc400M.

enumerator kCLOCK_CSI_ClockRoot_MuxOscRc16M
CSI mux from MuxOscRc16M.

enumerator kCLOCK_CSI_ClockRoot_MuxSysPll2Pfd2
CSI mux from MuxSysPll2Pfd2.

enumerator kCLOCK_CSI_ClockRoot_MuxSysPll3Out
CSI mux from MuxSysPll3Out.

enumerator kCLOCK_CSI_ClockRoot_MuxSysPll3Pfd1
CSI mux from MuxSysPll3Pfd1.

enumerator kCLOCK_CSI_ClockRoot_MuxVideoPllOut
CSI mux from MuxVideoPllOut.

enumerator kCLOCK_CKO1_ClockRoot_MuxOscRc48MDiv2
CKO1 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_CKO1_ClockRoot_MuxOsc24MOut
CKO1 mux from MuxOsc24MOut.

enumerator kCLOCK_CKO1_ClockRoot_MuxOscRc400M
CKO1 mux from MuxOscRc400M.

enumerator kCLOCK_CKO1_ClockRoot_MuxOscRc16M
CKO1 mux from MuxOscRc16M.

enumerator kCLOCK_CKO1_ClockRoot_MuxSysPll2Pfd2
CKO1 mux from MuxSysPll2Pfd2.

enumerator kCLOCK_CKO1_ClockRoot_MuxSysPll2Out
CKO1 mux from MuxSysPll2Out.

enumerator kCLOCK_CKO1_ClockRoot_MuxSysPll3Pfd1
CKO1 mux from MuxSysPll3Pfd1.

enumerator kCLOCK_CKO1_ClockRoot_MuxSysPll1Div5

CKO1 mux from MuxSysPll1Div5.

enumerator kCLOCK_CKO2_ClockRoot_MuxOscRc48MDiv2

CKO2 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_CKO2_ClockRoot_MuxOsc24MOut

CKO2 mux from MuxOsc24MOut.

enumerator kCLOCK_CKO2_ClockRoot_MuxOscRc400M

CKO2 mux from MuxOscRc400M.

enumerator kCLOCK_CKO2_ClockRoot_MuxOscRc16M

CKO2 mux from MuxOscRc16M.

enumerator kCLOCK_CKO2_ClockRoot_MuxSysPll2Pfd3

CKO2 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_CKO2_ClockRoot_MuxOscRc48M

CKO2 mux from MuxOscRc48M.

enumerator kCLOCK_CKO2_ClockRoot_MuxSysPll3Pfd1

CKO2 mux from MuxSysPll3Pfd1.

enumerator kCLOCK_CKO2_ClockRoot_MuxAudioPllOut

CKO2 mux from MuxAudioPllOut.

enum _clock_group

Clock group enumeration.

Values:

enumerator kCLOCK_Group_FlexRAM

FlexRAM clock group.

enumerator kCLOCK_Group_MipiDsi

Mipi Dsi clock group.

enumerator kCLOCK_Group_Last

Last clock group.

enum _clock_osc

OSC 24M source select.

Values:

enumerator kCLOCK_RcOsc

On chip OSC.

enumerator kCLOCK_XtalOsc

24M Xtal OSC

enum _clock_gate_value

Clock gate value.

Values:

enumerator kCLOCK_Off

Clock is off.

enumerator kCLOCK_On

Clock is on

enum `_clock_mode_t`

System clock mode.

Values:

enumerator `kCLOCK_ModeRun`

Remain in run mode.

enumerator `kCLOCK_ModeWait`

Transfer to wait mode.

enumerator `kCLOCK_ModeStop`

Transfer to stop mode.

enum `_clock_usb_src`

USB clock source definition.

Values:

enumerator `kCLOCK_Usb480M`

Use 480M.

enumerator `kCLOCK_UsbSrcUnused`

Used when the function does not care the clock source.

enum `_clock_usb_phy_src`

Source of the USB HS PHY.

Values:

enumerator `kCLOCK_Usbphy480M`

Use 480M.

enum `_clock_pll_clk_src`

PLL clock source, bypass cloco source also.

Values:

enumerator `kCLOCK_PllClkSrc24M`

Pll clock source 24M

enumerator `kCLOCK_PllSrcClkPN`

Pll clock source CLK1_P and CLK1_N

enum `_clock_pll_post_div`

PLL post divider enumeration.

Values:

enumerator `kCLOCK_PllPostDiv2`

Divide by 2.

enumerator `kCLOCK_PllPostDiv4`

Divide by 4.

enumerator `kCLOCK_PllPostDiv8`

Divide by 8.

enumerator `kCLOCK_PllPostDiv1`

Divide by 1.

enum `_clock_output1_selection`

The enumerator of clock output1's clock source.

Values:

enumerator kCLOCK_CKO1OutputMuxOscRc48MDiv2
CKO1 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_CKO1OutputMuxOsc24MOut
CKO1 mux from MuxOsc24MOut.

enumerator kCLOCK_CKO1OutputMuxOscRc400M
CKO1 mux from MuxOscRc400M.

enumerator kCLOCK_CKO1OutputMuxOscRc16M
CKO1 mux from MuxOscRc16M.

enumerator kCLOCK_CKO1OutputMuxSysPll2Pfd2
CKO1 mux from MuxSysPll2Pfd2.

enumerator kCLOCK_CKO1OutputMuxSysPll2Out
CKO1 mux from MuxSysPll2Out.

enumerator kCLOCK_CKO1OutputMuxSysPll3Pfd1
CKO1 mux from MuxSysPll3Pfd1.

enumerator kCLOCK_CKO1OutputMuxSysPll1Div5
CKO1 mux from MuxSysPll1Div5.

enum __clock_output2_selection

The enumerater of clock output2's clock source.

Values:

enumerator kCLOCK_CKO2OutputOscRc48MDiv2
CKO2 mux from MuxOscRc48MDiv2.

enumerator kCLOCK_CKO2OutputOsc24MOut
CKO2 mux from MuxOsc24MOut.

enumerator kCLOCK_CKO2OutputOscRc400M
CKO2 mux from MuxOscRc400M.

enumerator kCLOCK_CKO2OutputOscRc16M
CKO2 mux from MuxOscRc16M.

enumerator kCLOCK_CKO2OutputSysPll2Pfd3
CKO2 mux from MuxSysPll2Pfd3.

enumerator kCLOCK_CKO2OutputMuxOscRc48M
CKO2 mux from MuxOscRc48M.

enumerator kCLOCK_CKO2OutputMuxSysPll3Pfd1
CKO2 mux from MuxSysPll3Pfd1.

enumerator kCLOCK_CKO2OutputMuxAudioPllOut
CKO2 mux from MuxAudioPllOut.

enum __clock_pll

PLL name.

Values:

enumerator kCLOCK_PllArm
ARM PLL.

enumerator kCLOCK_PllSys1
SYS1 PLL, it has a dedicated frequency of 1GHz.

enumerator kCLOCK_PllSys2

SYS2 PLL, it has a dedicated frequency of 528MHz.

enumerator kCLOCK_PllSys3

SYS3 PLL, it has a dedicated frequency of 480MHz.

enumerator kCLOCK_PllAudio

Audio PLL.

enumerator kCLOCK_PllVideo

Video PLL.

enumerator kCLOCK_PllInvalid

Invalid value.

enum _clock_pfd

PLL PFD name.

Values:

enumerator kCLOCK_Pfd0

PLL PFD0

enumerator kCLOCK_Pfd1

PLL PFD1

enumerator kCLOCK_Pfd2

PLL PFD2

enumerator kCLOCK_Pfd3

PLL PFD3

enum _clock_control_mode

The enumeration of control mode.

Values:

enumerator kCLOCK_SoftwareMode

Software control mode.

enumerator kCLOCK_GpcMode

GPC control mode.

enum _clock_24MOsc_mode

The enumeration of 24MHz crystal oscillator mode.

Values:

enumerator kCLOCK_24MOscHighGainMode

24MHz crystal oscillator work as high gain mode.

enumerator kCLOCK_24MOscBypassMode

24MHz crystal oscillator work as bypass mode.

enumerator kCLOCK_24MOscLowPowerMode

24MHz crystal oscillator work as low power mode.

enum _clock_16MOsc_source

The enumeration of 16MHz RC oscillator clock source.

Values:

enumerator kCLOCK_16MOscSourceFrom16MOsc

Source from 16MHz RC oscialltor.

enumerator kCLOCK_16MOscSourceFrom24MOsc

Source from 24MHz crystal oscillator.

enum _clock_1MHzOut_behavior

The enumeration of 1MHz output clock behavior, including disabling 1MHz output, enabling locked 1MHz clock output, and enabling free-running 1MHz clock output.

Values:

enumerator kCLOCK_1MHzOutDisable

Disable 1MHz output clock.

enumerator kCLOCK_1MHzOutEnableLocked1Mhz

Enable 1MHz output clock, and select locked 1MHz to output.

enumerator kCLOCK_1MHzOutEnableFreeRunning1Mhz

Enable 1MHz output clock, and select free-running 1MHz to output.

enum _clock_level

The clock dependence level.

Values:

enumerator kCLOCK_Level0

Not needed in any mode.

enumerator kCLOCK_Level1

Needed in RUN mode.

enumerator kCLOCK_Level2

Needed in RUN and WAIT mode.

enumerator kCLOCK_Level3

Needed in RUN, WAIT and STOP mode.

enumerator kCLOCK_Level4

Always on in any mode.

typedef enum _clock_lpcg clock_lpcg_t

Clock LPCG index.

typedef enum _clock_name clock_name_t

Clock name.

typedef enum _clock_root clock_root_t

Root clock index.

typedef enum _clock_root_mux_source clock_root_mux_source_t

The enumerator of clock roots' clock source mux value.

typedef enum _clock_group clock_group_t

Clock group enumeration.

typedef struct _clock_group_config clock_group_config_t

The structure used to configure clock group.

typedef enum _clock_osc clock_osc_t

OSC 24M source select.

typedef enum _clock_gate_value clock_gate_value_t

Clock gate value.

typedef enum _clock_mode_t clock_mode_t

System clock mode.

typedef enum *_clock_usb_src* clock_usb_src_t

USB clock source definition.

typedef enum *_clock_usb_phy_src* clock_usb_phy_src_t

Source of the USB HS PHY.

typedef enum *_clock_pll_post_div* clock_pll_post_div_t

PLL post divider enumeration.

typedef enum *_clock_output1_selection* clock_output1_selection_t

The enumerater of clock output1's clock source.

typedef enum *_clock_output2_selection* clock_output2_selection_t

The enumerater of clock output2's clock source.

typedef struct *_clock_arm_pll_config* clock_arm_pll_config_t

PLL configuration for ARM.

The output clock frequency is:

$F_{out} = F_{in} * \text{loopDivider} / (2 * \text{postDivider})$.

F_{in} is always 24MHz.

typedef struct *_clock_usb_pll_config* clock_usb_pll_config_t

PLL configuration for USB.

typedef struct *_clock_pll_ss_config* clock_pll_ss_config_t

Spread specturm configure PLL.

typedef struct *_clock_sys_pll2_config* clock_sys_pll2_config_t

PLL configure for Sys PLL2.

typedef struct *_clock_sys_pll1_config* clock_sys_pll1_config_t

PLL configure for Sys PLL1.

typedef struct *_clock_audio_pll_config* clock_av_pll_config_t

PLL configuration for AUDIO and VIDEO.

typedef struct *_clock_audio_pll_config* clock_audio_pll_config_t

typedef struct *_clock_audio_pll_config* clock_video_pll_config_t

typedef struct *_clock_audio_pll_gpc_config* clock_audio_pll_gpc_config_t

PLL configuration fro AUDIO PLL, SYSTEM PLL1 and VIDEO PLL.

typedef struct *_clock_audio_pll_gpc_config* clock_video_pll_gpc_config_t

typedef struct *_clock_audio_pll_gpc_config* clock_sys_pll1_gpc_config_t

typedef struct *_clock_enet_pll_config* clock_enet_pll_config_t

PLL configuration for ENET.

typedef struct *_clock_root_config_t* clock_root_config_t

Clock root configuration.

typedef struct *_clock_root_setpoint_config_t* clock_root_setpoint_config_t

Clock root configuration in SetPoint Mode.

typedef enum *_clock_pll* clock_pll_t

PLL name.

typedef enum *_clock_pfd* clock_pfd_t

PLL PFD name.

typedef enum *_clock_control_mode* clock_control_mode_t

The enumeration of control mode.

typedef enum *_clock_24MOsc_mode* clock_24MOsc_mode_t

The enumeration of 24MHz crystal oscillator mode.

typedef enum *_clock_16MOsc_source* clock_16MOsc_source_t

The enumeration of 16MHz RC oscillator clock source.

typedef enum *_clock_1MHzOut_behavior* clock_1MHzOut_behavior_t

The enumeration of 1MHz output clock behavior, including disabling 1MHz output, enabling locked 1MHz clock output, and enabling free-running 1MHz clock output.

typedef enum *_clock_level* clock_level_t

The clock dependence level.

const *clock_name_t* s_clockSourceName[][8]

static inline void CLOCK_SetRootClockMux(*clock_root_t* root, uint8_t src)

Set CCM Root Clock MUX node to certain value.

Parameters

- *root* – Which root clock node to set, see *clock_root_t*.
- *src* – Clock mux value to set, different mux has different value range. See *clock_root_mux_source_t*.

static inline uint32_t CLOCK_GetRootClockMux(*clock_root_t* root)

Get CCM Root Clock MUX value.

Parameters

- *root* – Which root clock node to get, see *clock_root_t*.

Returns

Clock mux value.

static inline *clock_name_t* CLOCK_GetRootClockSource(*clock_root_t* root, uint32_t src)

Get CCM Root Clock Source.

Parameters

- *root* – Which root clock node to get, see *clock_root_t*.
- *src* – Clock mux value to get, see *clock_root_mux_source_t*.

Returns

Clock source

static inline void CLOCK_SetRootClockDiv(*clock_root_t* root, uint32_t div)

Set CCM Root Clock DIV certain value.

Parameters

- *root* – Which root clock to set, see *clock_root_t*.
- *div* – Clock div value to set range is 1-256, different divider has different value range.

static inline uint32_t CLOCK_GetRootClockDiv(*clock_root_t* root)

Get CCM DIV node value.

Parameters

- *root* – Which root clock node to get, see *clock_root_t*.

Returns

divider set for this root

static inline void CLOCK_PowerOffRootClock(*clock_root_t* root)

Power Off Root Clock.

Parameters

- root – Which root clock node to set, see *clock_root_t*.

static inline void CLOCK_PowerOnRootClock(*clock_root_t* root)

Power On Root Clock.

Parameters

- root – Which root clock node to set, see *clock_root_t*.

static inline void CLOCK_SetRootClock(*clock_root_t* root, const *clock_root_config_t* *config)

Configure Root Clock.

Parameters

- root – Which root clock node to set, see *clock_root_t*.
- config – root clock config, see *clock_root_config_t*

static inline void CLOCK_ControlGate(*clock_lpcg_t* name, *clock_gate_value_t* value)

Control the clock gate for specific IP.

Note: This API will not have any effect when this clock is in CPULPM or SetPoint Mode

Parameters

- name – Which clock to enable, see *clock_lpcg_t*.
- value – Clock gate value to set, see *clock_gate_value_t*.

static inline void CLOCK_EnableClock(*clock_lpcg_t* name)

Enable the clock for specific IP.

Parameters

- name – Which clock to enable, see *clock_lpcg_t*.

static inline void CLOCK_DisableClock(*clock_lpcg_t* name)

Disable the clock for specific IP.

Parameters

- name – Which clock to disable, see *clock_lpcg_t*.

void CLOCK_SetGroupConfig(*clock_group_t* group, const *clock_group_config_t* *config)

Set the clock group configuration.

Parameters

- group – Which group to configure, see *clock_group_t*.
- config – Configuration to set.

uint32_t CLOCK_GetFreq(*clock_name_t* name)

Gets the clock frequency for a specific clock name.

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

Parameters

- name – Clock names defined in *clock_name_t*

Returns

Clock frequency value in hertz

static inline uint32_t CLOCK_GetRootClockFreq(*clock_root_t* root)

Gets the clock frequency for a specific root clock name.

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

Parameters

- *root* – Clock names defined in *clock_root_t*

Returns

Clock frequency value in hertz

static inline uint32_t CLOCK_GetM7Freq(void)

Get the CCM CPU/core/system frequency.

Returns

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

static inline uint32_t CLOCK_GetM4Freq(void)

Get the CCM CPU/core/system frequency.

Returns

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

static inline bool CLOCK_IsPllBypassed(*clock_pll_t* pll)

Check if PLL is bypassed.

Parameters

- *pll* – PLL control name (see *clock_pll_t* enumeration)

Returns

PLL bypass status.

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

static inline bool CLOCK_IsPllEnabled(*clock_pll_t* pll)

Check if PLL is enabled.

Parameters

- *pll* – PLL control name (see *clock_pll_t* enumeration)

Returns

PLL bypass status.

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

FSL_CLOCK_DRIVER_VERSION

CLOCK driver version.

SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY

CCSR_OFFSET

CCM registers offset.

CBCDR_OFFSET

CBCMR_OFFSET

CSCMR1_OFFSET

CSCMR2_OFFSET

CSCDR1_OFFSET

CDCDR_OFFSET

CSCDR2_OFFSET

CSCDR3_OFFSET

CACRR_OFFSET

CS1CDR_OFFSET

CS2CDR_OFFSET

ARM_PLL_OFFSET

CCM Analog registers offset.

PLL_SYS_OFFSET

PLL_USB1_OFFSET

PLL_AUDIO_OFFSET

PLL_VIDEO_OFFSET

PLL_ENET_OFFSET

PLL_USB2_OFFSET

CCM_TUPLE(reg, shift, mask, busyShift)

CCM_TUPLE_REG(base, tuple)

CCM_TUPLE_SHIFT(tuple)

CCM_TUPLE_MASK(tuple)

CCM_TUPLE_BUSY_SHIFT(tuple)

CCM_BUSY_WAIT

CCM_ANALOG_TUPLE(reg, shift)

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

CCM_ANALOG_TUPLE_SHIFT(tuple)

CCM_ANALOG_TUPLE_REG_OFF(base, tuple, off)

CCM_ANALOG_TUPLE_REG(base, tuple)

SYS_PLL1_FREQ

SYS_PLL_FREQ frequency in Hz.

SYS_PLL2_MFI

SYS_PLL2_FREQ

SYS_PLL3_MFI

SYS_PLL3_FREQ

XTAL_FREQ

LPADC_CLOCKS

Clock gate name array for ADC.

ADC_ETC_CLOCKS

Clock gate name array for ADC.

AOI_CLOCKS

Clock gate name array for AOI.

DCDC_CLOCKS

Clock gate name array for DCDC.

Clock ip name array for DCDC.

DCDC_CLOCKS

Clock gate name array for DCDC.

Clock ip name array for DCDC.

SRC_CLOCKS

Clock gate name array for SRC.

GPC_CLOCKS

Clock gate name array for GPC.

SSARC_CLOCKS

Clock gate name array for SSARC.

WDOG_CLOCKS

Clock gate name array for WDOG.

EWM_CLOCKS

Clock gate name array for EWM.

SEMA_CLOCKS

Clock gate name array for Sema.

MU_CLOCKS

Clock gate name array for MU.

EDMA_CLOCKS

Clock gate name array for EDMA.

FLEXRAM_CLOCKS

Clock gate name array for FLEXRAM.

LMEM_CLOCKS

Clock gate name array for LMEM.

FLEXSPI_CLOCKS

Clock gate name array for FLEXSPI.

RDC_CLOCKS

Clock gate name array for RDC.

SEMC_CLOCKS

Clock ip name array for SEMC.

XECC_CLOCKS

Clock ip name array for XECC.

IEE_CLOCKS

Clock ip name array for IEE.

KEYMANAGER_CLOCKS

Clock ip name array for KEY_MANAGER.

PUF_CLOCKS

Clock ip name array for PUF.

OCOTP_CLOCKS

Clock ip name array for OCOTP.

CAAM_CLOCKS

Clock ip name array for CAAM.

XBAR_CLOCKS

Clock ip name array for XBAR.

IOMUXC_CLOCKS

Clock ip name array for IOMUXC.

GPIO_CLOCKS

Clock ip name array for GPIO.

KPP_CLOCKS

Clock ip name array for KPP.

FLEXIO_CLOCKS

Clock ip name array for FLEXIO.

DAC_CLOCKS

Clock ip name array for DAC.

CMP_CLOCKS

Clock ip name array for CMP.

PIT_CLOCKS

Clock ip name array for PIT.

GPT_CLOCKS

Clock ip name array for GPT.

TMR_CLOCKS

Clock ip name array for QTIMER.

ENC_CLOCKS

Clock ip name array for ENC.

PWM_CLOCKS

Clock ip name array for PWM.

FLEXCAN_CLOCKS

Clock ip name array for FLEXCAN.

LPUART_CLOCKS

Clock ip name array for LPUART.

LPI2C_CLOCKS

Clock ip name array for LPI2C.

LPSPi_CLOCKS

Clock ip name array for LPSPi.

EMVSIM_CLOCKS

Clock ip name array for EMVSIM.

ENET_CLOCKS

Clock ip name array for ENET.

USB_CLOCKS

Clock ip name array for USB.

CDOG_CLOCKS

Clock ip name array for CDOG.

USDHC_CLOCKS

Clock ip name array for USDHC.

ASRC_CLOCKS

Clock ip name array for ASRC.

MQS_CLOCKS

Clock ip name array for MQS.

PDM_CLOCKS

Clock ip name array for PDM.

SPDIF_CLOCKS

Clock ip name array for SPDIF.

SAI_CLOCKS

Clock ip name array for SAI.

PXP_CLOCKS

Clock ip name array for PXP.

GPU2D_CLOCKS

Clock ip name array for GPU2d.

LCDIF_CLOCKS

Clock ip name array for LCDIF.

LCDIFV2_CLOCKS

Clock ip name array for LCDIFV2.

MIPI_DSI_HOST_CLOCKS

Clock ip name array for MIPI_DSI.

MIPI_CSI2RX_CLOCKS

Clock ip name array for MIPI_CSI.

CSI_CLOCKS

Clock ip name array for CSI.

DCIC_CLOCKS

Clock ip name array for DCIC.

DMAMUX_CLOCKS

Clock ip name array for DMAMUX_CLOCKS.

XBARA_CLOCKS

Clock ip name array for XBARA.

XBARB_CLOCKS

Clock ip name array for XBARB.

CCM_OBS_M7_CLK_ROOT
CCM_OBS_M4_CLK_ROOT
CCM_OBS_BUS_CLK_ROOT
CCM_OBS_BUS_LPSR_CLK_ROOT
CCM_OBS_SEMC_CLK_ROOT
CCM_OBS_CSSYS_CLK_ROOT
CCM_OBS_CSTRACE_CLK_ROOT
CCM_OBS_M4_SYSTICK_CLK_ROOT
CCM_OBS_M7_SYSTICK_CLK_ROOT
CCM_OBS_ADC1_CLK_ROOT
CCM_OBS_ADC2_CLK_ROOT
CCM_OBS_ACMP_CLK_ROOT
CCM_OBS_FLEXIO1_CLK_ROOT
CCM_OBS_FLEXIO2_CLK_ROOT
CCM_OBS_GPT1_CLK_ROOT
CCM_OBS_GPT2_CLK_ROOT
CCM_OBS_GPT3_CLK_ROOT
CCM_OBS_GPT4_CLK_ROOT
CCM_OBS_GPT5_CLK_ROOT
CCM_OBS_GPT6_CLK_ROOT
CCM_OBS_FLEXSPI1_CLK_ROOT
CCM_OBS_FLEXSPI2_CLK_ROOT
CCM_OBS_CAN1_CLK_ROOT
CCM_OBS_CAN2_CLK_ROOT
CCM_OBS_CAN3_CLK_ROOT
CCM_OBS_LPUART1_CLK_ROOT
CCM_OBS_LPUART2_CLK_ROOT
CCM_OBS_LPUART3_CLK_ROOT
CCM_OBS_LPUART4_CLK_ROOT
CCM_OBS_LPUART5_CLK_ROOT
CCM_OBS_LPUART6_CLK_ROOT
CCM_OBS_LPUART7_CLK_ROOT
CCM_OBS_LPUART8_CLK_ROOT

CCM_OBS_LPUART9_CLK_ROOT
CCM_OBS_LPUART10_CLK_ROOT
CCM_OBS_LPUART11_CLK_ROOT
CCM_OBS_LPUART12_CLK_ROOT
CCM_OBS_LPI2C1_CLK_ROOT
CCM_OBS_LPI2C2_CLK_ROOT
CCM_OBS_LPI2C3_CLK_ROOT
CCM_OBS_LPI2C4_CLK_ROOT
CCM_OBS_LPI2C5_CLK_ROOT
CCM_OBS_LPI2C6_CLK_ROOT
CCM_OBS_LPSPI1_CLK_ROOT
CCM_OBS_LPSPI2_CLK_ROOT
CCM_OBS_LPSPI3_CLK_ROOT
CCM_OBS_LPSPI4_CLK_ROOT
CCM_OBS_LPSPI5_CLK_ROOT
CCM_OBS_LPSPI6_CLK_ROOT
CCM_OBS_EMV1_CLK_ROOT
CCM_OBS_EMV2_CLK_ROOT
CCM_OBS_ENET1_CLK_ROOT
CCM_OBS_ENET2_CLK_ROOT
CCM_OBS_ENET_25M_CLK_ROOT
CCM_OBS_ENET_TIMER1_CLK_ROOT
CCM_OBS_ENET_TIMER2_CLK_ROOT
CCM_OBS_USDHC1_CLK_ROOT
CCM_OBS_USDHC2_CLK_ROOT
CCM_OBS_ASRC_CLK_ROOT
CCM_OBS_MQS_CLK_ROOT
CCM_OBS_MIC_CLK_ROOT
CCM_OBS_SPDIF_CLK_ROOT
CCM_OBS_SAI1_CLK_ROOT
CCM_OBS_SAI2_CLK_ROOT
CCM_OBS_SAI3_CLK_ROOT
CCM_OBS_SAI4_CLK_ROOT

CCM_OBS_GC355_CLK_ROOT
CCM_OBS_LCDIF_CLK_ROOT
CCM_OBS_LCDIFV2_CLK_ROOT
CCM_OBS_MIPI_REF_CLK_ROOT
CCM_OBS_MIPI_ESC_CLK_ROOT
CCM_OBS_CSI2_CLK_ROOT
CCM_OBS_CSI2_ESC_CLK_ROOT
CCM_OBS_CSI2_UI_CLK_ROOT
CCM_OBS_CSI_CLK_ROOT
CCM_OBS_CCM_CK01_CLK_ROOT
CCM_OBS_CCM_CK02_CLK_ROOT
CCM_OBS_CM7_CORE_STCLKEN
CCM_OBS_CCM_FLEXRAM_CLK_ROOT
CCM_OBS_MIPI_DSI_TXESC
CCM_OBS_MIPI_DSI_RXESC
CCM_OBS_OSC_RC_16M
CCM_OBS_OSC_RC_48M
CCM_OBS_OSC_RC_48M_DIV2
CCM_OBS_OSC_RC_400M
CCM_OBS_OSC_24M_OUT
CCM_OBS_ARM_PLL_OUT
CCM_OBS_SYS_PLL2_OUT
CCM_OBS_SYS_PLL2_PFD0
CCM_OBS_SYS_PLL2_PFD1
CCM_OBS_SYS_PLL2_PFD2
CCM_OBS_SYS_PLL2_PFD3
CCM_OBS_SYS_PLL3_OUT
CCM_OBS_SYS_PLL3_DIV2
CCM_OBS_SYS_PLL3_PFD0
CCM_OBS_SYS_PLL3_PFD1
CCM_OBS_SYS_PLL3_PFD2
CCM_OBS_SYS_PLL3_PFD3
CCM_OBS_SYS_PLL1_OUT

CCM_OBS_SYS_PLL1_DIV2

CCM_OBS_SYS_PLL1_DIV5

CCM_OBS_PLL_AUDIO_OUT

CCM_OBS_PLL_VIDEO_OUT

CCM_OBS_DIV

clock_ip_name_t

CLOCK_GetCpuClkFreq

CLOCK_GetCoreSysClkFreq

For compatible with other platforms without CCM.

PLL_PFD_COUNT

static inline uint32_t CLOCK_GetRtcFreq(void)

Gets the RTC clock frequency.

Returns

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

static inline void CLOCK_OSC_SetOsc48MControlMode(*clock_control_mode_t* controlMode)

Set the control mode of 48MHz RC oscillator.

Parameters

- controlMode – The control mode to be set, please refer to *clock_control_mode_t*.

static inline void CLOCK_OSC_EnableOsc48M(bool enable)

Enable/disable 48MHz RC oscillator.

Parameters

- enable – Used to enable or disable the 48MHz RC oscillator.
 - **true** Enable the 48MHz RC oscillator.
 - **false** Dissable the 48MHz RC oscillator.

static inline void CLOCK_OSC_SetOsc48MDiv2ControlMode(*clock_control_mode_t* controlMode)

Set the control mode of the 24MHz clock sourced from 48MHz RC oscillator.

Parameters

- controlMode – The control mode to be set, please refer to *clock_control_mode_t*.

static inline void CLOCK_OSC_EnableOsc48MDiv2(bool enable)

Enable/disable the 24MHz clock sourced from 48MHz RC oscillator.

Note: The 48MHz RC oscillator must be enabled before enabling this 24MHz clock.

Parameters

- enable – Used to enable/disable the 24MHz clock sourced from 48MHz RC oscillator.
 - **true** Enable the 24MHz clock sourced from 48MHz.
 - **false** Disable the 24MHz clock sourced from 48MHz.

static inline void CLOCK_OSC_SetOsc24MControlMode(*clock_control_mode_t* controlMode)
Set the control mode of 24MHz crystal oscillator.

Parameters

- controlMode – The control mode to be set, please refer to *clock_control_mode_t*.

void CLOCK_OSC_EnableOsc24M(void)
Enable OSC 24Mhz.

This function enables OSC 24Mhz.

static inline void CLOCK_OSC_GateOsc24M(bool enableGate)
Gate/ungate the 24MHz crystal oscillator output.

Note: Gating the 24MHz crystal oscillator can save power.

Parameters

- enableGate – Used to gate/ungate the 24MHz crystal oscillator.
 - **true** Gate the 24MHz crystal oscillator to save power.
 - **false** Ungate the 24MHz crystal oscillator.

void CLOCK_OSC_SetOsc24MWorkMode(*clock_24MOsc_mode_t* workMode)
Set the work mode of 24MHz crystal oscillator, the available modes are high gian mode, low power mode, and bypass mode.

Parameters

- workMode – The work mode of 24MHz crystal oscillator, please refer to *clock_24MOsc_mode_t* for details.

static inline void CLOCK_OSC_SetOscRc400MControlMode(*clock_control_mode_t* controlMode)
Set the control mode of 400MHz RC oscillator.

Parameters

- controlMode – The control mode to be set, please refer to *clock_control_mode_t*.

void CLOCK_OSC_EnableOscRc400M(void)
Enable OSC RC 400Mhz.

This function enables OSC RC 400Mhz.

static inline void CLOCK_OSC_GateOscRc400M(bool enableGate)
Gate/ungate 400MHz RC oscillator.

Parameters

- enableGate – Used to gate/ungate 400MHz RC oscillator.
 - **true** Gate the 400MHz RC oscillator.
 - **false** Ungate the 400MHz RC oscillator.

void CLOCK_OSC_TrimOscRc400M(bool enable, bool bypass, uint16_t trim)
Trims OSC RC 400MHz.

Parameters

- enable – Used to enable trim function.
- bypass – Bypass the trim function.

- trim – Trim value.

void CLOCK_OSC_SetOscRc400MRefClkDiv(uint8_t divValue)

Set the divide value for ref_clk to generate slow clock.

Note: $\text{slow\_clk} = \text{ref\_clk} / (\text{divValue} + 1)$, and the recommend divide value is 24.

Parameters

- divValue – The divide value to be set, the available range is 0~63.

void CLOCK_OSC_SetOscRc400MFastClkCount(uint16_t targetCount)

Set the target count for the fast clock.

Parameters

- targetCount – The desired target for the fast clock, should be the number of clock cycles of the fast_clk per divided ref_clk.

void CLOCK_OSC_SetOscRc400MHysteresisValue(uint8_t negHysteresis, uint8_t posHysteresis)

Set the negative and positive hysteresis value for the tuned clock.

Note: The hysteresis value should be set after the clock is tuned.

Parameters

- negHysteresis – The negative hysteresis value for the turned clock, this value in number of clock cycles of the fast clock
- posHysteresis – The positive hysteresis value for the turned clock, this value in number of clock cycles of the fast clock

void CLOCK_OSC_BypassOscRc400MTuneLogic(bool enableBypass)

Bypass/un-bypass the tune logic.

Parameters

- enableBypass – Used to control whether to bypass the turn logic.
 - **true** Bypass the tune logic and use the programmed oscillator frequency to run the oscillator. Function CLOCK_OSC_SetOscRc400MTuneValue() can be used to set oscillator frequency.
 - **false** Use the output of tune logic to run the oscillator.

void CLOCK_OSC_EnableOscRc400MTuneLogic(bool enable)

Start/Stop the tune logic.

Parameters

- enable – Used to start or stop the tune logic.
 - **true** Start tuning
 - **false** Stop tuning and reset the tuning logic.

void CLOCK_OSC_FreezeOscRc400MTuneValue(bool enableFreeze)

Freeze/Unfreeze the tuning value.

Parameters

- enableFreeze – Used to control whether to freeze the tune value.
 - **true** Freeze the tune at the current tuned value and the oscillator runs at the frozen tune value.

- **false** Unfreezes and continues the tune operation.

void CLOCK_OSC_SetOscRc400MTuneValue(uint8_t tuneValue)

Set the 400MHz RC oscillator tune value when the tune logic is disabled.

Parameters

- tuneValue – The tune value to determine the frequency of Oscillator.

void CLOCK_OSC_Set1MHzOutputBehavior(*clock_1MHzOut_behavior_t* behavior)

Set the behavior of the 1MHz output clock, such as disable the 1MHz clock output, enable the free-running 1MHz clock output, enable the locked 1MHz clock output.

Note: The 1MHz clock is divided from 400M RC Oscillator.

Parameters

- behavior – The behavior of 1MHz output clock, please refer to *clock_1MHzOut_behavior_t* for details.

void CLOCK_OSC_SetLocked1MHzCount(uint16_t count)

Set the count for the locked 1MHz clock out.

Parameters

- count – Used to set the desired target for the locked 1MHz clock out, the value in number of clock cycles of the fast clock per divided ref_clk.

bool CLOCK_OSC_CheckLocked1MHzErrorFlag(void)

Check the error flag for locked 1MHz clock out.

Returns

The error flag for locked 1MHz clock out.

- **true** The count value has been reached within one divided ref clock period
- **false** No effect.

void CLOCK_OSC_ClearLocked1MHzErrorFlag(void)

Clear the error flag for locked 1MHz clock out.

uint16_t CLOCK_OSC_GetCurrentOscRc400MFastClockCount(void)

Get current count for the fast clock during the tune process.

Returns

The current count for the fast clock.

uint8_t CLOCK_OSC_GetCurrentOscRc400MTuneValue(void)

Get current tune value used by oscillator during tune process.

Returns

The current tune value.

static inline void CLOCK_OSC_SetOsc16MControlMode(*clock_control_mode_t* controlMode)

Set the control mode of 16MHz crystal oscillator.

Parameters

- controlMode – The control mode to be set, please refer to *clock_control_mode_t*.

void CLOCK_OSC_SetOsc16MConfig(*clock_16MOsc_source_t* source, bool enablePowerSave, bool enableClockOut)

Configure the 16MHz oscillator.

Parameters

- `source` – Used to select the source for 16MHz RC oscillator, please refer to `clock_16MOsc_source_t`.
- `enablePowerSave` – Enable/disable power save mode function at 16MHz OSC.
 - **true** Enable power save mode function at 16MHz osc.
 - **false** Disable power save mode function at 16MHz osc.
- `enableClockOut` – Enable/Disable clock output for 16MHz RCOSC.
 - **true** Enable clock output for 16MHz RCOSC.
 - **false** Disable clock output for 16MHz RCOSC.

`void CLOCK_InitArmPll(const clock_arm_pll_config_t *config)`

Initialize the ARM PLL.

This function initialize the ARM PLL with specific settings

Parameters

- `config` – configuration to set to PLL.

`status_t CLOCK_CalcArmPllFreq(clock_arm_pll_config_t *config, uint32_t freqInMhz)`

Calculate corresponding config values per given frequency.

This function calculates config valudes per given frequency for Arm PLL

Parameters

- `config` – pll config structure
- `freqInMhz` – target frequency

`status_t CLOCK_InitArmPllWithFreq(uint32_t freqInMhz)`

Initializes the Arm PLL with Specific Frequency (in Mhz).

This function initializes the Arm PLL with specific frequency

Parameters

- `freqInMhz` – target frequency

`void CLOCK_DeinitArmPll(void)`

De-initialize the ARM PLL.

`void CLOCK_CalcPllSpreadSpectrum(uint32_t factor, uint32_t range, uint32_t mod, clock_pll_ss_config_t *ss)`

Calculate spread spectrum step and stop.

This function calculate spread spectrum step and stop according to given parameters. For integer PLL (syspll2) the factor is mfd, while for other fractional PLLs (audio/video/syspll1), the factor is denominator.

Parameters

- `factor` – factor to calculate step/stop
- `range` – spread spectrum range
- `mod` – spread spectrum modulation frequency
- `ss` – calculated spread spectrum values

`void CLOCK_InitSysPll1(const clock_sys_pll1_config_t *config)`

Initialize the System PLL1.

This function initializes the System PLL1 with specific settings

Parameters

- config – Configuration to set to PLL1.

void CLOCK_DeinitSysPll1(void)

De-initialize the System PLL1.

void CLOCK_GPC_SetSysPll1OutputFreq(const *clock_sys_pll1_gpc_config_t* *config)

Set System PLL1 output frequency in GPC mode.

Parameters

- config – Pointer to System PLL1 configure structure.

void CLOCK_InitSysPll2(const *clock_sys_pll2_config_t* *config)

Initialize the System PLL2.

This function initializes the System PLL2 with specific settings

Parameters

- config – Configuration to configure spread spectrum. This parameter can be NULL, if no need to enabled spread spectrum

void CLOCK_DeinitSysPll2(void)

De-initialize the System PLL2.

bool CLOCK_IsSysPll2PfdEnabled(*clock_pfd_t* pfd)

Check if Sys PLL2 PFD is enabled.

Note: Only useful in software control mode.

Parameters

- pfd – PFD control name

Returns

PFD bypass status.

- true: power on.
- false: power off.

void CLOCK_InitSysPll3(void)

Initialize the System PLL3.

This function initializes the System PLL3 with specific settings

void CLOCK_DeinitSysPll3(void)

De-initialize the System PLL3.

bool CLOCK_IsSysPll3PfdEnabled(*clock_pfd_t* pfd)

Check if Sys PLL3 PFD is enabled.

Note: Only useful in software control mode.

Parameters

- pfd – PFD control name

Returns

PFD bypass status.

- true: power on.

- false: power off.

void CLOCK_SetPllBypass(*clock_pll_t* pll, bool bypass)

PLL bypass setting.

Parameters

- pll – PLL control name (see *clock_pll_t* enumeration)
- bypass – Bypass the PLL.
 - true: Bypass the PLL.
 - false: Not bypass the PLL.

status_t CLOCK_CalcAvPllFreq(*clock_av_pll_config_t* *config, uint32_t freqInMhz)

Calculate corresponding config values per given frequency.

This function calculates config values per given frequency for Audio/Video PLL.

Parameters

- config – pll config structure
- freqInMhz – target frequency

status_t CLOCK_InitAudioPllWithFreq(uint32_t freqInMhz, bool ssEnable, uint32_t ssRange, uint32_t ssMod)

Initializes the Audio PLL with Specific Frequency (in Mhz).

This function initializes the Audio PLL with specific frequency

Parameters

- freqInMhz – target frequency
- ssEnable – enable spread spectrum or not
- ssRange – range spread spectrum range
- ssMod – spread spectrum modulation frequency

void CLOCK_InitAudioPll(const *clock_audio_pll_config_t* *config)

Initializes the Audio PLL.

This function initializes the Audio PLL with specific settings

Parameters

- config – Configuration to set to PLL.

void CLOCK_DeinitAudioPll(void)

De-initialize the Audio PLL.

void CLOCK_GPC_SetAudioPllOutputFreq(const *clock_audio_pll_gpc_config_t* *config)

Set Audio PLL output frequency in GPC mode.

Parameters

- config – Pointer to *clock_audio_pll_gpc_config_t* structure.

status_t CLOCK_InitVideoPllWithFreq(uint32_t freqInMhz, bool ssEnable, uint32_t ssRange, uint32_t ssMod)

Initializes the Video PLL with Specific Frequency (in Mhz).

This function initializes the Video PLL with specific frequency

Parameters

- freqInMhz – target frequency
- ssEnable – enable spread spectrum or not

- ssRange – range spread spectrum range
- ssMod – spread spectrum modulation frequency

void CLOCK_InitVideoPll(const *clock_video_pll_config_t* *config)

Initialize the video PLL.

This function configures the Video PLL with specific settings

Parameters

- config – configuration to set to PLL.

void CLOCK_DeinitVideoPll(void)

De-initialize the Video PLL.

void CLOCK_GPC_SetVideoPllOutputFreq(const *clock_video_pll_gpc_config_t* *config)

Set Video PLL output frequency in GPC mode.

Parameters

- config – Pointer to Video PLL configure structure.

uint32_t CLOCK_GetPllFreq(*clock_pll_t* pll)

Get current PLL output frequency.

This function get current output frequency of specific PLL

Parameters

- pll – pll name to get frequency.

Returns

The PLL output frequency in hertz.

void CLOCK_InitPfd(*clock_pll_t* pll, *clock_pfd_t* pfd, uint8_t frac)

Initialize PLL PFD.

This function initializes the System PLL PFD. During new value setting, the clock output is disabled to prevent glitch.

Note: It is recommended that PFD settings are kept between 12-35.

Parameters

- pll – Which PLL of targeting PFD to be operated.
- pfd – Which PFD clock to enable.
- frac – The PFD FRAC value.

void CLOCK_DeinitPfd(*clock_pll_t* pll, *clock_pfd_t* pfd)

De-initialize selected PLL PFD.

Parameters

- pll – Which PLL of targeting PFD to be operated.
- pfd – Which PFD clock to enable.

uint32_t CLOCK_GetPfdFreq(*clock_pll_t* pll, *clock_pfd_t* pfd)

Get current PFD output frequency.

This function get current output frequency of specific System PLL PFD

Parameters

- pll – Which PLL of targeting PFD to be operated.

- pfd – pfd name to get frequency.

Returns

The PFD output frequency in hertz.

uint32_t CLOCK_GetFreqFromObs(uint32_t obsSigIndex, uint32_t obsIndex)

bool CLOCK_EnableUsbhs0Clock(*clock_usb_src_t* src, uint32_t freq)

Enable USB HS clock.

This function only enables the access to USB HS peripheral, upper layer should first call the CLOCK_EnableUsbhs0PhyPllClock to enable the PHY clock to use USB HS.

Parameters

- src – USB HS does not care about the clock source, here must be kCLOCK_UsbSrcUnused.
- freq – USB HS does not care about the clock source, so this parameter is ignored.

Return values

- true – The clock is set successfully.
- false – The clock source is invalid to get proper USB HS clock.

bool CLOCK_EnableUsbhs1Clock(*clock_usb_src_t* src, uint32_t freq)

Enable USB HS clock.

This function only enables the access to USB HS peripheral, upper layer should first call the CLOCK_EnableUsbhs0PhyPllClock to enable the PHY clock to use USB HS.

Parameters

- src – USB HS does not care about the clock source, here must be kCLOCK_UsbSrcUnused.
- freq – USB HS does not care about the clock source, so this parameter is ignored.

Return values

- true – The clock is set successfully.
- false – The clock source is invalid to get proper USB HS clock.

bool CLOCK_EnableUsbhs0PhyPllClock(*clock_usb_phy_src_t* src, uint32_t freq)

Enable USB HS PHY PLL clock.

This function enables the internal 480MHz USB PHY PLL clock.

Parameters

- src – USB HS PHY PLL clock source.
- freq – The frequency specified by src.

Return values

- true – The clock is set successfully.
- false – The clock source is invalid to get proper USB HS clock.

void CLOCK_DisableUsbhs0PhyPllClock(void)

Disable USB HS PHY PLL clock.

This function disables USB HS PHY PLL clock.

bool CLOCK_EnableUsbhs1PhyPllClock(*clock_usb_phy_src_t* src, uint32_t freq)

Enable USB HS PHY PLL clock.

This function enables the internal 480MHz USB PHY PLL clock.

Parameters

- src – USB HS PHY PLL clock source.
- freq – The frequency specified by src.

Return values

- true – The clock is set successfully.
- false – The clock source is invalid to get proper USB HS clock.

void CLOCK_DisableUsbhs1PhyPllClock(void)

Disable USB HS PHY PLL clock.

This function disables USB HS PHY PLL clock.

static inline void CLOCK_OSCPLL_LockControlMode(*clock_name_t* name)

Lock low power and access control mode for this clock.

Note: When this bit is set, bits 16-20 can not be changed until next system reset.

Parameters

- name – Clock source name, see *clock_name_t*.

static inline void CLOCK_OSCPLL_LockWhiteList(*clock_name_t* name)

Lock the value of Domain ID white list for this clock.

Note: Once locked, this bit and domain ID white list can not be changed until next system reset.

Parameters

- name – Clock source name, see *clock_name_t*.

static inline void CLOCK_OSCPLL_SetWhiteList(*clock_name_t* name, uint8_t domainId)

Set domain ID that can change this clock.

Note: If LOCK_LIST bit is set, domain ID white list can not be changed until next system reset.

Parameters

- name – Clock source name, see *clock_name_t*.
- domainId – Domains that on the whitelist can change this clock.

static inline bool CLOCK_OSCPLL_IsSetPointImplemented(*clock_name_t* name)

Check whether this clock implement SetPoint control scheme.

Parameters

- name – Clock source name, see *clock_name_t*.

Returns

Clock source SetPoint implement status.

- true: SetPoint is implemented.
- false: SetPoint is not implemented.

static inline void CLOCK__OSCPLL__ControlByUnassignedMode(*clock_name_t* name)

Set this clock works in Unassigned Mode.

Note: When LOCK_MODE bit is set, control mode can not be changed until next system reset.

Parameters

- name – Clock source name, see *clock_name_t*.

void CLOCK__OSCPLL__ControlBySetPointMode(*clock_name_t* name, uint16_t spValue, uint16_t stbyValue)

Set this clock works in SetPoint control Mode.

Note: When LOCK_MODE bit is set, control mode can not be changed until next system reset.

Parameters

- name – Clock source name, see *clock_name_t*.
- spValue – Bit0~Bit15 hold value for Setpoint 0~16 respectively. A bitfield value of 0 implies clock will be shutdown in this Setpoint. A bitfield value of 1 implies clock will be turn on in this Setpoint.
- stbyValue – Bit0~Bit15 hold value for Setpoint 0~16 standby. A bitfield value of 0 implies clock will be shutdown during standby. A bitfield value of 1 represent clock will keep Setpoint setting during standby.

void CLOCK__OSCPLL__ControlByCpuLowPowerMode(*clock_name_t* name, uint8_t domainId, *clock_level_t* level0, *clock_level_t* level1)

Set this clock works in CPU Low Power Mode.

Note: When LOCK_MODE bit is set, control mode can not be changed until next system reset.

Parameters

- name – Clock source name, see *clock_name_t*.
- domainId – Domains that on the whitelist can change this clock.
- level1 (level0,) – Depend level of this clock.

static inline void CLOCK__OSCPLL__SetCurrentClockLevel(*clock_name_t* name, *clock_level_t* level)

Set clock depend level for current accessing domain.

Note: This setting only take effects in CPU Low Power Mode.

Parameters

- name – Clock source name, see *clock_name_t*.

- level – Depend level of this clock.

```
static inline void CLOCK_OSCPLL_ControlByDomainMode(clock_name_t name, uint8_t  
                                                    domainId)
```

Set this clock works in Domain Mode.

Note: When LOCK_MODE bit is set, control mode can not be changed until next system reset.

Parameters

- name – Clock source name, see *clock_name_t*.
- domainId – Domains that on the whitelist can change this clock.

```
static inline void CLOCK_ROOT_LockControlMode(clock_root_t name)
```

Lock low power and access control mode for this clock.

Note: When this bit is set, bits 16-20 can not be changed until next system reset.

Parameters

- name – Clock root name, see *clock_root_t*.

```
static inline void CLOCK_ROOT_LockWhiteList(clock_root_t name)
```

Lock the value of Domain ID white list for this clock.

Note: Once locked, this bit and domain ID white list can not be changed until next system reset.

Parameters

- name – Clock root name, see *clock_root_t*.

```
static inline void CLOCK_ROOT_SetWhiteList(clock_root_t name, uint8_t domainId)
```

Set domain ID that can change this clock.

Note: If LOCK_LIST bit is set, domain ID white list can not be changed until next system reset.

Parameters

- name – Clock root name, see *clock_root_t*.
- domainId – Domains that on the whitelist can change this clock.

```
static inline bool CLOCK_ROOT_IsSetPointImplemented(clock_root_t name)
```

Check whether this clock implement SetPoint control scheme.

Parameters

- name – Clock root name, see *clock_root_t*.

Returns

Clock root SetPoint implement status.

- true: SetPoint is implemented.
- false: SetPoint is not implemented.

```
static inline void CLOCK_ROOT_ControlByUnassignedMode(clock_root_t name)
```

Set this clock works in Unassigned Mode.

Note: When LOCK_MODE bit is set, control mode can not be changed until next system reset.

Parameters

- name – Clock root name, see `clock_root_t`.

```
static inline void CLOCK_ROOT_ConfigSetPoint(clock_root_t name, uint16_t spIndex, const  
                                             clock_root_setpoint_config_t *config)
```

Configure one SetPoint for this clock.

Note: SetPoint value could only be changed in Unassigned Mode.

Parameters

- name – Which clock root to set, see `clock_root_t`.
- spIndex – Which SetPoint of this clock root to set.
- config – SetPoint config, see `clock_root_setpoint_config_t`

```
static inline void CLOCK_ROOT_EnableSetPointControl(clock_root_t name)
```

Enable SetPoint control for this clock root.

Note: When LOCK_MODE bit is set, control mode can not be changed until next system reset.

Parameters

- name – Clock root name, see `clock_root_t`.

```
void CLOCK_ROOT_ControlBySetPointMode(clock_root_t name, const  
                                       clock_root_setpoint_config_t *spTable)
```

Set this clock works in SetPoint controlled Mode.

Note: When LOCK_MODE bit is set, control mode can not be changed until next system reset.

Parameters

- name – Clock root name, see `clock_root_t`.
- spTable – Point to the array that stores clock root settings for each setpoint.
Note that the pointed array must have 16 elements.

```
static inline void CLOCK_ROOT_ControlByDomainMode(clock_root_t name, uint8_t domainId)
```

Set this clock works in CPU Low Power Mode.

Note: When LOCK_MODE bit is set, control mode can not be changed until next system reset.

Parameters

- name – Clock root name, see clock_root_t.
- domainId – Domains that on the whitelist can change this clock.

static inline void CLOCK_LPCG_LockControlMode(*clock_lpcg_t* name)

Lock low power and access control mode for this clock.

Note: When this bit is set, bits 16-20 can not be changed until next system reset.

Parameters

- name – Clock gate name, see clock_lpcg_t.

static inline void CLOCK_LPCG_LockWhiteList(*clock_lpcg_t* name)

Lock the value of Domain ID white list for this clock.

Note: Once locked, this bit and domain ID white list can not be changed until next system reset.

Parameters

- name – Clock gate name, see clock_lpcg_t.

static inline void CLOCK_LPCG_SetWhiteList(*clock_lpcg_t* name, uint8_t domainId)

Set domain ID that can change this clock.

Note: If LOCK_LIST bit is set, domain ID white list can not be changed until next system reset.

Parameters

- name – Clock gate name, see clock_lpcg_t.
- domainId – Domains that on the whitelist can change this clock.

static inline bool CLOCK_LPCG_IsSetPointImplemented(*clock_lpcg_t* name)

Check whether this clock implement SetPoint control scheme.

Parameters

- name – Clock gate name, see clock_lpcg_t.

Returns

Clock gate SetPoint implement status.

- true: SetPoint is implemented.
- false: SetPoint is not implemented.

static inline void CLOCK_LPCG_ControlByUnassignedMode(*clock_lpcg_t* name)

Set this clock works in Unassigned Mode.

Note: When LOCK_MODE bit is set, control mode can not be changed until next system reset.

Parameters

- name – Clock gate name, see clock_lpcg_t.

```
void CLOCK_LPCG_ControlBySetPointMode(clock_lpcg_t name, uint16_t spValue, uint16_t
                                     stbyValue)
```

Set this clock works in SetPoint control Mode.

Note: When LOCK_MODE bit is set, control mode can not be changed until next system reset.

Parameters

- name – Clock gate name, see `clock_lpcg_t`.
- spValue – Bit0~Bit15 hold value for Setpoint 0~16 respectively. A bitfield value of 0 implies clock will be shutdown in this Setpoint. A bitfield value of 1 implies clock will be turn on in this Setpoint.
- stbyValue – Bit0~Bit15 hold value for Setpoint 0~16 standby. A bitfield value of 0 implies clock will be shutdown during standby. A bitfield value of 1 represent clock will keep Setpoint setting during standby.

```
void CLOCK_LPCG_ControlByCpuLowPowerMode(clock_lpcg_t name, uint8_t domainId,
                                           clock_level_t level0, clock_level_t level1)
```

Set this clock works in CPU Low Power Mode.

Note: When LOCK_MODE bit is set, control mode can not be changed until next system reset.

Parameters

- name – Clock gate name, see `clock_lpcg_t`.
- domainId – Domains that on the whitelist can change this clock.
- level1 (level0,) – Depend level of this clock.

```
static inline void CLOCK_LPCG_SetCurrentClockLevel(clock_lpcg_t name, clock_level_t level)
```

Set clock depend level for current accessing domain.

Note: This setting only take effects in CPU Low Power Mode.

Parameters

- name – Clock gate name, see `clock_lpcg_t`.
- level – Depend level of this clock.

```
static inline void CLOCK_LPCG_ControlByDomainMode(clock_lpcg_t name, uint8_t domainId)
```

Set this clock works in Domain Mode.

Note: When LOCK_MODE bit is set, control mode can not be changed until next system reset.

Parameters

- name – Clock gate name, see `clock_lpcg_t`.
- domainId – Domains that on the whitelist can change this clock.

static inline void CLOCK_SetClockOutput1(*clock_output1_selection_t* selection, uint32_t divider)

Set the clock source and the divider of the clock output1.

param selection The clock source to be output, please refer to *clock_output1_selection_t*.

param divider The divider of the output clock signal.

static inline void CLOCK_SetClockOutput2(*clock_output2_selection_t* selection, uint32_t divider)

Set the clock source and the divider of the clock output2.

param selection The clock source to be output, please refer to *clock_output2_selection_t*.

param divider The divider of the output clock signal.

static inline uint32_t CLOCK_GetClockOutCLKO1Freq(void)

Get the frequency of clock output1 clock signal.

Returns

The frequency of clock output1 clock signal.

static inline uint32_t CLOCK_GetClockOutClkO2Freq(void)

Get the frequency of clock output2 clock signal.

Returns

The frequency of clock output2 clock signal.

bool clockOff

Turn off the clock.

uint16_t resetDiv

resetDiv + 1 should be common multiple of all dividers, valid range 0 ~ 255.

uint8_t div0

Divide root clock by div0 + 1, valid range: 0 ~ 15.

clock_pll_post_div_t postDivider

Post divider.

uint32_t loopDivider

PLL loop divider. Valid range: 104-208.

uint8_t loopDivider

PLL loop divider. 0 - Fout=Fref*20; 1 - Fout=Fref*22

uint8_t src

Pll clock source, reference *_clock_pll_clk_src*

uint16_t stop

Spread spectrum stop value to get frequency change.

uint16_t step

Spread spectrum step value to get frequency change step.

uint32_t mfd

Denominator of spread spectrum

clock_pll_ss_config_t *ss

Spread spectrum parameter, it can be NULL, if ssEnable is set to false

bool ssEnable

Enable spread spectrum flag

bool pllDiv2En

Enable Sys Pll1 divide-by-2 clock or not.

bool pllDiv5En

Enable Sys PLL1 divide-by-5 clock or not.

clock_pll_ss_config_t *ss

Spread spectrum parameter, it can be NULL, if ssEnable is set to false

bool ssEnable

Enable spread spectrum flag

uint8_t loopDivider

PLL loop divider. Valid range for DIV_SELECT divider value: 27~54.

uint8_t postDivider

Divider after the PLL, 0x0=divided by 1, 0x1=divided by 2, 0x2=divided by 4, 0x3=divided by 8, 0x4=divided by 16, 0x5=divided by 32.

uint32_t numerator

30 bit numerator of fractional loop divider.

uint32_t denominator

30 bit denominator of fractional loop divider

clock_pll_ss_config_t *ss

Spread spectrum parameter, it can be NULL, if ssEnable is set to false

bool ssEnable

Enable spread spectrum flag

uint8_t loopDivider

PLL loop divider.

uint32_t numerator

30 bit numerator of fractional loop divider.

uint32_t denominator

30 bit denominator of fractional loop divider

clock_pll_ss_config_t *ss

Spread spectrum parameter, it can be NULL, if ssEnable is set to false

bool ssEnable

Enable spread spectrum flag

bool enableClkOutput

Power on and enable PLL clock output for ENET0 (ref_enetpll0).

bool enableClkOutput25M

Power on and enable PLL clock output for ENET2 (ref_enetpll2).

uint8_t loopDivider

Controls the frequency of the ENET0 reference clock. b00 25MHz b01 50MHz b10 100MHz (not 50% duty cycle) b11 125MHz

uint8_t src

Pll clock source, reference _clock_pll_clk_src

bool enableClkOutput1

Power on and enable PLL clock output for ENET1 (ref_enetpll1).

uint8_t loopDivider1

Controls the frequency of the ENET1 reference clock. b00 25MHz b01 50MHz b10 100MHz (not 50% duty cycle) b11 125MHz

bool clockOff

uint8_t mux

See clock_root_mux_source_t for details.

uint8_t div

it's the actual divider

uint8_t grade

Indicate speed grade for each SetPoint

bool clockOff

uint8_t mux

See clock_root_mux_source_t for details.

uint8_t div

it's the actual divider

FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL

Configure whether driver controls clock.

When set to 0, peripheral drivers will enable clock in initialize function and disable clock in de-initialize function. When set to 1, peripheral driver will not control the clock, application could control the clock out of the driver.

Note: All drivers share this feature switcher. If it is set to 1, application should handle clock enable and disable for all drivers.

struct _clock_group_config

#include <fsl_clock.h> The structure used to configure clock group.

struct _clock_arm_pll_config

#include <fsl_clock.h> PLL configuration for ARM.

The output clock frequency is:

$F_{out} = F_{in} * \text{loopDivider} / (2 * \text{postDivider})$.

F_{in} is always 24MHz.

struct _clock_usb_pll_config

#include <fsl_clock.h> PLL configuration for USB.

struct _clock_pll_ss_config

#include <fsl_clock.h> Spread specturm configure Pll.

struct _clock_sys_pll2_config

#include <fsl_clock.h> PLL configure for Sys Pll2.

struct _clock_sys_pll1_config

#include <fsl_clock.h> PLL configure for Sys Pll1.

struct _clock_audio_pll_config

#include <fsl_clock.h> PLL configuration for AUDIO and VIDEO.

struct _clock_audio_pll_gpc_config

#include <fsl_clock.h> PLL configuration fro AUDIO PLL, SYSTEM PLL1 and VIDEO PLL.

struct _clock_enet_pll_config

#include <fsl_clock.h> PLL configuration for ENET.

```
struct _clock_root_config_t
    #include <fsl_clock.h> Clock root configuration.

struct _clock_root_setpoint_config_t
    #include <fsl_clock.h> Clock root configuration in SetPoint Mode.
```

2.30 MIPI CSI2 RX: MIPI CSI2 RX Driver

```
FSL_CSI2RX_DRIVER_VERSION
    CSI2RX driver version.

enum _csi2rx_data_lane
    CSI2RX data lanes.

    Values:

    enumerator kCSI2RX_DataLane0
        Data lane 0.

    enumerator kCSI2RX_DataLane1
        Data lane 1.

    enumerator kCSI2RX_DataLane2
        Data lane 2.

    enumerator kCSI2RX_DataLane3
        Data lane 3.

enum _csi2rx_payload
    CSI2RX payload type.

    Values:

    enumerator kCSI2RX_PayloadGroup0Null
        NULL.

    enumerator kCSI2RX_PayloadGroup0Blank
        Blank.

    enumerator kCSI2RX_PayloadGroup0Embedded
        Embedded.

    enumerator kCSI2RX_PayloadGroup0YUV420_8Bit
        Legacy YUV420 8 bit.

    enumerator kCSI2RX_PayloadGroup0YUV422_8Bit
        YUV422 8 bit.

    enumerator kCSI2RX_PayloadGroup0YUV422_10Bit
        YUV422 10 bit.

    enumerator kCSI2RX_PayloadGroup0RGB444
        RGB444.

    enumerator kCSI2RX_PayloadGroup0RGB555
        RGB555.

    enumerator kCSI2RX_PayloadGroup0RGB565
        RGB565.
```


enumerator kCSI2RX_PayloadGroup0RGB666
RGB666.

enumerator kCSI2RX_PayloadGroup0RGB888
RGB888.

enumerator kCSI2RX_PayloadGroup0Raw6
Raw 6.

enumerator kCSI2RX_PayloadGroup0Raw7
Raw 7.

enumerator kCSI2RX_PayloadGroup0Raw8
Raw 8.

enumerator kCSI2RX_PayloadGroup0Raw10
Raw 10.

enumerator kCSI2RX_PayloadGroup0Raw12
Raw 12.

enumerator kCSI2RX_PayloadGroup0Raw14
Raw 14.

enumerator kCSI2RX_PayloadGroup1UserDefined1
User defined 8-bit data type 1, 0x30.

enumerator kCSI2RX_PayloadGroup1UserDefined2
User defined 8-bit data type 2, 0x31.

enumerator kCSI2RX_PayloadGroup1UserDefined3
User defined 8-bit data type 3, 0x32.

enumerator kCSI2RX_PayloadGroup1UserDefined4
User defined 8-bit data type 4, 0x33.

enumerator kCSI2RX_PayloadGroup1UserDefined5
User defined 8-bit data type 5, 0x34.

enumerator kCSI2RX_PayloadGroup1UserDefined6
User defined 8-bit data type 6, 0x35.

enumerator kCSI2RX_PayloadGroup1UserDefined7
User defined 8-bit data type 7, 0x36.

enumerator kCSI2RX_PayloadGroup1UserDefined8
User defined 8-bit data type 8, 0x37.

enum _csi2rx_bit_error
MIPI CSI2RX bit errors.

Values:

enumerator kCSI2RX_BitErrorEccTwoBit
ECC two bit error has occurred.

enumerator kCSI2RX_BitErrorEccOneBit
ECC one bit error has occurred.

enum _csi2rx_ppi_error
MIPI CSI2RX PPI error types.

Values:

enumerator kCSI2RX_PpiErrorSotHs
CSI2RX DPHY PPI error ErrSotHS.

enumerator kCSI2RX_PpiErrorSotSyncHs
CSI2RX DPHY PPI error ErrSotSync_HS.

enumerator kCSI2RX_PpiErrorEsc
CSI2RX DPHY PPI error ErrEsc.

enumerator kCSI2RX_PpiErrorSyncEsc
CSI2RX DPHY PPI error ErrSyncEsc.

enumerator kCSI2RX_PpiErrorControl
CSI2RX DPHY PPI error ErrControl.

enum _csi2rx_interrupt
MIPI CSI2RX interrupt.

Values:

enumerator kCSI2RX_InterruptCrcError

enumerator kCSI2RX_InterruptEccOneBitError

enumerator kCSI2RX_InterruptEccTwoBitError

enumerator kCSI2RX_InterruptUlpsStatusChange

enumerator kCSI2RX_InterruptErrorSotHs

enumerator kCSI2RX_InterruptErrorSotSyncHs

enumerator kCSI2RX_InterruptErrorEsc

enumerator kCSI2RX_InterruptErrorSyncEsc

enumerator kCSI2RX_InterruptErrorControl

enum _csi2rx_ulps_status
MIPI CSI2RX D-PHY ULPS state.

Values:

enumerator kCSI2RX_ClockLaneUlps
Clock lane is in ULPS state.

enumerator kCSI2RX_DataLane0Ulps
Data lane 0 is in ULPS state.

enumerator kCSI2RX_DataLane1Ulps
Data lane 1 is in ULPS state.

enumerator kCSI2RX_DataLane2Ulps
Data lane 2 is in ULPS state.

enumerator kCSI2RX_DataLane3Ulps
Data lane 3 is in ULPS state.

enumerator kCSI2RX_ClockLaneMark
Clock lane is in mark state.

enumerator kCSI2RX_DataLane0Mark
Data lane 0 is in mark state.

enumerator kCSI2RX_DataLane1Mark

Data lane 1 is in mark state.

enumerator kCSI2RX_DataLane2Mark

Data lane 2 is in mark state.

enumerator kCSI2RX_DataLane3Mark

Data lane 3 is in mark state.

typedef struct *_csi2rx_config* csi2rx_config_t
CSI2RX configuration.

typedef enum *_csi2rx_ppi_error* csi2rx_ppi_error_t
MIPI CSI2RX PPI error types.

void CSI2RX_Init(MIPI_CSI2RX_Type *base, const *csi2rx_config_t* *config)
Enables and configures the CSI2RX peripheral module.

Parameters

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

void CSI2RX_Deinit(MIPI_CSI2RX_Type *base)
Disables the CSI2RX peripheral module.

Parameters

- base – CSI2RX peripheral address.

static inline uint32_t CSI2RX_GetBitError(MIPI_CSI2RX_Type *base)
Gets the MIPI CSI2RX bit error status.

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

Example:

```
uint32_t bitError;  
uint32_t bitErrorPosition;  
  
bitError = CSI2RX_GetBitError(MIPI_CSI2RX);  
  
if (kCSI2RX_BitErrorEccTwoBit & bitError)  
{  
    Two bits error;  
}  
else if (kCSI2RX_BitErrorEccOneBit & bitError)  
{  
    One bits error;  
    bitErrorPosition = CSI2RX_GetEccBitErrorPosition(bitError);  
}
```

Parameters

- base – CSI2RX peripheral address.

Returns

The RX bit error status.

```
static inline uint32_t CSI2RX_GetEccBitErrorPosition(uint32_t bitError)
```

Get ECC one bit error bit position.

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

Parameters

- bitError – The bit error returned by CSI2RX_GetBitError.

Returns

The position of error bit.

```
static inline uint32_t CSI2RX_GetUlpsStatus(MIPI_CSI2RX_Type *base)
```

Gets the MIPI CSI2RX D-PHY ULPS status.

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

```
uint32_t status = CSI2RX_GetUlpsStatus(MIPI_CSI2RX);

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

Parameters

- base – CSI2RX peripheral address.

Returns

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

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

Gets the MIPI CSI2RX D-PHY PPI error lanes.

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

```
uint32_t errorDataLanes = CSI2RX_GetPpiErrorDataLanes(MIPI_CSI2RX, kCSI2RX_
↪PpiErrorSotHS);

if (kCSI2RX_DataLane0 & errorDataLanes)
{
    ErrSotHS occurred on data lane 0.
}

if (kCSI2RX_DataLane1 & errorDataLanes)
{
    ErrSotHS occurred on data lane 1.
}
```

Parameters

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

Returns

The data lane mask that error errorType occurred.

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

Enable the MIPI CSI2RX interrupts.

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

```
CSI2RX_EnableInterrupts(MIPI_CSI2RX, kCSI2RX_InterruptEccOneBitError | kCSI2RX_InterruptEccTwoBitError);
```

Parameters

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

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

Disable the MIPI CSI2RX interrupts.

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

```
CSI2RX_DisableInterrupts(MIPI_CSI2RX, kCSI2RX_InterruptEccOneBitError | kCSI2RX_InterruptEccTwoBitError);
```

Parameters

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

`static inline uint32_t CSI2RX_GetInterruptStatus(MIPI_CSI2RX_Type *base)`

Get the MIPI CSI2RX interrupt status.

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

Parameters

- `base` – CSI2RX peripheral address.

Returns

OR'ed value of `_csi2rx_interrupt`.

`CSI2RX_REG_CFG_NUM_LANES(base)`

`CSI2RX_REG_CFG_DISABLE_DATA_LANES(base)`

`CSI2RX_REG_BIT_ERR(base)`

`CSI2RX_REG_IRQ_STATUS(base)`

`CSI2RX_REG_IRQ_MASK(base)`

`CSI2RX_REG_ULPS_STATUS(base)`

`CSI2RX_REG_PPI_ERRSOT_HS(base)`

`CSI2RX_REG_PPI_ERRSOTSYNC_HS(base)`

`CSI2RX_REG_PPI_ERRESC(base)`

`CSI2RX_REG_PPI_ERRSYNCESC(base)`

`CSI2RX_REG_PPI_ERRCONTROL(base)`

`CSI2RX_REG_CFG_DISABLE_PAYLOAD_0(base)`

```

CSI2RX_REG_CFG_DISABLE_PAYLOAD_1(base)
CSI2RX_REG_CFG_IGNORE_VC(base)
CSI2RX_REG_CFG_VID_VC(base)
CSI2RX_REG_CFG_VID_P_FIFO_SEND_LEVEL(base)
CSI2RX_REG_CFG_VID_VSYNC(base)
CSI2RX_REG_CFG_VID_HSYNC_FP(base)
CSI2RX_REG_CFG_VID_HSYNC(base)
CSI2RX_REG_CFG_VID_HSYNC_BP(base)
MIPI_CSI2RX_CSI2RX_CFG_NUM_LANES_csi2rx_cfg_num_lanes_MASK
MIPI_CSI2RX_CSI2RX_IRQ_MASK_csi2rx_irq_mask_MASK
struct _csi2rx_config
    #include <fsl_mipi_csi2rx.h> CSI2RX configuration.

```

Public Members

```

uint8_t laneNum
    Number of active lanes used for receiving data.

uint8_t tHsSettle_EscClk
    Number of rx_clk_esc clock periods for T_HS_SETTLE. The T_HS_SETTLE should be in
    the range of 85ns + 6UI to 145ns + 10UI.

```

2.31 CSI: CMOS Sensor Interface

```
status_t CSI_Init(CSI_Type *base, const csi_config_t *config)
```

Initialize the CSI.

This function enables the CSI peripheral clock, and resets the CSI registers.

Parameters

- base – CSI peripheral base address.
- config – Pointer to the configuration structure.

Return values

- kStatus_Success – Initialize successfully.
- kStatus_InvalidArgument – Initialize failed because of invalid argument.

```
void CSI_Deinit(CSI_Type *base)
```

De-initialize the CSI.

This function disables the CSI peripheral clock.

Parameters

- base – CSI peripheral base address.

void CSI_Reset(CSI_Type *base)

Reset the CSI.

This function resets the CSI peripheral registers to default status.

Parameters

- base – CSI peripheral base address.

void CSI_GetDefaultConfig(*csi_config_t* *config)

Get the default configuration for to initialize the CSI.

The default configuration value is:

```
config->width = 320U;
config->height = 240U;
config->polarityFlags = kCSI_HsyncActiveHigh | kCSI_DataLatchOnRisingEdge;
config->bytesPerPixel = 2U;
config->linePitch_Bytes = 320U * 2U;
config->workMode = kCSI_GatedClockMode;
config->dataBus = kCSI_DataBus8Bit;
config->useExtVsync = true;
```

Parameters

- config – Pointer to the CSI configuration.

void CSI_ClearFifo(CSI_Type *base, *csi_fifo_t* fifo)

Clear the CSI FIFO.

This function clears the CSI FIFO.

Parameters

- base – CSI peripheral base address.
- fifo – The FIFO to clear.

void CSI_ReflashFifoDma(CSI_Type *base, *csi_fifo_t* fifo)

Reflash the CSI FIFO DMA.

This function reflashes the CSI FIFO DMA.

For RXFIFO, there are two frame buffers. When the CSI module started, it saves the frames to frame buffer 0 then frame buffer 1, the two buffers will be written by turns. After reflash DMA using this function, the CSI is reset to save frame to buffer 0.

Parameters

- base – CSI peripheral base address.
- fifo – The FIFO DMA to reflash.

void CSI_EnableFifoDmaRequest(CSI_Type *base, *csi_fifo_t* fifo, bool enable)

Enable or disable the CSI FIFO DMA request.

Parameters

- base – CSI peripheral base address.
- fifo – The FIFO DMA request to enable or disable.
- enable – True to enable, false to disable.

static inline void CSI_Start(CSI_Type *base)

Start to receive data.

Parameters

- base – CSI peripheral base address.

static inline void CSI_Stop(CSI_Type *base)

Stop to receiving data.

Parameters

- base – CSI peripheral base address.

void CSI_SetRxBufferAddr(CSI_Type *base, uint8_t index, uint32_t addr)

Set the RX frame buffer address.

Parameters

- base – CSI peripheral base address.
- index – Buffer index.
- addr – Frame buffer address to set.

void CSI_EnableInterrupts(CSI_Type *base, uint32_t mask)

Enables CSI interrupt requests.

Parameters

- base – CSI peripheral base address.
- mask – The interrupts to enable, pass in as OR'ed value of _csi_interrupt_enable.

void CSI_DisableInterrupts(CSI_Type *base, uint32_t mask)

Disable CSI interrupt requests.

Parameters

- base – CSI peripheral base address.
- mask – The interrupts to disable, pass in as OR'ed value of _csi_interrupt_enable.

static inline uint32_t CSI_GetStatusFlags(CSI_Type *base)

Gets the CSI status flags.

Parameters

- base – CSI peripheral base address.

Returns

status flag, it is OR'ed value of _csi_flags.

static inline void CSI_ClearStatusFlags(CSI_Type *base, uint32_t statusMask)

Clears the CSI status flag.

The flags to clear are passed in as OR'ed value of _csi_flags. The following flags are cleared automatically by hardware:

- kCSI_RxFifoFullFlag,
- kCSI_StatFifoFullFlag,
- kCSI_Field0PresentFlag,
- kCSI_Field1PresentFlag,
- kCSI_RxFifoDataReadyFlag,

Parameters

- base – CSI peripheral base address.
- statusMask – The status flags mask, OR'ed value of _csi_flags.

status_t CSI_TransferCreateHandle(CSI_Type *base, *csi_handle_t* *handle, *csi_transfer_callback_t* callback, void *userData)

Initializes the CSI handle.

This function initializes CSI handle, it should be called before any other CSI transactional functions.

Parameters

- base – CSI peripheral base address.
- handle – Pointer to the handle structure.
- callback – Callback function for CSI transfer.
- userData – Callback function parameter.

Return values

kStatus_Success – Handle created successfully.

status_t CSI_TransferStart(CSI_Type *base, *csi_handle_t* *handle)

Start the transfer using transactional functions.

When the empty frame buffers have been submit to CSI driver using function CSI_TransferSubmitEmptyBuffer, user could call this function to start the transfer. The incoming frame will be saved to the empty frame buffer, and user could be optionally notified through callback function.

Parameters

- base – CSI peripheral base address.
- handle – Pointer to the handle structure.

Return values

- kStatus_Success – Started successfully.
- kStatus_CSI_NoEmptyBuffer – Could not start because no empty frame buffer in queue.

status_t CSI_TransferStop(CSI_Type *base, *csi_handle_t* *handle)

Stop the transfer using transactional functions.

The driver does not clean the full frame buffers in queue. In other words, after calling this function, user still could get the full frame buffers in queue using function CSI_TransferGetFullBuffer.

Parameters

- base – CSI peripheral base address.
- handle – Pointer to the handle structure.

Return values

kStatus_Success – Stopped successfully.

status_t CSI_TransferSubmitEmptyBuffer(CSI_Type *base, *csi_handle_t* *handle, uint32_t frameBuffer)

Submit empty frame buffer to queue.

This function could be called before CSI_TransferStart or after CSI_TransferStart. If there is no room in queue to store the empty frame buffer, this function returns error.

Parameters

- base – CSI peripheral base address.
- handle – Pointer to the handle structure.

- `frameBuffer` – Empty frame buffer to submit.

Return values

- `kStatus__Success` – Started successfully.
- `kStatus__CSI_QueueFull` – Could not submit because there is no room in queue.

`status_t` `CSI_TransferGetFullBuffer(CSI_Type *base, csi_handle_t *handle, uint32_t *frameBuffer)`
Get one full frame buffer from queue.

After the transfer started using function `CSI_TransferStart`, the incoming frames will be saved to the empty frame buffers in queue. This function gets the full-filled frame buffer from the queue. If there is no full frame buffer in queue, this function returns error.

Parameters

- `base` – CSI peripheral base address.
- `handle` – Pointer to the handle structure.
- `frameBuffer` – Full frame buffer.

Return values

- `kStatus__Success` – Started successfully.
- `kStatus__CSI_NoFullBuffer` – There is no full frame buffer in queue.

`void` `CSI_TransferHandleIRQ(CSI_Type *base, csi_handle_t *handle)`
CSI IRQ handle function.

This function handles the CSI IRQ request to work with CSI driver transactional APIs.

Parameters

- `base` – CSI peripheral base address.
- `handle` – CSI handle pointer.

`FSL_CSI_DRIVER_VERSION`

Error codes for the CSI driver.

Values:

enumerator `kStatus__CSI_NoEmptyBuffer`
No empty frame buffer in queue to load to CSI.

enumerator `kStatus__CSI_NoFullBuffer`
No full frame buffer in queue to read out.

enumerator `kStatus__CSI_QueueFull`
Queue is full, no room to save new empty buffer.

enumerator `kStatus__CSI_FrameDone`
New frame received and saved to queue.

enum `_csi_work_mode`
CSI work mode.

The CCIR656 interlace mode is not supported currently.

Values:

enumerator `kCSI_GatedClockMode`
HSYNC, VSYNC, and PIXCLK signals are used.

enumerator kCSI_NonGatedClockMode

VSYNC, and PIXCLK signals are used.

enumerator kCSI_CCIR656ProgressiveMode

CCIR656 progressive mode.

enum __csi_data_bus

CSI data bus width.

Values:

enumerator kCSI_DataBus8Bit

8-bit data bus.

enumerator kCSI_DataBus16Bit

16-bit data bus.

enumerator kCSI_DataBus24Bit

24-bit data bus.

enum __csi_polarity_flags

CSI signal polarity.

Values:

enumerator kCSI_HsyncActiveLow

HSYNC is active low.

enumerator kCSI_HsyncActiveHigh

HSYNC is active high.

enumerator kCSI_DataLatchOnRisingEdge

Pixel data latched at rising edge of pixel clock.

enumerator kCSI_DataLatchOnFallingEdge

Pixel data latched at falling edge of pixel clock.

enumerator kCSI_VsyncActiveHigh

VSYNC is active high.

enumerator kCSI_VsyncActiveLow

VSYNC is active low.

enum __csi_fifo

The CSI FIFO, used for FIFO operation.

Values:

enumerator kCSI_RxFifo

RXFIFO.

enumerator kCSI_StatFifo

STAT FIFO.

enumerator kCSI_AllFifo

Both RXFIFO and STAT FIFO.

enum __csi_interrupt_enable

CSI feature interrupt source.

Values:

enumerator kCSI_EndOfFrameInterruptEnable

End of frame interrupt enable.

enumerator kCSI_ChangeOfFieldInterruptEnable

Change of field interrupt enable.

enumerator kCSI_StatFifoOverrunInterruptEnable

STAT FIFO overrun interrupt enable.

enumerator kCSI_RxFifoOverrunInterruptEnable

RXFIFO overrun interrupt enable.

enumerator kCSI_StatFifoDmaDoneInterruptEnable

STAT FIFO DMA done interrupt enable.

enumerator kCSI_StatFifoFullInterruptEnable

STAT FIFO full interrupt enable.

enumerator kCSI_RxBuffer1DmaDoneInterruptEnable

RX frame buffer 1 DMA transfer done.

enumerator kCSI_RxBuffer0DmaDoneInterruptEnable

RX frame buffer 0 DMA transfer done.

enumerator kCSI_RxFifoFullInterruptEnable

RXFIFO full interrupt enable.

enumerator kCSI_StartOfFrameInterruptEnable

Start of frame (SOF) interrupt enable.

enumerator kCSI_EccErrorInterruptEnable

ECC error detection interrupt enable.

enumerator kCSI_AhbResErrorInterruptEnable

AHB response Error interrupt enable.

enumerator kCSI_BaseAddrChangeErrorInterruptEnable

The DMA output buffer base address changes before DMA completed.

enumerator kCSI_Field0DoneInterruptEnable

Field 0 done interrupt enable.

enumerator kCSI_Field1DoneInterruptEnable

Field 1 done interrupt enable.

enum _csi_flags

CSI status flags.

The following status register flags can be cleared:

- kCSI_EccErrorFlag
- kCSI_AhbResErrorFlag
- kCSI_ChangeOfFieldFlag
- kCSI_StartOfFrameFlag
- kCSI_EndOfFrameFlag
- kCSI_RxBuffer1DmaDoneFlag
- kCSI_RxBuffer0DmaDoneFlag
- kCSI_StatFifoDmaDoneFlag
- kCSI_StatFifoOverrunFlag
- kCSI_RxFifoOverrunFlag
- kCSI_Field0DoneFlag

- kCSI_Field1DoneFlag
- kCSI_BaseAddrChangeErrorFlag

Values:

enumerator kCSI_RxFifoDataReadyFlag
RXFIFO data ready.

enumerator kCSI_EccErrorFlag
ECC error detected.

enumerator kCSI_AhbResErrorFlag
Hresponse (AHB bus response) Error.

enumerator kCSI_ChangeOfFieldFlag
Change of field.

enumerator kCSI_Field0PresentFlag
Field 0 present in CCIR mode.

enumerator kCSI_Field1PresentFlag
Field 1 present in CCIR mode.

enumerator kCSI_StartOfFrameFlag
Start of frame (SOF) detected.

enumerator kCSI_EndOfFrameFlag
End of frame (EOF) detected.

enumerator kCSI_RxFifoFullFlag
RXFIFO full (Number of data reaches trigger level).

enumerator kCSI_RxBuffer1DmaDoneFlag
RX frame buffer 1 DMA transfer done.

enumerator kCSI_RxBuffer0DmaDoneFlag
RX frame buffer 0 DMA transfer done.

enumerator kCSI_StatFifoFullFlag
STAT FIFO full (Reach trigger level).

enumerator kCSI_StatFifoDmaDoneFlag
STAT FIFO DMA transfer done.

enumerator kCSI_StatFifoOverrunFlag
STAT FIFO overrun.

enumerator kCSI_RxFifoOverrunFlag
RXFIFO overrun.

enumerator kCSI_Field0DoneFlag
Field 0 transfer done.

enumerator kCSI_Field1DoneFlag
Field 1 transfer done.

enumerator kCSI_BaseAddrChangeErrorFlag
The DMA output buffer base address changes before DMA completed.

typedef enum *_csi_work_mode* csi_work_mode_t
CSI work mode.

The CCIR656 interlace mode is not supported currently.

typedef enum *_csi_data_bus* csi_data_bus_t

CSI data bus width.

typedef struct *_csi_config* csi_config_t

Configuration to initialize the CSI module.

typedef enum *_csi_fifo* csi_fifo_t

The CSI FIFO, used for FIFO operation.

typedef struct *_csi_handle* csi_handle_t

typedef void (*csi_transfer_callback_t)(CSI_Type *base, *csi_handle_t* *handle, *status_t* status, void *userData)

CSI transfer callback function.

When a new frame is received and saved to the frame buffer queue, the callback is called and the pass the status kStatus_CSI_FrameDone to upper layer.

CSI_REG_CR1(base)

CSI_REG_CR2(base)

CSI_REG_CR3(base)

CSI_REG_CR18(base)

CSI_REG_SR(base)

CSI_REG_DMASA_FB1(base)

CSI_REG_DMASA_FB2(base)

CSI_REG_IMAG_PARA(base)

CSI_REG_FBUF_PARA(base)

CSI_DRIVER_QUEUE_SIZE

Size of the frame buffer queue used in CSI transactional function.

CSI_DRIVER_FRAG_MODE

Enable fragment capture function or not.

CSI_CR1_INT_EN_MASK

CSI_CR3_INT_EN_MASK

CSI_CR18_INT_EN_MASK

struct *_csi_config*

#include <fsl_csi.h> Configuration to initialize the CSI module.

Public Members

uint16_t width

Pixels of the input frame.

uint16_t height

Lines of the input frame.

uint32_t polarityFlags

Timing signal polarity flags, OR'ed value of *_csi_polarity_flags*.

uint8_t bytesPerPixel

Bytes per pixel, valid values are:

- 2: Used for RGB565, YUV422, and so on.
- 4: Used for XRGB8888, XYUV444, and so on.

uint16_t linePitch_Bytes

Frame buffer line pitch, must be 8-byte aligned.

csi_work_mode_t workMode

CSI work mode.

csi_data_bus_t dataBus

Data bus width.

bool useExtVsync

In CCIR656 progressive mode, set true to use external VSYNC signal, set false to use internal VSYNC signal decoded from SOF.

struct buf_queue_t

#include <fsl_csi.h>

struct _csi_handle

#include <fsl_csi.h> CSI handle structure.

Please see the user guide for the details of the CSI driver queue mechanism.

Public Members

volatile uint8_t activeBufferNum

How many frame buffers are in progress currently.

volatile uint8_t dmaDoneBufferIdx

Index of the current full-filled framebuffer.

volatile bool transferStarted

User has called CSI_TransferStart to start frame receiving.

csi_transfer_callback_t callback

Callback function.

void *userData

CSI callback function parameter.

2.32 DAC12: 12-bit Digital-to-Analog Converter Driver

void DAC12_GetHardwareInfo(DAC_Type *base, dac12_hardware_info_t *info)

Get hardware information about this module.

Parameters

- base – DAC12 peripheral base address.
- info – Pointer to info structure, see to dac12_hardware_info_t.

void DAC12_Init(DAC_Type *base, const dac12_config_t *config)

Initialize the DAC12 module.

Parameters

- base – DAC12 peripheral base address.

- config – Pointer to configuration structure, see to `dac12_config_t`.

`void DAC12_GetDefaultConfig(dac12_config_t *config)`

Initializes the DAC12 user configuration structure.

This function initializes the user configuration structure to a default value. The default values are:

```
config->fifoWatermarkLevel = 0U;
config->fifoWorkMode = kDAC12_FIFODisabled;
config->referenceVoltageSource = kDAC12_ReferenceVoltageSourceAlt1;
config->fifoTriggerMode = kDAC12_FIFOTriggerByHardwareMode;
config->referenceCurrentSource = kDAC12_ReferenceCurrentSourceAlt0;
config->speedMode = kDAC12_SpeedLowMode;
config->speedMode = false;
config->currentReferenceInternalTrimValue = 0x4;
```

Parameters

- config – Pointer to the configuration structure. See “`dac12_config_t`”.

`void DAC12_Deinit(DAC_Type *base)`

De-initialize the DAC12 module.

Parameters

- base – DAC12 peripheral base address.

`static inline void DAC12_Enable(DAC_Type *base, bool enable)`

Enable the DAC12's converter or not.

Parameters

- base – DAC12 peripheral base address.
- enable – Enable the DAC12's converter or not.

`static inline void DAC12_ResetConfig(DAC_Type *base)`

Reset all internal logic and registers.

Parameters

- base – DAC12 peripheral base address.

`static inline void DAC12_ResetFIFO(DAC_Type *base)`

Reset the FIFO pointers.

FIFO pointers should only be reset when the DAC12 is disabled. This function can be used to configure both pointers to the same address to reset the FIFO as empty.

Parameters

- base – DAC12 peripheral base address.

`static inline uint32_t DAC12_GetStatusFlags(DAC_Type *base)`

Get status flags.

Parameters

- base – DAC12 peripheral base address.

Returns

Mask of current status flags. See to `_dac12_status_flags`.

`static inline void DAC12_ClearStatusFlags(DAC_Type *base, uint32_t flags)`

Clear status flags.

Note: Not all the flags can be cleared by this API. Several flags need special condition to clear them according to target chip's reference manual document.

Parameters

- base – DAC12 peripheral base address.
- flags – Mask of status flags to be cleared. See to `_dac12_status_flags`.

```
static inline void DAC12_EnableInterrupts(DAC_Type *base, uint32_t mask)
```

Enable interrupts.

Parameters

- base – DAC12 peripheral base address.
- mask – Mask value of interrupts to be enabled. See to `_dac12_interrupt_enable`.

```
static inline void DAC12_DisableInterrupts(DAC_Type *base, uint32_t mask)
```

Disable interrupts.

Parameters

- base – DAC12 peripheral base address.
- mask – Mask value of interrupts to be disabled. See to `_dac12_interrupt_enable`.

```
static inline void DAC12_EnableDMA(DAC_Type *base, bool enable)
```

Enable DMA or not.

When DMA is enabled, the DMA request will be generated by original interrupts. The interrupts will not be presented on this module at the same time.

```
static inline void DAC12_SetData(DAC_Type *base, uint32_t value)
```

Set data into the entry of FIFO buffer.

When the DAC FIFO is disabled, and the one entry buffer is enabled, the DAC converts the data in the buffer to analog output voltage. Any write to the DATA register will replace the data in the buffer and push data to analog conversion without trigger support. When the DAC FIFO is enabled. Writing data would increase the write pointer of FIFO. Also, the data would be restored into the FIFO buffer.

Parameters

- base – DAC12 peripheral base address.
- value – Setting value into FIFO buffer.

```
static inline void DAC12_DoSoftwareTrigger(DAC_Type *base)
```

Do trigger the FIFO by software.

When the DAC FIFO is enabled, and software trigger is used. Doing trigger would increase the read pointer, and the data in the entry pointed by read pointer would be converted as new output.

Parameters

- base – DAC12 peripheral base address.

```
static inline uint32_t DAC12_GetFIFOReadPointer(DAC_Type *base)
```

Get the current read pointer of FIFO.

Parameters

- base – DAC12 peripheral base address.

Returns

Read pointer index of FIFO buffer.

static inline uint32_t DAC12_GetFIFOWritePointer(DAC_Type *base)

Get the current write pointer of FIFO.

Parameters

- base – DAC12 peripheral base address.

Returns

Write pointer index of FIFO buffer

FSL_DAC12_DRIVER_VERSION

DAC12 driver version 2.1.2.

enum __dac12_status_flags

DAC12 flags.

Values:

enumerator kDAC12_OverflowFlag

FIFO overflow status flag, which indicates that more data has been written into FIFO than it can hold.

enumerator kDAC12_UnderflowFlag

FIFO underflow status flag, which means that there is a new trigger after the FIFO is nearly empty.

enumerator kDAC12_WatermarkFlag

FIFO watermark status flag, which indicates the remaining FIFO data is less than the watermark setting.

enumerator kDAC12_NearlyEmptyFlag

FIFO nearly empty flag, which means there is only one data remaining in FIFO.

enumerator kDAC12_FullFlag

FIFO full status flag, which means that the FIFO read pointer equals the write pointer, as the write pointer increase.

enum __dac12_interrupt_enable

DAC12 interrupts.

Values:

enumerator kDAC12_UnderOrOverflowInterruptEnable

Underflow and overflow interrupt enable.

enumerator kDAC12_WatermarkInterruptEnable

Watermark interrupt enable.

enumerator kDAC12_NearlyEmptyInterruptEnable

Nearly empty interrupt enable.

enumerator kDAC12_FullInterruptEnable

Full interrupt enable.

enum __dac12_fifo_size_info

DAC12 FIFO size information provided by hardware.

Values:

enumerator kDAC12_FIFOSize2

FIFO depth is 2.

enumerator kDAC12_FIFOSize4

FIFO depth is 4.

enumerator kDAC12__FIFOSize8

FIFO depth is 8.

enumerator kDAC12__FIFOSize16

FIFO depth is 16.

enumerator kDAC12__FIFOSize32

FIFO depth is 32.

enumerator kDAC12__FIFOSize64

FIFO depth is 64.

enumerator kDAC12__FIFOSize128

FIFO depth is 128.

enumerator kDAC12__FIFOSize256

FIFO depth is 256.

enum _dac12_fifo_work_mode

DAC12 FIFO work mode.

Values:

enumerator kDAC12__FIFODisabled

FIFO disabled and only one level buffer is enabled. Any data written from this buffer goes to conversion.

enumerator kDAC12__FIFOWorkAsNormalMode

Data will first read from FIFO to buffer then go to conversion.

enumerator kDAC12__FIFOWorkAsSwingMode

In Swing mode, the FIFO must be set up to be full. In Swing back mode, a trigger changes the read pointer to make it swing between the FIFO Full and Nearly Empty state. That is, the trigger increases the read pointer till FIFO is nearly empty and decreases the read pointer till the FIFO is full.

enum _dac12_reference_voltage_source

DAC12 reference voltage source.

Values:

enumerator kDAC12__ReferenceVoltageSourceAlt1

The DAC selects DACREF_1 as the reference voltage.

enumerator kDAC12__ReferenceVoltageSourceAlt2

The DAC selects DACREF_2 as the reference voltage.

enum _dac12_fifo_trigger_mode

DAC12 FIFO trigger mode.

Values:

enumerator kDAC12__FIFOTriggerByHardwareMode

Buffer would be triggered by hardware.

enumerator kDAC12__FIFOTriggerBySoftwareMode

Buffer would be triggered by software.

enum _dac12_reference_current_source

DAC internal reference current source.

Analog module needs reference current to keep working . Such reference current can generated by IP itself, or by on-chip PMC's "reference part". If no current reference be selected, analog module can't working normally ,even when other register can still

be assigned, DAC would waste current but no function. To make the DAC work, either `kDAC12_ReferenceCurrentSourceAltX` should be selected.

Values:

enumerator `kDAC12_ReferenceCurrentSourceDisabled`

None of reference current source is enabled.

enumerator `kDAC12_ReferenceCurrentSourceAlt0`

Use the internal reference current generated by the module itself.

enumerator `kDAC12_ReferenceCurrentSourceAlt1`

Use the ZTC(Zero Temperature Coefficient) reference current generated by on-chip power management module.

enumerator `kDAC12_ReferenceCurrentSourceAlt2`

Use the PTAT(Proportional To Absolution Temperature) reference current generated by power management module.

enum `_dac12_speed_mode`

DAC analog buffer speed mode for conversion.

Values:

enumerator `kDAC12_SpeedLowMode`

Low speed mode.

enumerator `kDAC12_SpeedMiddleMode`

Middle speed mode.

enumerator `kDAC12_SpeedHighMode`

High speed mode.

typedef enum `_dac12_fifo_size_info` `dac12_fifo_size_info_t`

DAC12 FIFO size information provided by hardware.

typedef enum `_dac12_fifo_work_mode` `dac12_fifo_work_mode_t`

DAC12 FIFO work mode.

typedef enum `_dac12_reference_voltage_source` `dac12_reference_voltage_source_t`

DAC12 reference voltage source.

typedef enum `_dac12_fifo_trigger_mode` `dac12_fifo_trigger_mode_t`

DAC12 FIFO trigger mode.

typedef enum `_dac12_reference_current_source` `dac12_reference_current_source_t`

DAC internal reference current source.

Analog module needs reference current to keep working . Such reference current can generated by IP itself, or by on-chip PMC's "reference part". If no current reference be selected, analog module can't working normally ,even when other register can still be assigned, DAC would waste current but no function. To make the DAC work, either `kDAC12_ReferenceCurrentSourceAltX` should be selected.

typedef enum `_dac12_speed_mode` `dac12_speed_mode_t`

DAC analog buffer speed mode for conversion.

typedef struct `_dac12_hardware_info` `dac12_hardware_info_t`

DAC12 hardware information.

`DAC12_CR_W1C_FLAGS_MASK`

Define "write 1 to clear" flags.

DAC12_CR_ALL_FLAGS_MASK

Define all the flag bits in DACx_CR register.

struct __dac12_hardware_info

#include <fsl_dac12.h> DAC12 hardware information.

Public Members

dac12_fifo_size_info_t fifoSizeInfo

The number of words in this device's DAC buffer.

struct dac12_config_t

#include <fsl_dac12.h> DAC12 module configuration.

Actually, the most fields are for FIFO buffer.

Public Members

uint32_t fifoWatermarkLevel

FIFO's watermark, the max value can be the hardware FIFO size.

dac12_fifo_work_mode_t fifoWorkMode

FIFO's work mode about pointers.

dac12_reference_voltage_source_t referenceVoltageSource

Select the reference voltage source.

dac12_reference_current_source_t referenceCurrentSource

Select the trigger mode for FIFO. Select the reference current source.

dac12_speed_mode_t speedMode

Select the speed mode for conversion.

bool enableAnalogBuffer

Enable analog buffer for high drive.

2.33 DCIC: Display Content Integrity Checker

2.34 DCIC

void DCIC_Init(DCIC_Type *base, const *dcic_config_t* *config)

Initializes the DCIC.

This function resets DCIC registers to default value, then set the configurations. This function does not start the DCIC to work, application should call DCIC_DisableRegion to configure regions, then call DCIC_Enable to start the DCIC to work.

Parameters

- base – DCIC peripheral base address.
- config – Pointer to the configuration.

void DCIC_Deinit(DCIC_Type *base)

Deinitialize the DCIC.

Disable the DCIC functions.

Parameters

- base – DCIC peripheral base address.

void DCIC_GetDefaultConfig(*dcic_config_t* *config)

Get the default configuration to initialize DCIC.

The default configuration is:

```
config->polarityFlags = kDCIC_VsyncActiveLow | kDCIC_HsyncActiveLow |
                    kDCIC_DataEnableActiveLow | kDCIC_DriveDataOnFallingClkEdge;
config->enableExternalSignal = false;
config->enableInterrupts = 0;
```

Parameters

- config – Pointer to the configuration.

static inline void DCIC_Enable(DCIC_Type *base, bool enable)

Enable or disable the DCIC module.

Parameters

- base – DCIC peripheral base address.
- enable – Use true to enable, false to disable.

static inline uint32_t DCIC_GetStatusFlags(DCIC_Type *base)

Get status flags.

The flag kDCIC_ErrorInterruptStatus is asserted if any region mismatch flag asserted.

base DCIC peripheral base address.

Returns

Masks of asserted status flags, _DCIC_status_flags.

static inline void DCIC_ClearStatusFlags(DCIC_Type *base, uint32_t mask)

Clear status flags.

The flag kDCIC_ErrorInterruptStatus should be cleared by clearing all asserted region mismatch flags.

base DCIC peripheral base address.

mask Mask of status values that would be cleared, _DCIC_status_flags.

static inline void DCIC_LockInterruptEnabledStatus(DCIC_Type *base)

Lock the interrupt enabled status.

Once this function is called, the interrupt enabled status could not be changed until reset.

Parameters

- base – DCIC peripheral base address.

static inline void DCIC_EnableInterrupts(DCIC_Type *base, uint32_t mask)

Enable interrupts.

Parameters

- base – DCIC peripheral base address.
- mask – Mask of interrupt events that would be enabled. See to “_dcic_interrupt_enable_t”.

```
static inline void DCIC_DisableInterrupts(DCIC_Type *base, uint32_t mask)
```

Disable interrupts.

Parameters

- `base` – DCIC peripheral base address.
- `mask` – Mask of interrupt events that would be disabled. See to “`_dcic_interrupt_enable_t`”.

```
void DCIC_EnableRegion(DCIC_Type *base, uint8_t regionIdx, const dcic_region_config_t *config)
```

Enable the region of interest (ROI) with configuration.

Enable the ROI with configuration. To change the configuration except reference CRC value, the region should be disabled first by `DCIC_DisableRegion`, then call this function again. The reference CRC value could be changed by `DCIC_SetRegionRefCrc` without disabling the region. If the configuration is locked, only the reference CRC value could be changed, the region size and position, enable status could not be changed until reset.

Parameters

- `base` – DCIC peripheral base address.
- `regionIdx` – Region index, from 0 to (`DCIC_REGION_COUNT - 1`).
- `config` – Pointer to the configuration.

```
static inline void DCIC_DisableRegion(DCIC_Type *base, uint8_t regionIdx)
```

Disable the region of interest (ROI).

Parameters

- `base` – DCIC peripheral base address.
- `regionIdx` – Region index, from 0 to (`DCIC_REGION_COUNT - 1`).

```
static inline void DCIC_SetRegionRefCrc(DCIC_Type *base, uint8_t regionIdx, uint32_t crc)
```

Set the reference CRC of interest (ROI).

Parameters

- `base` – DCIC peripheral base address.
- `regionIdx` – Region index, from 0 to (`DCIC_REGION_COUNT - 1`).
- `crc` – The reference CRC value.

```
static inline uint32_t DCIC_GetRegionCalculatedCrc(DCIC_Type *base, uint8_t regionIdx)
```

Get the DCIC calculated CRC.

Parameters

- `base` – DCIC peripheral base address.
- `regionIdx` – Region index, from 0 to (`DCIC_REGION_COUNT - 1`).

Returns

The calculated CRC value.

```
static inline void DCIC_EnableMismatchExternalSignal(DCIC_Type *base, bool enable)
```

Enable or disable output the mismatch external signal.

The mismatch status can be output to external pins. If enabled:

- If `kDCIC_ErrorInterruptStatus` asserted, the output signal frequency is DCIC clock / 16.
- If `kDCIC_ErrorInterruptStatus` not asserted, the output signal frequency is DCIC clock / 4.
- If integrity check is disabled, the signal is idle.

Parameters

- base – DCIC peripheral base address.
- enable – Use true to enable, false to disable.

enum `_DCIC_polarity_flags`

DCIC display signal polarity flags .

Values:

enumerator `kDCIC_VsyncActiveHigh`

VSYNC active high.

enumerator `kDCIC_HsyncActiveHigh`

HSYNC active high.

enumerator `kDCIC_DataEnableActiveHigh`

Data enable line active high.

enumerator `kDCIC_DriveDataOnFallingClkEdge`

Output data on rising clock edge, capture data on falling clock edge.

enumerator `kDCIC_VsyncActiveLow`

VSYNC active low.

enumerator `kDCIC_HsyncActiveLow`

HSYNC active low.

enumerator `kDCIC_DataEnableActiveLow`

Data enable line active low.

enumerator `kDCIC_DriveDataOnRisingClkEdge`

Output data on falling clock edge, capture data on rising clock edge.

enum `_DCIC_status_flags`

Status flags. .

Values:

enumerator `kDCIC_FunctionalInterruptStatus`

Asserted when match results ready.

enumerator `kDCIC_ErrorInterruptStatus`

Asserted when there is a signature mismatch.

enumerator `kDCIC_Region0MismatchStatus`

Region 0 CRC32 value mismatch.

enumerator `kDCIC_Region1MismatchStatus`

Region 1 CRC32 value mismatch.

enumerator `kDCIC_Region2MismatchStatus`

Region 2 CRC32 value mismatch.

enumerator `kDCIC_Region3MismatchStatus`

Region 3 CRC32 value mismatch.

enumerator `kDCIC_Region4MismatchStatus`

Region 4 CRC32 value mismatch.

enumerator `kDCIC_Region5MismatchStatus`

Region 5 CRC32 value mismatch.

enumerator kDCIC_Region6MismatchStatus
Region 6 CRC32 value mismatch.

enumerator kDCIC_Region7MismatchStatus
Region 7 CRC32 value mismatch.

enumerator kDCIC_Region8MismatchStatus
Region 8 CRC32 value mismatch.

enumerator kDCIC_Region9MismatchStatus
Region 9 CRC32 value mismatch.

enumerator kDCIC_Region10MismatchStatus
Region 10 CRC32 value mismatch.

enumerator kDCIC_Region11MismatchStatus
Region 11 CRC32 value mismatch.

enumerator kDCIC_Region12MismatchStatus
Region 12 CRC32 value mismatch.

enumerator kDCIC_Region13MismatchStatus
Region 13 CRC32 value mismatch.

enumerator kDCIC_Region14MismatchStatus
Region 14 CRC32 value mismatch.

enumerator kDCIC_Region15MismatchStatus
Region 15 CRC32 value mismatch.

enum _dcic_interrupt_enable
Interrupts. .

Values:

enumerator kDCIC_FunctionalInterruptEnable
Interrupt when match results ready.

enumerator kDCIC_ErrorInterruptEnable
Interrupt when there is a signature mismatch.

typedef struct *_dcic_config* dcic_config_t
DCIC configuration.

typedef struct *_dcic_region_config* dcic_region_config_t
Region of interest (ROI) configuration.

DCIC_REGION_COUNT

FSL_DCIC_DRIVER_VERSION
DCIC driver version.

DCIC_CRC32_POLYNOMIAL
CRC32 calculation polynomial.

DCIC_CRC32_INIT_VALUE
CRC32 calculation initialize value.

DCIC_REGION_MISMATCH_STATUS(region)
ROI CRC32 value mismatch status.

struct _dcic_config
#include <fsl_dcic.h> DCIC configuration.

Public Members

bool enableExternalSignal

Enable the mismatch external signal. When enabled, the mismatch status could be monitored from the extern pin.

uint8_t polarityFlags

Display signal polarity, logical OR'ed of _DCIC_polarity_flags.

uint32_t enableInterrupts

Interrupts to enable, should be OR'ed of _dcic_interrupt_enable.

struct _dcic_region_config

#include <fsl_dcic.h> Region of interest (ROI) configuration.

Public Members

bool lock

Lock the region configuration except reference CRC32 value setting.

uint16_t upperLeftX

X of upper left corner. Range: 0 to $2^{13}-1$.

uint16_t upperLeftY

Y of upper left corner. Range: 0 to $2^{12}-1$.

uint16_t lowerRightX

X of lower right corner. Range: 0 to $2^{13}-1$.

uint16_t lowerRightY

Y of lower right corner. Range: 0 to $2^{12}-1$.

uint32_t refCrc

Reference CRC32 value.

2.35 DMAMUX: Direct Memory Access Multiplexer Driver

void DMAMUX_Init(DMAMUX_Type *base)

Initializes the DMAMUX peripheral.

This function ungates the DMAMUX clock.

Parameters

- base – DMAMUX peripheral base address.

void DMAMUX_Deinit(DMAMUX_Type *base)

Deinitializes the DMAMUX peripheral.

This function gates the DMAMUX clock.

Parameters

- base – DMAMUX peripheral base address.

static inline void DMAMUX_EnableChannel(DMAMUX_Type *base, uint32_t channel)

Enables the DMAMUX channel.

This function enables the DMAMUX channel.

Parameters

- base – DMAMUX peripheral base address.
- channel – DMAMUX channel number.

static inline void DMAMUX_DisableChannel(DMAMUX_Type *base, uint32_t channel)

Disables the DMAMUX channel.

This function disables the DMAMUX channel.

Note: The user must disable the DMAMUX channel before configuring it.

Parameters

- base – DMAMUX peripheral base address.
- channel – DMAMUX channel number.

static inline void DMAMUX_SetSource(DMAMUX_Type *base, uint32_t channel, int32_t source)

Configures the DMAMUX channel source.

Parameters

- base – DMAMUX peripheral base address.
- channel – DMAMUX channel number.
- source – Channel source, which is used to trigger the DMA transfer. User need to use the dma_request_source_t type as the input parameter.

static inline void DMAMUX_EnablePeriodTrigger(DMAMUX_Type *base, uint32_t channel)

Enables the DMAMUX period trigger.

This function enables the DMAMUX period trigger feature.

Parameters

- base – DMAMUX peripheral base address.
- channel – DMAMUX channel number.

static inline void DMAMUX_DisablePeriodTrigger(DMAMUX_Type *base, uint32_t channel)

Disables the DMAMUX period trigger.

This function disables the DMAMUX period trigger.

Parameters

- base – DMAMUX peripheral base address.
- channel – DMAMUX channel number.

static inline void DMAMUX_EnableAlwaysOn(DMAMUX_Type *base, uint32_t channel, bool enable)

Enables the DMA channel to be always ON.

This function enables the DMAMUX channel always ON feature.

Parameters

- base – DMAMUX peripheral base address.
- channel – DMAMUX channel number.
- enable – Switcher of the always ON feature. “true” means enabled, “false” means disabled.

FSL_DMAMUX_DRIVER_VERSION

DMAMUX driver version 2.1.1.

2.36 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

`void EDMA_Init(DMA_Type *base, const edma_config_t *config)`

Initializes the eDMA peripheral.

This function ungates the eDMA clock and configures the eDMA peripheral according to the configuration structure. 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(DMA_Type *base)`

Deinitializes the eDMA peripheral.

This function gates the eDMA clock.

Parameters

- `base` – eDMA peripheral base address.

`void EDMA_InstallTCD(DMA_Type *base, uint32_t channel, edma_tcd_t *tcd)`

Push content of TCD structure into hardware TCD register.

Parameters

- `base` – EDMA peripheral base address.
- `channel` – EDMA channel number.
- `tcd` – Point to TCD structure.

`void EDMA_GetDefaultConfig(edma_config_t *config)`

Gets the eDMA default configuration structure.

This function sets the configuration structure to default values. The default configuration is set to the following values.

```
config.enableContinuousLinkMode = false;
config.enableHaltOnError = true;
config.enableRoundRobinArbitration = false;
config.enableDebugMode = false;
```

Parameters

- `config` – A pointer to the eDMA configuration structure.

`static inline void EDMA_EnableContinuousChannelLinkMode(DMA_Type *base, bool enable)`

Enable/Disable continuous channel link mode.

Note: Do not use continuous link mode with a channel linking to itself if there is only one minor loop iteration per service request, for example, if the channel's NBYTES value is the same as either the source or destination size. The same data transfer profile can be achieved by simply increasing the NBYTES value, which provides more efficient, faster processing.

Parameters

- base – EDMA peripheral base address.
- enable – true is enable, false is disable.

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

Enable/Disable minor loop mapping.

The TCDn.word2 is redefined to include individual enable fields, an offset field, and the NBYTES field.

Parameters

- base – EDMA peripheral base address.
- enable – true is enable, false is disable.

void EDMA_ResetChannel(DMA_Type *base, uint32_t channel)

Sets all TCD registers to default values.

This function sets TCD registers for this channel to default values.

Note: This function must not be called while the channel transfer is ongoing or it causes unpredictable results.

Note: This function enables the auto stop request feature.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

void EDMA_SetTransferConfig(DMA_Type *base, uint32_t channel, const *edma_transfer_config_t* *config, *edma_tcd_t* *nextTcd)

Configures the eDMA transfer attribute.

This function configures the transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the TCD address. Example:

```
edma_transfer_t config;
edma_tcd_t tcd;
config.srcAddr = ..;
config.destAddr = ..;
...
EDMA_SetTransferConfig(DMA0, channel, &config, &tcd);
```

Note: If nextTcd is not NULL, it means scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the EDMA_ResetChannel.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- config – Pointer to eDMA transfer configuration structure.
- nextTcd – Point to TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

```
void EDMA_SetMinorOffsetConfig(DMA_Type *base, uint32_t channel, const
                               edma_minor_offset_config_t *config)
```

Configures the eDMA minor offset feature.

The minor offset means that the signed-extended value is added to the source address or destination address after each minor loop.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- config – A pointer to the minor offset configuration structure.

```
void EDMA_SetChannelPreemptionConfig(DMA_Type *base, uint32_t channel, const
                                      edma_channel_preemption_config_t *config)
```

Configures the eDMA channel preemption feature.

This function configures the channel preemption attribute and the priority of the channel.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number
- config – A pointer to the channel preemption configuration structure.

```
void EDMA_SetChannelLink(DMA_Type *base, uint32_t channel, edma_channel_link_type_t
                        linkType, 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.
- linkType – A channel link type, which can be one of the following:
 - kEDMA_LinkNone
 - kEDMA_MinorLink
 - kEDMA_MajorLink
- linkedChannel – The linked channel number.

```
void EDMA_SetBandWidth(DMA_Type *base, uint32_t channel, edma_bandwidth_t bandWidth)
```

Sets the bandwidth for the eDMA transfer.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- base – eDMA peripheral base address.

- channel – eDMA channel number.
- bandWidth – A bandwidth setting, which can be one of the following:
 - kEDMABandwidthStallNone
 - kEDMABandwidthStall4Cycle
 - kEDMABandwidthStall8Cycle

```
void EDMA_SetModulo(DMA_Type *base, uint32_t channel, edma_modulo_t srcModulo,  
                   edma_modulo_t destModulo)
```

Sets the source modulo and the destination modulo for the eDMA transfer.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- srcModulo – A source modulo value.
- destModulo – A destination modulo value.

```
static inline void EDMA_EnableAsyncRequest(DMA_Type *base, uint32_t channel, bool enable)
```

Enables an async request for the eDMA transfer.

Parameters

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

```
static inline void EDMA_EnableAutoStopRequest(DMA_Type *base, uint32_t channel, bool  
                                              enable)
```

Enables an auto stop request for the eDMA transfer.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

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

```
void EDMA_EnableChannelInterrupts(DMA_Type *base, uint32_t channel, uint32_t mask)
```

Enables the interrupt source for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

```
void EDMA_DisableChannelInterrupts(DMA_Type *base, uint32_t channel, uint32_t mask)
```

Disables the interrupt source for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.

- channel – eDMA channel number.
- mask – The mask of the interrupt source to be set. Use the defined `edma_interrupt_enable_t` type.

```
void EDMA_SetMajorOffsetConfig(DMA_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_TcdReset(edma_tcd_t *tcd)
```

Sets all fields to default values for the TCD structure.

This function sets all fields for this TCD structure to default value.

Note: This function enables the auto stop request feature.

Parameters

- tcd – Pointer to the TCD structure.

```
void EDMA_TcdSetTransferConfig(edma_tcd_t *tcd, const edma_transfer_config_t *config,  
                               edma_tcd_t *nextTcd)
```

Configures the eDMA TCD transfer attribute.

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

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 linkType, uint32_t linkedChannel)
```

Sets the channel link for the eDMA TCD.

This function configures either a minor link or a major link. The minor link means the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note: Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

- `tcd` – Point to the TCD structure.
- `linkType` – Channel link type, it can be one of:
 - `kEDMA_LinkNone`
 - `kEDMA_MinorLink`
 - `kEDMA_MajorLink`
- `linkedChannel` – The linked channel number.

```
static inline void EDMA__TcdSetBandWidth(edma_tcd_t *tcd, edma_bandwidth_t bandWidth)
```

Sets the bandwidth for the eDMA TCD.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- `tcd` – A pointer to the TCD structure.
- `bandWidth` – A bandwidth setting, which can be one of the following:
 - `kEDMABandwidthStallNone`
 - `kEDMABandwidthStall4Cycle`
 - `kEDMABandwidthStall8Cycle`

```
void EDMA__TcdSetModulo(edma_tcd_t *tcd, edma_modulo_t srcModulo, edma_modulo_t destModulo)
```

Sets the source modulo and the destination modulo for the eDMA TCD.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- `tcd` – A pointer to the TCD structure.
- `srcModulo` – A source modulo value.
- `destModulo` – A destination modulo value.

`static inline void EDMA__TcdEnableAutoStopRequest(edma_tcd_t *tcd, bool enable)`

Sets the auto stop request for the eDMA TCD.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

- `tcd` – A pointer to the TCD structure.
- `enable` – The command to enable (true) or disable (false).

`void EDMA__TcdEnableInterrupts(edma_tcd_t *tcd, uint32_t mask)`

Enables the interrupt source for the eDMA TCD.

Parameters

- `tcd` – Point to the TCD structure.
- `mask` – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

`void EDMA__TcdDisableInterrupts(edma_tcd_t *tcd, uint32_t mask)`

Disables the interrupt source for the eDMA TCD.

Parameters

- `tcd` – Point to the TCD structure.
- `mask` – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

`void EDMA__TcdSetMajorOffsetConfig(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

- `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(DMA_Type *base, uint32_t channel)`

Enables the eDMA hardware channel request.

This function enables the hardware channel request.

Parameters

- `base` – eDMA peripheral base address.
- `channel` – eDMA channel number.

```
static inline void EDMA_DisableChannelRequest(DMA_Type *base, uint32_t channel)
```

Disables the eDMA hardware channel request.

This function disables the hardware channel request.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

```
static inline void EDMA_TriggerChannelStart(DMA_Type *base, uint32_t channel)
```

Starts the eDMA transfer by using the software trigger.

This function starts a minor loop transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

```
uint32_t EDMA_GetRemainingMajorLoopCount(DMA_Type *base, uint32_t channel)
```

Gets the remaining major loop count from the eDMA current channel TCD.

This function checks the TCD (Task Control Descriptor) status for a specified eDMA channel and returns the number of major loop count that has not finished.

Note: 1. This function can only be used to get unfinished major loop count of transfer without the next TCD, or it might be inaccuracy.

- a. The unfinished/remaining transfer bytes cannot be obtained directly from registers while the channel is running. Because to calculate the remaining bytes, the initial NBYTES configured in DMA_TCDn_NBYTES_MLNO register is needed while the eDMA IP does not support getting it while a channel is active. In another word, the NBYTES value reading is always the actual (decrementing) NBYTES value the dma_engine is working with while a channel is running. Consequently, to get the remaining transfer bytes, a software-saved initial value of NBYTES (for example copied before enabling the channel) is needed. The formula to calculate it is shown below: $\text{RemainingBytes} = \text{RemainingMajorLoopCount} * \text{NBYTES}(\text{initially configured})$
-

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(DMA_Type *base)
```

Gets the eDMA channel error status flags.

Parameters

- base – eDMA peripheral base address.

Returns

The mask of error status flags. Users need to use the `_edma_error_status_flags` type to decode the return variables.

```
uint32_t EDMA_GetChannelStatusFlags(DMA_Type *base, uint32_t channel)
```

Gets the eDMA channel status flags.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

Returns

The mask of channel status flags. Users need to use the `_edma_channel_status_flags` type to decode the return variables.

`void EDMA_ClearChannelStatusFlags(DMA_Type *base, uint32_t channel, uint32_t mask)`

Clears the eDMA channel status flags.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of channel status to be cleared. Users need to use the defined `_edma_channel_status_flags` type.

`void EDMA_CreateHandle(edma_handle_t *handle, DMA_Type *base, uint32_t channel)`

Creates the eDMA handle.

This function is called if using the transactional API for eDMA. This function initializes the internal state of the eDMA handle.

Parameters

- handle – eDMA handle pointer. The eDMA handle stores callback function and parameters.
- base – eDMA peripheral base address.
- channel – eDMA channel number.

`void EDMA_InstallTCDMemory(edma_handle_t *handle, edma_tcd_t *tcdPool, uint32_t tcdSize)`

Installs the TCDs memory pool into the eDMA handle.

This function is called after the `EDMA_CreateHandle` to use scatter/gather feature. 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).

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 transferType)
```

Prepares the eDMA transfer structure.

This function prepares the transfer configuration structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

Parameters

- config – The user configuration structure of type `edma_transfer_t`.
- srcAddr – eDMA transfer source address.
- srcWidth – eDMA transfer source address width(bytes).
- destAddr – eDMA transfer destination address.
- destWidth – eDMA transfer destination address width(bytes).
- bytesEachRequest – eDMA transfer bytes per channel request.
- transferBytes – eDMA transfer bytes to be transferred.
- transferType – eDMA transfer type.

status_t EDMA_SubmitTransfer(*edma_handle_t* *handle, const *edma_transfer_config_t* *config)

Submits the eDMA transfer request.

This function submits the eDMA transfer request according to the transfer configuration structure. 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.

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 2.4.7.

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

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_ChannelPriorityErrorFlag
Channel priority is not unique.

enumerator kEDMA_TransferCanceledFlag
Transfer cancelled

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

enum _edma_interrupt_enable
eDMA interrupt source

Values:

enumerator kEDMA_ErrorInterruptEnable

Enable interrupt while channel error occurs.

enumerator kEDMA_MajorInterruptEnable

Enable interrupt while major count exhausted.

enumerator kEDMA_HalfInterruptEnable

Enable interrupt while major count to half value.

enum _edma_transfer_type

eDMA transfer type

Values:

enumerator kEDMA_MemoryToMemory

Transfer from memory to memory

enumerator kEDMA_PeripheralToMemory

Transfer from peripheral to memory

enumerator kEDMA_MemoryToPeripheral

Transfer from memory to peripheral

enumerator kEDMA_PeripheralToPeripheral

Transfer from Peripheral to peripheral

_edma_transfer_status eDMA transfer status

Values:

enumerator kStatus_EDMA_QueueFull

TCD queue is full.

enumerator kStatus_EDMA_Busy

Channel is busy and can't handle the transfer request.

typedef enum _edma_transfer_size edma_transfer_size_t

eDMA transfer configuration

typedef enum _edma_modulo edma_modulo_t

eDMA modulo configuration

typedef enum _edma_bandwidth edma_bandwidth_t

Bandwidth control.

typedef enum _edma_channel_link_type edma_channel_link_type_t

Channel link type.

typedef enum _edma_interrupt_enable edma_interrupt_enable_t

eDMA interrupt source

typedef enum _edma_transfer_type edma_transfer_type_t

eDMA transfer type

typedef struct _edma_config edma_config_t

eDMA global configuration structure.

typedef struct _edma_transfer_config edma_transfer_config_t

eDMA transfer configuration

This structure configures the source/destination transfer attribute.

```
typedef struct _edma_channel_Preemption_config edma_channel_Preemption_config_t
    eDMA channel priority configuration
```

```
typedef struct _edma_minor_offset_config edma_minor_offset_config_t
    eDMA minor offset configuration
```

```
typedef struct _edma_tcd edma_tcd_t
    eDMA TCD.
```

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

```
typedef void (*edma_callback)(struct _edma_handle *handle, void *userData, bool transferDone,
uint32_t tcds)
```

Define callback function for eDMA.

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
```

```
DMA_DCHPRI_INDEX(channel)
    Compute the offset unit from DCHPRI3.
```

```
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 enableHaltOnError
```

Enable (true) transfer halt on error. Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

bool enableRoundRobinArbitration

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

bool enableDebugMode

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

struct _edma_transfer_config

#include <fsl_edma.h> eDMA transfer configuration

This structure configures the source/destination transfer attribute.

Public Members

uint32_t srcAddr

Source data address.

uint32_t destAddr

Destination data address.

edma_transfer_size_t srcTransferSize

Source data transfer size.

edma_transfer_size_t destTransferSize

Destination data transfer size.

int16_t srcOffset

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

int16_t destOffset

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

uint32_t minorLoopBytes

Bytes to transfer in a minor loop

uint32_t majorLoopCounts

Major loop iteration count.

struct _edma_channel_Preemption_config

#include <fsl_edma.h> eDMA channel priority configuration

Public Members

bool enableChannelPreemption

If true: a channel can be suspended by other channel with higher priority

bool enablePreemptAbility

If true: a channel can suspend other channel with low priority

uint8_t channelPriority

Channel priority

struct _edma_minor_offset_config

#include <fsl_edma.h> eDMA minor offset configuration

Public Members

bool enableSrcMinorOffset

Enable(true) or Disable(false) source minor loop offset.

bool enableDestMinorOffset

Enable(true) or Disable(false) destination minor loop offset.

uint32_t minorOffset

Offset for a minor loop mapping.

struct __edma_tcd

#include <fsl_edma.h> eDMA TCD.

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

Public Members

__IO uint32_t SADDR

SADDR register, used to save source address

__IO uint16_t SOFF

SOFF register, save offset bytes every transfer

__IO uint16_t ATTR

ATTR register, source/destination transfer size and modulo

__IO uint32_t NBYTES

Nbytes register, minor loop length in bytes

__IO uint32_t SLAST

SLAST register

__IO uint32_t DADDR

DADDR register, used for destination address

__IO uint16_t DOFF

DOFF register, used for destination offset

__IO uint16_t CITER

CITER register, current minor loop numbers, for unfinished minor loop.

__IO uint32_t DLAST_SGA

DLASTSGA register, next 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_handle

#include <fsl_edma.h> eDMA transfer handle structure

Public Members

edma_callback callback

Callback function for major count exhausted.

`void *userData`

Callback function parameter.

`DMA_Type *base`

eDMA peripheral base address.

`edma_tcd_t *tcdPool`

Pointer to memory stored TCDs.

`uint8_t channel`

eDMA channel number.

`volatile int8_t header`

The first TCD index. 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.

`uint8_t flags`

The status of the current channel.

2.37 eLCDIF: Enhanced LCD Interface

`void ELCDIF_RgbModeInit(LCDIF_Type *base, const elcdif_rgb_mode_config_t *config)`

Initializes the eLCDIF to work in RGB mode (DOTCLK mode).

This function ungates the eLCDIF clock and configures the eLCDIF peripheral according to the configuration structure.

Parameters

- `base` – eLCDIF peripheral base address.
- `config` – Pointer to the configuration structure.

`void ELCDIF_RgbModeGetDefaultConfig(elcdif_rgb_mode_config_t *config)`

Gets the eLCDIF default configuration structure for RGB (DOTCLK) mode.

This function sets the configuration structure to default values. The default configuration is set to the following values.

```
config->panelWidth = 480U;
config->panelHeight = 272U;
config->hsw = 41;
config->hfp = 4;
config->hbp = 8;
config->vsw = 10;
config->vfp = 4;
config->vbp = 2;
config->polarityFlags = kELCDIF_VsyncActiveLow |
                      kELCDIF_HsyncActiveLow |
                      kELCDIF_DataEnableActiveLow |
                      kELCDIF_DriveDataOnFallingClkEdge;
config->bufferAddr = 0U;
```

(continues on next page)

(continued from previous page)

```
config->pixelFormat = kELCDIF_PixelFormatRGB888;
config->dataBus = kELCDIF_DataBus24Bit;
```

Parameters

- config – Pointer to the eLCDIF configuration structure.

void ELCDIF_Deinit(LCDIF_Type *base)
Deinitializes the eLCDIF peripheral.

Parameters

- base – eLCDIF peripheral base address.

void ELCDIF_RgbModeSetPixelFormat(LCDIF_Type *base, *elcdif_pixel_format_t* pixelFormat)
Set the pixel format in RGB (DOTCLK) mode.

Parameters

- base – eLCDIF peripheral base address.
- pixelFormat – The pixel format.

static inline void ELCDIF_RgbModeStart(LCDIF_Type *base)
Start to display in RGB (DOTCLK) mode.

Parameters

- base – eLCDIF peripheral base address.

void ELCDIF_RgbModeStop(LCDIF_Type *base)
Stop display in RGB (DOTCLK) mode and wait until finished.

Parameters

- base – eLCDIF peripheral base address.

static inline void ELCDIF_SetNextBufferAddr(LCDIF_Type *base, uint32_t bufferAddr)
Set the next frame buffer address to display.

Parameters

- base – eLCDIF peripheral base address.
- bufferAddr – The frame buffer address to set.

void ELCDIF_Reset(LCDIF_Type *base)
Reset the eLCDIF peripheral.

Parameters

- base – eLCDIF peripheral base address.

void ELCDIF_SetPixelComponentOrder(LCDIF_Type *base, *elcdif_pixel_component_order_t* order)
Set the order of the RGB components of each pixel in lines.

Parameters

- base – eLCDIF peripheral base address.
- order – The pixel component order

static inline uint32_t ELCDIF_GetCrcValue(const LCDIF_Type *base)

Get the CRC value of the frame sent out.

When a frame is sent complete (the interrupt kELCDIF_CurFrameDone assert), this function can be used to get the CRC value of the frame sent.

Note: The CRC value is dependent on the LCD_DATABUS_WIDTH.

Parameters

- base – eLCDIF peripheral base address.

Returns

The CRC value.

static inline uint32_t ELCDIF_GetBusMasterErrorAddr(const LCDIF_Type *base)

Get the bus master error virtual address.

When bus master error occurs (the interrupt kELCDIF_BusMasterError assert), this function can get the virtual address at which the AXI master received an error response from the slave.

Parameters

- base – eLCDIF peripheral base address.

Returns

The error virtual address.

static inline uint32_t ELCDIF_GetStatus(const LCDIF_Type *base)

Get the eLCDIF status.

The status flags are returned as a mask value, application could check the corresponding bit. Example:

```
uint32_t statusFlags;
statusFlags = ELCDIF_GetStatus(LCDIF);

if (kELCDIF_LFifoFull & statusFlags)
{
}

if (kELCDIF_TxFifoEmpty & statusFlags)
{
}
```

Parameters

- base – eLCDIF peripheral base address.

Returns

The mask value of status flags, it is OR'ed value of _elcdif_status_flags.

static inline uint32_t ELCDIF_GetLFifoCount(const LCDIF_Type *base)

Get current count in Latency buffer (LFIFO).

Parameters

- base – eLCDIF peripheral base address.

Returns

The LFIFO current count

static inline void ELCDIF_EnableInterrupts(LCDIF_Type *base, uint32_t mask)

Enables eLCDIF interrupt requests.

Parameters

- base – eLCDIF peripheral base address.
- mask – interrupt source, OR'ed value of `_elcdif_interrupt_enable`.

static inline void ELCDIF_DisableInterrupts(LCDIF_Type *base, uint32_t mask)

Disables eLCDIF interrupt requests.

Parameters

- base – eLCDIF peripheral base address.
- mask – interrupt source, OR'ed value of `_elcdif_interrupt_enable`.

static inline uint32_t ELCDIF_GetInterruptStatus(const LCDIF_Type *base)

Get eLCDIF interrupt pending status.

Parameters

- base – eLCDIF peripheral base address.

Returns

Interrupt pending status, OR'ed value of `_elcdif_interrupt_flags`.

static inline void ELCDIF_ClearInterruptStatus(LCDIF_Type *base, uint32_t mask)

Clear eLCDIF interrupt pending status.

Parameters

- base – eLCDIF peripheral base address.
- mask – of the flags to clear, OR'ed value of `_elcdif_interrupt_flags`.

static inline void ELCDIF_EnableLut(LCDIF_Type *base, bool enable)

Enable or disable the LUT.

Parameters

- base – eLCDIF peripheral base address.
- enable – True to enable, false to disable.

status_t ELCDIF_UpdateLut(LCDIF_Type *base, *elcdif_lut_t* lut, uint16_t startIndex, const uint32_t *lutData, uint16_t count)

Load the LUT value.

This function loads the LUT value to the specific LUT memory, user can specify the start entry index.

Parameters

- base – eLCDIF peripheral base address.
- lut – Which LUT to load.
- startIndex – The start index of the LUT entry to update.
- lutData – The LUT data to load.
- count – Count of lutData.

Return values

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

FSL_ELCDIF_DRIVER_VERSION

eLCDIF driver version

enum __elcdif_polarity_flags

eLCDIF signal polarity flags

Values:

enumerator kELCDIF_VsyncActiveLow

VSYNC active low.

enumerator kELCDIF_HsyncActiveLow

HSYNC active low.

enumerator kELCDIF_DataEnableActiveLow

Data enable line active low.

enumerator kELCDIF_DriveDataOnFallingClkEdge

Drive data on falling clock edge, capture data on rising clock edge.

enumerator kELCDIF_VsyncActiveHigh

VSYNC active high.

enumerator kELCDIF_HsyncActiveHigh

HSYNC active high.

enumerator kELCDIF_DataEnableActiveHigh

Data enable line active high.

enumerator kELCDIF_DriveDataOnRisingClkEdge

Drive data on falling clock edge, capture data on rising clock edge.

enum __elcdif_interrupt_enable

The eLCDIF interrupts to enable.

Values:

enumerator kELCDIF_BusMasterErrorInterruptEnable

Bus master error interrupt.

enumerator kELCDIF_TxFifoOverflowInterruptEnable

TXFIFO overflow interrupt.

enumerator kELCDIF_TxFifoUnderflowInterruptEnable

TXFIFO underflow interrupt.

enumerator kELCDIF_CurFrameDoneInterruptEnable

Interrupt when hardware enters vertical blanking state.

enumerator kELCDIF_VsyncEdgeInterruptEnable

Interrupt when hardware encounters VSYNC edge.

enum __elcdif_interrupt_flags

The eLCDIF interrupt status flags.

Values:

enumerator kELCDIF_BusMasterError

Bus master error interrupt.

enumerator kELCDIF_TxFifoOverflow

TXFIFO overflow interrupt.

enumerator kELCDIF__TxFifoUnderflow

TXFIFO underflow interrupt.

enumerator kELCDIF__CurFrameDone

Interrupt when hardware enters vertical blanking state.

enumerator kELCDIF__VsyncEdge

Interrupt when hardware encounters VSYNC edge.

enum __elcdif_status_flags

eLCDIF status flags

Values:

enumerator kELCDIF__LFifoFull

LFIFO full.

enumerator kELCDIF__LFifoEmpty

LFIFO empty.

enumerator kELCDIF__TxFifoFull

TXFIFO full.

enumerator kELCDIF__TxFifoEmpty

TXFIFO empty.

enum __elcdif_pixel_format

The pixel format.

This enumerator should be defined together with the array s_pixelFormatReg. To support new pixel format, enhance this enumerator and s_pixelFormatReg.

Values:

enumerator kELCDIF__PixelFormatRAW8

RAW 8 bit, four data use 32 bits.

enumerator kELCDIF__PixelFormatRGB565

RGB565, two pixel use 32 bits.

enumerator kELCDIF__PixelFormatRGB666

RGB666 unpacked, one pixel uses 32 bits, high byte unused, upper 2 bits of other bytes unused.

enumerator kELCDIF__PixelFormatXRGB8888

XRGB8888 unpacked, one pixel uses 32 bits, high byte unused.

enumerator kELCDIF__PixelFormatRGB888

RGB888 packed, one pixel uses 24 bits.

enum __elcdif_lcd_data_bus

The LCD data bus type.

Values:

enumerator kELCDIF__DataBus8Bit

8-bit data bus.

enumerator kELCDIF__DataBus16Bit

16-bit data bus, support RGB565.

enumerator kELCDIF__DataBus18Bit

18-bit data bus, support RGB666.

enumerator kELCDIF_DataBus24Bit
24-bit data bus, support RGB888.

enum _elcdif_as_pixel_format
eLCDIF alpha surface pixel format.

Values:

enumerator kELCDIF_AsPixelFormatARGB8888
32-bit pixels with alpha.

enumerator kELCDIF_AsPixelFormatRGB888
32-bit pixels without alpha (unpacked 24-bit format)

enumerator kELCDIF_AsPixelFormatARGB1555
16-bit pixels with alpha.

enumerator kELCDIF_AsPixelFormatARGB4444
16-bit pixels with alpha.

enumerator kELCDIF_AsPixelFormatRGB555
16-bit pixels without alpha.

enumerator kELCDIF_AsPixelFormatRGB444
16-bit pixels without alpha.

enumerator kELCDIF_AsPixelFormatRGB565
16-bit pixels without alpha.

enum _elcdif_alpha_mode
eLCDIF alpha mode during blending.

Values:

enumerator kELCDIF_AlphaEmbedded
The alpha surface pixel alpha value will be used for blend.

enumerator kELCDIF_AlphaOverride
The user defined alpha value will be used for blend directly.

enumerator kELCDIF_AlphaMultiply
The alpha surface pixel alpha value scaled the user defined alpha value will be used for blend, for example, pixel alpha set to 200, user defined alpha set to 100, then the result alpha is $200 * 100 / 255$.

enumerator kELCDIF_AlphaRop
Raster operation.

enum _elcdif_rop_mode
eLCDIF ROP mode during blending.

Explanation:

- AS: Alpha surface
- PS: Process surface
- nAS: Alpha surface NOT value
- nPS: Process surface NOT value

Values:

enumerator kELCDIF_RopMaskAs
AS AND PS.

enumerator kELCDIF__RopMaskNotAs
nAS AND PS.

enumerator kELCDIF__RopMaskAsNot
AS AND nPS.

enumerator kELCDIF__RopMergeAs
AS OR PS.

enumerator kELCDIF__RopMergeNotAs
nAS OR PS.

enumerator kELCDIF__RopMergeAsNot
AS OR nPS.

enumerator kELCDIF__RopNotCopyAs
nAS.

enumerator kELCDIF__RopNot
nPS.

enumerator kELCDIF__RopNotMaskAs
AS NAND PS.

enumerator kELCDIF__RopNotMergeAs
AS NOR PS.

enumerator kELCDIF__RopXorAs
AS XOR PS.

enumerator kELCDIF__RopNotXorAs
AS XNOR PS.

enum _elcdif_lut
eLCDIF LUT

The Lookup Table (LUT) is used to expand the 8 bits pixel to 24 bits pixel before output to external displayer.

There are two 256x24 bits LUT memory in LCDIF, the LSB of frame buffer address determines which memory to use.

Values:

enumerator kELCDIF__Lut0
LUT 0.

enumerator kELCDIF__Lut1
LUT 1.

enum _elcdif_pixel_component_order
eLCDIF pixel component order.

Values:

enumerator kELCDIF__PixelComponentOrderRGB
Input order RGB.

enumerator kELCDIF__PixelComponentOrderRBG
Input order RBG.

enumerator kELCDIF__PixelComponentOrderGBR
Input order GBR.

enumerator `kELCDIF_PixelComponentOrderGRB`
Input order GRB.

enumerator `kELCDIF_PixelComponentOrderBRG`
Input order BRG.

enumerator `kELCDIF_PixelComponentOrderBGR`
Input order BGR.

typedef enum `_elcdif_pixel_format` `elcdif_pixel_format_t`
The pixel format.

This enumerator should be defined together with the array `s_pixelFormatReg`. To support new pixel format, enhance this enumerator and `s_pixelFormatReg`.

typedef enum `_elcdif_lcd_data_bus` `elcdif_lcd_data_bus_t`
The LCD data bus type.

typedef struct `_elcdif_pixel_format_reg` `elcdif_pixel_format_reg_t`
The register value when using different pixel format.

These register bits control the pixel format:

- `CTRL[DATA_FORMAT_24_BIT]`
- `CTRL[DATA_FORMAT_18_BIT]`
- `CTRL[DATA_FORMAT_16_BIT]`
- `CTRL[WORD_LENGTH]`
- `CTRL1[BYTE_PACKING_FORMAT]`

typedef struct `_elcdif_rgb_mode_config` `elcdif_rgb_mode_config_t`
eLCDIF configure structure for RGB mode (DOTCLK mode).

typedef enum `_elcdif_as_pixel_format` `elcdif_as_pixel_format_t`
eLCDIF alpha surface pixel format.

typedef struct `_elcdif_as_buffer_config` `elcdif_as_buffer_config_t`
eLCDIF alpha surface buffer configuration.

typedef enum `_elcdif_alpha_mode` `elcdif_alpha_mode_t`
eLCDIF alpha mode during blending.

typedef enum `_elcdif_rop_mode` `elcdif_rop_mode_t`
eLCDIF ROP mode during blending.

Explanation:

- AS: Alpha surface
- PS: Process surface
- nAS: Alpha surface NOT value
- nPS: Process surface NOT value

typedef struct `_elcdif_as_blend_config` `elcdif_as_blend_config_t`
eLCDIF alpha surface blending configuration.

typedef enum `_elcdif_lut` `elcdif_lut_t`
eLCDIF LUT

The Lookup Table (LUT) is used to expand the 8 bits pixel to 24 bits pixel before output to external display.

There are two 256x24 bits LUT memory in LCDIF, the LSB of frame buffer address determines which memory to use.

```
typedef enum _elcdif_pixel_component_order elcdif_pixel_component_order_t
```

eLCDIF pixel component order.

```
ELCDIF_CTRL1_IRQ_MASK
```

```
ELCDIF_CTRL1_IRQ_EN_MASK
```

```
ELCDIF_AS_CTRL_IRQ_MASK
```

```
ELCDIF_AS_CTRL_IRQ_EN_MASK
```

```
FSL_FEATURE_LCDIF_HAS_PXP_HANDSHAKE
```

```
ELCDIF_ADDR_CPU_2_IP(addr)
```

```
ELCDIF_LUT_ENTRY_NUM
```

```
struct _elcdif_pixel_format_reg
```

#include <fsl_elcdif.h> The register value when using different pixel format.

These register bits control the pixel format:

- CTRL[DATA_FORMAT_24_BIT]
- CTRL[DATA_FORMAT_18_BIT]
- CTRL[DATA_FORMAT_16_BIT]
- CTRL[WORD_LENGTH]
- CTRL1[BYTE_PACKING_FORMAT]

Public Members

```
uint32_t regCtrl
```

Value of register CTRL.

```
uint32_t regCtrl1
```

Value of register CTRL1.

```
struct _elcdif_rgb_mode_config
```

#include <fsl_elcdif.h> eLCDIF configure structure for RGB mode (DOTCLK mode).

Public Members

```
uint16_t panelWidth
```

Display panel width, pixels per line.

```
uint16_t panelHeight
```

Display panel height, how many lines per panel.

```
uint8_t hsw
```

HSYNC pulse width.

```
uint8_t hfp
```

Horizontal front porch.

```
uint8_t hbp
```

Horizontal back porch.

```
uint8_t vsw
```

VSYNC pulse width.

uint8_t vfp

Vertical front porch.

uint8_t vbp

Vertical back porch.

uint32_t polarityFlags

OR'ed value of `_elcdif_polarity_flags`, used to control the signal polarity.

uint32_t bufferAddr

Frame buffer address.

elcdif_pixel_format_t pixelFormat

Pixel format.

elcdif_lcd_data_bus_t dataBus

LCD data bus.

struct `_elcdif_as_buffer_config`

#include <fsl_elcdif.h> eLCDIF alpha surface buffer configuration.

Public Members

uint32_t bufferAddr

Buffer address.

elcdif_as_pixel_format_t pixelFormat

Pixel format.

struct `_elcdif_as_blend_config`

#include <fsl_elcdif.h> eLCDIF alpha surface blending configuration.

Public Members

uint8_t alpha

User defined alpha value, only used when `alphaMode` is `kELCDIF_AlphaOverride` or `kELCDIF_AlphaRop`.

bool invertAlpha

Set true to invert the alpha.

elcdif_alpha_mode_t alphaMode

Alpha mode.

elcdif_rop_mode_t ropMode

ROP mode, only valid when `alphaMode` is `kELCDIF_AlphaRop`.

2.38 ENC: Quadrature Encoder/Decoder

void ENC_Init(ENC_Type *base, const *enc_config_t* *config)

Initialization for the ENC module.

This function is to make the initialization for the ENC module. It should be called firstly before any operation to the ENC with the operations like:

- Enable the clock for ENC module.
- Configure the ENC's working attributes.

Parameters

- base – ENC peripheral base address.
- config – Pointer to configuration structure. See to “enc_config_t”.

void ENC_Deinit(ENC_Type *base)

De-initialization for the ENC module.

This function is to make the de-initialization for the ENC module. It could be called when ENC is no longer used with the operations like:

- Disable the clock for ENC module.

Parameters

- base – ENC peripheral base address.

void ENC_GetDefaultConfig(*enc_config_t* *config)

Get an available pre-defined settings for ENC's configuration.

This function initializes the ENC configuration structure with an available settings, the default value are:

```
config->enableReverseDirection      = false;
config->decoderWorkMode             = kENC_DecoderWorkAsNormalMode;
config->HOMETriggerMode             = kENC_HOMETriggerDisabled;
config->INDEXTriggerMode            = kENC_INDEXTriggerDisabled;
config->enableTRIGGERClearPositionCounter = false;
config->enableTRIGGERClearHoldPositionCounter = false;
config->enableWatchdog              = false;
config->watchdogTimeoutValue        = 0U;
config->filterCount                 = 0U;
config->filterSamplePeriod          = 0U;
config->positionMatchMode           = kENC_
↪ POSMATCHOnPositionCounterEqualToComapreValue;
config->positionCompareValue        = 0xFFFFFFFFU;
config->revolutionCountCondition     = kENC_RevolutionCountOnINDEXPulse;
config->enableModuloCountMode       = false;
config->positionModulusValue        = 0U;
config->positionInitialValue        = 0U;
config->prescalerValue              = kENC_ClockDiv1;
config->enablePeriodMeasurementFunction = true;
```

Parameters

- config – Pointer to a variable of configuration structure. See to “enc_config_t”.

void ENC_DoSoftwareLoadInitialPositionValue(ENC_Type *base)

Load the initial position value to position counter.

This function is to transfer the initial position value (UNIT and LINIT) contents to position counter (UPOS and LPOS), so that to provide the consistent operation the position counter registers.

Parameters

- base – ENC peripheral base address.

void ENC_SetSelfTestConfig(ENC_Type *base, const *enc_self_test_config_t* *config)

Enable and configure the self test function.

This function is to enable and configuration the self test function. It controls and sets the frequency of a quadrature signal generator. It provides a quadrature test signal to the in-

puts of the quadrature decoder module. It is a factory test feature; however, it may be useful to customers' software development and testing.

Parameters

- base – ENC peripheral base address.
- config – Pointer to configuration structure. See to “enc_self_test_config_t”. Pass “NULL” to disable.

void ENC_EnableWatchdog(ENC_Type *base, bool enable)

Enable watchdog for ENC module.

Parameters

- base – ENC peripheral base address
- enable – Enables or disables the watchdog

void ENC_SetInitialPositionValue(ENC_Type *base, uint32_t value)

Set initial position value for ENC module.

Parameters

- base – ENC peripheral base address
- value – Positive initial value

uint32_t ENC_GetStatusFlags(ENC_Type *base)

Get the status flags.

Parameters

- base – ENC peripheral base address.

Returns

Mask value of status flags. For available mask, see to “_enc_status_flags”.

void ENC_ClearStatusFlags(ENC_Type *base, uint32_t mask)

Clear the status flags.

Parameters

- base – ENC peripheral base address.
- mask – Mask value of status flags to be cleared. For available mask, see to “_enc_status_flags”.

static inline uint16_t ENC_GetSignalStatusFlags(ENC_Type *base)

Get the signals' real-time status.

Parameters

- base – ENC peripheral base address.

Returns

Mask value of signals' real-time status. For available mask, see to “_enc_signal_status_flags”

void ENC_EnableInterrupts(ENC_Type *base, uint32_t mask)

Enable the interrupts.

Parameters

- base – ENC peripheral base address.
- mask – Mask value of interrupts to be enabled. For available mask, see to “_enc_interrupt_enable”.

`void ENC_DisableInterrupts(ENC_Type *base, uint32_t mask)`

Disable the interrupts.

Parameters

- `base` – ENC peripheral base address.
- `mask` – Mask value of interrupts to be disabled. For available mask, see to “`_enc_interrupt_enable`”.

`uint32_t ENC_GetEnabledInterrupts(ENC_Type *base)`

Get the enabled interrupts' flags.

Parameters

- `base` – ENC peripheral base address.

Returns

Mask value of enabled interrupts.

`uint32_t ENC_GetPositionValue(ENC_Type *base)`

Get the current position counter's value.

Parameters

- `base` – ENC peripheral base address.

Returns

Current position counter's value.

`uint32_t ENC_GetHoldPositionValue(ENC_Type *base)`

Get the hold position counter's value.

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

Parameters

- `base` – ENC peripheral base address.

Returns

Hold position counter's value.

`static inline uint16_t ENC_GetPositionDifferenceValue(ENC_Type *base)`

Get the position difference counter's value.

Parameters

- `base` – ENC peripheral base address.

Returns

The position difference counter's value.

`static inline uint16_t ENC_GetHoldPositionDifferenceValue(ENC_Type *base)`

Get the hold position difference counter's value.

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

Parameters

- `base` – ENC peripheral base address.

Returns

Hold position difference counter's value.

```
static inline uint16_t ENC_GetRevolutionValue(ENC_Type *base)
```

Get the position revolution counter's value.

Parameters

- base – ENC peripheral base address.

Returns

The position revolution counter's value.

```
static inline uint16_t ENC_GetHoldRevolutionValue(ENC_Type *base)
```

Get the hold position revolution counter's value.

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

Parameters

- base – ENC peripheral base address.

Returns

Hold position revolution counter's value.

```
static inline uint16_t ENC_GetLastEdgeTimeValue(ENC_Type *base)
```

Get the last edge time value.

Parameters

- base – ENC peripheral base address.

Returns

The last edge time hold value.

```
static inline uint16_t ENC_GetHoldLastEdgeTimeValue(ENC_Type *base)
```

Get the last edge time hold value.

Parameters

- base – ENC peripheral base address.

Returns

The last edge time hold value.

```
static inline uint16_t ENC_GetPositionDifferencePeriodValue(ENC_Type *base)
```

Get the position difference period value.

Parameters

- base – ENC peripheral base address.

Returns

The position difference period hold value.

```
static inline uint16_t ENC_GetPositionDifferencePeriodBufferValue(ENC_Type *base)
```

Get the position difference period buffer value.

Parameters

- base – ENC peripheral base address.

Returns

The position difference period hold value.

```
static inline uint16_t ENC_GetHoldPositionDifferencePeriodValue(ENC_Type *base)
```

Get the position difference period hold value.

Parameters

- base – ENC peripheral base address.

Returns

The position difference period hold value.

enum `_enc_interrupt_enable`

Interrupt enable/disable mask.

Values:

enumerator `kENC_HOMETransitionInterruptEnable`

HOME interrupt enable.

enumerator `kENC_INDEXPulseInterruptEnable`

INDEX pulse interrupt enable.

enumerator `kENC_WatchdogTimeoutInterruptEnable`

Watchdog timeout interrupt enable.

enumerator `kENC_PositionCompareInterruptEnable`

Position compare interrupt enable.

enumerator `kENC_PositionRollOverInterruptEnable`

Roll-over interrupt enable.

enumerator `kENC_PositionRollUnderInterruptEnable`

Roll-under interrupt enable.

enum `_enc_status_flags`

Status flag mask.

These flags indicate the counter's events.

Values:

enumerator `kENC_HOMETransitionFlag`

HOME signal transition interrupt request.

enumerator `kENC_INDEXPulseFlag`

INDEX Pulse Interrupt Request.

enumerator `kENC_WatchdogTimeoutFlag`

Watchdog timeout interrupt request.

enumerator `kENC_PositionCompareFlag`

Position compare interrupt request.

enumerator `kENC_PositionRollOverFlag`

Roll-over interrupt request.

enumerator `kENC_PositionRollUnderFlag`

Roll-under interrupt request.

enumerator `kENC_LastCountDirectionFlag`

Last count was in the up direction, or the down direction.

enum `_enc_signal_status_flags`

Signal status flag mask.

These flags indicate the counter's signal.

Values:

enumerator `kENC_RawHOMEStatusFlag`

Raw HOME input.

enumerator kENC_RawINDEXStatusFlag

Raw INDEX input.

enumerator kENC_RawPHBStatusFlag

Raw PHASEB input.

enumerator kENC_RawPHAEXStatusFlag

Raw PHASEA input.

enumerator kENC_FilteredHOMESTatusFlag

The filtered version of HOME input.

enumerator kENC_FilteredINDEXStatusFlag

The filtered version of INDEX input.

enumerator kENC_FilteredPHBStatusFlag

The filtered version of PHASEB input.

enumerator kENC_FilteredPHASTatusFlag

The filtered version of PHASEA input.

enum _enc_home_trigger_mode

Define HOME signal's trigger mode.

The ENC would count the trigger from HOME signal line.

Values:

enumerator kENC_HOMETriggerDisabled

HOME signal's trigger is disabled.

enumerator kENC_HOMETriggerOnRisingEdge

Use positive going edge-to-trigger initialization of position counters.

enumerator kENC_HOMETriggerOnFallingEdge

Use negative going edge-to-trigger initialization of position counters.

enum _enc_index_trigger_mode

Define INDEX signal's trigger mode.

The ENC would count the trigger from INDEX signal line.

Values:

enumerator kENC_INDEXTriggerDisabled

INDEX signal's trigger is disabled.

enumerator kENC_INDEXTriggerOnRisingEdge

Use positive going edge-to-trigger initialization of position counters.

enumerator kENC_INDEXTriggerOnFallingEdge

Use negative going edge-to-trigger initialization of position counters.

enum _enc_decoder_work_mode

Define type for decoder work mode.

The normal work mode uses the standard quadrature decoder with PHASEA and PHASEB. When in signal phase count mode, a positive transition of the PHASEA input generates a count signal while the PHASEB input and the reverse direction control the counter direction. If the reverse direction is not enabled, PHASEB = 0 means counting up and PHASEB = 1 means counting down. Otherwise, the direction is reversed.

Values:

enumerator kENC_DecoderWorkAsNormalMode

Use standard quadrature decoder with PHASEA and PHASEB.

enumerator kENC_DecoderWorkAsSignalPhaseCountMode

PHASEA input generates a count signal while PHASEB input control the direction.

enum _enc_position_match_mode

Define type for the condition of POSMATCH pulses.

Values:

enumerator kENC_POSMATCHOnPositionCounterEqualToCompareValue

POSMATCH pulses when a match occurs between the position counters (POS) and the compare value (COMP).

enumerator kENC_POSMATCHOnReadingAnyPositionCounter

POSMATCH pulses when any position counter register is read.

enum _enc_revolution_count_condition

Define type for determining how the revolution counter (REV) is incremented/decremented.

Values:

enumerator kENC_RevolutionCountOnINDEXPulse

Use INDEX pulse to increment/decrement revolution counter.

enumerator kENC_RevolutionCountOnRollOverModulus

Use modulus counting roll-over/under to increment/decrement revolution counter.

enum _enc_self_test_direction

Define type for direction of self test generated signal.

Values:

enumerator kENC_SelfTestDirectionPositive

Self test generates the signal in positive direction.

enumerator kENC_SelfTestDirectionNegative

Self test generates the signal in negative direction.

enum _enc_prescaler

Define prescaler value for clock in CTRL3.

The clock is prescaled by a value of 2^{PRSC} which means that the prescaler logic can divide the clock by a minimum of 1 and a maximum of 32,768.

Values:

enumerator kENC_ClockDiv1

enumerator kENC_ClockDiv2

enumerator kENC_ClockDiv4

enumerator kENC_ClockDiv8

enumerator kENC_ClockDiv16

enumerator kENC_ClockDiv32

enumerator kENC_ClockDiv64

enumerator kENC_ClockDiv128

enumerator kENC_ClockDiv256
enumerator kENC_ClockDiv512
enumerator kENC_ClockDiv1024
enumerator kENC_ClockDiv2048
enumerator kENC_ClockDiv4096
enumerator kENC_ClockDiv8192
enumerator kENC_ClockDiv16384
enumerator kENC_ClockDiv32768

enum _enc_filter_prescaler

Define input filter prescaler value.

The input filter prescaler value is to prescale the IPBus clock. (Frequency of FILT clock) = (Frequency of IPBus clock) / $2^{\text{FILT\_PRSC}}$.

Values:

enumerator kENC_FilterPrescalerDiv1
Input filter prescaler is 1.
enumerator kENC_FilterPrescalerDiv2
Input filter prescaler is 2.
enumerator kENC_FilterPrescalerDiv4
Input filter prescaler is 4.
enumerator kENC_FilterPrescalerDiv8
Input filter prescaler is 8.
enumerator kENC_FilterPrescalerDiv16
Input filter prescaler is 16.
enumerator kENC_FilterPrescalerDiv32
Input filter prescaler is 32.
enumerator kENC_FilterPrescalerDiv64
Input filter prescaler is 64.
enumerator kENC_FilterPrescalerDiv128
Input filter prescaler is 128.

typedef enum _enc_home_trigger_mode enc_home_trigger_mode_t

Define HOME signal's trigger mode.

The ENC would count the trigger from HOME signal line.

typedef enum _enc_index_trigger_mode enc_index_trigger_mode_t

Define INDEX signal's trigger mode.

The ENC would count the trigger from INDEX signal line.

typedef enum _enc_decoder_work_mode enc_decoder_work_mode_t

Define type for decoder work mode.

The normal work mode uses the standard quadrature decoder with PHASEA and PHASEB. When in signal phase count mode, a positive transition of the PHASEA input generates a count signal while the PHASEB input and the reverse direction control the counter direction. If the reverse direction is not enabled, PHASEB = 0 means counting up and PHASEB = 1 means counting down. Otherwise, the direction is reversed.

```
typedef enum _enc_position_match_mode enc_position_match_mode_t
```

Define type for the condition of POSMATCH pulses.

```
typedef enum _enc_revolution_count_condition enc_revolution_count_condition_t
```

Define type for determining how the revolution counter (REV) is incremented/decremented.

```
typedef enum _enc_self_test_direction enc_self_test_direction_t
```

Define type for direction of self test generated signal.

```
typedef enum _enc_prescaler enc_prescaler_t
```

Define prescaler value for clock in CTRL3.

The clock is prescaled by a value of 2^{PRSC} which means that the prescaler logic can divide the clock by a minimum of 1 and a maximum of 32,768.

```
typedef enum _enc_filter_prescaler enc_filter_prescaler_t
```

Define input filter prescaler value.

The input filter prescaler value is to prescale the IPBus clock. (Frequency of FILT clock) = (Frequency of IPBus clock) / $2^{\text{FILT\_PRSC}}$.

```
typedef struct _enc_config enc_config_t
```

Define user configuration structure for ENC module.

```
typedef struct _enc_self_test_config enc_self_test_config_t
```

Define configuration structure for self test module.

The self test module provides a quadrature test signal to the inputs of the quadrature decoder module. This is a factory test feature. It is also useful to customers' software development and testing.

```
FSL_ENC_DRIVER_VERSION
```

```
struct _enc_config
```

`#include <fsl_enc.h>` Define user configuration structure for ENC module.

Public Members

```
bool enableReverseDirection
```

Enable reverse direction counting.

```
enc_decoder_work_mode_t decoderWorkMode
```

Enable signal phase count mode.

```
enc_home_trigger_mode_t HOMETriggerMode
```

Enable HOME to initialize position counters.

```
enc_index_trigger_mode_t INDEXTriggerMode
```

Enable INDEX to initialize position counters.

```
bool enableTRIGGERClearPositionCounter
```

Clear POSD, REV, UPOS and LPOS on rising edge of TRIGGER, or not.

```
bool enableTRIGGERClearHoldPositionCounter
```

Enable update of hold registers on rising edge of TRIGGER, or not.

```
bool enableWatchdog
```

Enable the watchdog to detect if the target is moving or not.

uint16_t watchdogTimeoutValue

Watchdog timeout count value. It stores the timeout count for the quadrature decoder module watchdog timer. This field is only available when “enableWatchdog” = true. The available value is a 16-bit unsigned number.

enc_filter_prescaler_t filterPrescaler

Input filter prescaler.

uint16_t filterCount

Input Filter Sample Count. This value should be chosen to reduce the probability of noisy samples causing an incorrect transition to be recognized. The value represent the number of consecutive samples that must agree prior to the input filter accepting an input transition. A value of 0x0 represents 3 samples. A value of 0x7 represents 10 samples. The Available range is 0 - 7.

uint16_t filterSamplePeriod

Input Filter Sample Period. This value should be set such that the sampling period is larger than the period of the expected noise. This value represents the sampling period (in IPBus clock cycles) of the decoder input signals. The available range is 0 - 255.

enc_position_match_mode_t positionMatchMode

The condition of POSMATCH pulses.

uint32_t positionCompareValue

Position compare value. The available value is a 32-bit number.

enc_revolution_count_condition_t revolutionCountCondition

Revolution Counter Modulus Enable.

bool enableModuloCountMode

Enable Modulo Counting.

uint32_t positionModulusValue

Position modulus value. This value would be available only when “enableModuloCountMode” = true. The available value is a 32-bit number.

uint32_t positionInitialValue

Position initial value. The available value is a 32-bit number.

bool enablePeriodMeasurementFunction

Enable period measurement function.

enc_prescaler_t prescalerValue

The value of prescaler.

struct __enc_self_test_config

#include <fsl_enc.h> Define configuration structure for self test module.

The self test module provides a quadrature test signal to the inputs of the quadrature decoder module. This is a factory test feature. It is also useful to customers’ software development and testing.

Public Members

enc_self_test_direction_t signalDirection

Direction of self test generated signal.

uint16_t signalCount

Hold the number of quadrature advances to generate. The available range is 0 - 255.

uint16_t signalPeriod

Hold the period of quadrature phase in IPBus clock cycles. The available range is 0 - 31.

2.39 ENET: Ethernet MAC Driver

void ENET_GetDefaultConfig(*enet_config_t* *config)

Gets the ENET default configuration structure.

The purpose of this API is to get the default ENET MAC controller configure structure for ENET_Init(). User may use the initialized structure unchanged in ENET_Init(), or modify some fields of the structure before calling ENET_Init(). Example:

```
enet_config_t config;
ENET_GetDefaultConfig(&config);
```

Parameters

- config – The ENET mac controller configuration structure pointer.

status_t ENET_Up(*ENET_Type* *base, *enet_handle_t* *handle, const *enet_config_t* *config, const *enet_buffer_config_t* *bufferConfig, uint8_t *macAddr, uint32_t srcClock_Hz)

Initializes the ENET module.

This function initializes the module with the ENET configuration.

Note: ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining “ENET_ENHANCEDBUFFERDESCRIPTOR_MODE” and calling ENET_Ptp1588Configure() to configure the 1588 feature and related buffers after calling ENET_Up().

Parameters

- base – ENET peripheral base address.
- handle – ENET handler pointer.
- config – ENET mac configuration structure pointer. The “enet_config_t” type mac configuration return from ENET_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.
- bufferConfig – ENET buffer configuration structure pointer. The buffer configuration should be prepared for ENET Initialization. It is the start address of “ringNum” enet_buffer_config structures. To support added multi-ring features in some soc and compatible with the previous enet driver version. For single ring supported, this bufferConfig is a buffer configure structure pointer, for multi-ring supported and used case, this bufferConfig pointer should be a buffer configure structure array pointer.
- macAddr – ENET mac address of Ethernet device. This MAC address should be provided.
- srcClock_Hz – The internal module clock source for MII clock.

Return values

- kStatus_Success – Succeed to initialize the ethernet driver.

- `kStatus_ENET_InitMemoryFail` – Init fails since buffer memory is not enough.

`status_t ENET_Init(ENET_Type *base, enet_handle_t *handle, const enet_config_t *config, const enet_buffer_config_t *bufferConfig, uint8_t *macAddr, uint32_t srcClock_Hz)`

Initializes the ENET module.

This function ungates the module clock and initializes it with the ENET configuration.

Note: ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining “ENET_ENHANCEDBUFFERDESCRIPTOR_MODE” and calling `ENET_Ptp1588Configure()` to configure the 1588 feature and related buffers after calling `ENET_Init()`.

Parameters

- `base` – ENET peripheral base address.
- `handle` – ENET handler pointer.
- `config` – ENET mac configuration structure pointer. The “`enet_config_t`” type mac configuration return from `ENET_GetDefaultConfig` can be used directly. It is also possible to verify the Mac configuration using other methods.
- `bufferConfig` – ENET buffer configuration structure pointer. The buffer configuration should be prepared for ENET Initialization. It is the start address of “`ringNum`” `enet_buffer_config` structures. To support added multi-ring features in some soc and compatible with the previous enet driver version. For single ring supported, this `bufferConfig` is a buffer configure structure pointer, for multi-ring supported and used case, this `bufferConfig` pointer should be a buffer configure structure array pointer.
- `macAddr` – ENET mac address of Ethernet device. This MAC address should be provided.
- `srcClock_Hz` – The internal module clock source for MII clock.

Return values

- `kStatus_Success` – Succeed to initialize the ethernet driver.
- `kStatus_ENET_InitMemoryFail` – Init fails since buffer memory is not enough.

`void ENET_Down(ENET_Type *base)`

Stops the ENET module.

This function disables the ENET module.

Parameters

- `base` – ENET peripheral base address.

`void ENET_Deinit(ENET_Type *base)`

Deinitializes the ENET module.

This function gates the module clock, clears ENET interrupts, and disables the ENET module.

Parameters

- `base` – ENET peripheral base address.

static inline void ENET_Reset(ENET_Type *base)

Resets the ENET module.

This function restores the ENET module to reset state. Note that this function sets all registers to reset state. As a result, the ENET module can't work after calling this function.

Parameters

- base – ENET peripheral base address.

void ENET_SetMII(ENET_Type *base, *enet_mii_speed_t* speed, *enet_mii_duplex_t* duplex)

Sets the ENET MII speed and duplex.

This API is provided to dynamically change the speed and duplex for MAC.

Parameters

- base – ENET peripheral base address.
- speed – The speed of the RMII mode.
- duplex – The duplex of the RMII mode.

void ENET_SetSMI(ENET_Type *base, uint32_t srcClock_Hz, bool isPreambleDisabled)

Sets the ENET SMI(serial management interface)- MII management interface.

Parameters

- base – ENET peripheral base address.
- srcClock_Hz – This is the ENET module clock frequency. See clock distribution.
- isPreambleDisabled – The preamble disable flag.
 - true Enables the preamble.
 - false Disables the preamble.

static inline bool ENET_GetSMI(ENET_Type *base)

Gets the ENET SMI- MII management interface configuration.

This API is used to get the SMI configuration to check whether the MII management interface has been set.

Parameters

- base – ENET peripheral base address.

Returns

The SMI setup status true or false.

static inline uint32_t ENET_ReadSMIData(ENET_Type *base)

Reads data from the PHY register through an SMI interface.

Parameters

- base – ENET peripheral base address.

Returns

The data read from PHY

static inline void ENET_StartSMIWrite(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, *enet_mii_write_t* operation, uint16_t data)

Sends the MDIO IEEE802.3 Clause 22 format write command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated ENET_MDIOWrite() can be called. For customized requirements, implement with combining separated APIs.

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address. Range from 0 ~ 31.
- regAddr – The PHY register address. Range from 0 ~ 31.
- operation – The write operation.
- data – The data written to PHY.

```
static inline void ENET_StartSMIRRead(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr,  
                                     enet_mii_read_t operation)
```

Sends the MDIO IEEE802.3 Clause 22 format read command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated ENET_MDIORead() can be called. For customized requirements, implement with combining separated APIs.

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address. Range from 0 ~ 31.
- regAddr – The PHY register address. Range from 0 ~ 31.
- operation – The read operation.

```
status_t ENET_MDIOWrite(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t data)  
MDIO write with IEEE802.3 Clause 22 format.
```

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address. Range from 0 ~ 31.
- regAddr – The PHY register. Range from 0 ~ 31.
- data – The data written to PHY.

Returns

kStatus_Success MDIO access succeeds.

Returns

kStatus_Timeout MDIO access timeout.

```
status_t ENET_MDIORead(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t  
                       *pData)
```

MDIO read with IEEE802.3 Clause 22 format.

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address. Range from 0 ~ 31.
- regAddr – The PHY register. Range from 0 ~ 31.
- pData – The data read from PHY.

Returns

kStatus_Success MDIO access succeeds.

Returns

kStatus_Timeout MDIO access timeout.

```
static inline void ENET_StartExtC45SMIWriteReg(ENET_Type *base, uint8_t portAddr, uint8_t
                                              devAddr, uint16_t regAddr)
```

Sends the MDIO IEEE802.3 Clause 45 format write register command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated ENET_MDIO_C45_Write()/ENET_MDIO_C45_Read() can be called. For customized requirements, implement with combining separated APIs.

Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.

```
static inline void ENET_StartExtC45SMIWriteData(ENET_Type *base, uint8_t portAddr, uint8_t
                                              devAddr, uint16_t data)
```

Sends the MDIO IEEE802.3 Clause 45 format write data command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated ENET_MDIO_C45_Write() can be called. For customized requirements, implement with combining separated APIs.

Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- data – The data written to PHY.

```
static inline void ENET_StartExtC45SMIReadData(ENET_Type *base, uint8_t portAddr, uint8_t
                                              devAddr)
```

Sends the MDIO IEEE802.3 Clause 45 format read data command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated ENET_MDIO_C45_Read() can be called. For customized requirements, implement with combining separated APIs.

Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.

```
status_t ENET_MDIO_C45_Write(ENET_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t
                             regAddr, uint16_t data)
```

MDIO write with IEEE802.3 Clause 45 format.

Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.
- data – The data written to PHY.

Returns

kStatus_Success MDIO access succeeds.

Returns

kStatus_Timeout MDIO access timeout.

status_t ENET_MDIORead(ENET_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr, uint16_t *pData)

MDIO read with IEEE802.3 Clause 45 format.

Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.
- pData – The data read from PHY.

Returns

kStatus_Success MDIO access succeeds.

Returns

kStatus_Timeout MDIO access timeout.

static inline void ENET_SetRGMIIClockDelay(ENET_Type *base, bool txEnabled, bool rxEnabled)

Control the usage of the delayed tx/rx RGMII clock.

Parameters

- base – ENET peripheral base address.
- txEnabled – Enable or disable to generate the delayed version of RGMII_TXC.
- rxEnabled – Enable or disable to use the delayed version of RGMII_RXC.

void ENET_SetMacAddr(ENET_Type *base, uint8_t *macAddr)

Sets the ENET module Mac address.

Parameters

- base – ENET peripheral base address.
- macAddr – The six-byte Mac address pointer. The pointer is allocated by application and input into the API.

void ENET_GetMacAddr(ENET_Type *base, uint8_t *macAddr)

Gets the ENET module Mac address.

Parameters

- base – ENET peripheral base address.
- macAddr – The six-byte Mac address pointer. The pointer is allocated by application and input into the API.

void ENET_AddMulticastGroup(ENET_Type *base, uint8_t *address)

Adds the ENET device to a multicast group.

Parameters

- base – ENET peripheral base address.
- address – The six-byte multicast group address which is provided by application.

```
void ENET_LeaveMulticastGroup(ENET_Type *base, uint8_t *address)
```

Moves the ENET device from a multicast group.

Parameters

- base – ENET peripheral base address.
- address – The six-byte multicast group address which is provided by application.

```
static inline void ENET_ActiveRead(ENET_Type *base)
```

Activates frame reception for multiple rings.

This function is to active the enet read process.

Note: This must be called after the MAC configuration and state are ready. It must be called after the ENET_Init(). This should be called when the frame reception is required.

Parameters

- base – ENET peripheral base address.

```
static inline void ENET_EnableSleepMode(ENET_Type *base, bool enable)
```

Enables/disables the MAC to enter sleep mode. This function is used to set the MAC enter sleep mode. When entering sleep mode, the magic frame wakeup interrupt should be enabled to wake up MAC from the sleep mode and reset it to normal mode.

Parameters

- base – ENET peripheral base address.
- enable – True enable sleep mode, false disable sleep mode.

```
static inline void ENET_GetAccelFunction(ENET_Type *base, uint32_t *txAccelOption, uint32_t *rxAccelOption)
```

Gets ENET transmit and receive accelerator functions from MAC controller.

Parameters

- base – ENET peripheral base address.
- txAccelOption – The transmit accelerator option. The “enet_tx_accelerator_t” is recommended to be used to as the mask to get the exact the accelerator option.
- rxAccelOption – The receive accelerator option. The “enet_rx_accelerator_t” is recommended to be used to as the mask to get the exact the accelerator option.

```
static inline void ENET_EnableInterrupts(ENET_Type *base, uint32_t mask)
```

Enables the ENET interrupt.

This function enables the ENET interrupt according to the provided mask. The mask is a logical OR of enumeration members. See enet_interrupt_enable_t. For example, to enable the TX frame interrupt and RX frame interrupt, do the following.

```
ENET_EnableInterrupts(ENET, kENET_TxFrameInterrupt | kENET_RxFrameInterrupt);
```

Parameters

- base – ENET peripheral base address.
- mask – ENET interrupts to enable. This is a logical OR of the enumeration enet_interrupt_enable_t.

```
static inline void ENET_DisableInterrupts(ENET_Type *base, uint32_t mask)
```

Disables the ENET interrupt.

This function disables the ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See `enet_interrupt_enable_t`. For example, to disable the TX frame interrupt and RX frame interrupt, do the following.

```
ENET_DisableInterrupts(ENET, kENET_TxFrameInterrupt | kENET_RxFrameInterrupt);
```

Parameters

- `base` – ENET peripheral base address.
- `mask` – ENET interrupts to disable. This is a logical OR of the enumeration `enet_interrupt_enable_t`.

```
static inline uint32_t ENET_GetInterruptStatus(ENET_Type *base)
```

Gets the ENET interrupt status flag.

Parameters

- `base` – ENET peripheral base address.

Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration `enet_interrupt_enable_t`.

```
static inline void ENET_ClearInterruptStatus(ENET_Type *base, uint32_t mask)
```

Clears the ENET interrupt events status flag.

This function clears enabled ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See the `enet_interrupt_enable_t`. For example, to clear the TX frame interrupt and RX frame interrupt, do the following.

```
ENET_ClearInterruptStatus(ENET, kENET_TxFrameInterrupt | kENET_RxFrameInterrupt);
```

Parameters

- `base` – ENET peripheral base address.
- `mask` – ENET interrupt source to be cleared. This is the logical OR of members of the enumeration `enet_interrupt_enable_t`.

```
void ENET_SetRxISRHandler(ENET_Type *base, enet_isr_t ISRHandler)
```

Set the second level Rx IRQ handler.

Parameters

- `base` – ENET peripheral base address.
- `ISRHandler` – The handler to install.

```
void ENET_SetTxISRHandler(ENET_Type *base, enet_isr_t ISRHandler)
```

Set the second level Tx IRQ handler.

Parameters

- `base` – ENET peripheral base address.
- `ISRHandler` – The handler to install.

```
void ENET_SetErrISRHandler(ENET_Type *base, enet_isr_t ISRHandler)
```

Set the second level Err IRQ handler.

Parameters

- `base` – ENET peripheral base address.

- ISRHandler – The handler to install.

void ENET_GetRxErrBeforeReadFrame(*enet_handle_t* *handle, *enet_data_error_stats_t* *eErrorStatic, uint8_t ringId)

Gets the error statistics of a received frame for ENET specified ring.

This API must be called after the ENET_GetRxFrameSize and before the ENET_ReadFrame(). If the ENET_GetRxFrameSize returns kStatus_ENET_RxFrameError, the ENET_GetRxErrBeforeReadFrame can be used to get the exact error statistics. This is an example.

```
status = ENET_GetRxFrameSize(&g_handle, &length, 0);
if (status == kStatus_ENET_RxFrameError)
{
    Comments: Get the error information of the received frame.
    ENET_GetRxErrBeforeReadFrame(&g_handle, &eErrStatic, 0);
    Comments: update the receive buffer.
    ENET_ReadFrame(EXAMPLE_ENET, &g_handle, NULL, 0);
}
```

Parameters

- handle – The ENET handler structure pointer. This is the same handler pointer used in the ENET_Init.
- eErrorStatic – The error statistics structure pointer.
- ringId – The ring index, range from 0 ~ (FSL_FEATURE_ENET_INSTANCE_QUEUEEn(x) - 1).

void ENET_EnableStatistics(ENET_Type *base, bool enable)

Enables/disables collection of transfer statistics.

Note that this function does not reset any of the already collected data, use the function ENET_ResetStatistics to clear the transfer statistics if needed.

Parameters

- base – ENET peripheral base address.
- enable – True enable statistics collection, false disable statistics collection.

void ENET_GetStatistics(ENET_Type *base, *enet_transfer_stats_t* *statistics)

Gets transfer statistics.

Copies the actual value of hardware counters into the provided structure. Calling this function does not reset the counters in hardware.

Parameters

- base – ENET peripheral base address.
- statistics – The statistics structure pointer.

void ENET_ResetStatistics(ENET_Type *base)

Resets transfer statistics.

Sets the value of hardware transfer counters to zero.

Parameters

- base – ENET peripheral base address.

status_t ENET_GetRxFrameSize(*enet_handle_t* *handle, uint32_t *length, uint8_t ringId)

Gets the size of the read frame for specified ring.

This function gets a received frame size from the ENET buffer descriptors.

Note: The FCS of the frame is automatically removed by MAC and the size is the length without the FCS. After calling ENET_GetRxFrameSize, ENET_ReadFrame() should be called to receive frame and update the BD if the result is not “kStatus_ENET_RxFrameEmpty”.

Parameters

- `handle` – The ENET handler structure. This is the same handler pointer used in the ENET_Init.
- `length` – The length of the valid frame received.
- `ringId` – The ring index or ring number.

Return values

- `kStatus_ENET_RxFrameEmpty` – No frame received. Should not call ENET_ReadFrame to read frame.
- `kStatus_ENET_RxFrameError` – Data error happens. ENET_ReadFrame should be called with NULL data and NULL length to update the receive buffers.
- `kStatus_Success` – Receive a frame Successfully then the ENET_ReadFrame should be called with the right data buffer and the captured data length input.

`status_t` ENET_ReadFrame(ENET_Type *base, *enet_handle_t* *handle, uint8_t *data, uint32_t length, uint8_t ringId, uint32_t *ts)

Reads a frame from the ENET device. This function reads a frame (both the data and the length) from the ENET buffer descriptors. User can get timestamp through ts pointer if the ts is not NULL.

Note: It doesn't store the timestamp in the receive timestamp queue. The ENET_GetRxFrameSize should be used to get the size of the prepared data buffer. This API uses memcpy to copy data from DMA buffer to application buffer; 4 bytes aligned data buffer in 32 bits platforms provided by user may let compiler use optimization instruction to reduce time consumption. This is an example:

```
uint32_t length;
enet_handle_t g_handle;
Comments: Get the received frame size firstly.
status = ENET_GetRxFrameSize(&g_handle, &length, 0);
if (length != 0)
{
    Comments: Allocate memory here with the size of "length"
    uint8_t *data = memory allocate interface;
    if (!data)
    {
        ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0, NULL);
        Comments: Add the console warning log.
    }
    else
    {
        status = ENET_ReadFrame(ENET, &g_handle, data, length, 0, NULL);
        Comments: Call stack input API to deliver the data to stack
    }
}
else if (status == kStatus_ENET_RxFrameError)
{
    Comments: Update the received buffer when a error frame is received.
```

(continues on next page)

(continued from previous page)

```

    ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0, NULL);
}

```

Parameters

- `base` – ENET peripheral base address.
- `handle` – The ENET handler structure. This is the same handler pointer used in the `ENET_Init`.
- `data` – The data buffer provided by user to store the frame which memory size should be at least “length”.
- `length` – The size of the data buffer which is still the length of the received frame.
- `ringId` – The ring index or ring number.
- `ts` – The timestamp address to store received timestamp.

Returns

The execute status, successful or failure.

```

status_t ENET_SendFrame(ENET_Type *base, enet_handle_t *handle, const uint8_t *data,
                        uint32_t length, uint8_t ringId, bool tsFlag, void *context)

```

Transmits an ENET frame for specified ring.

Note: The CRC is automatically appended to the data. Input the data to send without the CRC. This API uses memcpy to copy data from DMA buffer to application buffer; 4 bytes aligned data buffer in 32 bits platforms provided by user may let compiler use optimization instruction to reduce time consumption.

Parameters

- `base` – ENET peripheral base address.
- `handle` – The ENET handler pointer. This is the same handler pointer used in the `ENET_Init`.
- `data` – The data buffer provided by user to send.
- `length` – The length of the data to send.
- `ringId` – The ring index or ring number.
- `tsFlag` – Timestamp enable flag.
- `context` – Used by user to handle some events after transmit over.

Return values

- `kStatus_Success` – Send frame succeed.
- `kStatus_ENET_TxFrameBusy` – Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity. The waiting mechanism is recommended to be added after each call return with `kStatus_ENET_TxFrameBusy`.

```

status_t ENET_SetTxReclaim(enet_handle_t *handle, bool isEnabled, uint8_t ringId)

```

Enable or disable tx descriptors reclaim mechanism.

Note: This function must be called when no pending send frame action. Set enable if you want to reclaim context or timestamp in interrupt.

Parameters

- `handle` – The ENET handler pointer. This is the same handler pointer used in the `ENET_Init`.
- `isEnabled` – Enable or disable flag.
- `ringId` – The ring index or ring number.

Return values

- `kStatus_Success` – Succeed to enable/disable Tx reclaim.
- `kStatus_Fail` – Fail to enable/disable Tx reclaim.

`void ENET_ReclaimTxDescriptor(ENET_Type *base, enet_handle_t *handle, uint8_t ringId)`

Reclaim tx descriptors. This function is used to update the tx descriptor status and store the tx timestamp when the 1588 feature is enabled. This is called by the transmit interrupt IRQ handler after the complete of a frame transmission.

Parameters

- `base` – ENET peripheral base address.
- `handle` – The ENET handler pointer. This is the same handler pointer used in the `ENET_Init`.
- `ringId` – The ring index or ring number.

`status_t ENET_GetRxFrame(ENET_Type *base, enet_handle_t *handle, enet_rx_frame_struct_t *rxFrame, uint8_t ringId)`

Receives one frame in specified BD ring with zero copy.

This function uses the user-defined allocation and free callbacks. Every time application gets one frame through this function, driver stores the buffer address(es) in `enet_buffer_struct_t` and allocate new buffer(s) for the BD(s). If there's no memory buffer in the pool, this function drops current one frame to keep the Rx frame in BD ring is as fresh as possible.

Note: Application must provide a memory pool including at least BD number + n buffers in order for this function to work properly, because each BD must always take one buffer while driver is running, then other extra n buffer(s) can be taken by application. Here n is the `ceil(max_frame_length(set by RCR) / bd_rx_size(set by MRBR))`. Application must also provide an array structure in `rxFrame->rxBuffArray` with n index to receive one complete frame in any case.

Parameters

- `base` – ENET peripheral base address.
- `handle` – The ENET handler pointer. This is the same handler pointer used in the `ENET_Init`.
- `rxFrame` – The received frame information structure provided by user.
- `ringId` – The ring index or ring number.

Return values

- `kStatus_Success` – Succeed to get one frame and allocate new memory for Rx buffer.

- `kStatus_ENET_RxFrameEmpty` – There's no Rx frame in the BD.
- `kStatus_ENET_RxFrameError` – There's issue in this receiving.
- `kStatus_ENET_RxFrameDrop` – There's no new buffer memory for BD, drop this frame.

`status_t` ENET_StartTxFrame(ENET_Type *base, *enet_handle_t* *handle, *enet_tx_frame_struct_t* *txFrame, `uint8_t` ringId)

Sends one frame in specified BD ring with zero copy.

This function supports scattered buffer transmit, user needs to provide the buffer array.

Note: Tx reclaim should be enabled to ensure the Tx buffer ownership can be given back to application after Tx is over.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer. This is the same handler pointer used in the `ENET_Init`.
- txFrame – The Tx frame structure.
- ringId – The ring index or ring number.

Return values

- `kStatus_Success` – Succeed to send one frame.
- `kStatus_ENET_TxFrameBusy` – The BD is not ready for Tx or the reclaim operation still not finishes.
- `kStatus_ENET_TxFrameOverLen` – The Tx frame length is over max ethernet frame length.

`void` ENET_TransmitIRQHandler(ENET_Type *base, *enet_handle_t* *handle)

The transmit IRQ handler.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer.

`void` ENET_ReceiveIRQHandler(ENET_Type *base, *enet_handle_t* *handle)

The receive IRQ handler.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer.

`void` ENET_ErrorIRQHandler(ENET_Type *base, *enet_handle_t* *handle)

Some special IRQ handler including the error, mii, wakeup irq handler.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer.

`void` ENET_Ptp1588IRQHandler(ENET_Type *base)

the common IRQ handler for the 1588 irq handler.

This is used for the 1588 timer interrupt.

Parameters

- base – ENET peripheral base address.

void ENET_CommonFrame0IRQHandler(ENET_Type *base)

the common IRQ handler for the tx/rx/error etc irq handler.

This is used for the combined tx/rx/error interrupt for single/multi-ring (frame 0).

Parameters

- base – ENET peripheral base address.

FSL_ENET_DRIVER_VERSION

Defines the driver version.

ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK

Empty bit mask.

ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK

Software owner one mask.

ENET_BUFFDESCRIPTOR_RX_WRAP_MASK

Next buffer descriptor is the start address.

ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask

Software owner two mask.

ENET_BUFFDESCRIPTOR_RX_LAST_MASK

Last BD of the frame mask.

ENET_BUFFDESCRIPTOR_RX_MISS_MASK

Received because of the promiscuous mode.

ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK

Broadcast packet mask.

ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK

Multicast packet mask.

ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK

Length violation mask.

ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK

Non-octet aligned frame mask.

ENET_BUFFDESCRIPTOR_RX_CRC_MASK

CRC error mask.

ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK

FIFO overrun mask.

ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK

Frame is truncated mask.

ENET_BUFFDESCRIPTOR_TX_READY_MASK

Ready bit mask.

ENET_BUFFDESCRIPTOR_TX_SOFTOWNER1_MASK

Software owner one mask.

ENET_BUFFDESCRIPTOR_TX_WRAP_MASK

Wrap buffer descriptor mask.

ENET_BUFFDESCRIPTOR_TX_SOFTOWNER2_MASK

Software owner two mask.

ENET_BUFFDESCRIPTOR_TX_LAST_MASK

Last BD of the frame mask.

ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK

Transmit CRC mask.

ENET_FRAME_MAX_FRAMELEN

Default maximum Ethernet frame size without VLAN tag.

ENET_FRAME_VLAN_TAGLEN

Ethernet single VLAN tag size.

ENET_FRAME_CRC_LEN

CRC size in a frame.

ENET_FRAME_TX_LEN_LIMITATION(x)

ENET_FIFO_MIN_RX_FULL

ENET minimum receive FIFO full.

ENET_RX_MIN_BUFFERSIZE

ENET minimum buffer size.

ENET_PHY_MAXADDRESS

Maximum PHY address.

ENET_TX_INTERRUPT

Enet Tx interrupt flag.

ENET_RX_INTERRUPT

Enet Rx interrupt flag.

ENET_TS_INTERRUPT

Enet timestamp interrupt flag.

ENET_ERR_INTERRUPT

Enet error interrupt flag.

Defines the status return codes for transaction.

Values:

enumerator kStatus_ENET_InitMemoryFail

Init fails since buffer memory is not enough.

enumerator kStatus_ENET_RxFrameError

A frame received but data error happen.

enumerator kStatus_ENET_RxFrameFail

Failed to receive a frame.

enumerator kStatus_ENET_RxFrameEmpty

No frame arrive.

enumerator kStatus_ENET_RxFrameDrop

Rx frame is dropped since no buffer memory.

enumerator kStatus_ENET_TxFrameOverLen

Tx frame over length.

enumerator kStatus_ENET_TxFrameBusy

Tx buffer descriptors are under process.

enumerator kStatus_ENET_TxFrameFail

Transmit frame fail.

enum _enet_mii_mode

Defines the MII/RMII/RGMII mode for data interface between the MAC and the PHY.

Values:

enumerator kENET_MiiMode

MII mode for data interface.

enumerator kENET_RmiiMode

RMII mode for data interface.

enumerator kENET_RgmiiMode

RGMII mode for data interface.

enum _enet_mii_speed

Defines the 10/100/1000 Mbps speed for the MII data interface.

Notice: “kENET_MiiSpeed1000M” only supported when mii mode is “kENET_RgmiiMode”.

Values:

enumerator kENET_MiiSpeed10M

Speed 10 Mbps.

enumerator kENET_MiiSpeed100M

Speed 100 Mbps.

enumerator kENET_MiiSpeed1000M

Speed 1000M bps.

enum _enet_mii_duplex

Defines the half or full duplex for the MII data interface.

Values:

enumerator kENET_MiiHalfDuplex

Half duplex mode.

enumerator kENET_MiiFullDuplex

Full duplex mode.

enum _enet_mii_write

Define the MII opcode for normal MDIO_CLAUSES_22 Frame.

Values:

enumerator kENET_MiiWriteNoCompliant

Write frame operation, but not MII-compliant.

enumerator kENET_MiiWriteValidFrame

Write frame operation for a valid MII management frame.

enum _enet_mii_read

Defines the read operation for the MII management frame.

Values:

enumerator kENET_MiiReadValidFrame

Read frame operation for a valid MII management frame.

enumerator kENET_MiiReadNoCompliant
Read frame operation, but not MII-compliant.

enum _enet_mii_extend_opcode
Define the MII opcode for extended MDIO_CLAUSES_45 Frame.

Values:

enumerator kENET_MiiAddrWrite_C45
Address Write operation.

enumerator kENET_MiiWriteFrame_C45
Write frame operation for a valid MII management frame.

enumerator kENET_MiiReadFrame_C45
Read frame operation for a valid MII management frame.

enum _enet_special_control_flag
Defines a special configuration for ENET MAC controller.

These control flags are provided for special user requirements. Normally, these control flags are unused for ENET initialization. For special requirements, set the flags to mac-SpecialConfig in the enet_config_t. The kENET_ControlStoreAndFwdDisable is used to disable the FIFO store and forward. FIFO store and forward means that the FIFO read/send is started when a complete frame is stored in TX/RX FIFO. If this flag is set, configure rxFifo-FullThreshold and txFifoWatermark in the enet_config_t.

Values:

enumerator kENET_ControlFlowControlEnable
Enable ENET flow control: pause frame.

enumerator kENET_ControlRxPayloadCheckEnable
Enable ENET receive payload length check.

enumerator kENET_ControlRxPadRemoveEnable
Padding is removed from received frames.

enumerator kENET_ControlRxBroadCastRejectEnable
Enable broadcast frame reject.

enumerator kENET_ControlMacAddrInsert
Enable MAC address insert.

enumerator kENET_ControlStoreAndFwdDisable
Enable FIFO store and forward.

enumerator kENET_ControlSMIPreambleDisable
Enable SMI preamble.

enumerator kENET_ControlPromiscuousEnable
Enable promiscuous mode.

enumerator kENET_ControlMIILoopEnable
Enable ENET MII loop back.

enumerator kENET_ControlVLANTagEnable
Enable normal VLAN (single vlan tag).

enumerator kENET_ControlSVLANEnable
Enable S-VLAN.

enumerator kENET_ControlVLANUseSecondTag
Enable extracting the second vlan tag for further processing.

enum _enet_interrupt_enable

List of interrupts supported by the peripheral. This enumeration uses one-bit encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

Values:

enumerator kENET__BabrInterrupt
 Babbling receive error interrupt source

enumerator kENET__BabtInterrupt
 Babbling transmit error interrupt source

enumerator kENET__GraceStopInterrupt
 Graceful stop complete interrupt source

enumerator kENET__TxFrameInterrupt
 TX FRAME interrupt source

enumerator kENET__TxBufferInterrupt
 TX BUFFER interrupt source

enumerator kENET__RxFrameInterrupt
 RX FRAME interrupt source

enumerator kENET__RxBufferInterrupt
 RX BUFFER interrupt source

enumerator kENET__MiiInterrupt
 MII interrupt source

enumerator kENET__EBusERInterrupt
 Ethernet bus error interrupt source

enumerator kENET__LateCollisionInterrupt
 Late collision interrupt source

enumerator kENET__RetryLimitInterrupt
 Collision Retry Limit interrupt source

enumerator kENET__UnderrunInterrupt
 Transmit FIFO underrun interrupt source

enumerator kENET__PayloadRxInterrupt
 Payload Receive error interrupt source

enumerator kENET__WakeupInterrupt
 WAKEUP interrupt source

enumerator kENET__TsAvailInterrupt
 TS AVAIL interrupt source for PTP

enumerator kENET__TsTimerInterrupt
 TS WRAP interrupt source for PTP

enum _enet_event

Defines the common interrupt event for callback use.

Values:

enumerator kENET__RxEvent
 Receive event.

enumerator kENET_TxEvent

Transmit event.

enumerator kENET_ErrEvent

Error event: BABR/BABT/EBERR/LC/RL/UN/PLR .

enumerator kENET_WakeUpEvent

Wake up from sleep mode event.

enumerator kENET_TimeStampEvent

Time stamp event.

enumerator kENET_TimeStampAvailEvent

Time stamp available event.

enum _enet_idle_slope

Defines certain idle slope for bandwidth fraction.

Values:

enumerator kENET_IdleSlope1

The bandwidth fraction is about 0.002.

enumerator kENET_IdleSlope2

The bandwidth fraction is about 0.003.

enumerator kENET_IdleSlope4

The bandwidth fraction is about 0.008.

enumerator kENET_IdleSlope8

The bandwidth fraction is about 0.02.

enumerator kENET_IdleSlope16

The bandwidth fraction is about 0.03.

enumerator kENET_IdleSlope32

The bandwidth fraction is about 0.06.

enumerator kENET_IdleSlope64

The bandwidth fraction is about 0.11.

enumerator kENET_IdleSlope128

The bandwidth fraction is about 0.20.

enumerator kENET_IdleSlope256

The bandwidth fraction is about 0.33.

enumerator kENET_IdleSlope384

The bandwidth fraction is about 0.43.

enumerator kENET_IdleSlope512

The bandwidth fraction is about 0.50.

enumerator kENET_IdleSlope640

The bandwidth fraction is about 0.56.

enumerator kENET_IdleSlope768

The bandwidth fraction is about 0.60.

enumerator kENET_IdleSlope896

The bandwidth fraction is about 0.64.

enumerator kENET_IdleSlope1024

The bandwidth fraction is about 0.67.

enumerator kENET_IdleSlope1152

The bandwidth fraction is about 0.69.

enumerator kENET_IdleSlope1280

The bandwidth fraction is about 0.71.

enumerator kENET_IdleSlope1408

The bandwidth fraction is about 0.73.

enumerator kENET_IdleSlope1536

The bandwidth fraction is about 0.75.

enum _enet_tx_accelerator

Defines the transmit accelerator configuration.

Note that the hardware does not insert ICMPv6 protocol checksums as mentioned in errata ERR052152.

Values:

enumerator kENET_TxAccelIsShift16Enabled

Transmit FIFO shift-16.

enumerator kENET_TxAccelIpCheckEnabled

Insert IP header checksum.

enumerator kENET_TxAccelProtoCheckEnabled

Insert protocol checksum (TCP, UDP, ICMPv4).

enum _enet_rx_accelerator

Defines the receive accelerator configuration.

Note that the hardware does not validate ICMPv6 protocol checksums as mentioned in errata ERR052152.

Values:

enumerator kENET_RxAccelPadRemoveEnabled

Padding removal for short IP frames.

enumerator kENET_RxAccelIpCheckEnabled

Discard with wrong IP header checksum.

enumerator kENET_RxAccelProtoCheckEnabled

Discard with wrong protocol checksum (TCP, UDP, ICMPv4).

enumerator kENET_RxAccelMacCheckEnabled

Discard with Mac layer errors.

enumerator kENET_RxAccelIsShift16Enabled

Receive FIFO shift-16.

typedef enum _enet_mii_mode enet_mii_mode_t

Defines the MII/RMII/RGMII mode for data interface between the MAC and the PHY.

typedef enum _enet_mii_speed enet_mii_speed_t

Defines the 10/100/1000 Mbps speed for the MII data interface.

Notice: “kENET_MiiSpeed1000M” only supported when mii mode is “kENET_RgmiiMode”.

typedef enum *_enet_mii_duplex* enet_mii_duplex_t

Defines the half or full duplex for the MII data interface.

typedef enum *_enet_mii_write* enet_mii_write_t

Define the MII opcode for normal MDIO_CLAUSES_22 Frame.

typedef enum *_enet_mii_read* enet_mii_read_t

Defines the read operation for the MII management frame.

typedef enum *_enet_mii_extend_opcode* enet_mii_extend_opcode

Define the MII opcode for extended MDIO_CLAUSES_45 Frame.

typedef enum *_enet_special_control_flag* enet_special_control_flag_t

Defines a special configuration for ENET MAC controller.

These control flags are provided for special user requirements. Normally, these control flags are unused for ENET initialization. For special requirements, set the flags to macSpecialConfig in the enet_config_t. The kENET_ControlStoreAndFwdDisable is used to disable the FIFO store and forward. FIFO store and forward means that the FIFO read/send is started when a complete frame is stored in TX/RX FIFO. If this flag is set, configure rxFifoFullThreshold and txFifoWatermark in the enet_config_t.

typedef enum *_enet_interrupt_enable* enet_interrupt_enable_t

List of interrupts supported by the peripheral. This enumeration uses one-bit encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

typedef enum *_enet_event* enet_event_t

Defines the common interrupt event for callback use.

typedef enum *_enet_idle_slope* enet_idle_slope_t

Defines certain idle slope for bandwidth fraction.

typedef enum *_enet_tx_accelerator* enet_tx_accelerator_t

Defines the transmit accelerator configuration.

Note that the hardware does not insert ICMPv6 protocol checksums as mentioned in errata ERR052152.

typedef enum *_enet_rx_accelerator* enet_rx_accelerator_t

Defines the receive accelerator configuration.

Note that the hardware does not validate ICMPv6 protocol checksums as mentioned in errata ERR052152.

typedef struct *_enet_rx_bd_struct* enet_rx_bd_struct_t

Defines the receive buffer descriptor structure for the little endian system.

typedef struct *_enet_tx_bd_struct* enet_tx_bd_struct_t

Defines the enhanced transmit buffer descriptor structure for the little endian system.

typedef struct *_enet_data_error_stats* enet_data_error_stats_t

Defines the ENET data error statistics structure.

typedef struct *_enet_rx_frame_error* enet_rx_frame_error_t

Defines the Rx frame error structure.

typedef struct *_enet_transfer_stats* enet_transfer_stats_t

Defines the ENET transfer statistics structure.

typedef struct *enet_frame_info* enet_frame_info_t

Defines the frame info structure.


```
typedef struct _enet_tx_dirty_ring enet_tx_dirty_ring_t
```

Defines the ENET transmit dirty addresses ring/queue structure.

```
typedef void (*enet_rx_alloc_callback_t)(ENET_Type *base, void *userData, uint8_t ringId)
```

Defines the ENET Rx memory buffer alloc function pointer.

```
typedef void (*enet_rx_free_callback_t)(ENET_Type *base, void *buffer, void *userData, uint8_t ringId)
```

Defines the ENET Rx memory buffer free function pointer.

```
typedef struct _enet_buffer_config enet_buffer_config_t
```

Defines the receive buffer descriptor configuration structure.

Note that for the internal DMA requirements, the buffers have a corresponding alignment requirements.

- a. The aligned receive and transmit buffer size must be evenly divisible by ENET_BUFF_ALIGNMENT. when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of “ENET_BUFF_ALIGNMENT” and the cache line size.
- b. The aligned transmit and receive buffer descriptor start address must be at least 64 bit aligned. However, it's recommended to be evenly divisible by ENET_BUFF_ALIGNMENT. buffer descriptors should be put in non-cacheable region when cache is enabled.
- c. The aligned transmit and receive data buffer start address must be evenly divisible by ENET_BUFF_ALIGNMENT. Receive buffers should be continuous with the total size equal to “rxBdNumber * rxBuffSizeAlign”. Transmit buffers should be continuous with the total size equal to “txBdNumber * txBuffSizeAlign”. when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of “ENET_BUFF_ALIGNMENT” and the cache line size.

```
typedef struct _enet_intcoalesce_config enet_intcoalesce_config_t
```

Defines the interrupt coalescing configure structure.

```
typedef struct _enet_avb_config enet_avb_config_t
```

Defines the ENET AVB Configure structure.

This is used for to configure the extended ring 1 and ring 2.

- a. The classification match format is $(CMP3 \ll 12) \mid (CMP2 \ll 8) \mid (CMP1 \ll 4) \mid CMP0$. composed of four 3-bit compared VLAN priority field cmp0~cmp3, cm0 ~ cmp3 are used in parallel.

If CMP1,2,3 are not unused, please set them to the same value as CMP0.

- a. The idleSlope is used to calculate the Band Width fraction, $BW \text{ fraction} = 1 / (1 + 512/idleSlope)$. For avb configuration, the BW fraction of Class 1 and Class 2 combined must not exceed 0.75.

```
typedef struct _enet_handle enet_handle_t
```

```
typedef void (*enet_callback_t)(ENET_Type *base, enet_handle_t *handle, enet_event_t event, enet_frame_info_t *frameInfo, void *userData)
```

ENET callback function.

```
typedef struct _enet_config enet_config_t
```

Defines the basic configuration structure for the ENET device.

Note:

- a. macSpecialConfig is used for a special control configuration, A logical OR of “enet_special_control_flag_t”. For a special configuration for MAC, set this parameter to 0.

- b. txWatermark is used for a cut-through operation. It is in steps of 64 bytes: 0/1 - 64 bytes written to TX FIFO before transmission of a frame begins. 2 - 128 bytes written to TX FIFO 3 - 192 bytes written to TX FIFO The maximum of txWatermark is 0x2F - 4032 bytes written to TX FIFO txWatermark allows minimizing the transmit latency to set the txWatermark to 0 or 1 or for larger bus access latency 3 or larger due to contention for the system bus.
- c. rxFifoFullThreshold is similar to the txWatermark for cut-through operation in RX. It is in 64-bit words. The minimum is ENET_FIFO_MIN_RX_FULL and the maximum is 0xFF. If the end of the frame is stored in FIFO and the frame size is smaller than the txWatermark, the frame is still transmitted. The rule is the same for rxFifoFullThreshold in the receive direction.
- d. When “kENET_ControlFlowControlEnable” is set in the macSpecialConfig, ensure that the pauseDuration, rxFifoEmptyThreshold, and rxFifoStatEmptyThreshold are set for flow control enabled case.
- e. When “kENET_ControlStoreAndFwdDisabled” is set in the macSpecialConfig, ensure that the rxFifoFullThreshold and txFifoWatermark are set for store and forward disable.
- f. The rxAccelerConfig and txAccelerConfig default setting with 0 - accelerator are disabled. The “enet_tx_accelerator_t” and “enet_rx_accelerator_t” are recommended to be used to enable the transmit and receive accelerator. After the accelerators are enabled, the store and forward feature should be enabled. As a result, kENET_ControlStoreAndFwdDisabled should not be set.
- g. The intCoalesceCfg can be used in the rx or tx enabled cases to decrease the CPU loading.

```
typedef struct _enet_tx_bd_ring enet_tx_bd_ring_t
```

Defines the ENET transmit buffer descriptor ring/queue structure.

```
typedef struct _enet_rx_bd_ring enet_rx_bd_ring_t
```

Defines the ENET receive buffer descriptor ring/queue structure.

```
typedef struct _enet_buffer_struct enet_buffer_struct_t
```

```
typedef struct _enet_rx_frame_attribute_struct enet_rx_frame_attribute_t
```

```
typedef struct _enet_rx_frame_struct enet_rx_frame_struct_t
```

```
typedef struct _enet_tx_frame_struct enet_tx_frame_struct_t
```

```
typedef void (*enet_isr_t)(ENET_Type *base, enet_handle_t *handle)
```

Define interrupt IRQ handler.

```
const clock_ip_name_t s_enetClock[]
```

Pointers to enet clocks for each instance.

```
const clock_ip_name_t s_enetExtraClock[]
```

```
uint32_t ENET_GetInstance(ENET_Type *base)
```

Get the ENET instance from peripheral base address.

Parameters

- base – ENET peripheral base address.

Returns

ENET instance.

```
ENET_BUFFDESCRIPTOR_RX_ERR_MASK
```

Defines the receive error status flag mask.

struct _enet_rx_bd_struct

#include <fsl_enet.h> Defines the receive buffer descriptor structure for the little endian system.

Public Members

uint16_t length

Buffer descriptor data length.

uint16_t control

Buffer descriptor control and status.

uint32_t buffer

Data buffer pointer.

struct _enet_tx_bd_struct

#include <fsl_enet.h> Defines the enhanced transmit buffer descriptor structure for the little endian system.

Public Members

uint16_t length

Buffer descriptor data length.

uint16_t control

Buffer descriptor control and status.

uint32_t buffer

Data buffer pointer.

struct _enet_data_error_stats

#include <fsl_enet.h> Defines the ENET data error statistics structure.

Public Members

uint32_t statsRxLenGreaterErr

Receive length greater than RCR[MAX_FL].

uint32_t statsRxAlignErr

Receive non-octet alignment/

uint32_t statsRxFcsErr

Receive CRC error.

uint32_t statsRxOverRunErr

Receive over run.

uint32_t statsRxTruncateErr

Receive truncate.

struct _enet_rx_frame_error

#include <fsl_enet.h> Defines the Rx frame error structure.

Public Members

bool statsRxTruncateErr

Receive truncate.

bool statsRxOverRunErr

Receive over run.

bool statsRxFcsErr

Receive CRC error.

bool statsRxAlignErr

Receive non-octet alignment.

bool statsRxLenGreaterErr

Receive length greater than RCR[MAX_FL].

struct _enet_transfer_stats

#include <fsl_enet.h> Defines the ENET transfer statistics structure.

Public Members

uint32_t statsRxFrameCount

Rx frame number.

uint32_t statsRxFrameOk

Good Rx frame number.

uint32_t statsRxCrcErr

Rx frame number with CRC error.

uint32_t statsRxAlignErr

Rx frame number with alignment error.

uint32_t statsRxDropInvalidSFD

Dropped frame number due to invalid SFD.

uint32_t statsRx FifoOverflowErr

Rx FIFO overflow count.

uint32_t statsTxFrameCount

Tx frame number.

uint32_t statsTxFrameOk

Good Tx frame number.

uint32_t statsTxCrcAlignErr

The transmit frame is error.

uint32_t statsTx FifoUnderRunErr

Tx FIFO underrun count.

struct enet_frame_info

#include <fsl_enet.h> Defines the frame info structure.

Public Members

void *context

User specified data

struct _enet_tx_dirty_ring

#include <fsl_enet.h> Defines the ENET transmit dirty addresses ring/queue structure.

Public Members

enet_frame_info_t *txDirtyBase

Dirty buffer descriptor base address pointer.

uint16_t txGenIdx

tx generate index.

uint16_t txConsumIdx

tx consume index.

uint16_t txRingLen

tx ring length.

bool isFull

tx ring is full flag.

struct _enet_buffer_config

#include <fsl_enet.h> Defines the receive buffer descriptor configuration structure.

Note that for the internal DMA requirements, the buffers have a corresponding alignment requirements.

- a. The aligned receive and transmit buffer size must be evenly divisible by ENET_BUFF_ALIGNMENT. when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of “ENET_BUFF_ALIGNMENT” and the cache line size.
- b. The aligned transmit and receive buffer descriptor start address must be at least 64 bit aligned. However, it's recommended to be evenly divisible by ENET_BUFF_ALIGNMENT. buffer descriptors should be put in non-cacheable region when cache is enabled.
- c. The aligned transmit and receive data buffer start address must be evenly divisible by ENET_BUFF_ALIGNMENT. Receive buffers should be continuous with the total size equal to “rxBdNumber * rxBuffSizeAlign”. Transmit buffers should be continuous with the total size equal to “txBdNumber * txBuffSizeAlign”. when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of “ENET_BUFF_ALIGNMENT” and the cache line size.

Public Members

uint16_t rxBdNumber

Receive buffer descriptor number.

uint16_t txBdNumber

Transmit buffer descriptor number.

uint16_t rxBuffSizeAlign

Aligned receive data buffer size.

uint16_t txBuffSizeAlign

Aligned transmit data buffer size.

volatile *enet_rx_bd_struct_t* *rxBdStartAddrAlign

Aligned receive buffer descriptor start address: should be non-cacheable.

volatile *enet_tx_bd_struct_t* *txBdStartAddrAlign

Aligned transmit buffer descriptor start address: should be non-cacheable.

uint8_t *rxBufferAlign

Receive data buffer start address.

```

uint8_t *txBufferAlign
    Transmit data buffer start address.

bool rxMaintainEnable
    Receive buffer cache maintain.

bool txMaintainEnable
    Transmit buffer cache maintain.

enet_frame_info_t *txFrameInfo
    Transmit frame information start address.

struct __enet_intcoalesce_config
    #include <fsl_enet.h> Defines the interrupt coalescing configure structure.

```

Public Members

```

uint8_t txCoalesceFrameCount[1]
    Transmit interrupt coalescing frame count threshold.

uint16_t txCoalesceTimeCount[1]
    Transmit interrupt coalescing timer count threshold.

uint8_t rxCoalesceFrameCount[1]
    Receive interrupt coalescing frame count threshold.

uint16_t rxCoalesceTimeCount[1]
    Receive interrupt coalescing timer count threshold.

struct __enet_avb_config
    #include <fsl_enet.h> Defines the ENET AVB Configure structure.

    This is used for to configure the extended ring 1 and ring 2.
    a. The classification match format is (CMP3 « 12) | (CMP2 « 8) | (CMP1 « 4) | CMP0. composed of four 3-bit compared VLAN priority field cmp0~cmp3, cm0 ~ cmp3 are used in parallel.

    If CMP1,2,3 are not unused, please set them to the same value as CMP0.
    a. The idleSlope is used to calculate the Band Width fraction, BW fraction = 1 / (1 + 512/idleSlope). For avb configuration, the BW fraction of Class 1 and Class 2 combined must not exceed 0.75.

```

Public Members

```

uint16_t rxClassifyMatch[1 - 1]
    The classification match value for the ring.

enet_idle_slope_t idleSlope[1 - 1]
    The idle slope for certian bandwidth fraction.

struct __enet_config
    #include <fsl_enet.h> Defines the basic configuration structure for the ENET device.

```

Note:

- a. macSpecialConfig is used for a special control configuration, A logical OR of “enet_special_control_flag_t”. For a special configuration for MAC, set this parameter to 0.

- b. txWatermark is used for a cut-through operation. It is in steps of 64 bytes: 0/1 - 64 bytes written to TX FIFO before transmission of a frame begins. 2 - 128 bytes written to TX FIFO 3 - 192 bytes written to TX FIFO The maximum of txWatermark is 0x2F - 4032 bytes written to TX FIFO txWatermark allows minimizing the transmit latency to set the txWatermark to 0 or 1 or for larger bus access latency 3 or larger due to contention for the system bus.
- c. rxFifoFullThreshold is similar to the txWatermark for cut-through operation in RX. It is in 64-bit words. The minimum is ENET_FIFO_MIN_RX_FULL and the maximum is 0xFF. If the end of the frame is stored in FIFO and the frame size is smaller than the txWatermark, the frame is still transmitted. The rule is the same for rxFifoFullThreshold in the receive direction.
- d. When “kENET_ControlFlowControlEnable” is set in the macSpecialConfig, ensure that the pauseDuration, rxFifoEmptyThreshold, and rxFifoStatEmptyThreshold are set for flow control enabled case.
- e. When “kENET_ControlStoreAndFwdDisabled” is set in the macSpecialConfig, ensure that the rxFifoFullThreshold and txFifoWatermark are set for store and forward disable.
- f. The rxAccelerConfig and txAccelerConfig default setting with 0 - accelerator are disabled. The “enet_tx_accelerator_t” and “enet_rx_accelerator_t” are recommended to be used to enable the transmit and receive accelerator. After the accelerators are enabled, the store and forward feature should be enabled. As a result, kENET_ControlStoreAndFwdDisabled should not be set.
- g. The intCoalesceCfg can be used in the rx or tx enabled cases to decrease the CPU loading.

Public Members

uint32_t macSpecialConfig

Mac special configuration. A logical OR of “enet_special_control_flag_t”.

uint32_t interrupt

Mac interrupt source. A logical OR of “enet_interrupt_enable_t”.

uint16_t rxMaxFrameLen

Receive maximum frame length.

enet_mii_mode_t miiMode

MII mode.

enet_mii_speed_t miiSpeed

MII Speed.

enet_mii_duplex_t miiDuplex

MII duplex.

uint8_t rxAccelerConfig

Receive accelerator, A logical OR of “enet_rx_accelerator_t”.

uint8_t txAccelerConfig

Transmit accelerator, A logical OR of “enet_tx_accelerator_t”.

uint16_t pauseDuration

For flow control enabled case: Pause duration.

uint8_t rxFifoEmptyThreshold

For flow control enabled case: when RX FIFO level reaches this value, it makes MAC generate XOFF pause frame.

uint8_t rxFifoStatEmptyThreshold

For flow control enabled case: number of frames in the receive FIFO, independent of size, that can be accept. If the limit is reached, reception continues and a pause frame is triggered.

uint8_t rxFifoFullThreshold

For store and forward disable case, the data required in RX FIFO to notify the MAC receive ready status.

uint8_t txFifoWatermark

For store and forward disable case, the data required in TX FIFO before a frame transmit start.

enet_intcoalesce_config_t *intCoalesceCfg

If the interrupt coalesce is not required in the ring n(0,1,2), please set to NULL.

uint8_t ringNum

Number of used rings. default with 1 — single ring.

enet_rx_alloc_callback_t rxBuffAlloc

Callback function to alloc memory, must be provided for zero-copy Rx.

enet_rx_free_callback_t rxBuffFree

Callback function to free memory, must be provided for zero-copy Rx.

enet_callback_t callback

General callback function.

void *userData

Callback function parameter.

struct _enet_tx_bd_ring

#include <fsl_enet.h> Defines the ENET transmit buffer descriptor ring/queue structure.

Public Members

volatile *enet_tx_bd_struct_t* *txBdBase

Buffer descriptor base address pointer.

uint16_t txGenIdx

The current available transmit buffer descriptor pointer.

uint16_t txConsumIdx

Transmit consume index.

volatile uint16_t txDescUsed

Transmit descriptor used number.

uint16_t txRingLen

Transmit ring length.

struct _enet_rx_bd_ring

#include <fsl_enet.h> Defines the ENET receive buffer descriptor ring/queue structure.

Public Members

volatile *enet_rx_bd_struct_t* *rxBdBase

Buffer descriptor base address pointer.

uint16_t rxGenIdx

The current available receive buffer descriptor pointer.

uint16_t rxRingLen

Receive ring length.

struct __enet_handle

#include <fsl_enet.h> Defines the ENET handler structure.

Public Members

enet_rx_bd_ring_t rxBdRing[1]

Receive buffer descriptor.

enet_tx_bd_ring_t txBdRing[1]

Transmit buffer descriptor.

uint16_t rxBuffSizeAlign[1]

Receive buffer size alignment.

uint16_t txBuffSizeAlign[1]

Transmit buffer size alignment.

bool rxMaintainEnable[1]

Receive buffer cache maintain.

bool txMaintainEnable[1]

Transmit buffer cache maintain.

uint8_t ringNum

Number of used rings.

enet_callback_t callback

Callback function.

void *userData

Callback function parameter.

enet_tx_dirty_ring_t txDirtyRing[1]

Ring to store tx frame information.

bool txReclaimEnable[1]

Tx reclaim enable flag.

enet_rx_alloc_callback_t rxBuffAlloc

Callback function to alloc memory for zero copy Rx.

enet_rx_free_callback_t rxBuffFree

Callback function to free memory for zero copy Rx.

uint8_t multicastCount[64]

Multicast collisions counter

uint32_t enetClock

The clock of enet peripheral, to caculate core cycles for PTP timestamp.

uint32_t tsDelayCount

The count of core cycles for PTP timestamp capture delay.

struct __enet_buffer_struct

#include <fsl_enet.h>

Public Members

void *buffer

The buffer store the whole or partial frame.

uint16_t length

The byte length of this buffer.

struct __enet_rx_frame_attribute_struct
#include <fsl_enet.h>

Public Members

bool promiscuous

This frame is received because of promiscuous mode.

struct __enet_rx_frame_struct
#include <fsl_enet.h>

Public Members

enet_buffer_struct_t *rxBuffArray
Rx frame buffer structure.

uint16_t totLen
Rx frame total length.

enet_rx_frame_attribute_t rxAttribute
Rx frame attribute structure.

enet_rx_frame_error_t rxFrameError
Rx frame error.

struct __enet_tx_frame_struct
#include <fsl_enet.h>

Public Members

enet_buffer_struct_t *txBuffArray
Tx frame buffer structure.

uint32_t txBuffNum
Buffer number of this Tx frame.

void *context
Driver reclaims and gives it in Tx over callback, usually store network packet header.

2.40 EWM: External Watchdog Monitor Driver

void EWM_Init(EWM_Type *base, const ewm_config_t *config)
Initializes the EWM peripheral.

This function is used to initialize the EWM. After calling, the EWM runs immediately according to the configuration. Note that, except for the interrupt enable control bit, other control bits and registers are write once after a CPU reset. Modifying them more than once generates a bus transfer error.

This is an example.

```
ewm_config_t config;  
EWM_GetDefaultConfig(&config);  
config.compareHighValue = 0xAAU;  
EWM_Init(ewm_base,&config);
```

Parameters

- base – EWM peripheral base address
- config – The configuration of the EWM

`void EWM_Deinit(EWM_Type *base)`

Deinitializes the EWM peripheral.

This function is used to shut down the EWM.

Parameters

- base – EWM peripheral base address

`void EWM_GetDefaultConfig(ewm_config_t *config)`

Initializes the EWM configuration structure.

This function initializes the EWM configuration structure to default values. The default values are as follows.

```
ewmConfig->enableEwm = true;  
ewmConfig->enableEwmInput = false;  
ewmConfig->setInputAssertLogic = false;  
ewmConfig->enableInterrupt = false;  
ewmConfig->ewm_lpo_clock_source_t = kEWM_LpoClockSource0;  
ewmConfig->prescaler = 0;  
ewmConfig->compareLowValue = 0;  
ewmConfig->compareHighValue = 0xFEU;
```

See also:

`ewm_config_t`

Parameters

- config – Pointer to the EWM configuration structure.

`static inline void EWM_EnableInterrupts(EWM_Type *base, uint32_t mask)`

Enables the EWM interrupt.

This function enables the EWM interrupt.

Parameters

- base – EWM peripheral base address
- mask – The interrupts to enable The parameter can be combination of the following source if defined
 - kEWM_InterruptEnable

`static inline void EWM_DisableInterrupts(EWM_Type *base, uint32_t mask)`

Disables the EWM interrupt.

This function enables the EWM interrupt.

Parameters

- base – EWM peripheral base address

- mask – The interrupts to disable The parameter can be combination of the following source if defined
 - kEWM_InterruptEnable

static inline uint32_t EWM_GetStatusFlags(EWM_Type *base)

Gets all status flags.

This function gets all status flags.

This is an example for getting the running flag.

```
uint32_t status;
status = EWM_GetStatusFlags(ewm_base) & kEWM_RunningFlag;
```

See also:

_ewm_status_flags_t

- True: a related status flag has been set.
- False: a related status flag is not set.

Parameters

- base – EWM peripheral base address

Returns

State of the status flag: asserted (true) or not-asserted (false).

void EWM_Refresh(EWM_Type *base)

Services the EWM.

This function resets the EWM counter to zero.

Parameters

- base – EWM peripheral base address

FSL_EWM_DRIVER_VERSION

EWM driver version 2.0.4.

enum _ewm_lpo_clock_source

Describes EWM clock source.

Values:

enumerator kEWM_LpoClockSource0

EWM clock sourced from lpo_clk[0]

enumerator kEWM_LpoClockSource1

EWM clock sourced from lpo_clk[1]

enumerator kEWM_LpoClockSource2

EWM clock sourced from lpo_clk[2]

enumerator kEWM_LpoClockSource3

EWM clock sourced from lpo_clk[3]

enum _ewm_interrupt_enable_t

EWM interrupt configuration structure with default settings all disabled.

This structure contains the settings for all of EWM interrupt configurations.

Values:

enumerator kEWM_InterruptEnable

Enable the EWM to generate an interrupt

enum `_ewm_status_flags_t`

EWM status flags.

This structure contains the constants for the EWM status flags for use in the EWM functions.

Values:

enumerator `kEWM_RunningFlag`

Running flag, set when EWM is enabled

typedef enum `_ewm_lpo_clock_source` `ewm_lpo_clock_source_t`

Describes EWM clock source.

typedef struct `_ewm_config` `ewm_config_t`

Data structure for EWM configuration.

This structure is used to configure the EWM.

struct `_ewm_config`

#include <fsl_ewm.h> Data structure for EWM configuration.

This structure is used to configure the EWM.

Public Members

bool `enableEwm`

Enable EWM module

bool `enableEwmInput`

Enable EWM_in input

bool `setInputAssertLogic`

EWM_in signal assertion state

bool `enableInterrupt`

Enable EWM interrupt

`ewm_lpo_clock_source_t` `clockSource`

Clock source select

uint8_t `prescaler`

Clock prescaler value

uint8_t `compareLowValue`

Compare low-register value

uint8_t `compareHighValue`

Compare high-register value

2.41 FlexCAN: Flex Controller Area Network Driver

2.42 FlexCAN Driver

bool `FLEXCAN_IsInstanceHasFDMode(CAN_Type *base)`

Determine whether the FlexCAN instance support CAN FD mode at run time.

Note: Use this API only if different soc parts share the SOC part name macro define. Otherwise, a different SOC part name can be used to determine at compile time whether the

FlexCAN instance supports CAN FD mode or not. If need use this API to determine if CAN FD mode is supported, the FLEXCAN_Init function needs to be executed first, and then call this API and use the return to value determines whether to supports CAN FD mode, if return true, continue calling FLEXCAN_FDInit to enable CAN FD mode.

Parameters

- base – FlexCAN peripheral base address.

Returns

return TRUE if instance support CAN FD mode, FALSE if instance only support classic CAN (2.0) mode.

uint32_t FLEXCAN_GetFDMailboxOffset(CAN_Type *base, uint8_t mbIdx)

Get Mailbox offset number by dword.

This function gets the offset number of the specified mailbox. Mailbox is not consecutive between memory regions when payload is not 8 bytes so need to calculate the specified mailbox address. For example, in the first memory region, MB[0].CS address is 0x4002_4080. For 32 bytes payload frame, the second mailbox is $((1/12)*512 + 1\%12*40)/4 = 10$, meaning 10 dword after the 0x4002_4080, which is actually the address of mailbox MB[1].CS.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – Mailbox index.

Returns

Mailbox address offset in word.

status_t FLEXCAN_EnterFreezeMode(CAN_Type *base)

Enter FlexCAN Freeze Mode.

This function makes the FlexCAN work under Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

Returns

kStatus_Success Enter Freeze Mode successful kStatus_Timeout Timeout when wait for Freeze Mode Acknowledge

status_t FLEXCAN_ExitFreezeMode(CAN_Type *base)

Exit FlexCAN Freeze Mode.

This function makes the FlexCAN leave Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

Returns

kStatus_Success Enter Freeze Mode successful kStatus_Timeout Timeout when wait for Freeze Mode Acknowledge

uint32_t FLEXCAN_GetInstance(CAN_Type *base)

Get the FlexCAN instance from peripheral base address.

Parameters

- base – FlexCAN peripheral base address.

Returns

FlexCAN instance.

```
bool FLEXCAN_CalculateImprovedTimingValues(CAN_Type *base, uint32_t bitRate, uint32_t
                                           sourceClock_Hz, flexcan_timing_config_t
                                           *pTimingConfig)
```

Calculates the improved timing values by specific bit Rates for classical CAN.

This function use to calculates the Classical CAN timing values according to the given bit rate. The Calculated timing values will be set in CTRL1/CBT/ENCBT register. The calculation is based on the recommendation of the CiA 301 v4.2.0 and previous version document.

Parameters

- base – FlexCAN peripheral base address.
- bitRate – The classical CAN speed in bps defined by user, should be less than or equal to 1Mbps.
- sourceClock_Hz – The Source clock frequency in Hz.
- pTimingConfig – Pointer to the FlexCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration.

```
void FLEXCAN_Init(CAN_Type *base, const flexcan_config_t *pConfig, uint32_t sourceClock_Hz)
```

Initializes a FlexCAN instance.

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the flexcan_config_t parameters and how to call the FLEXCAN_Init function by passing in these parameters.

```
flexcan_config_t flexcanConfig;
flexcanConfig.clkSrc      = kFLEXCAN_ClkSrc0;
flexcanConfig.bitRate     = 1000000U;
flexcanConfig.maxMbNum    = 16;
flexcanConfig.enableLoopBack = false;
flexcanConfig.enableSelfWakeup = false;
flexcanConfig.enableIndividMask = false;
flexcanConfig.enableDoze    = false;
flexcanConfig.disableSelfReception = false;
flexcanConfig.enableListenOnlyMode = false;
flexcanConfig.timingConfig = timingConfig;
FLEXCAN_Init(CAN0, &flexcanConfig, 4000000UL);
```

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the user-defined configuration structure.
- sourceClock_Hz – FlexCAN Protocol Engine clock source frequency in Hz.

```
bool FLEXCAN_FDCalculateImprovedTimingValues(CAN_Type *base, uint32_t bitRate, uint32_t
                                              bitRateFD, uint32_t sourceClock_Hz,
                                              flexcan_timing_config_t *pTimingConfig)
```

Calculates the improved timing values by specific bit rates for CANFD.

This function use to calculates the CANFD timing values according to the given nominal phase bit rate and data phase bit rate. The Calculated timing values will be set in CBT/ENCBT and FDCBT/EDCBT registers. The calculation is based on the recommendation of the CiA 1301 v1.0.0 document.

Parameters

- base – FlexCAN peripheral base address.
- bitRate – The CANFD bus control speed in bps defined by user.

- bitRateFD – The CAN FD data phase speed in bps defined by user. Equal to bitRate means disable bit rate switching.
- sourceClock_Hz – The Source clock frequency in Hz.
- pTimingConfig – Pointer to the FlexCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration

```
void FLEXCAN_FDInit(CAN_Type *base, const flexcan_config_t *pConfig, uint32_t
                    sourceClock_Hz, flexcan_mb_size_t dataSize, bool brs)
```

Initializes a FlexCAN instance.

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the flexcan_config_t parameters and how to call the FLEXCAN_FDInit function by passing in these parameters.

```
flexcan_config_t flexcanConfig;
flexcanConfig.clkSrc      = kFLEXCAN_ClkSrc0;
flexcanConfig.bitRate     = 1000000U;
flexcanConfig.bitRateFD   = 2000000U;
flexcanConfig.maxMbNum    = 16;
flexcanConfig.enableLoopBack = false;
flexcanConfig.enableSelfWakeup = false;
flexcanConfig.enableIndividMask = false;
flexcanConfig.disableSelfReception = false;
flexcanConfig.enableListenOnlyMode = false;
flexcanConfig.enableDoze    = false;
flexcanConfig.timingConfig = timingConfig;
FLEXCAN_FDInit(CAN0, &flexcanConfig, 8000000UL, kFLEXCAN_16BperMB, true);
```

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the user-defined configuration structure.
- sourceClock_Hz – FlexCAN Protocol Engine clock source frequency in Hz.
- dataSize – FlexCAN Message Buffer payload size. The actual transmitted or received CAN FD frame data size needs to be less than or equal to this value.
- brs – True if bit rate switch is enabled in FD mode.

```
void FLEXCAN_Deinit(CAN_Type *base)
```

De-initializes a FlexCAN instance.

This function disables the FlexCAN module clock and sets all register values to the reset value.

Parameters

- base – FlexCAN peripheral base address.

```
void FLEXCAN_GetDefaultConfig(flexcan_config_t *pConfig)
```

Gets the default configuration structure.

This function initializes the FlexCAN configuration structure to default values. The default values are as follows. flexcanConfig->clkSrc = kFLEXCAN_ClkSrc0; flexcanConfig->bitRate = 1000000U; flexcanConfig->bitRateFD = 2000000U; flexcanConfig->maxMbNum = 16; flexcanConfig->enableLoopBack = false; flexcanConfig->enableSelfWakeup = false; flexcanConfig->enableIndividMask = false; flexcanConfig->disableSelfReception = false; flexcanConfig->enableListenOnlyMode = false; flexcanConfig->enableDoze = false; flexcanConfig->


```
>enablePretendedeNetworking = false; flexcanConfig->enableMemoryErrorControl  
= true; flexcanConfig->enableNonCorrectableErrorEnterFreeze = true;  
flexcanConfig->enableTransceiverDelayMeasure = true; flexcanConfig->  
>enableRemoteRequestFrameStored = true; flexcanConfig->payloadEndianness = kFLEX-  
CAN_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_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 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_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.

void FLEXCAN__TransferCreateHandle(CAN_Type *base, flexcan_handle_t *handle,
flexcan_transfer_callback_t callback, void *userData)

Initializes the FlexCAN handle.

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

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- callback – The callback function.
- userData – The parameter of the callback function.

status_t FLEXCAN__TransferSendNonBlocking(CAN_Type *base, flexcan_handle_t *handle,
flexcan_mb_transfer_t *pMbXfer)

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus_Success – Start Tx Message Buffer sending process successfully.
- kStatus_Fail – Write Tx Message Buffer failed.
- kStatus_FLEXCAN__TxBusy – Tx Message Buffer is in use.

status_t FLEXCAN__TransferReceiveNonBlocking(CAN_Type *base, flexcan_handle_t *handle,
flexcan_mb_transfer_t *pMbXfer)

Receives a message using IRQ.

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus_Success – Start Rx Message Buffer receiving process successfully.
- kStatus_FLEXCAN__RxBusy – Rx Message Buffer is in use.

status_t FLEXCAN_TransferReceiveFifoNonBlocking(*CAN_Type* *base, *flexcan_handle_t* *handle, *flexcan_fifo_transfer_t* *pFifoXfer)

Receives a message from Rx FIFO using IRQ.

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pFifoXfer – FlexCAN Rx FIFO transfer structure. See the *flexcan_fifo_transfer_t*.

Return values

- kStatus_Success – Start Rx FIFO receiving process successfully.
- kStatus_FLEXCAN_RxFifoBusy – Rx FIFO is currently in use.

status_t FLEXCAN_TransferGetReceiveFifoCount(*CAN_Type* *base, *flexcan_handle_t* *handle, *size_t* *count)

Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

uint32_t FLEXCAN_GetTimeStamp(*flexcan_handle_t* *handle, *uint8_t* mbIdx)

Gets the detail index of Mailbox's Timestamp by handle.

Then function can only be used when calling non-blocking Data transfer (TX/RX) API, After TX/RX data transfer done (User can get the status by handler's callback function), we can get the detail index of Mailbox's timestamp by handle, Detail non-blocking data transfer API (TX/RX) contain. -FLEXCAN_TransferSendNonBlocking -FLEXCAN_TransferFDSendNonBlocking -FLEXCAN_TransferReceiveNonBlocking -FLEXCAN_TransferFDReceiveNonBlocking -FLEXCAN_TransferReceiveFifoNonBlocking

Parameters

- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

Return values

the – index of mailbox 's timestamp stored in the handle.

void FLEXCAN_TransferAbortSend(*CAN_Type* *base, *flexcan_handle_t* *handle, *uint8_t* mbIdx)

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

- mbIdx – The FlexCAN Message Buffer index.

void FLEXCAN_TransferAbortReceive(CAN_Type *base, flexcan_handle_t *handle, uint8_t mbIdx)

Aborts the interrupt driven message receive process.

This function aborts the interrupt driven message receive process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

void FLEXCAN_TransferAbortReceiveFifo(CAN_Type *base, flexcan_handle_t *handle)

Aborts the interrupt driven message receive from Rx FIFO process.

This function aborts the interrupt driven message receive from Rx FIFO process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_TransferHandleIRQ(CAN_Type *base, flexcan_handle_t *handle)

FlexCAN IRQ handle function.

This function handles the FlexCAN Error, the Message Buffer, and the Rx FIFO IRQ request.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_MbHandleIRQ(CAN_Type *base, flexcan_handle_t *handle, uint32_t startMbIdx, uint32_t endMbIdx)

FlexCAN Message Buffer IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- startMbIdx – First Message Buffer to handle.
- endMbIdx – Last Message Buffer to handle.

void FLEXCAN_BusoffErrorHandleIRQ(CAN_Type *base, flexcan_handle_t *handle)

FlexCAN Bus Off, Error and Warning IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

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`

UnHadled Interrupt asserted.

enumerator `kStatus_FLEXCAN_RxRemote`

Rx Remote Message Received in Mail box.

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_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.

enumerator kFLEXCAN_HostAccessNCErrInterruptEnable

Host Access With Non-Correctable Errors interrupt, use high word bit 0.

enumerator kFLEXCAN_FlexCanAccessNCErrInterruptEnable

FlexCAN Access With Non-Correctable Errors interrupt, use high word bit 2.

enumerator kFLEXCAN_HostOrFlexCanCErrInterruptEnable

Host or FlexCAN Access With Correctable Errors interrupt, use high word bit 3.

enum _flexcan_flags

FlexCAN status flags.

This provides constants for the FlexCAN status flags for use in the FlexCAN functions.

Note: The CPU read action clears the bits corresponding to the FLEXCAN_ErrorFlag macro, therefore user need to read status flags and distinguish which error is occur using _flexcan_error_flags enumerations.

Values:

enumerator kFLEXCAN_ErrorOverrunFlag

Error Overrun Status.

enumerator kFLEXCAN_FDErrorIntFlag

CAN FD Error Interrupt Flag.

enumerator kFLEXCAN_BusoffDoneIntFlag

Bus Off process completed Interrupt Flag.

enumerator kFLEXCAN_SynchFlag

CAN Synchronization Status.

enumerator kFLEXCAN_TxWarningIntFlag

Tx Warning Interrupt Flag.

enumerator kFLEXCAN_RxWarningIntFlag

Rx Warning Interrupt Flag.

enumerator kFLEXCAN_IdleFlag

FlexCAN In IDLE Status.

enumerator kFLEXCAN_FaultConfinementFlag

FlexCAN Fault Confinement State.

enumerator kFLEXCAN_TransmittingFlag

FlexCAN In Transmission Status.

enumerator kFLEXCAN_ReceivingFlag

FlexCAN In Reception Status.

enumerator kFLEXCAN_BusOffIntFlag

Bus Off Interrupt Flag.

enumerator kFLEXCAN_ErrorIntFlag
CAN Error Interrupt Flag.

enumerator kFLEXCAN_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_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_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_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.

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_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.

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`.

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_mb_transfer`

#include <fsl_flexcan.h> FlexCAN Message Buffer transfer.

Public Members

`flexcan_frame_t *frame`

The buffer of CAN Message to be transfer.

`uint8_t mbIdx`

The index of Message buffer used to transfer Message.

`struct _flexcan_fifo_transfer`

#include <fsl_flexcan.h> FlexCAN Rx FIFO transfer.

Public Members

`flexcan_frame_t *frame`

The buffer of CAN Message to be received from Legacy Rx FIFO.

`size_t frameNum`

Number of CAN Message need to be received from Legacy or 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.

`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 ____unnamed27____

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 ____unnamed29____

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 ____unnamed31____

Public Members

struct __flexcan_frame

struct __flexcan_frame

struct ____unnamed33____

Public Members

uint32_t dataWord0

CAN Frame payload word0.

uint32_t dataWord1

CAN Frame payload word1.

struct ____unnamed35____

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 ____unnamed37____

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 ____unnamed39____

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 ____unnamed41____

Public Members

struct _flexcan_fd_frame

struct _flexcan_fd_frame

struct ____unnamed43____

Public Members

uint32_t dataWord[16]
CAN FD Frame payload, 16 double word maximum.

struct ____unnamed45____

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 __unnamed47__
```

Public Members

```
struct __flexcan_config
struct __flexcan_config
struct __unnamed49__
```

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 __unnamed51__
```

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 __unnamed53__
```

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 __unnamed57__
    < 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 ____unnamed59____

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 ____unnamed55____

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 ____unnamed61____

< 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 ____unnamed63____
```

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.
```

2.43 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.

```
FSL_FLEXCAN_EDMA_DRIVER_VERSION
```

FlexCAN EDMA driver version.

```
typedef struct _flexcan_edma_handle flexcan_edma_handle_t
```

```
typedef void (*flexcan_edma_transfer_callback_t)(CAN_Type *base, flexcan_edma_handle_t
*handle, status_t status, void *userData)
```

FlexCAN transfer callback function.

```
struct _flexcan_edma_handle
```

```
#include <fsl_flexcan_edma.h> FlexCAN eDMA handle.
```

Public Members

```
flexcan_edma_transfer_callback_t callback
```

Callback function.

```
void *userData
```

FlexCAN callback function parameter.

```
edma_handle_t *rxFifoEdmaHandle
```

The EDMA handler for Rx FIFO.

```
volatile uint8_t rxFifoState
```

Rx FIFO transfer state.

```
size_t frameNum
```

The number of messages that need to be received.

2.44 FlexIO: FlexIO Driver

2.45 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_SHIFToNSTAT(0),
    .triggerPolarity = kFLEXIO_TimerTriggerPolarityActiveLow,
    .triggerSource = kFLEXIO_TimerTriggerSourceInternal,
    .pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
    .pinSelect = 0,
    .pinPolarity = kFLEXIO_PinActiveHigh,
    .timerMode = kFLEXIO_TimerModeDual8BitBaudBit,
    .timerOutput = kFLEXIO_TimerOutputZeroNotAffectedByReset,
    .timerDecrement = kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput,
    .timerReset = kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput,
    .timerDisable = kFLEXIO_TimerDisableOnTimerCompare,
    .timerEnable = kFLEXIO_TimerEnableOnTriggerHigh,
    .timerStop = kFLEXIO_TimerStopBitEnableOnTimerDisable,
    .timerStart = kFLEXIO_TimerStartBitEnabled
};
FLEXIO_SetTimerConfig(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- index – Timer index
- timerConfig – Pointer to the flexio_timer_config_t structure

```
static inline void FLEXIO_SetClockMode(FLEXIO_Type *base, uint8_t index,  
                                       flexio_timer_decrement_source_t clocksource)
```

This function set the value of the prescaler on flexio channels.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.
- index – Timer index
- clocksource – Set clock value

```
static inline void FLEXIO_EnableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
```

Enables the shifter status interrupt. The interrupt generates when the corresponding SSF is set.

Note: For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using $((1 \ll \text{shifter index0}) \mid (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$

```
static inline void FLEXIO_DisableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
```

Disables the shifter status interrupt. The interrupt won't generate when the corresponding SSF is set.

Note: For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using $((1 \ll \text{shifter index0}) \mid (1 \ll \text{shifter index1}))$

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 $((1 \ll \text{shifter index0}) \mid (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$

static inline void FLEXIO_DisableShifterErrorInterrupts(FLEXIO_Type *base, uint32_t mask)

Disables the shifter error interrupt. The interrupt won't generate when the corresponding SEF is set.

Note: For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$

static inline void FLEXIO_EnableTimerStatusInterrupts(FLEXIO_Type *base, uint32_t mask)

Enables the timer status interrupt. The interrupt generates when the corresponding SSF is set.

Note: For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by $(1 \ll \text{timer index})$

static inline void FLEXIO_DisableTimerStatusInterrupts(FLEXIO_Type *base, uint32_t mask)

Disables the timer status interrupt. The interrupt won't generate when the corresponding SSF is set.

Note: For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by $(1 \ll \text{timer index})$

static inline uint32_t FLEXIO_GetShifterStatusFlags(FLEXIO_Type *base)

Gets the shifter status flags.

Parameters

- base – FlexIO peripheral base address

Returns

Shifter status flags

static inline void FLEXIO_ClearShifterStatusFlags(FLEXIO_Type *base, uint32_t mask)

Clears the shifter status flags.

Note: For clearing multiple shifter status flags, for example, two shifter status flags, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address

- mask – The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$

static inline uint32_t FLEXIO_GetShifterErrorFlags(FLEXIO_Type *base)

Gets the shifter error flags.

Parameters

- base – FlexIO peripheral base address

Returns

Shifter error flags

static inline void FLEXIO_ClearShifterErrorFlags(FLEXIO_Type *base, uint32_t mask)

Clears the shifter error flags.

Note: For clearing multiple shifter error flags, for example, two shifter error flags, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$

static inline uint32_t FLEXIO_GetTimerStatusFlags(FLEXIO_Type *base)

Gets the timer status flags.

Parameters

- base – FlexIO peripheral base address

Returns

Timer status flags

static inline void FLEXIO_ClearTimerStatusFlags(FLEXIO_Type *base, uint32_t mask)

Clears the timer status flags.

Note: For clearing multiple timer status flags, for example, two timer status flags, can calculate the mask by using $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by $(1 \ll \text{timer index})$

static inline void FLEXIO_EnableShifterStatusDMA(FLEXIO_Type *base, uint32_t mask, bool enable)

Enables/disables the shifter status DMA. The DMA request generates when the corresponding SSF is set.

Note: For multiple shifter status DMA enables, for example, calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$

- enable – True to enable, false to disable.

uint32_t FLEXIO_GetShifterBufferAddress(FLEXIO_Type *base, flexio_shifter_buffer_type_t type, uint8_t index)

Gets the shifter buffer address for the DMA transfer usage.

Parameters

- base – FlexIO peripheral base address
- type – Shifter type of flexio_shifter_buffer_type_t
- index – Shifter index

Returns

Corresponding shifter buffer index

status_t FLEXIO_RegisterHandleIRQ(void *base, void *handle, flexio_isr_t isr)

Registers the handle and the interrupt handler for the FlexIO-simulated peripheral.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.
- handle – Pointer to the handler for FlexIO simulated peripheral.
- isr – FlexIO simulated peripheral interrupt handler.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

status_t FLEXIO_UnregisterHandleIRQ(void *base)

Unregisters the handle and the interrupt handler for the FlexIO-simulated peripheral.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

static inline void FLEXIO_ClearPortOutput(FLEXIO_Type *base, uint32_t mask)

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

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

static inline void FLEXIO_SetPortOutput(FLEXIO_Type *base, uint32_t mask)

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

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

static inline void FLEXIO_TogglePortOutput(FLEXIO_Type *base, uint32_t mask)

Reverses the current output logic of the multiple FLEXIO pins.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

static inline void FLEXIO_PinWrite(FLEXIO_Type *base, uint32_t pin, uint8_t output)

Sets the output level of the FLEXIO pins to the logic 1 or 0.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.
- output – FLEXIO pin output logic level.
 - 0: corresponding pin output low-logic level.
 - 1: corresponding pin output high-logic level.

static inline void FLEXIO_EnablePinOutput(FLEXIO_Type *base, uint32_t pin)

Enables the FLEXIO output pin function.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

static inline uint32_t FLEXIO_PinRead(FLEXIO_Type *base, uint32_t pin)

Reads the current input value of the FLEXIO pin.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

Return values

FLEXIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

static inline uint32_t FLEXIO_GetPinStatus(FLEXIO_Type *base, uint32_t pin)

Gets the FLEXIO input pin status.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

Return values

FLEXIO – port input status

- 0: corresponding pin input capture no status.
- 1: corresponding pin input capture rising or falling edge.

static inline void FLEXIO_SetPinLevel(FLEXIO_Type *base, uint8_t pin, bool level)

Sets the FLEXIO output pin level.

Parameters

- base – FlexIO peripheral base address
- pin – FlexIO pin number.
- level – FlexIO output pin level to set, can be either 0 or 1.

static inline bool FLEXIO_GetPinOverride(const FLEXIO_Type *const base, uint8_t pin)

Gets the enabled status of a FLEXIO output pin.

Parameters

- base – FlexIO peripheral base address

- pin – FlexIO pin number.

Return values

FlexIO – port enabled status

- 0: corresponding output pin is in disabled state.
- 1: corresponding output pin is in enabled state.

static inline void FLEXIO_ConfigPinOverride(FLEXIO_Type *base, uint8_t pin, bool enabled)

Enables or disables a FLEXIO output pin.

Parameters

- base – FlexIO peripheral base address
- pin – Flexio pin number.
- enabled – Enable or disable the FlexIO pin.

static inline void FLEXIO_ClearPortStatus(FLEXIO_Type *base, uint32_t mask)

Clears the multiple FLEXIO input pins status.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

FSL_FLEXIO_DRIVER_VERSION

FlexIO driver version.

enum _flexio_timer_trigger_polarity

Define time of timer trigger polarity.

Values:

enumerator kFLEXIO_TimerTriggerPolarityActiveHigh
Active high.

enumerator kFLEXIO_TimerTriggerPolarityActiveLow
Active low.

enum _flexio_timer_trigger_source

Define type of timer trigger source.

Values:

enumerator kFLEXIO_TimerTriggerSourceExternal
External trigger selected.

enumerator kFLEXIO_TimerTriggerSourceInternal
Internal trigger selected.

enum _flexio_pin_config

Define type of timer/shifter pin configuration.

Values:

enumerator kFLEXIO_PinConfigOutputDisabled
Pin output disabled.

enumerator kFLEXIO_PinConfigOpenDrainOrBidirection
Pin open drain or bidirectional output enable.

enumerator kFLEXIO_PinConfigBidirectionOutputData
Pin bidirectional output data.

enumerator kFLEXIO_PinConfigOutput
Pin output.

enum _flexio_pin_polarity
Definition of pin polarity.

Values:

enumerator kFLEXIO_PinActiveHigh
Active high.

enumerator kFLEXIO_PinActiveLow
Active low.

enum _flexio_timer_mode
Define type of timer work mode.

Values:

enumerator kFLEXIO_TimerModeDisabled
Timer Disabled.

enumerator kFLEXIO_TimerModeDual8BitBaudBit
Dual 8-bit counters baud/bit mode.

enumerator kFLEXIO_TimerModeDual8BitPWM
Dual 8-bit counters PWM mode.

enumerator kFLEXIO_TimerModeSingle16Bit
Single 16-bit counter mode.

enumerator kFLEXIO_TimerModeDual8BitPWMLow
Dual 8-bit counters PWM Low mode.

enum _flexio_timer_output
Define type of timer initial output or timer reset condition.

Values:

enumerator kFLEXIO_TimerOutputOneNotAffectedByReset
Logic one when enabled and is not affected by timer reset.

enumerator kFLEXIO_TimerOutputZeroNotAffectedByReset
Logic zero when enabled and is not affected by timer reset.

enumerator kFLEXIO_TimerOutputOneAffectedByReset
Logic one when enabled and on timer reset.

enumerator kFLEXIO_TimerOutputZeroAffectedByReset
Logic zero when enabled and on timer reset.

enum _flexio_timer_decrement_source
Define type of timer decrement.

Values:

enumerator kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput
Decrement counter on FlexIO clock, Shift clock equals Timer output.

enumerator kFLEXIO_TimerDecSrcOnTriggerInputShiftTimerOutput
Decrement counter on Trigger input (both edges), Shift clock equals Timer output.

enumerator kFLEXIO_TimerDecSrcOnPinInputShiftPinInput
Decrement counter on Pin input (both edges), Shift clock equals Pin input.

enumerator kFLEXIO__TimerDecSrcOnTriggerInputShiftTriggerInput

Decrement counter on Trigger input (both edges), Shift clock equals Trigger input.

enum __flexio_timer_reset_condition

Define type of timer reset condition.

Values:

enumerator kFLEXIO__TimerResetNever

Timer never reset.

enumerator kFLEXIO__TimerResetOnTimerPinEqualToTimerOutput

Timer reset on Timer Pin equal to Timer Output.

enumerator kFLEXIO__TimerResetOnTimerTriggerEqualToTimerOutput

Timer reset on Timer Trigger equal to Timer Output.

enumerator kFLEXIO__TimerResetOnTimerPinRisingEdge

Timer reset on Timer Pin rising edge.

enumerator kFLEXIO__TimerResetOnTimerTriggerRisingEdge

Timer reset on Trigger rising edge.

enumerator kFLEXIO__TimerResetOnTimerTriggerBothEdge

Timer reset on Trigger rising or falling edge.

enum __flexio_timer_disable_condition

Define type of timer disable condition.

Values:

enumerator kFLEXIO__TimerDisableNever

Timer never disabled.

enumerator kFLEXIO__TimerDisableOnPreTimerDisable

Timer disabled on Timer N-1 disable.

enumerator kFLEXIO__TimerDisableOnTimerCompare

Timer disabled on Timer compare.

enumerator kFLEXIO__TimerDisableOnTimerCompareTriggerLow

Timer disabled on Timer compare and Trigger Low.

enumerator kFLEXIO__TimerDisableOnPinBothEdge

Timer disabled on Pin rising or falling edge.

enumerator kFLEXIO__TimerDisableOnPinBothEdgeTriggerHigh

Timer disabled on Pin rising or falling edge provided Trigger is high.

enumerator kFLEXIO__TimerDisableOnTriggerFallingEdge

Timer disabled on Trigger falling edge.

enum __flexio_timer_enable_condition

Define type of timer enable condition.

Values:

enumerator kFLEXIO__TimerEnabledAlways

Timer always enabled.

enumerator kFLEXIO__TimerEnableOnPrevTimerEnable

Timer enabled on Timer N-1 enable.

enumerator kFLEXIO__TimerEnableOnTriggerHigh

Timer enabled on Trigger high.

enumerator kFLEXIO__TimerEnableOnTriggerHighPinHigh

Timer enabled on Trigger high and Pin high.

enumerator kFLEXIO__TimerEnableOnPinRisingEdge

Timer enabled on Pin rising edge.

enumerator kFLEXIO__TimerEnableOnPinRisingEdgeTriggerHigh

Timer enabled on Pin rising edge and Trigger high.

enumerator kFLEXIO__TimerEnableOnTriggerRisingEdge

Timer enabled on Trigger rising edge.

enumerator kFLEXIO__TimerEnableOnTriggerBothEdge

Timer enabled on Trigger rising or falling edge.

enum _flexio_timer_stop_bit_condition

Define type of timer stop bit generate condition.

Values:

enumerator kFLEXIO__TimerStopBitDisabled

Stop bit disabled.

enumerator kFLEXIO__TimerStopBitEnableOnTimerCompare

Stop bit is enabled on timer compare.

enumerator kFLEXIO__TimerStopBitEnableOnTimerDisable

Stop bit is enabled on timer disable.

enumerator kFLEXIO__TimerStopBitEnableOnTimerCompareDisable

Stop bit is enabled on timer compare and timer disable.

enum _flexio_timer_start_bit_condition

Define type of timer start bit generate condition.

Values:

enumerator kFLEXIO__TimerStartBitDisabled

Start bit disabled.

enumerator kFLEXIO__TimerStartBitEnabled

Start bit enabled.

enum _flexio_timer_output_state

FlexIO as PWM channel output state.

Values:

enumerator kFLEXIO__PwmLow

The output state of PWM channel is low

enumerator kFLEXIO__PwmHigh

The output state of PWM channel is high

enum _flexio_shifter_timer_polarity

Define type of timer polarity for shifter control.

Values:

enumerator kFLEXIO__ShifterTimerPolarityOnPositive

Shift on positive edge of shift clock.

enumerator kFLEXIO__ShifterTimerPolarityOnNegative

Shift on negative edge of shift clock.

enum _flexio__shifter__mode

Define type of shifter working mode.

Values:

enumerator kFLEXIO__ShifterDisabled

Shifter is disabled.

enumerator kFLEXIO__ShifterModeReceive

Receive mode.

enumerator kFLEXIO__ShifterModeTransmit

Transmit mode.

enumerator kFLEXIO__ShifterModeMatchStore

Match store mode.

enumerator kFLEXIO__ShifterModeMatchContinuous

Match continuous mode.

enumerator kFLEXIO__ShifterModeState

SHIFTBUF contents are used for storing programmable state attributes.

enumerator kFLEXIO__ShifterModeLogic

SHIFTBUF contents are used for implementing programmable logic look up table.

enum _flexio__shifter__input__source

Define type of shifter input source.

Values:

enumerator kFLEXIO__ShifterInputFromPin

Shifter input from pin.

enumerator kFLEXIO__ShifterInputFromNextShifterOutput

Shifter input from Shifter N+1.

enum _flexio__shifter__stop__bit

Define of STOP bit configuration.

Values:

enumerator kFLEXIO__ShifterStopBitDisable

Disable shifter stop bit.

enumerator kFLEXIO__ShifterStopBitLow

Set shifter stop bit to logic low level.

enumerator kFLEXIO__ShifterStopBitHigh

Set shifter stop bit to logic high level.

enum _flexio__shifter__start__bit

Define type of START bit configuration.

Values:

enumerator kFLEXIO__ShifterStartBitDisabledLoadDataOnEnable

Disable shifter start bit, transmitter loads data on enable.

enumerator kFLEXIO__ShifterStartBitDisabledLoadDataOnShift

Disable shifter start bit, transmitter loads data on first shift.

enumerator kFLEXIO__ShifterStartBitLow

Set shifter start bit to logic low level.

enumerator kFLEXIO__ShifterStartBitHigh

Set shifter start bit to logic high level.

enum _flexio_shifter_buffer_type

Define FlexIO shifter buffer type.

Values:

enumerator kFLEXIO__ShifterBuffer

Shifter Buffer N Register.

enumerator kFLEXIO__ShifterBufferBitSwapped

Shifter Buffer N Bit Byte Swapped Register.

enumerator kFLEXIO__ShifterBufferByteSwapped

Shifter Buffer N Byte Swapped Register.

enumerator kFLEXIO__ShifterBufferBitByteSwapped

Shifter Buffer N Bit Swapped Register.

enumerator kFLEXIO__ShifterBufferNibbleByteSwapped

Shifter Buffer N Nibble Byte Swapped Register.

enumerator kFLEXIO__ShifterBufferHalfWordSwapped

Shifter Buffer N Half Word Swapped Register.

enumerator kFLEXIO__ShifterBufferNibbleSwapped

Shifter Buffer N Nibble Swapped Register.

enum _flexio_gpio_direction

FLEXIO gpio direction definition.

Values:

enumerator kFLEXIO__DigitalInput

Set current pin as digital input

enumerator kFLEXIO__DigitalOutput

Set current pin as digital output

enum _flexio_pin_input_config

FLEXIO gpio input config.

Values:

enumerator kFLEXIO__InputInterruptDisabled

Interrupt request is disabled.

enumerator kFLEXIO__InputInterruptEnable

Interrupt request is enable.

enumerator kFLEXIO__FlagRisingEdgeEnable

Input pin flag on rising edge.

enumerator kFLEXIO__FlagFallingEdgeEnable

Input pin flag on falling edge.

typedef enum _flexio_timer_trigger_polarity flexio_timer_trigger_polarity_t

Define time of timer trigger polarity.

`typedef enum _flexio_timer_trigger_source flexio_timer_trigger_source_t`
Define type of timer trigger source.

`typedef enum _flexio_pin_config flexio_pin_config_t`
Define type of timer/shifter pin configuration.

`typedef enum _flexio_pin_polarity flexio_pin_polarity_t`
Definition of pin polarity.

`typedef enum _flexio_timer_mode flexio_timer_mode_t`
Define type of timer work mode.

`typedef enum _flexio_timer_output flexio_timer_output_t`
Define type of timer initial output or timer reset condition.

`typedef enum _flexio_timer_decrement_source flexio_timer_decrement_source_t`
Define type of timer decrement.

`typedef enum _flexio_timer_reset_condition flexio_timer_reset_condition_t`
Define type of timer reset condition.

`typedef enum _flexio_timer_disable_condition flexio_timer_disable_condition_t`
Define type of timer disable condition.

`typedef enum _flexio_timer_enable_condition flexio_timer_enable_condition_t`
Define type of timer enable condition.

`typedef enum _flexio_timer_stop_bit_condition flexio_timer_stop_bit_condition_t`
Define type of timer stop bit generate condition.

`typedef enum _flexio_timer_start_bit_condition flexio_timer_start_bit_condition_t`
Define type of timer start bit generate condition.

`typedef enum _flexio_timer_output_state flexio_timer_output_state_t`
FlexIO as PWM channel output state.

`typedef enum _flexio_shifter_timer_polarity flexio_shifter_timer_polarity_t`
Define type of timer polarity for shifter control.

`typedef enum _flexio_shifter_mode flexio_shifter_mode_t`
Define type of shifter working mode.

`typedef enum _flexio_shifter_input_source flexio_shifter_input_source_t`
Define type of shifter input source.

`typedef enum _flexio_shifter_stop_bit flexio_shifter_stop_bit_t`
Define of STOP bit configuration.

`typedef enum _flexio_shifter_start_bit flexio_shifter_start_bit_t`
Define type of START bit configuration.

`typedef enum _flexio_shifter_buffer_type flexio_shifter_buffer_type_t`
Define FlexIO shifter buffer type.

`typedef struct _flexio_config flexio_config_t`
Define FlexIO user configuration structure.

`typedef struct _flexio_timer_config flexio_timer_config_t`
Define FlexIO timer configuration structure.

`typedef struct _flexio_shifter_config flexio_shifter_config_t`
Define FlexIO shifter configuration structure.

typedef enum *_flexio_gpio_direction* flexio_gpio_direction_t

FLEXIO gpio direction definition.

typedef enum *_flexio_pin_input_config* flexio_pin_input_config_t

FLEXIO gpio input config.

typedef struct *_flexio_gpio_config* flexio_gpio_config_t

The FLEXIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, use inputConfig param. If configured as an output pin, use outputLogic.

typedef void (*flexio_isr_t)(void *base, void *handle)

typedef for FlexIO simulated driver interrupt handler.

FLEXIO_Type *const s_flexioBases[]

Pointers to flexio bases for each instance.

const *clock_ip_name_t* s_flexioClocks[]

Pointers to flexio clocks for each instance.

void FLEXIO_SetPinConfig(FLEXIO_Type *base, uint32_t pin, *flexio_gpio_config_t* *config)

Configure a FLEXIO pin used by the board.

To Config the FLEXIO PIN, define a pin configuration, as either input or output, in the user file. Then, call the FLEXIO_SetPinConfig() function.

This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
flexio_gpio_config_t config =
{
    kFLEXIO_DigitalInput,
    0U,
    kFLEXIO_FlagRisingEdgeEnable | kFLEXIO_InputInterruptEnable,
}
Define a digital output pin configuration,
flexio_gpio_config_t config =
{
    kFLEXIO_DigitalOutput,
    0U,
    0U
}
```

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.
- config – FLEXIO pin configuration pointer.

FLEXIO_TIMER_TRIGGER_SEL_PININPUT(x)

Calculate FlexIO timer trigger.

FLEXIO_TIMER_TRIGGER_SEL_SHIFThSTAT(x)

FLEXIO_TIMER_TRIGGER_SEL_TIMn(x)

struct _flexio_config_

#include <fsl_flexio.h> Define FlexIO user configuration structure.

Public Members**bool** enableFlexio

Enable/disable FlexIO module

bool enableInDoze

Enable/disable FlexIO operation in doze mode

bool enableInDebug

Enable/disable FlexIO operation in debug mode

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

struct _flexio_timer_config

#include <fsl_flexio.h> Define FlexIO timer configuration structure.

Public Members**uint32_t** triggerSelect

The internal trigger selection number using MACROs.

flexio_timer_trigger_polarity_t triggerPolarity

Trigger Polarity.

flexio_timer_trigger_source_t triggerSource

Trigger Source, internal (see 'trgsel') or external.

flexio_pin_config_t pinConfig

Timer Pin Configuration.

uint32_t pinSelect

Timer Pin number Select.

flexio_pin_polarity_t pinPolarity

Timer Pin Polarity.

flexio_timer_mode_t timerMode

Timer work Mode.

flexio_timer_output_t timerOutput

Configures the initial state of the Timer Output and whether it is affected by the Timer reset.

flexio_timer_decrement_source_t timerDecrement

Configures the source of the Timer decrement and the source of the Shift clock.

flexio_timer_reset_condition_t timerReset

Configures the condition that causes the timer counter (and optionally the timer output) to be reset.

flexio_timer_disable_condition_t timerDisable

Configures the condition that causes the Timer to be disabled and stop decrementing.

flexio_timer_enable_condition_t timerEnable

Configures the condition that causes the Timer to be enabled and start decrementing.

flexio_timer_stop_bit_condition_t timerStop

Timer STOP Bit generation.

flexio_timer_start_bit_condition_t timerStart

Timer STRAT Bit generation.

uint32_t timerCompare

Value for Timer Compare N Register.

struct *_flexio_shifter_config*

#include <fsl_flexio.h> Define FlexIO shifter configuration structure.

Public Members

uint32_t timerSelect

Selects which Timer is used for controlling the logic/shift register and generating the Shift clock.

flexio_shifter_timer_polarity_t timerPolarity

Timer Polarity.

flexio_pin_config_t pinConfig

Shifter Pin Configuration.

uint32_t pinSelect

Shifter Pin number Select.

flexio_pin_polarity_t pinPolarity

Shifter Pin Polarity.

flexio_shifter_mode_t shifterMode

Configures the mode of the Shifter.

uint32_t parallelWidth

Configures the parallel width when using parallel mode.

flexio_shifter_input_source_t inputSource

Selects the input source for the shifter.

flexio_shifter_stop_bit_t shifterStop

Shifter STOP bit.

flexio_shifter_start_bit_t shifterStart

Shifter START bit.

struct *_flexio_gpio_config*

#include <fsl_flexio.h> The FLEXIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, use inputConfig param. If configured as an output pin, use outputLogic.

Public Members

flexio_gpio_direction_t pinDirection

FLEXIO pin direction, input or output

uint8_t outputLogic

Set a default output logic, which has no use in input

uint8_t inputConfig

Set an input config

2.46 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

2.47 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

2.48 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

2.49 FlexIO I2C Master Driver

status_t FLEXIO_I2C_CheckForBusyBus(*FLEXIO_I2C_Type* *base)

Make sure the bus isn't already pulled down.

Check the FLEXIO pin status to see whether either of SDA and SCL pin is pulled down.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure..

Return values

- kStatus_Success –
- kStatus_FLEXIO_I2C_Busy –

status_t FLEXIO_I2C_MasterInit(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO I2C hardware configuration.

Example

```
FLEXIO_I2C_Type base = {  
.flexioBase = FLEXIO,  
.SDAPinIndex = 0,  
.SCLPinIndex = 1,  
.shifterIndex = {0,1},  
.timerIndex = {0,1}  
};  
flexio_i2c_master_config_t config = {  
.enableInDoze = false,
```

(continues on next page)

(continued from previous page)

```
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 100000
};
FLEXIO_I2C_MasterInit(base, &config, srcClock_Hz);
```

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- masterConfig – Pointer to flexio_i2c_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Initialization successful
- kStatus_InvalidArgument – The source clock exceed upper range limitation

void FLEXIO_I2C_MasterDeinit(*FLEXIO_I2C_Type* *base)

De-initializes the FlexIO I2C master peripheral. Calling this API Resets the FlexIO I2C master shifer and timer config, module can't work unless the FLEXIO_I2C_MasterInit is called.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterGetDefaultConfig(*flexio_i2c_master_config_t* *masterConfig)

Gets the default configuration to configure the FlexIO module. The configuration can be used directly for calling the FLEXIO_I2C_MasterInit().

Example:

```
flexio_i2c_master_config_t config;
FLEXIO_I2C_MasterGetDefaultConfig(&config);
```

Parameters

- masterConfig – Pointer to flexio_i2c_master_config_t structure.

static inline void FLEXIO_I2C_MasterEnable(*FLEXIO_I2C_Type* *base, bool enable)

Enables/disables the FlexIO module operation.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- enable – Pass true to enable module, false does not have any effect.

uint32_t FLEXIO_I2C_MasterGetStatusFlags(*FLEXIO_I2C_Type* *base)

Gets the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure

Returns

Status flag, use status flag to AND `_flexio_i2c_master_status_flags` can get the related status.

void FLEXIO_I2C_MasterClearStatusFlags(*FLEXIO_I2C_Type* *base, uint32_t mask)

Clears the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

- mask – Status flag. The parameter can be any combination of the following values:
 - kFLEXIO_I2C_RxFullFlag
 - kFLEXIO_I2C_ReceiveNakFlag

void FLEXIO_I2C_MasterEnableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)

Enables the FlexIO i2c master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source. Currently only one interrupt request source:
 - kFLEXIO_I2C_TransferCompleteInterruptEnable

void FLEXIO_I2C_MasterDisableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)

Disables the FlexIO I2C master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source.

void FLEXIO_I2C_MasterSetBaudRate(*FLEXIO_I2C_Type* *base, uint32_t baudRate_Bps,
uint32_t srcClock_Hz)

Sets the FlexIO I2C master transfer baudrate.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- baudRate_Bps – the baud rate value in HZ
- srcClock_Hz – source clock in HZ

void FLEXIO_I2C_MasterStart(*FLEXIO_I2C_Type* *base, uint8_t address, *flexio_i2c_direction_t*
direction)

Sends START + 7-bit address to the bus.

Note: This API should be called when the transfer configuration is ready to send a START signal and 7-bit address to the bus. This is a non-blocking API, which returns directly after the address is put into the data register but the address transfer is not finished on the bus. Ensure that the kFLEXIO_I2C_RxFullFlag status is asserted before calling this API.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- address – 7-bit address.
- direction – transfer direction. This parameter is one of the values in *flexio_i2c_direction_t*:
 - kFLEXIO_I2C_Write: Transmit
 - kFLEXIO_I2C_Read: Receive

void FLEXIO_I2C_MasterStop(*FLEXIO_I2C_Type* *base)

Sends the stop signal on the bus.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterRepeatedStart(*FLEXIO_I2C_Type* *base)

Sends the repeated start signal on the bus.

Parameters

- base – Pointer to *FLEXIO_I2C_Type* structure.

void FLEXIO_I2C_MasterAbortStop(*FLEXIO_I2C_Type* *base)

Sends the stop signal when transfer is still on-going.

Parameters

- base – Pointer to *FLEXIO_I2C_Type* structure.

void FLEXIO_I2C_MasterEnableAck(*FLEXIO_I2C_Type* *base, bool enable)

Configures the sent ACK/NAK for the following byte.

Parameters

- base – Pointer to *FLEXIO_I2C_Type* structure.
- enable – True to configure send ACK, false configure to send NAK.

status_t FLEXIO_I2C_MasterSetTransferCount(*FLEXIO_I2C_Type* *base, uint16_t count)

Sets the number of bytes to be transferred from a start signal to a stop signal.

Note: Call this API before a transfer begins because the timer generates a number of clocks according to the number of bytes that need to be transferred.

Parameters

- base – Pointer to *FLEXIO_I2C_Type* structure.
- count – Number of bytes need to be transferred from a start signal to a re-start/stop signal

Return values

- kStatus_Success – Successfully configured the count.
- kStatus_InvalidArgument – Input argument is invalid.

static inline void FLEXIO_I2C_MasterWriteByte(*FLEXIO_I2C_Type* *base, uint32_t data)

Writes one byte of data to the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to *FLEXIO_I2C_Type* structure.
- data – a byte of data.

static inline uint8_t FLEXIO_I2C_MasterReadByte(*FLEXIO_I2C_Type* *base)

Reads one byte of data from the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the data is ready in the register.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

Returns

data byte read.

status_t FLEXIO_I2C_MasterWriteBlocking(*FLEXIO_I2C_Type* *base, const uint8_t *txBuff, uint8_t txSize)

Sends a buffer of data in bytes.

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

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- txBuff – The data bytes to send.
- txSize – The number of data bytes to send.

Return values

- kStatus_Success – Successfully write data.
- kStatus_FLEXIO_I2C_Nak – Receive NAK during writing data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterReadBlocking(*FLEXIO_I2C_Type* *base, uint8_t *rxBuff, uint8_t rxSize)

Receives a buffer of bytes.

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

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- rxBuff – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus_Success – Successfully read data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterTransferBlocking(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_transfer_t* *xfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to receiving NAK.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.
- xfer – pointer to flexio_i2c_master_transfer_t structure.

Returns

status of status_t.

```
status_t FLEXIO_I2C_MasterTransferCreateHandle(FLEXIO_I2C_Type *base,  
                                              flexio_i2c_master_handle_t *handle,  
                                              flexio_i2c_master_transfer_callback_t  
                                              callback, void *userData)
```

Initializes the I2C handle which is used in transactional functions.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- handle – Pointer to flexio_i2c_master_handle_t structure to store the transfer state.
- callback – Pointer to user callback function.
- userData – User param passed to the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/isr table out of range.

```
status_t FLEXIO_I2C_MasterTransferNonBlocking(FLEXIO_I2C_Type *base,  
                                              flexio_i2c_master_handle_t *handle,  
                                              flexio_i2c_master_transfer_t *xfer)
```

Performs a master interrupt non-blocking transfer on the I2C bus.

Note: The API returns immediately after the transfer initiates. Call FLEXIO_I2C_MasterTransferGetCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_FLEXIO_I2C_Busy, the transfer is finished.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state
- xfer – pointer to flexio_i2c_master_transfer_t structure

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_FLEXIO_I2C_Busy – FlexIO I2C is not idle, is running another transfer.

```
status_t FLEXIO_I2C_MasterTransferGetCount(FLEXIO_I2C_Type *base,  
                                              flexio_i2c_master_handle_t *handle, size_t  
                                              *count)
```

Gets the master transfer status during a interrupt non-blocking transfer.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.

- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.
- `kStatus_Success` – Successfully return the count.

`void FLEXIO_I2C_MasterTransferAbort(FLEXIO_I2C_Type *base, flexio_i2c_master_handle_t *handle)`

Aborts an interrupt non-blocking transfer early.

Note: This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- `base` – Pointer to `FLEXIO_I2C_Type` structure
- `handle` – Pointer to `flexio_i2c_master_handle_t` structure which stores the transfer state

`void FLEXIO_I2C_MasterTransferHandleIRQ(void *i2cType, void *i2cHandle)`

Master interrupt handler.

Parameters

- `i2cType` – Pointer to `FLEXIO_I2C_Type` structure
- `i2cHandle` – Pointer to `flexio_i2c_master_transfer_t` structure

`FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION`

FlexIO I2C transfer status.

Values:

enumerator `kStatus_FLEXIO_I2C_Busy`
I2C is busy doing transfer.

enumerator `kStatus_FLEXIO_I2C_Idle`
I2C is busy doing transfer.

enumerator `kStatus_FLEXIO_I2C_Nak`
NAK received during transfer.

enumerator `kStatus_FLEXIO_I2C_Timeout`
Timeout polling status flags.

`enum _flexio_i2c_master_interrupt`

Define FlexIO I2C master interrupt mask.

Values:

enumerator `kFLEXIO_I2C_TxEmptyInterruptEnable`
Tx buffer empty interrupt enable.

enumerator `kFLEXIO_I2C_RxFullInterruptEnable`
Rx buffer full interrupt enable.

`enum _flexio_i2c_master_status_flags`

Define FlexIO I2C master status mask.

Values:

```

enumerator kFLEXIO_I2C_TxEmptyFlag
    Tx shifter empty flag.
enumerator kFLEXIO_I2C_RxFullFlag
    Rx shifter full/Transfer complete flag.
enumerator kFLEXIO_I2C_ReceiveNakFlag
    Receive NAK flag.
enum _flexio_i2c_direction
    Direction of master transfer.
    Values:
enumerator kFLEXIO_I2C_Write
    Master send to slave.
enumerator kFLEXIO_I2C_Read
    Master receive from slave.
typedef enum _flexio_i2c_direction flexio_i2c_direction_t
    Direction of master transfer.
typedef struct _flexio_i2c_type FLEXIO_I2C_Type
    Define FlexIO I2C master access structure typedef.
typedef struct _flexio_i2c_master_config flexio_i2c_master_config_t
    Define FlexIO I2C master user configuration structure.
typedef struct _flexio_i2c_master_transfer flexio_i2c_master_transfer_t
    Define FlexIO I2C master transfer structure.
typedef struct _flexio_i2c_master_handle flexio_i2c_master_handle_t
    FlexIO I2C master handle typedef.
typedef void (*flexio_i2c_master_transfer_callback_t)(FLEXIO_I2C_Type *base,
flexio_i2c_master_handle_t *handle, status_t status, void *userData)
    FlexIO I2C master transfer callback typedef.
I2C_RETRY_TIMES
    Retry times for waiting flag.
struct _flexio_i2c_type
    #include <fsl_flexio_i2c_master.h> Define FlexIO I2C master access structure typedef.

```

Public Members

```

FLEXIO_Type *flexioBase
    FlexIO base pointer.
uint8_t SDAPinIndex
    Pin select for I2C SDA.
uint8_t SCLPinIndex
    Pin select for I2C SCL.
uint8_t shifterIndex[2]
    Shifter index used in FlexIO I2C.
uint8_t timerIndex[3]
    Timer index used in FlexIO I2C.

```

uint32_t baudrate

Master transfer baudrate, used to calculate delay time.

struct _flexio_i2c_master_config

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master user configuration structure.

Public Members

bool enableMaster

Enables the FlexIO I2C peripheral at initialization time.

bool enableInDoze

Enable/disable FlexIO operation in doze mode.

bool enableInDebug

Enable/disable FlexIO operation in debug mode.

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps

Baud rate in Bps.

struct _flexio_i2c_master_transfer

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master transfer structure.

Public Members

uint32_t flags

Transfer flag which controls the transfer, reserved for FlexIO I2C.

uint8_t slaveAddress

7-bit slave address.

flexio_i2c_direction_t direction

Transfer direction, read or write.

uint32_t subaddress

Sub address. Transferred MSB first.

uint8_t subaddressSize

Size of sub address.

uint8_t volatile *data

Transfer buffer.

volatile size_t dataSize

Transfer size.

struct _flexio_i2c_master_handle

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master handle structure.

Public Members

flexio_i2c_master_transfer_t transfer

FlexIO I2C master transfer copy.

`size_t` transferSize

Total bytes to be transferred.

`uint8_t` state

Transfer state maintained during transfer.

flexio_i2c_master_transfer_callback_t completionCallback

Callback function called at transfer event. Callback function called at transfer event.

`void *`userData

Callback parameter passed to callback function.

`bool` needRestart

Whether master needs to send re-start signal.

2.50 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

2.51 FlexIO SPI Driver

```
void FLEXIO_SPI_MasterInit(FLEXIO_SPI_Type *base, flexio_spi_master_config_t
                           *masterConfig, uint32_t srcClock_Hz)
```

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI master hardware, and configures the FlexIO SPI with FlexIO SPI master configuration. The configuration structure can be filled by the user, or be set with default values by the FLEXIO_SPI_MasterGetDefaultConfig().

Example

```
FLEXIO_SPI_Type spiDev = {
    .flexioBase = FLEXIO,
    .SDOPinIndex = 0,
    .SDIPinIndex = 1,
    .SCKPinIndex = 2,
    .CSnPinIndex = 3,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_spi_master_config_t config = {
    .enableMaster = true,
    .enableInDoze = false,
    .enableInDebug = true,
```

(continues on next page)

(continued from previous page)

```
.enableFastAccess = false,
.baudRate_Bps = 500000,
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
.direction = kFLEXIO_SPI_MsbFirst,
.dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_MasterInit(&spiDev, &config, srcClock_Hz);
```

Note: 1.FlexIO SPI master only support CPOL = 0, which means clock inactive low. 2.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI master communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by $2*2=4$. If FlexIO SPI master communicates with FlexIO SPI slave, the maximum baud rate is FlexIO clock frequency divided by $(1.5+2.5)*2=8$.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- masterConfig – Pointer to the flexio_spi_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

void FLEXIO_SPI_MasterDeinit(*FLEXIO_SPI_Type* *base)

Resets the FlexIO SPI timer and shifter config.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.

void FLEXIO_SPI_MasterGetDefaultConfig(*flexio_spi_master_config_t* *masterConfig)

Gets the default configuration to configure the FlexIO SPI master. The configuration can be used directly by calling the FLEXIO_SPI_MasterConfigure(). Example:

```
flexio_spi_master_config_t masterConfig;
FLEXIO_SPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – Pointer to the flexio_spi_master_config_t structure.

void FLEXIO_SPI_SlaveInit(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_config_t* *slaveConfig)

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI slave hardware configuration, and configures the FlexIO SPI with FlexIO SPI slave configuration. The configuration structure can be filled by the user, or be set with default values by the FLEXIO_SPI_SlaveGetDefaultConfig().

Note: 1.Only one timer is needed in the FlexIO SPI slave. As a result, the second timer index is ignored. 2.FlexIO SPI slave only support CPOL = 0, which means clock inactive low. 3.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI slave communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by $3*2=6$. If FlexIO SPI slave communicates with FlexIO SPI master, the maximum baud rate is FlexIO clock frequency divided by $(1.5+2.5)*2=8$. Example

```
FLEXIO_SPI_Type spiDev = {
.flexioBase = FLEXIO,
.SDOPinIndex = 0,
```

(continues on next page)

(continued from previous page)

```
.SDIPinIndex = 1,
.SCKPinIndex = 2,
.CSnPinIndex = 3,
.shifterIndex = {0,1},
.timerIndex = {0}
};
flexio_spi_slave_config_t config = {
.enableSlave = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
.direction = kFLEXIO_SPI_MsbFirst,
.dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_SlaveInit(&spiDev, &config);
```

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- slaveConfig – Pointer to the flexio_spi_slave_config_t structure.

void FLEXIO_SPI_SlaveDeinit(*FLEXIO_SPI_Type* *base)

Gates the FlexIO clock.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.

void FLEXIO_SPI_SlaveGetDefaultConfig(*flexio_spi_slave_config_t* *slaveConfig)

Gets the default configuration to configure the FlexIO SPI slave. The configuration can be used directly for calling the FLEXIO_SPI_SlaveConfigure(). Example:

```
flexio_spi_slave_config_t slaveConfig;
FLEXIO_SPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – Pointer to the flexio_spi_slave_config_t structure.

uint32_t FLEXIO_SPI_GetStatusFlags(*FLEXIO_SPI_Type* *base)

Gets FlexIO SPI status flags.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

Returns

status flag; Use the status flag to AND the following flag mask and get the status.

- kFLEXIO_SPI_TxEmptyFlag
- kFLEXIO_SPI_RxEmptyFlag

void FLEXIO_SPI_ClearStatusFlags(*FLEXIO_SPI_Type* *base, uint32_t mask)

Clears FlexIO SPI status flags.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

- **mask** – status flag The parameter can be any combination of the following values:
 - kFLEXIO_SPI_TxEmptyFlag
 - kFLEXIO_SPI_RxEmptyFlag

`void FLEXIO_SPI_EnableInterrupts(FLEXIO_SPI_Type *base, uint32_t mask)`

Enables the FlexIO SPI interrupt.

This function enables the FlexIO SPI interrupt.

Parameters

- **base** – Pointer to the `FLEXIO_SPI_Type` structure.
- **mask** – interrupt source. The parameter can be any combination of the following values:
 - kFLEXIO_SPI_RxFullInterruptEnable
 - kFLEXIO_SPI_TxEmptyInterruptEnable

`void FLEXIO_SPI_DisableInterrupts(FLEXIO_SPI_Type *base, uint32_t mask)`

Disables the FlexIO SPI interrupt.

This function disables the FlexIO SPI interrupt.

Parameters

- **base** – Pointer to the `FLEXIO_SPI_Type` structure.
- **mask** – interrupt source The parameter can be any combination of the following values:
 - kFLEXIO_SPI_RxFullInterruptEnable
 - kFLEXIO_SPI_TxEmptyInterruptEnable

`void FLEXIO_SPI_EnableDMA(FLEXIO_SPI_Type *base, uint32_t mask, bool enable)`

Enables/disables the FlexIO SPI transmit DMA. This function enables/disables the FlexIO SPI Tx DMA, which means that asserting the `kFLEXIO_SPI_TxEmptyFlag` does/doesn't trigger the DMA request.

Parameters

- **base** – Pointer to the `FLEXIO_SPI_Type` structure.
- **mask** – SPI DMA source.
- **enable** – True means enable DMA, false means disable DMA.

`static inline uint32_t FLEXIO_SPI_GetTxDataRegisterAddress(FLEXIO_SPI_Type *base,
flexio_spi_shift_direction_t
direction)`

Gets the FlexIO SPI transmit data register address for MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- **base** – Pointer to the `FLEXIO_SPI_Type` structure.
- **direction** – Shift direction of MSB first or LSB first.

Returns

FlexIO SPI transmit data register address.

```
static inline uint32_t FLEXIO_SPI_GetRxDataRegisterAddress(FLEXIO_SPI_Type *base,  
                                                         flexio_spi_shift_direction_t  
                                                         direction)
```

Gets the FlexIO SPI receive data register address for the MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- direction – Shift direction of MSB first or LSB first.

Returns

FlexIO SPI receive data register address.

```
static inline void FLEXIO_SPI_Enable(FLEXIO_SPI_Type *base, bool enable)
```

Enables/disables the FlexIO SPI module operation.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type*.
- enable – True to enable, false does not have any effect.

```
void FLEXIO_SPI_MasterSetBaudRate(FLEXIO_SPI_Type *base, uint32_t baudRate_Bps,  
                                  uint32_t srcClockHz)
```

Sets baud rate for the FlexIO SPI transfer, which is only used for the master.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- baudRate_Bps – Baud Rate needed in Hz.
- srcClockHz – SPI source clock frequency in Hz.

```
static inline void FLEXIO_SPI_WriteData(FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t  
                                       direction, uint32_t data)
```

Writes one byte of data, which is sent using the MSB method.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- direction – Shift direction of MSB first or LSB first.
- data – 8/16/32 bit data.

```
static inline uint32_t FLEXIO_SPI_ReadData(FLEXIO_SPI_Type *base,  
                                           flexio_spi_shift_direction_t direction)
```

Reads 8 bit/16 bit data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.

- direction – Shift direction of MSB first or LSB first.

Returns

8 bit/16 bit data received.

status_t FLEXIO_SPI_WriteBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, const uint8_t *buffer, size_t size)

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The data bytes to send.
- size – The number of data bytes to send.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

status_t FLEXIO_SPI_ReadBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, uint8_t *buffer, size_t size)

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The buffer to store the received bytes.
- size – The number of data bytes to be received.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

status_t FLEXIO_SPI_MasterTransferBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_transfer_t* *xfer)

Receives a buffer of bytes.

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

Parameters

- base – pointer to FLEXIO_SPI_Type structure
- xfer – FlexIO SPI transfer structure, see flexio_spi_transfer_t.

Return values

- kStatus_Success – Successfully create the handle.

- `kStatus_FLEXIO_SPI_Timeout` – The transfer timed out and was aborted.

`void FLEXIO_SPI_FlushShifters(FLEXIO_SPI_Type *base)`

Flush tx/rx shifters.

Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.

`status_t FLEXIO_SPI_MasterTransferCreateHandle(FLEXIO_SPI_Type *base,
flexio_spi_master_handle_t *handle,
flexio_spi_master_transfer_callback_t
callback, void *userData)`

Initializes the FlexIO SPI Master handle, which is used in transactional functions.

Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to the `flexio_spi_master_handle_t` structure to store the transfer state.
- `callback` – The callback function.
- `userData` – The parameter of the callback function.

Return values

- `kStatus_Success` – Successfully create the handle.
- `kStatus_OutOfRange` – The FlexIO type/handle/ISR table out of range.

`status_t FLEXIO_SPI_MasterTransferNonBlocking(FLEXIO_SPI_Type *base,
flexio_spi_master_handle_t *handle,
flexio_spi_transfer_t *xfer)`

Master transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to the `flexio_spi_master_handle_t` structure to store the transfer state.
- `xfer` – FlexIO SPI transfer structure. See `flexio_spi_transfer_t`.

Return values

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

`void FLEXIO_SPI_MasterTransferAbort(FLEXIO_SPI_Type *base, flexio_spi_master_handle_t *handle)`

Aborts the master data transfer, which used IRQ.

Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to the `flexio_spi_master_handle_t` structure to store the transfer state.

```
status_t FLEXIO_SPI_MasterTransferGetCount(FLEXIO_SPI_Type *base,  
                                           flexio_spi_master_handle_t *handle, size_t  
                                           *count)
```

Gets the data transfer status which used IRQ.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- handle – Pointer to the *flexio_spi_master_handle_t* structure to store the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- *kStatus_InvalidArgument* – count is Invalid.
- *kStatus_Success* – Successfully return the count.

```
void FLEXIO_SPI_MasterTransferHandleIRQ(void *spiType, void *spiHandle)
```

FlexIO SPI master IRQ handler function.

Parameters

- spiType – Pointer to the *FLEXIO_SPI_Type* structure.
- spiHandle – Pointer to the *flexio_spi_master_handle_t* structure to store the transfer state.

```
status_t FLEXIO_SPI_SlaveTransferCreateHandle(FLEXIO_SPI_Type *base,  
                                              flexio_spi_slave_handle_t *handle,  
                                              flexio_spi_slave_transfer_callback_t callback,  
                                              void *userData)
```

Initializes the FlexIO SPI Slave handle, which is used in transactional functions.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- handle – Pointer to the *flexio_spi_slave_handle_t* structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- *kStatus_Success* – Successfully create the handle.
- *kStatus_OutOfRange* – The FlexIO type/handle/ISR table out of range.

```
status_t FLEXIO_SPI_SlaveTransferNonBlocking(FLEXIO_SPI_Type *base,  
                                              flexio_spi_slave_handle_t *handle,  
                                              flexio_spi_transfer_t *xfer)
```

Slave transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- handle – Pointer to the *flexio_spi_slave_handle_t* structure to store the transfer state.
- base – Pointer to the *FLEXIO_SPI_Type* structure.
- xfer – FlexIO SPI transfer structure. See *flexio_spi_transfer_t*.

Return values

- `kStatus_Success` – Successfully start a transfer.
- `kStatus_InvalidArgument` – Input argument is invalid.
- `kStatus_FLEXIO_SPI_Busy` – SPI is not idle; it is running another transfer.

```
static inline void FLEXIO_SPI_SlaveTransferAbort(FLEXIO_SPI_Type *base,
                                                flexio_spi_slave_handle_t *handle)
```

Aborts the slave data transfer which used IRQ, share same API with master.

Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.

```
static inline status_t FLEXIO_SPI_SlaveTransferGetCount(FLEXIO_SPI_Type *base,
                                                         flexio_spi_slave_handle_t *handle,
                                                         size_t *count)
```

Gets the data transfer status which used IRQ, share same API with master.

Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_InvalidArgument` – count is Invalid.
- `kStatus_Success` – Successfully return the count.

```
void FLEXIO_SPI_SlaveTransferHandleIRQ(void *spiType, void *spiHandle)
```

FlexIO SPI slave IRQ handler function.

Parameters

- `spiType` – Pointer to the `FLEXIO_SPI_Type` structure.
- `spiHandle` – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.

```
FSL_FLEXIO_SPI_DRIVER_VERSION
```

FlexIO SPI driver version.

Error codes for the FlexIO SPI driver.

Values:

```
enumerator kStatus_FLEXIO_SPI_Busy
```

FlexIO SPI is busy.

```
enumerator kStatus_FLEXIO_SPI_Idle
```

SPI is idle

```
enumerator kStatus_FLEXIO_SPI_Error
```

FlexIO SPI error.

```
enumerator kStatus_FLEXIO_SPI_Timeout
```

FlexIO SPI timeout polling status flags.

enum _flexio_spi_clock_phase

FlexIO SPI clock phase configuration.

Values:

enumerator kFLEXIO_SPI_ClockPhaseFirstEdge

First edge on SPCK occurs at the middle of the first cycle of a data transfer.

enumerator kFLEXIO_SPI_ClockPhaseSecondEdge

First edge on SPCK occurs at the start of the first cycle of a data transfer.

enum _flexio_spi_shift_direction

FlexIO SPI data shifter direction options.

Values:

enumerator kFLEXIO_SPI_MsbFirst

Data transfers start with most significant bit.

enumerator kFLEXIO_SPI_LsbFirst

Data transfers start with least significant bit.

enum _flexio_spi_data_bitcount_mode

FlexIO SPI data length mode options.

Values:

enumerator kFLEXIO_SPI_8BitMode

8-bit data transmission mode.

enumerator kFLEXIO_SPI_16BitMode

16-bit data transmission mode.

enumerator kFLEXIO_SPI_32BitMode

32-bit data transmission mode.

enum _flexio_spi_interrupt_enable

Define FlexIO SPI interrupt mask.

Values:

enumerator kFLEXIO_SPI_TxEmptyInterruptEnable

Transmit buffer empty interrupt enable.

enumerator kFLEXIO_SPI_RxFullInterruptEnable

Receive buffer full interrupt enable.

enum _flexio_spi_status_flags

Define FlexIO SPI status mask.

Values:

enumerator kFLEXIO_SPI_TxBufferEmptyFlag

Transmit buffer empty flag.

enumerator kFLEXIO_SPI_RxBufferFullFlag

Receive buffer full flag.

enum _flexio_spi_dma_enable

Define FlexIO SPI DMA mask.

Values:

enumerator kFLEXIO_SPI_TxDmaEnable

Tx DMA request source

enumerator kFLEXIO_SPI_RxDmaEnable
Rx DMA request source

enumerator kFLEXIO_SPI_DmaAllEnable
All DMA request source

enum *_flexio_spi_transfer_flags*
Define FlexIO SPI transfer flags.

Note: Use kFLEXIO_SPI_csContinuous and one of the other flags to OR together to form the transfer flag.

Values:

enumerator kFLEXIO_SPI_8bitMsb
FlexIO SPI 8-bit MSB first

enumerator kFLEXIO_SPI_8bitLsb
FlexIO SPI 8-bit LSB first

enumerator kFLEXIO_SPI_16bitMsb
FlexIO SPI 16-bit MSB first

enumerator kFLEXIO_SPI_16bitLsb
FlexIO SPI 16-bit LSB first

enumerator kFLEXIO_SPI_32bitMsb
FlexIO SPI 32-bit MSB first

enumerator kFLEXIO_SPI_32bitLsb
FlexIO SPI 32-bit LSB first

enumerator kFLEXIO_SPI_csContinuous
Enable the CS signal continuous mode

typedef enum *_flexio_spi_clock_phase* flexio_spi_clock_phase_t
FlexIO SPI clock phase configuration.

typedef enum *_flexio_spi_shift_direction* flexio_spi_shift_direction_t
FlexIO SPI data shifter direction options.

typedef enum *_flexio_spi_data_bitcount_mode* flexio_spi_data_bitcount_mode_t
FlexIO SPI data length mode options.

typedef struct *_flexio_spi_type* FLEXIO_SPI_Type
Define FlexIO SPI access structure typedef.

typedef struct *_flexio_spi_master_config* flexio_spi_master_config_t
Define FlexIO SPI master configuration structure.

typedef struct *_flexio_spi_slave_config* flexio_spi_slave_config_t
Define FlexIO SPI slave configuration structure.

typedef struct *_flexio_spi_transfer* flexio_spi_transfer_t
Define FlexIO SPI transfer structure.

typedef struct *_flexio_spi_master_handle* flexio_spi_master_handle_t
typedef for flexio_spi_master_handle_t in advance.

typedef *flexio_spi_master_handle_t* flexio_spi_slave_handle_t
Slave handle is the same with master handle.


```
typedef void (*flexio_spi_master_transfer_callback_t)(FLEXIO_SPI_Type *base,  
flexio_spi_master_handle_t *handle, status_t status, void *userData)
```

FlexIO SPI master callback for finished transmit.

```
typedef void (*flexio_spi_slave_transfer_callback_t)(FLEXIO_SPI_Type *base,  
flexio_spi_slave_handle_t *handle, status_t status, void *userData)
```

FlexIO SPI slave callback for finished transmit.

```
FLEXIO_SPI_DUMMYDATA
```

FlexIO SPI dummy transfer data, the data is sent while txData is NULL.

```
SPI_RETRY_TIMES
```

Retry times for waiting flag.

```
FLEXIO_SPI_XFER_DATA_FORMAT(flag)
```

Get the transfer data format of width and bit order.

```
struct _flexio_spi_type
```

#include <fsl_flexio_spi.h> Define FlexIO SPI access structure typedef.

Public Members

```
FLEXIO_Type *flexioBase
```

FlexIO base pointer.

```
uint8_t SDOPinIndex
```

Pin select for data output. To set SDO pin in Hi-Z state, user needs to mux the pin as GPIO input and disable all pull up/down in application.

```
uint8_t SDIPinIndex
```

Pin select for data input.

```
uint8_t SCKPinIndex
```

Pin select for clock.

```
uint8_t CSnPinIndex
```

Pin select for enable.

```
uint8_t shifterIndex[2]
```

Shifter index used in FlexIO SPI.

```
uint8_t timerIndex[2]
```

Timer index used in FlexIO SPI.

```
struct _flexio_spi_master_config
```

#include <fsl_flexio_spi.h> Define FlexIO SPI master configuration structure.

Public Members

```
bool enableMaster
```

Enable/disable FlexIO SPI master after configuration.

```
bool enableInDoze
```

Enable/disable FlexIO operation in doze mode.

```
bool enableInDebug
```

Enable/disable FlexIO operation in debug mode.

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps

Baud rate in Bps.

flexio_spi_clock_phase_t phase

Clock phase.

flexio_spi_data_bitcount_mode_t dataMode

8bit or 16bit mode.

struct *_flexio_spi_slave_config*

#include <fsl_flexio_spi.h> Define FlexIO SPI slave configuration structure.

Public Members

bool enableSlave

Enable/disable FlexIO SPI slave after configuration.

bool enableInDoze

Enable/disable FlexIO operation in doze mode.

bool enableInDebug

Enable/disable FlexIO operation in debug mode.

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

flexio_spi_clock_phase_t phase

Clock phase.

flexio_spi_data_bitcount_mode_t dataMode

8bit or 16bit mode.

struct *_flexio_spi_transfer*

#include <fsl_flexio_spi.h> Define FlexIO SPI transfer structure.

Public Members

const uint8_t *txData

Send buffer.

uint8_t *rxData

Receive buffer.

size_t dataSize

Transfer bytes.

uint8_t flags

FlexIO SPI control flag, MSB first or LSB first.

struct *_flexio_spi_master_handle*

#include <fsl_flexio_spi.h> Define FlexIO SPI handle structure.

Public Members

`const uint8_t *txData`
Transfer buffer.

`uint8_t *rxData`
Receive buffer.

`size_t transferSize`
Total bytes to be transferred.

`volatile size_t txRemainingBytes`
Send data remaining in bytes.

`volatile size_t rxRemainingBytes`
Receive data remaining in bytes.

`volatile uint32_t state`
FlexIO SPI internal state.

`uint8_t bytePerFrame`
SPI mode, 2bytes or 1byte in a frame

`flexio_spi_shift_direction_t direction`
Shift direction.

`flexio_spi_master_transfer_callback_t callback`
FlexIO SPI callback.

`void *userData`
Callback parameter.

`bool isCsContinuous`
Is current transfer using CS continuous mode.

`uint32_t timer1Cfg`
TIMER1 TIMCFG regiser value backup.

2.52 FlexIO UART Driver

`status_t FLEXIO_UART_Init(FLEXIO_UART_Type *base, const flexio_uart_config_t *userConfig, uint32_t srcClock_Hz)`

Ungates the FlexIO clock, resets the FlexIO module, configures FlexIO UART hardware, and configures the FlexIO UART with FlexIO UART configuration. The configuration structure can be filled by the user or be set with default values by `FLEXIO_UART_GetDefaultConfig()`.

Example

```
FLEXIO_UART_Type base = {
    .flexioBase = FLEXIO,
    .TxPinIndex = 0,
    .RxPinIndex = 1,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_uart_config_t config = {
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .baudRate_Bps = 115200U,
```

(continues on next page)

(continued from previous page)

```
.bitCountPerChar = 8
};
FLEXIO_UART_Init(base, &config, srcClock_Hz);
```

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- userConfig – Pointer to the flexio_uart_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Configuration success.
- kStatus_FLEXIO_UART_BaudrateNotSupport – Baudrate is not supported for current clock source frequency.

```
void FLEXIO_UART_Deinit(FLEXIO_UART_Type *base)
```

Resets the FlexIO UART shifter and timer config.

Note: After calling this API, call the FLEXIO_UART_Init to use the FlexIO UART module.

Parameters

- base – Pointer to FLEXIO_UART_Type structure

```
void FLEXIO_UART_GetDefaultConfig(flexio_uart_config_t *userConfig)
```

Gets the default configuration to configure the FlexIO UART. The configuration can be used directly for calling the FLEXIO_UART_Init(). Example:

```
flexio_uart_config_t config;
FLEXIO_UART_GetDefaultConfig(&userConfig);
```

Parameters

- userConfig – Pointer to the flexio_uart_config_t structure.

```
uint32_t FLEXIO_UART_GetStatusFlags(FLEXIO_UART_Type *base)
```

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART status flags.

```
void FLEXIO_UART_ClearStatusFlags(FLEXIO_UART_Type *base, uint32_t mask)
```

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Status flag. The parameter can be any combination of the following values:
 - kFLEXIO_UART_TxDataRegEmptyFlag
 - kFLEXIO_UART_RxEmptyFlag

– kFLEXIO_UART_RxOverRunFlag

void FLEXIO_UART_EnableInterrupts(*FLEXIO_UART_Type* *base, uint32_t mask)

Enables the FlexIO UART interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Interrupt source.

void FLEXIO_UART_DisableInterrupts(*FLEXIO_UART_Type* *base, uint32_t mask)

Disables the FlexIO UART interrupt.

This function disables the FlexIO UART interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Interrupt source.

static inline uint32_t FLEXIO_UART_GetTxDataRegisterAddress(*FLEXIO_UART_Type* *base)

Gets the FlexIO UART transmit data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART transmit data register address.

static inline uint32_t FLEXIO_UART_GetRxDataRegisterAddress(*FLEXIO_UART_Type* *base)

Gets the FlexIO UART receive data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART receive data register address.

static inline void FLEXIO_UART_EnableTxDMA(*FLEXIO_UART_Type* *base, bool enable)

Enables/disables the FlexIO UART transmit DMA. This function enables/disables the FlexIO UART Tx DMA, which means asserting the kFLEXIO_UART_TxDataRegEmptyFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- enable – True to enable, false to disable.

static inline void FLEXIO_UART_EnableRxDMA(*FLEXIO_UART_Type* *base, bool enable)

Enables/disables the FlexIO UART receive DMA. This function enables/disables the FlexIO UART Rx DMA, which means asserting kFLEXIO_UART_RxDataRegFullFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- enable – True to enable, false to disable.

```
static inline void FLEXIO_UART_Enable(FLEXIO_UART_Type *base, bool enable)
```

Enables/disables the FlexIO UART module operation.

Parameters

- base – Pointer to the *FLEXIO_UART_Type*.
- enable – True to enable, false does not have any effect.

```
static inline void FLEXIO_UART_WriteByte(FLEXIO_UART_Type *base, const uint8_t *buffer)
```

Writes one byte of data.

Note: This is a non-blocking API, which returns directly after the data is put into the data register. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to the *FLEXIO_UART_Type* structure.
- buffer – The data bytes to send.

```
static inline void FLEXIO_UART_ReadByte(FLEXIO_UART_Type *base, uint8_t *buffer)
```

Reads one byte of data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

- base – Pointer to the *FLEXIO_UART_Type* structure.
- buffer – The buffer to store the received bytes.

```
status_t FLEXIO_UART_WriteBlocking(FLEXIO_UART_Type *base, const uint8_t *txData, size_t txSize)
```

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the *FLEXIO_UART_Type* structure.
- txData – The data bytes to send.
- txSize – The number of data bytes to send.

Return values

- kStatus_FLEXIO_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

```
status_t FLEXIO_UART_ReadBlocking(FLEXIO_UART_Type *base, uint8_t *rxData, size_t rxSize)
```

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- rxData – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus_FLEXIO_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t FLEXIO_UART_TransferCreateHandle(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_callback_t* callback, void *userData)

Initializes the UART handle.

This function initializes the FlexIO UART handle, which can be used for other FlexIO UART transactional APIs. Call this API once to get the initialized handle.

The UART driver supports the “background” receiving, which means that users can set up a RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn’t call the FLEXIO_UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as ringBuffer.

Parameters

- base – to FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

void FLEXIO_UART_TransferStartRingBuffer(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, uint8_t *ringBuffer, size_t ringBufferSize)

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn’t call the UART_ReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly.

Note: When using the RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, only 31 bytes are used for saving data.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – Size of the ring buffer.

```
void FLEXIO_UART_TransferStopRingBuffer(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle)
```

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

```
status_t FLEXIO_UART_TransferSendNonBlocking(FLEXIO_UART_Type *base,
                                              flexio_uart_handle_t *handle,
                                              flexio_uart_transfer_t *xfer)
```

Transmits a buffer of data using the interrupt method.

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in ISR, the FlexIO UART driver calls the callback function and passes the kStatus_FLEXIO_UART_TxIdle as status parameter.

Note: The kStatus_FLEXIO_UART_TxIdle is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- xfer – FlexIO UART transfer structure. See flexio_uart_transfer_t.

Return values

- kStatus_Success – Successfully starts the data transmission.
- kStatus_UART_TxBusy – Previous transmission still not finished, data not written to the TX register.

```
void FLEXIO_UART_TransferAbortSend(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle)
```

Aborts the interrupt-driven data transmit.

This function aborts the interrupt-driven data sending. Get the remainBytes to find out how many bytes are still not sent out.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

```
status_t FLEXIO_UART_TransferGetSendCount(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, size_t *count)
```

Gets the number of bytes sent.

This function gets the number of bytes sent driven by interrupt.

Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.
- `count` – Number of bytes sent so far by the non-blocking transaction.

Return values

- `kStatus_NoTransferInProgress` – transfer has finished or no transfer in progress.
- `kStatus_Success` – Successfully return the count.

```
status_t FLEXIO_UART_TransferReceiveNonBlocking(FLEXIO_UART_Type *base,  
                                                flexio_uart_handle_t *handle,  
                                                flexio_uart_transfer_t *xfer, size_t  
                                                *receivedBytes)
```

Receives a buffer of data using the interrupt method.

This function receives data using the interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in ring buffer is not enough to read, the receive request is saved by the UART driver. When new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter `kStatus_UART_RxIdle`. For example, if the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer, the 5 bytes are copied to `xfer->data`. This function returns with the parameter `receivedBytes` set to 5. For the last 5 bytes, newly arrived data is saved from the `xfer->data[5]`. When 5 bytes are received, the UART driver notifies upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to `xfer->data`. When all data is received, the upper layer is notified.

Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.
- `xfer` – UART transfer structure. See `flexio_uart_transfer_t`.
- `receivedBytes` – Bytes received from the ring buffer directly.

Return values

- `kStatus_Success` – Successfully queue the transfer into the transmit queue.
- `kStatus_FLEXIO_UART_RxBusy` – Previous receive request is not finished.

```
void FLEXIO_UART_TransferAbortReceive(FLEXIO_UART_Type *base, flexio_uart_handle_t  
                                     *handle)
```

Aborts the receive data which was using IRQ.

This function aborts the receive data which was using IRQ.

Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.

status_t FLEXIO_UART_TransferGetReceiveCount(*FLEXIO_UART_Type* *base,
flexio_uart_handle_t *handle, size_t *count)

Gets the number of bytes received.

This function gets the number of bytes received driven by interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- count – Number of bytes received so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

void FLEXIO_UART_TransferHandleIRQ(void *uartType, void *uartHandle)

FlexIO UART IRQ handler function.

This function processes the FlexIO UART transmit and receives the IRQ request.

Parameters

- uartType – Pointer to the FLEXIO_UART_Type structure.
- uartHandle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

void FLEXIO_UART_FlushShifters(*FLEXIO_UART_Type* *base)

Flush tx/rx shifters.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

FSL_FLEXIO_UART_DRIVER_VERSION

FlexIO UART driver version.

Error codes for the UART driver.

Values:

enumerator kStatus_FLEXIO_UART_TxBusy

Transmitter is busy.

enumerator kStatus_FLEXIO_UART_RxBusy

Receiver is busy.

enumerator kStatus_FLEXIO_UART_TxIdle

UART transmitter is idle.

enumerator kStatus_FLEXIO_UART_RxIdle

UART receiver is idle.

enumerator kStatus_FLEXIO_UART_ERROR

ERROR happens on UART.

enumerator kStatus_FLEXIO_UART_RxRingBufferOverrun

UART RX software ring buffer overrun.

enumerator kStatus_FLEXIO_UART_RxHardwareOverrun
UART RX receiver overrun.

enumerator kStatus_FLEXIO_UART_Timeout
UART times out.

enumerator kStatus_FLEXIO_UART_BaudrateNotSupport
Baudrate is not supported in current clock source

enum _flexio_uart_bit_count_per_char
FlexIO UART bit count per char.

Values:

enumerator kFLEXIO_UART_7BitsPerChar
7-bit data characters

enumerator kFLEXIO_UART_8BitsPerChar
8-bit data characters

enumerator kFLEXIO_UART_9BitsPerChar
9-bit data characters

enum _flexio_uart_interrupt_enable
Define FlexIO UART interrupt mask.

Values:

enumerator kFLEXIO_UART_TxDataRegEmptyInterruptEnable
Transmit buffer empty interrupt enable.

enumerator kFLEXIO_UART_RxDataRegFullInterruptEnable
Receive buffer full interrupt enable.

enum _flexio_uart_status_flags
Define FlexIO UART status mask.

Values:

enumerator kFLEXIO_UART_TxDataRegEmptyFlag
Transmit buffer empty flag.

enumerator kFLEXIO_UART_RxDataRegFullFlag
Receive buffer full flag.

enumerator kFLEXIO_UART_RxOverRunFlag
Receive buffer over run flag.

typedef enum _flexio_uart_bit_count_per_char flexio_uart_bit_count_per_char_t
FlexIO UART bit count per char.

typedef struct _flexio_uart_type FLEXIO_UART_Type
Define FlexIO UART access structure typedef.

typedef struct _flexio_uart_config flexio_uart_config_t
Define FlexIO UART user configuration structure.

typedef struct _flexio_uart_transfer flexio_uart_transfer_t
Define FlexIO UART transfer structure.

typedef struct _flexio_uart_handle flexio_uart_handle_t

```
typedef void (*flexio_uart_transfer_callback_t)(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, status_t status, void *userData)
```

FlexIO UART transfer callback function.

```
UART_RETRY_TIMES
```

Retry times for waiting flag.

```
struct _flexio_uart_type
```

#include <fsl_flexio_uart.h> Define FlexIO UART access structure typedef.

Public Members

```
FLEXIO_Type *flexioBase
```

FlexIO base pointer.

```
uint8_t TxPinIndex
```

Pin select for UART_Tx.

```
uint8_t RxPinIndex
```

Pin select for UART_Rx.

```
uint8_t shifterIndex[2]
```

Shifter index used in FlexIO UART.

```
uint8_t timerIndex[2]
```

Timer index used in FlexIO UART.

```
struct _flexio_uart_config
```

#include <fsl_flexio_uart.h> Define FlexIO UART user configuration structure.

Public Members

```
bool enableUart
```

Enable/disable FlexIO UART TX & RX.

```
bool enableInDoze
```

Enable/disable FlexIO operation in doze mode

```
bool enableInDebug
```

Enable/disable FlexIO operation in debug mode

```
bool enableFastAccess
```

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

```
uint32_t baudRate_Bps
```

Baud rate in Bps.

```
flexio_uart_bit_count_per_char_t bitCountPerChar
```

number of bits, 7/8/9 -bit

```
struct _flexio_uart_transfer
```

#include <fsl_flexio_uart.h> Define FlexIO UART transfer structure.

Public Members

```
size_t dataSize
```

Transfer size

struct `_flexio_uart_handle`
#include <fsl_flexio_uart.h> Define FLEXIO UART handle structure.

Public Members

`const uint8_t *volatile txData`
Address of remaining data to send.

`volatile size_t txDataSize`
Size of the remaining data to send.

`uint8_t *volatile rxData`
Address of remaining data to receive.

`volatile size_t rxDataSize`
Size of the remaining data to receive.

`size_t txDataSizeAll`
Total bytes to be sent.

`size_t rxDataSizeAll`
Total bytes to be received.

`uint8_t *rxRingBuffer`
Start address of the receiver ring buffer.

`size_t rxRingBufferSize`
Size of the ring buffer.

`volatile uint16_t rxRingBufferHead`
Index for the driver to store received data into ring buffer.

`volatile uint16_t rxRingBufferTail`
Index for the user to get data from the ring buffer.

flexio_uart_transfer_callback_t callback
Callback function.

`void *userData`
UART callback function parameter.

`volatile uint8_t txState`
TX transfer state.

`volatile uint8_t rxState`
RX transfer state

union `__unnamed144__`

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.

2.53 FLEXRAM: on-chip RAM manager

FLEXRAM bank type.

Values:

enumerator kFLEXRAM_BankNotUsed
bank is not used

enumerator kFLEXRAM_BankOCRAM
bank is OCRM

enumerator kFLEXRAM_BankDTCM
bank is DTCM

enumerator kFLEXRAM_BankITCM
bank is ITCM

enum _flexram_bank_allocate_src
FLEXRAM bank allocate source.

Values:

enumerator kFLEXRAM_BankAllocateThroughHardwareFuse
allocate ram through hardware fuse value

enumerator kFLEXRAM_BankAllocateThroughBankCfg
allocate ram through FLEXRAM_BANK_CFG

typedef enum _flexram_bank_allocate_src flexram_bank_allocate_src_t
FLEXRAM bank allocate source.

typedef struct _flexram_allocate_ram flexram_allocate_ram_t
FLEXRAM allocates OCRM, ITCM, DTCM size.

status_t FLEXRAM_AllocateRam(flexram_allocate_ram_t *config)
FLEXRAM allocates an on-chip ram for OCRM, ITCM and DTCM. This function is independent from FLEXRAM_Init, and can be called directly if ram re-allocate is needed.

Parameters

- config – Allocate configuration.

Return values

- kStatus_InvalidArgument – When the argument is invalid.
- kStatus_Success – Upon allocate success.

static inline void FLEXRAM_SetAllocateRamSrc(flexram_bank_allocate_src_t src)
FLEXRAM set allocate on-chip ram source.

Parameters

- src – Bank config source select value.

FSL_SOC_FLEXRAM_ALLOCATE_DRIVER_VERSION
SOC_FLEXRAM_ALLOCATE driver version 2.0.2.

void FLEXRAM_Init(FLEXRAM_Type *base)
FLEXRAM module initialization function.

Parameters

- base – FLEXRAM base address.

`void FLEXRAM_Deinit(FLEXRAM_Type *base)`

De-initializes the FLEXRAM.

`static inline uint32_t FLEXRAM_GetInterruptStatus(FLEXRAM_Type *base)`

FLEXRAM module gets interrupt status.

Parameters

- `base` – FLEXRAM base address.

`static inline void FLEXRAM_ClearInterruptStatus(FLEXRAM_Type *base, uint32_t status)`

FLEXRAM module clears interrupt status.

Parameters

- `base` – FLEXRAM base address.
- `status` – Status to be cleared.

`static inline void FLEXRAM_EnableInterruptStatus(FLEXRAM_Type *base, uint32_t status)`

FLEXRAM module enables interrupt status.

Parameters

- `base` – FLEXRAM base address.
- `status` – Status to be enabled.

`static inline void FLEXRAM_DisableInterruptStatus(FLEXRAM_Type *base, uint32_t status)`

FLEXRAM module disable interrupt status.

Parameters

- `base` – FLEXRAM base address.
- `status` – Status to be disabled.

`static inline void FLEXRAM_EnableInterruptSignal(FLEXRAM_Type *base, uint32_t status)`

FLEXRAM module enables interrupt.

Parameters

- `base` – FLEXRAM base address.
- `status` – Status interrupt to be enabled.

`static inline void FLEXRAM_DisableInterruptSignal(FLEXRAM_Type *base, uint32_t status)`

FLEXRAM module disables interrupt.

Parameters

- `base` – FLEXRAM base address.
- `status` – Status interrupt to be disabled.

`FSL_FLEXRAM_DRIVER_VERSION`

Driver version.

Flexram write/read selection.

Values:

enumerator `kFLEXRAM_Read`

read

enumerator `kFLEXRAM_Write`

write

Interrupt status flag mask.

Values:

enumerator kFLEXRAM_OCRAMAccessError
OCRAM accesses unallocated address

enumerator kFLEXRAM_DTCMAccessError
DTCM accesses unallocated address

enumerator kFLEXRAM_ITCMAccessError
ITCM accesses unallocated address

enumerator kFLEXRAM_OCRAMMagicAddrMatch
OCRAM magic address match

enumerator kFLEXRAM_DTCMMagicAddrMatch
DTCM magic address match

enumerator kFLEXRAM_ITCMMagicAddrMatch
ITCM magic address match

enumerator kFLEXRAM_OCRAMECCMultiError

enumerator kFLEXRAM_OCRAMECCSingleError

enumerator kFLEXRAM_ITCMECCMultiError

enumerator kFLEXRAM_ITCMECCSingleError

enumerator kFLEXRAM_D0TCMECCMultiError

enumerator kFLEXRAM_D0TCMECCSingleError

enumerator kFLEXRAM_D1TCMECCMultiError

enumerator kFLEXRAM_D1TCMECCSingleError

enumerator kFLEXRAM_InterruptStatusAll

enum _flexram_tcm_access_mode

FLEXRAM TCM access mode. Fast access mode expected to be finished in 1-cycle; Wait access mode expected to be finished in 2-cycle. Wait access mode is a feature of the flexram and it should be used when the CPU clock is too fast to finish TCM access in 1-cycle. Normally, fast mode is the default mode, the efficiency of the TCM access will better.

Values:

enumerator kFLEXRAM_TCMAccessFastMode
fast access mode

enumerator kFLEXRAM_TCMAccessWaitMode
wait access mode

FLEXRAM TCM support size.

Values:

enumerator kFLEXRAM_TCMSize32KB
TCM total size be 32KB

enumerator kFLEXRAM_TCMSize64KB
TCM total size be 64KB

enumerator kFLEXRAM_TCMSize128KB
TCM total size be 128KB

enumerator kFLEXRAM_TCMSize256KB
TCM total size be 256KB

enumerator kFLEXRAM_TCMSize512KB
TCM total size be 512KB

enum *_flexram_memory_type*

FLEXRAM memory type, such as OCRAM/ITCM/D0TCM/D1TCM.

Values:

enumerator kFLEXRAM_OCRAM
Memory type OCRAM

enumerator kFLEXRAM_ITCM
Memory type ITCM

enumerator kFLEXRAM_D0TCM
Memory type D0TCM

enumerator kFLEXRAM_D1TCM
Memory type D1TCM

typedef enum *_flexram_tcm_access_mode* flexram_tcm_access_mode_t

FLEXRAM TCM access mode. Fast access mode expected to be finished in 1-cycle; Wait access mode expected to be finished in 2-cycle. Wait access mode is a feature of the flexram and it should be used when the CPU clock is too fast to finish TCM access in 1-cycle. Normally, fast mode is the default mode, the efficiency of the TCM access will better.

typedef enum *_flexram_memory_type* flexram_memory_type_t

FLEXRAM memory type, such as OCRAM/ITCM/D0TCM/D1TCM.

typedef struct *_flexram_ecc_error_type* flexram_ecc_error_type_t

FLEXRAM error type, such as single bit error position, multi-bit error position.

typedef struct *_flexram_ocram_ecc_single_error_info* flexram_ocram_ecc_single_error_info_t

FLEXRAM ocram ecc single error information, including single error information, error address, error data.

typedef struct *_flexram_ocram_ecc_multi_error_info* flexram_ocram_ecc_multi_error_info_t

FLEXRAM ocram ecc multiple error information, including multiple error information, error address, error data.

typedef struct *_flexram_itcm_ecc_single_error_info* flexram_itcm_ecc_single_error_info_t

FLEXRAM itcm ecc single error information, including single error information, error address, error data.

typedef struct *_flexram_itcm_ecc_multi_error_info* flexram_itcm_ecc_multi_error_info_t

FLEXRAM itcm ecc multiple error information, including multiple error information, error address, error data.

typedef struct *_flexram_dtcn_ecc_single_error_info* flexram_dtcn_ecc_single_error_info_t

FLEXRAM dtcm ecc single error information, including single error information, error address, error data.

typedef struct *_flexram_dtcn_ecc_multi_error_info* flexram_dtcn_ecc_multi_error_info_t

FLEXRAM dtcm ecc multiple error information, including multiple error information, error address, error data.

const uint8_t ocramBankNum

OCRAM banknumber which the SOC support.

const uint8_t dtcmBankNum

DTCM bank number to allocate, the number should be power of 2.

const uint8_t itcmBankNum

ITCM bank number to allocate, the number should be power of 2.

static inline void FLEXRAM_SetTCMReadAccessMode(FLEXRAM_Type *base,
flexram_tcm_access_mode_t mode)

FLEXRAM module sets TCM read access mode.

Parameters

- base – FLEXRAM base address.
- mode – Access mode.

static inline void FLEXRAM_SetTCMWriteAccessMode(FLEXRAM_Type *base,
flexram_tcm_access_mode_t mode)

FLEXRAM module set TCM write access mode.

Parameters

- base – FLEXRAM base address.
- mode – Access mode.

static inline void FLEXRAM_EnableForceRamClockOn(FLEXRAM_Type *base, bool enable)

FLEXRAM module force ram clock on.

Parameters

- base – FLEXRAM base address.
- enable – Enable or disable clock force on.

static inline void FLEXRAM_SetOCRAMMagicAddr(FLEXRAM_Type *base, uint16_t magicAddr,
uint32_t rwSel)

FLEXRAM OCRAM magic addr configuration. When read/write access hit magic address, it will generate interrupt.

Parameters

- base – FLEXRAM base address.
- magicAddr – Magic address, the actual address bits [18:3] is corresponding to the register field [16:1].
- rwSel – Read/write selection. 0 for read access while 1 for write access.

static inline void FLEXRAM_SetDTCMMagicAddr(FLEXRAM_Type *base, uint16_t magicAddr,
uint32_t rwSel)

FLEXRAM DTCM magic addr configuration. When read/write access hits magic address, it will generate interrupt.

Parameters

- base – FLEXRAM base address.
- magicAddr – Magic address, the actual address bits [18:3] is corresponding to the register field [16:1].
- rwSel – Read/write selection. 0 for read access while 1 write access.

```
static inline void FLEXRAM_SetITCMMagicAddr(FLEXRAM_Type *base, uint16_t magicAddr,  
                                             uint32_t rwSel)
```

FLEXRAM ITCM magic addr configuration. When read/write access hits magic address, it will generate interrupt.

Parameters

- base – FLEXRAM base address.
- magicAddr – Magic address, the actual address bits [18:3] is corresponding to the register field [16:1].
- rwSel – Read/write selection. 0 for read access while 1 for write access.

```
void FLEXRAM_EnableECC(FLEXRAM_Type *base, bool OcramECCEnable, bool  
                      TcmECCEnable)
```

FLEXRAM get ocram ecc single error information.

Parameters

- base – FLEXRAM base address.
- OcramECCEnable – ocram ecc enablement.
- TcmECCEnable – tcm(itcm/d0tcm/d1tcm) ecc enablement.

```
void FLEXRAM_ErrorInjection(FLEXRAM_Type *base, flexram_memory_type_t memory,  
                           flexram_ecc_error_type_t *error)
```

FLEXRAM ECC error injection.

Parameters

- base – FLEXRAM base address.
- memory – memory type, such as OCRAM/ITCM/DTCM.
- error – ECC error type.

```
void FLEXRAM_GetOcramSingleErrInfo(FLEXRAM_Type *base,  
                                   flexram_ocram_ecc_single_error_info_t *info)
```

FLEXRAM get ocram ecc single error information.

Parameters

- base – FLEXRAM base address.
- info – ecc error information.

```
void FLEXRAM_GetOcramMultiErrInfo(FLEXRAM_Type *base,  
                                  flexram_ocram_ecc_multi_error_info_t *info)
```

FLEXRAM get ocram ecc multiple error information.

Parameters

- base – FLEXRAM base address.
- info – ecc error information.

```
void FLEXRAM_GetItcmSingleErrInfo(FLEXRAM_Type *base,  
                                  flexram_itcm_ecc_single_error_info_t *info)
```

FLEXRAM get itcm ecc single error information.

Parameters

- base – FLEXRAM base address.
- info – ecc error information.

```
void FLEXRAM_GetItcmMultiErroInfo(FLEXRAM_Type *base,  
                                   flexram_itcm_ecc_multi_error_info_t *info)
```

FLEXRAM get itcm ecc multiple error information.

Parameters

- base – FLEXRAM base address.
- info – ecc error information.

```
void FLEXRAM_GetDtcnSingleErroInfo(FLEXRAM_Type *base,  
                                    flexram_dtcn_ecc_single_error_info_t *info, uint8_t  
                                    bank)
```

FLEXRAM get d0tcn ecc single error information.

Parameters

- base – FLEXRAM base address.
- info – ecc error information.
- bank – DTCN bank, 0 is D0TCN, 1 is D1TCN.

```
void FLEXRAM_GetDtcnMultiErroInfo(FLEXRAM_Type *base,  
                                   flexram_dtcn_ecc_multi_error_info_t *info, uint8_t  
                                   bank)
```

FLEXRAM get d0tcn ecc multiple error information.

Parameters

- base – FLEXRAM base address.
- info – ecc error information.
- bank – DTCN bank, 0 is D0TCN, 1 is D1TCN.

FLEXRAM_ECC_ERROR_DETAILED_INFO

Get ECC error detailed information.

```
struct flexram_allocate_ram
```

#include <fsl_flexram_allocate.h> FLEXRAM allocates OCRAM, ITCM, DTCN size.

```
struct flexram_ecc_error_type
```

#include <fsl_flexram.h> FLEXRAM error type, such as single bit error position, multi-bit error position.

Public Members

uint8_t SingleBitPos

Bit position of the bit to inject ECC Error.

uint8_t SecondBitPos

Bit position of the second bit to inject multi-bit ECC Error

bool Force1BitDataInversion

Force One 1-Bit Data Inversion (single-bit ECC error) on memory write access

bool ForceOneNCDataInversion

Force One Non-correctable Data Inversion(multi-bit ECC error) on memory write access

bool ForceConti1BitDataInversion

Force Continuous 1-Bit Data Inversions (single-bit ECC error) on memory write access

bool FourceContiNCDataInversion

Force Continuous Non-correctable Data Inversions (multi-bit ECC error) on memory write access

struct _flexram_ocram_ecc_single_error_info

#include <fsl_flexram.h> FLEXRAM ocram ecc single error information, including single error information, error address, error data.

Public Members

uint32_t OcramSingleErrorInfo

Ocram single error information, user should parse it by themself.

uint32_t OcramSingleErrorAddr

Ocram single error address

uint32_t OcramSingleErrorDataLSB

Ocram single error data LSB

uint32_t OcramSingleErrorDataMSB

Ocram single error data MSB

struct _flexram_ocram_ecc_multi_error_info

#include <fsl_flexram.h> FLEXRAM ocram ecc multiple error information, including multiple error information, error address, error data.

Public Members

uint32_t OcramMultiErrorInfo

Ocram single error information, user should parse it by themself.

uint32_t OcramMultiErrorAddr

Ocram multiple error address

uint32_t OcramMultiErrorDataLSB

Ocram multiple error data LSB

uint32_t OcramMultiErrorDataMSB

Ocram multiple error data MSB

struct _flexram_itcm_ecc_single_error_info

#include <fsl_flexram.h> FLEXRAM itcm ecc single error information, including single error information, error address, error data.

Public Members

uint32_t ItcmSingleErrorInfo

itcm single error information, user should parse it by themself.

uint32_t ItcmSingleErrorAddr

itcm single error address

uint32_t ItcmSingleErrorDataLSB

itcm single error data LSB

uint32_t ItcmSingleErrorDataMSB

itcm single error data MSB

```
struct _flexram_itcm_ecc_multi_error_info
```

#include <fsl_flexram.h> FLEXRAM itcm ecc multiple error information, including multiple error information, error address, error data.

Public Members

```
uint32_t ItcmMultiErrorInfo
```

itcm multiple error information, user should parse it by themself.

```
uint32_t ItcmMultiErrorAddr
```

itcm multiple error address

```
uint32_t ItcmMultiErrorDataLSB
```

itcm multiple error data LSB

```
uint32_t ItcmMultiErrorDataMSB
```

itcm multiple error data MSB

```
struct _flexram_dtcn_ecc_single_error_info
```

#include <fsl_flexram.h> FLEXRAM dtcm ecc single error information, including single error information, error address, error data.

Public Members

```
uint32_t DtcnSingleErrorInfo
```

dtcm single error information, user should parse it by themself.

```
uint32_t DtcnSingleErrorAddr
```

dtcm single error address

```
uint32_t DtcnSingleErrorData
```

dtcm single error data

```
struct _flexram_dtcn_ecc_multi_error_info
```

#include <fsl_flexram.h> FLEXRAM dtcm ecc multiple error information, including multiple error information, error address, error data.

Public Members

```
uint32_t DtcnMultiErrorInfo
```

dtcm multiple error information, user should parse it by themself.

```
uint32_t DtcnMultiErrorAddr
```

dtcm multiple error address

```
uint32_t DtcnMultiErrorData
```

dtcm multiple error data

2.54 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 enableCombination

Enable/disable combining PORT A and B Data Pins (SIOA[3:0] and SIOB[3:0]) to support Flash Octal mode.

bool enableDoze

Enable/disable doze mode support.

bool enableHalfSpeedAccess

Enable/disable divide by 2 of the clock for half speed commands.

bool enableSckBDiffOpt

Enable/disable SCKB pad use as SCKA differential clock output, when enable, Port B flash access is not available.

bool enableSameConfigForAll

Enable/disable same configuration for all connected devices when enabled, same configuration in FLASHA1CRx is applied to all.

uint16_t seqTimeoutCycle

Timeout wait cycle for command sequence execution, timeout after 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.

2.55 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.

`enum _flexspi_edma_ntransfer_size`

eDMA transfer configuration

Values:

`enumerator kFLEXPSI_EDMAAnSize1Bytes`

Source/Destination data transfer size is 1 byte every time

`enumerator kFLEXPSI_EDMAAnSize2Bytes`

Source/Destination data transfer size is 2 bytes every time

`enumerator kFLEXPSI_EDMAAnSize4Bytes`

Source/Destination data transfer size is 4 bytes every time

`enumerator kFLEXPSI_EDMAAnSize8Bytes`

Source/Destination data transfer size is 8 bytes every time

`enumerator kFLEXPSI_EDMAAnSize32Bytes`

Source/Destination data transfer size is 32 bytes every time

`typedef struct _flexspi_edma_handle flexspi_edma_handle_t`

`typedef void (*flexspi_edma_callback_t)(FLEXSPI_Type *base, flexspi_edma_handle_t *handle, status_t status, void *userData)`

FLEXSPI eDMA transfer callback function for finish and error.

`typedef enum _flexspi_edma_ntransfer_size flexspi_edma_transfer_nsize_t`

eDMA transfer configuration

`struct _flexspi_edma_handle`

#include <fsl_flexspi_edma.h> FLEXSPI DMA transfer handle, users should not touch the content of the handle.

Public Members

`edma_handle_t *txDmaHandle`

eDMA handler for FLEXSPI Tx.

`edma_handle_t *rxDmaHandle`

eDMA handler for FLEXSPI Rx.

`size_t transferSize`

Bytes need to transfer.

`flexspi_edma_transfer_nsize_t nsize`

eDMA SSIZE/DSIZE in each transfer.

`uint8_t nbytes`

eDMA minor byte transfer count initially configured.

`uint8_t count`

The transfer data count in a DMA request.

`uint32_t state`

Internal state for FLEXSPI eDMA transfer.

flexspi_edma_callback_t completionCallback

A callback function called after the eDMA transfer is finished.

void *userData

User callback parameter

2.56 GPIO: General-Purpose Input/Output Driver

void GPIO_PinInit(GPIO_Type *base, uint32_t pin, const *gpio_pin_config_t* *Config)

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

Parameters

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

void GPIO_PinWrite(GPIO_Type *base, uint32_t pin, uint8_t output)

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

Parameters

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

static inline void GPIO_WritePinOutput(GPIO_Type *base, uint32_t pin, uint8_t output)

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

Deprecated:

Do not use this function. It has been superseded by GPIO_PinWrite.

static inline void GPIO_PortSet(GPIO_Type *base, uint32_t mask)

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

Parameters

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

static inline void GPIO_SetPinsOutput(GPIO_Type *base, uint32_t mask)

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

Deprecated:

Do not use this function. It has been superseded by GPIO_PortSet.

static inline void GPIO_PortClear(GPIO_Type *base, uint32_t mask)

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

Parameters

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

- mask – GPIO pin number macro

static inline void GPIO_ClearPinsOutput(GPIO_Type *base, uint32_t mask)

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

Deprecated:

Do not use this function. It has been superceded by GPIO_PortClear.

static inline void GPIO_PortToggle(GPIO_Type *base, uint32_t mask)

Reverses the current output logic of the multiple GPIO pins.

Parameters

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

static inline uint32_t GPIO_PinRead(GPIO_Type *base, uint32_t pin)

Reads the current input value of the GPIO port.

Parameters

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

Return values

GPIO – port input value.

static inline uint32_t GPIO_ReadPinInput(GPIO_Type *base, uint32_t pin)

Reads the current input value of the GPIO port.

Deprecated:

Do not use this function. It has been superceded by GPIO_PinRead.

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

Reads the current GPIO pin pad status.

Parameters

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

Return values

GPIO – pin pad status value.

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

Reads the current GPIO pin pad status.

Deprecated:

Do not use this function. It has been superceded by GPIO_PinReadPadStatus.

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

Sets the current pin interrupt mode.

Parameters

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

- `pinInterruptMode` – pointer to a `gpio_interrupt_mode_t` structure that contains the interrupt mode information.

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

Sets the current pin interrupt mode.

Deprecated:

Do not use this function. It has been superceded by `GPIO_PinSetInterruptConfig`.

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

Enables the specific pin interrupt.

Parameters

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

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

Enables the specific pin interrupt.

Parameters

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

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

Disables the specific pin interrupt.

Parameters

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

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

Disables the specific pin interrupt.

Deprecated:

Do not use this function. It has been superceded by `GPIO_PortDisableInterrupts`.

```
static inline uint32_t GPIO_PortGetInterruptFlags(GPIO_Type *base)
```

Reads individual pin interrupt status.

Parameters

- `base` – GPIO base pointer.

Return values

`current` – pin interrupt status flag.

```
static inline uint32_t GPIO_GetPinsInterruptFlags(GPIO_Type *base)
```

Reads individual pin interrupt status.

Parameters

- `base` – GPIO base pointer.

Return values

`current` – pin interrupt status flag.

static inline void GPIO_PortClearInterruptFlags(GPIO_Type *base, uint32_t mask)

Clears pin interrupt flag. Status flags are cleared by writing a 1 to the corresponding bit position.

Parameters

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

static inline void GPIO_ClearPinsInterruptFlags(GPIO_Type *base, uint32_t mask)

Clears pin interrupt flag. Status flags are cleared by writing a 1 to the corresponding bit position.

Parameters

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

FSL_GPIO_DRIVER_VERSION

GPIO driver version.

enum _gpio_pin_direction

GPIO direction definition.

Values:

enumerator kGPIO_DigitalInput

Set current pin as digital input.

enumerator kGPIO_DigitalOutput

Set current pin as digital output.

enum _gpio_interrupt_mode

GPIO interrupt mode definition.

Values:

enumerator kGPIO_NoIntmode

Set current pin general IO functionality.

enumerator kGPIO_IntLowLevel

Set current pin interrupt is low-level sensitive.

enumerator kGPIO_IntHighLevel

Set current pin interrupt is high-level sensitive.

enumerator kGPIO_IntRisingEdge

Set current pin interrupt is rising-edge sensitive.

enumerator kGPIO_IntFallingEdge

Set current pin interrupt is falling-edge sensitive.

enumerator kGPIO_IntRisingOrFallingEdge

Enable the edge select bit to override the ICR register's configuration.

typedef enum _gpio_pin_direction gpio_pin_direction_t

GPIO direction definition.

typedef enum _gpio_interrupt_mode gpio_interrupt_mode_t

GPIO interrupt mode definition.

typedef struct _gpio_pin_config gpio_pin_config_t

GPIO Init structure definition.

struct `_gpio_pin_config`
`#include <fsl_gpio.h>` GPIO Init structure definition.

Public Members

`gpio_pin_direction_t` direction
 Specifies the pin direction.

`uint8_t` outputLogic
 Set a default output logic, which has no use in input

`gpio_interrupt_mode_t` interruptMode
 Specifies the pin interrupt mode, a value of `gpio_interrupt_mode_t`.

2.57 GPT: General Purpose Timer

void `GPT_Init(GPT_Type *base, const gpt_config_t *initConfig)`
 Initialize GPT to reset state and initialize running mode.

Parameters

- `base` – GPT peripheral base address.
- `initConfig` – GPT mode setting configuration.

void `GPT_Deinit(GPT_Type *base)`
 Disables the module and gates the GPT clock.

Parameters

- `base` – GPT peripheral base address.

void `GPT_GetDefaultConfig(gpt_config_t *config)`
 Fills in the GPT configuration structure with default settings.
 The default values are:

```
config->clockSource = kGPT_ClockSource_Periph;
config->divider = 1U;
config->enableRunInStop = true;
config->enableRunInWait = true;
config->enableRunInDoze = false;
config->enableRunInDbg = false;
config->enableFreeRun = false;
config->enableMode = true;
```

Parameters

- `config` – Pointer to the user configuration structure.

static inline void `GPT_SoftwareReset(GPT_Type *base)`
 Software reset of GPT module.

Parameters

- `base` – GPT peripheral base address.

static inline void `GPT_SetClockSource(GPT_Type *base, gpt_clock_source_t gptClkSource)`
 Set clock source of GPT.

Parameters

- base – GPT peripheral base address.
- gptClkSource – Clock source (see `gpt_clock_source_t` typedef enumeration).

static inline *gpt_clock_source_t* GPT_GetClockSource(*GPT_Type* *base)

Get clock source of GPT.

Parameters

- base – GPT peripheral base address.

Returns

clock source (see `gpt_clock_source_t` typedef enumeration).

static inline void GPT_SetClockDivider(*GPT_Type* *base, *uint32_t* divider)

Set pre scaler of GPT.

Parameters

- base – GPT peripheral base address.
- divider – Divider of GPT (1-4096).

static inline *uint32_t* GPT_GetClockDivider(*GPT_Type* *base)

Get clock divider in GPT module.

Parameters

- base – GPT peripheral base address.

Returns

clock divider in GPT module (1-4096).

static inline void GPT_SetOscClockDivider(*GPT_Type* *base, *uint32_t* divider)

OSC 24M pre-scaler before selected by clock source.

Parameters

- base – GPT peripheral base address.
- divider – OSC Divider(1-16).

static inline *uint32_t* GPT_GetOscClockDivider(*GPT_Type* *base)

Get OSC 24M clock divider in GPT module.

Parameters

- base – GPT peripheral base address.

Returns

OSC clock divider in GPT module (1-16).

static inline void GPT_StartTimer(*GPT_Type* *base)

Start GPT timer.

Parameters

- base – GPT peripheral base address.

static inline void GPT_StopTimer(*GPT_Type* *base)

Stop GPT timer.

Parameters

- base – GPT peripheral base address.

static inline *uint32_t* GPT_GetCurrentTimerCount(*GPT_Type* *base)

Reads the current GPT counting value.

Parameters

- base – GPT peripheral base address.

Returns

Current GPT counter value.

```
static inline void GPT_SetInputOperationMode(GPT_Type *base, gpt_input_capture_channel_t
                                             channel, gpt_input_operation_mode_t mode)
```

Set GPT operation mode of input capture channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT capture channel (see gpt_input_capture_channel_t typedef enumeration).
- mode – GPT input capture operation mode (see gpt_input_operation_mode_t typedef enumeration).

```
static inline gpt_input_operation_mode_t GPT_GetInputOperationMode(GPT_Type *base,
                                                                    gpt_input_capture_channel_t
                                                                    channel)
```

Get GPT operation mode of input capture channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT capture channel (see gpt_input_capture_channel_t typedef enumeration).

Returns

GPT input capture operation mode (see gpt_input_operation_mode_t typedef enumeration).

```
static inline uint32_t GPT_GetInputCaptureValue(GPT_Type *base, gpt_input_capture_channel_t
                                                channel)
```

Get GPT input capture value of certain channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT capture channel (see gpt_input_capture_channel_t typedef enumeration).

Returns

GPT input capture value.

```
static inline void GPT_SetOutputOperationMode(GPT_Type *base,
                                                gpt_output_compare_channel_t channel,
                                                gpt_output_operation_mode_t mode)
```

Set GPT operation mode of output compare channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).
- mode – GPT output operation mode (see gpt_output_operation_mode_t typedef enumeration).

```
static inline gpt_output_operation_mode_t GPT_GetOutputOperationMode(GPT_Type *base,
                                                                        gpt_output_compare_channel_t
                                                                        channel)
```

Get GPT operation mode of output compare channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see `gpt_output_compare_channel_t` typedef enumeration).

Returns

GPT output operation mode (see `gpt_output_operation_mode_t` typedef enumeration).

```
static inline void GPT_SetOutputCompareValue(GPT_Type *base, gpt_output_compare_channel_t channel, uint32_t value)
```

Set GPT output compare value of output compare channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see `gpt_output_compare_channel_t` typedef enumeration).
- value – GPT output compare value.

```
static inline uint32_t GPT_GetOutputCompareValue(GPT_Type *base, gpt_output_compare_channel_t channel)
```

Get GPT output compare value of output compare channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see `gpt_output_compare_channel_t` typedef enumeration).

Returns

GPT output compare value.

```
static inline void GPT_ForceOutput(GPT_Type *base, gpt_output_compare_channel_t channel)
```

Force GPT output action on output compare channel, ignoring comparator.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see `gpt_output_compare_channel_t` typedef enumeration).

```
static inline void GPT_EnableInterrupts(GPT_Type *base, uint32_t mask)
```

Enables the selected GPT interrupts.

Parameters

- base – GPT peripheral base address.
- mask – The interrupts to enable. This is a logical OR of members of the enumeration `gpt_interrupt_enable_t`

```
static inline void GPT_DisableInterrupts(GPT_Type *base, uint32_t mask)
```

Disables the selected GPT interrupts.

Parameters

- base – GPT peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration `gpt_interrupt_enable_t`

static inline uint32_t GPT_GetEnabledInterrupts(GPT_Type *base)

Gets the enabled GPT interrupts.

Parameters

- base – GPT peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration `gpt_interrupt_enable_t`

static inline uint32_t GPT_GetStatusFlags(GPT_Type *base, *gpt_status_flag_t* flags)

Get GPT status flags.

Parameters

- base – GPT peripheral base address.
- flags – GPT status flag mask (see `gpt_status_flag_t` for bit definition).

Returns

GPT status, each bit represents one status flag.

static inline void GPT_ClearStatusFlags(GPT_Type *base, *gpt_status_flag_t* flags)

Clears the GPT status flags.

Parameters

- base – GPT peripheral base address.
- flags – GPT status flag mask (see `gpt_status_flag_t` for bit definition).

FSL_GPT_DRIVER_VERSION

enum `_gpt_clock_source`

List of clock sources.

Note: Actual number of clock sources is SoC dependent

Values:

enumerator `kGPT_ClockSource_Off`

GPT Clock Source Off.

enumerator `kGPT_ClockSource_Periph`

GPT Clock Source from Peripheral Clock.

enumerator `kGPT_ClockSource_HighFreq`

GPT Clock Source from High Frequency Reference Clock.

enumerator `kGPT_ClockSource_Ext`

GPT Clock Source from external pin.

enumerator `kGPT_ClockSource_LowFreq`

GPT Clock Source from Low Frequency Reference Clock.

enumerator `kGPT_ClockSource_Osc`

GPT Clock Source from Crystal oscillator.

enum `_gpt_input_capture_channel`

List of input capture channel number.

Values:

enumerator kGPT_InputCapture_Channel1
GPT Input Capture Channel1.

enumerator kGPT_InputCapture_Channel2
GPT Input Capture Channel2.

enum _gpt_input_operation_mode
List of input capture operation mode.

Values:

enumerator kGPT_InputOperation_Disabled
Don't capture.

enumerator kGPT_InputOperation_RiseEdge
Capture on rising edge of input pin.

enumerator kGPT_InputOperation_FallEdge
Capture on falling edge of input pin.

enumerator kGPT_InputOperation_BothEdge
Capture on both edges of input pin.

enum _gpt_output_compare_channel
List of output compare channel number.

Values:

enumerator kGPT_OutputCompare_Channel1
Output Compare Channel1.

enumerator kGPT_OutputCompare_Channel2
Output Compare Channel2.

enumerator kGPT_OutputCompare_Channel3
Output Compare Channel3.

enum _gpt_output_operation_mode
List of output compare operation mode.

Values:

enumerator kGPT_OutputOperation_Disconnected
Don't change output pin.

enumerator kGPT_OutputOperation_Toggle
Toggle output pin.

enumerator kGPT_OutputOperation_Clear
Set output pin low.

enumerator kGPT_OutputOperation_Set
Set output pin high.

enumerator kGPT_OutputOperation_Activelow
Generate a active low pulse on output pin.

enum _gpt_interrupt_enable
List of GPT interrupts.

Values:

enumerator kGPT_OutputCompare1InterruptEnable
Output Compare Channel1 interrupt enable

enumerator kGPT_OutputCompare2InterruptEnable

Output Compare Channel2 interrupt enable

enumerator kGPT_OutputCompare3InterruptEnable

Output Compare Channel3 interrupt enable

enumerator kGPT_InputCapture1InterruptEnable

Input Capture Channel1 interrupt enable

enumerator kGPT_InputCapture2InterruptEnable

Input Capture Channel1 interrupt enable

enumerator kGPT_RollOverFlagInterruptEnable

Counter rolled over interrupt enable

enum _gpt_status_flag

Status flag.

Values:

enumerator kGPT_OutputCompare1Flag

Output compare channel 1 event.

enumerator kGPT_OutputCompare2Flag

Output compare channel 2 event.

enumerator kGPT_OutputCompare3Flag

Output compare channel 3 event.

enumerator kGPT_InputCapture1Flag

Input Capture channel 1 event.

enumerator kGPT_InputCapture2Flag

Input Capture channel 2 event.

enumerator kGPT_RollOverFlag

Counter reaches maximum value and rolled over to 0 event.

typedef enum _gpt_clock_source gpt_clock_source_t

List of clock sources.

Note: Actual number of clock sources is SoC dependent

typedef enum _gpt_input_capture_channel gpt_input_capture_channel_t

List of input capture channel number.

typedef enum _gpt_input_operation_mode gpt_input_operation_mode_t

List of input capture operation mode.

typedef enum _gpt_output_compare_channel gpt_output_compare_channel_t

List of output compare channel number.

typedef enum _gpt_output_operation_mode gpt_output_operation_mode_t

List of output compare operation mode.

typedef enum _gpt_interrupt_enable gpt_interrupt_enable_t

List of GPT interrupts.

typedef enum _gpt_status_flag gpt_status_flag_t

Status flag.

```
typedef struct _gpt_init_config gpt_config_t
```

Structure to configure the running mode.

```
struct _gpt_init_config
```

#include <fsl_gpt.h> Structure to configure the running mode.

Public Members

gpt_clock_source_t clockSource

clock source for GPT module.

uint32_t divider

clock divider (prescaler+1) from clock source to counter.

bool enableFreeRun

true: FreeRun mode, false: Restart mode.

bool enableRunInWait

GPT enabled in wait mode.

bool enableRunInStop

GPT enabled in stop mode.

bool enableRunInDoze

GPT enabled in doze mode.

bool enableRunInDbg

GPT enabled in debug mode.

bool enableMode

true: counter reset to 0 when enabled; false: counter retain its value when enabled.

2.58 IEE: Inline Encryption Engine

FSL_IEE_DRIVER_VERSION

IEE driver version. Version 2.1.1.

Current version: 2.1.1

Change log:

- Version 2.0.0
 - Initial version
- Version 2.1.0
 - Add region lock function IEE_LockRegionConfig() and driver clock control
- Version 2.1.1
 - Fixed MISRA issues.
- Version 2.2.0
 - Add ELE (EdgeLock Enclave) key provisioning feature.

enum *_iee_region*

IEE region.

Values:

enumerator kIEE_Region0

IEE region 0

enumerator kIEE_Region1

IEE region 1

enumerator kIEE_Region2

IEE region 2

enumerator kIEE_Region3

IEE region 3

enumerator kIEE_Region4

IEE region 4

enumerator kIEE_Region5

IEE region 5

enumerator kIEE_Region6

IEE region 6

enumerator kIEE_Region7

IEE region 7

enum _iee_aes_bypass

IEE AES enablement/bypass.

Values:

enumerator kIEE_AesUseMdField

AES encryption/decryption enabled

enumerator kIEE_AesBypass

AES encryption/decryption bypass

enum _iee_aes_mode

IEE AES mode.

Values:

enumerator kIEE_ModeNone

AES NONE mode

enumerator kIEE_ModeAesXTS

AES XTS mode

enumerator kIEE_ModeAesCTRWAddress

AES CTR w address binding mode

enumerator kIEE_ModeAesCTRWOAddress

AES CTR w/o address binding mode

enumerator kIEE_ModeAesCTRkeystream

AES CTR keystream only

enum _iee_aes_key_size

IEE AES key size.

Values:

enumerator kIEE_AesCTR128XTS256

AES 128 bits (CTR), 256 bits (XTS)

enumerator kIEE_AesCTR256XTS512
AES 256 bits (CTR), 512 bits (XTS)

enum _iee_aes_key_num
IEE AES key number.

Values:

enumerator kIEE_AesKey1
AES Key 1

enumerator kIEE_AesKey2
AES Key 2

typedef enum _iee_region iee_region_t
IEE region.

typedef enum _iee_aes_bypass iee_aes_bypass_t
IEE AES enablement/bypass.

typedef enum _iee_aes_mode iee_aes_mode_t
IEE AES mode.

typedef enum _iee_aes_key_size iee_aes_key_size_t
IEE AES key size.

typedef enum _iee_aes_key_num iee_aes_key_num_t
IEE AES key number.

typedef struct _iee_config iee_config_t
IEE configuration structure.

void IEE_Init(IEE_Type *base)
Resets IEE module to factory default values.

This function performs hardware reset of IEE module. Attributes and keys of all regions are cleared.

Parameters

- base – IEER peripheral address.

void IEE_GetDefaultConfig(iee_config_t *config)
Loads default values to the IEE configuration structure.

Loads default values to the IEE region configuration structure. The default values are as follows.

```
config->bypass = kIEE_AesUseMdField;  
config->mode = kIEE_ModeNone;  
config->keySize = kIEE_AesCTR128XTS256;  
config->pageOffset = 0U;
```

Parameters

- config – Configuration for the selected IEE region.

void IEE_SetRegionConfig(IEE_Type *base, iee_region_t region, iee_config_t *config)
Sets the IEE module according to the configuration structure.

This function configures IEE region according to configuration structure.

Parameters

- base – IEE peripheral address.

- `region` – Selection of the IEE region to be configured.
- `config` – Configuration for the selected IEE region.

```
status_t IEE_SetRegionKey(IEE_Type *base, iee_region_t region, iee_aes_key_num_t keyNum,
                          const uint8_t *key, size_t keySize)
```

Sets the IEE module key.

This function sets specified AES key for the given region.

Parameters

- `base` – IEE peripheral address.
- `region` – Selection of the IEE region to be configured.
- `keyNum` – Selection of AES KEY1 or KEY2.
- `key` – AES key.
- `keySize` – Size of AES key.

```
static inline uint32_t IEE_GetOffset(uint32_t addressIee, uint32_t addressMemory)
```

Computes IEE offset to be set for specified memory location.

This function calculates offset that must be set for IEE region to access physical memory location.

Parameters

- `addressIee` – Address of IEE peripheral.
- `addressMemory` – Address of physical memory location.

```
void IEE_LockRegionConfig(IEE_Type *base, iee_region_t region)
```

Lock the IEE region configuration.

This function locks IEE region registers for Key, Offset and Attribute. Only system reset can clear the Lock bit.

Parameters

- `base` – IEE peripheral address.
- `region` – Selection of the IEE region to be locked.

```
struct _iee_config
```

`#include <fsl_iee.h>` IEE configuration structure.

Public Members

iee_aes_bypass_t `bypass`

AES encryption/decryption bypass

iee_aes_mode_t `mode`

AES mode

iee_aes_key_size_t `keySize`

size of AES key

uint32_t `pageOffset`

Offset to physical memory location from IEE start address

2.59 Ieer

FSL_IEE_APC_DRIVER_VERSION

IEE_APC driver version. Version 2.0.2.

Current version: 2.0.2

Change log:

- Version 2.0.0
 - Initial version
- Version 2.0.1
 - Fixed MISRA issues.
- Version 2.0.2
 - Update to newer version of implementation in HW.

enum _iee_apc_region

APC IEE regions.

Values:

enumerator kIEE_APC_Region0

APC IEE region 0

enumerator kIEE_APC_Region1

APC IEE region 1

enumerator kIEE_APC_Region2

APC IEE region 2

enumerator kIEE_APC_Region3

APC IEE region 3

enumerator kIEE_APC_Region4

APC IEE region 4

enumerator kIEE_APC_Region5

APC IEE region 5

enumerator kIEE_APC_Region6

APC IEE region 6

enumerator kIEE_APC_Region7

APC IEE region 7

enum _apc_iee_domain

APC IEE domains.

Values:

enumerator kIEE_APC_Domain0

APC IEE region 0

enumerator kIEE_APC_Domain1

APC IEE region 1

typedef enum _iee_apc_region iee_apc_region_t

APC IEE regions.

typedef enum _apc_iee_domain iee_apc_domain_t

APC IEE domains.

`void IEE__APC_GlobalEnable(IEE__APC_Type *base)`

Enable the APC IEE Region setting.

This function enables IOMUXC LPSR GPR and APC IEE for setting the region.

Parameters

- `base` – APC IEE peripheral address.

`void IEE__APC_GlobalDisable(IEE__APC_Type *base)`

Disables the APC IEE Region setting.

This function disables IOMUXC LPSR GPR and APC IEE for setting the region.

Parameters

- `base` – APC IEE peripheral address.

`status_t IEE__APC_SetRegionConfig(IEE__APC_Type *base, iee_apc_region_t region, uint32_t startAddr, uint32_t endAddr)`

Sets the APC IEE Memory Region Descriptors.

This function configures APC IEE Memory Region Descriptor according to region configuration structure.

Parameters

- `base` – APC IEE peripheral address.
- `region` – Selection of the APC IEE region to be configured.
- `startAddr` – Start encryption address for the selected APC IEE region.
- `endAddr` – End encryption address for the selected APC IEE region.

`status_t IEE__APC_LockRegionConfig(IEE__APC_Type *base, iee_apc_region_t region, iee_apc_domain_t domain)`

Lock the LPSR GPR and APC IEE configuration.

This function locks writing to IOMUXC LPSR GPR and APC IEE encryption region setting registers. Only system reset can clear the LPSR GPR and APC IEE-RDC_D0/1 Lock bit

Parameters

- `base` – APC IEE peripheral address.
- `region` – Selection of the APC IEE region to be locked.
- `domain` – Core domain ID

`void IEE__APC_RegionEnable(IEE__APC_Type *base, iee_apc_region_t region)`

Enable the IEE encryption/decryption and can lock this setting.

This function enables encryption/decryption by writing to IOMUXC LPSR GPR.

Parameters

- `base` – APC IEE peripheral address.
- `region` – Selection of the APC IEE region to be enabled.

2.60 IOMUXC: IOMUX Controller

`enum __iomuxc_gpr_saimclk`

Values:

```
enumerator kIOMUXC_GPR_SAI1MClk1Sel
enumerator kIOMUXC_GPR_SAI1MClk2Sel
enumerator kIOMUXC_GPR_SAI1MClk3Sel
enumerator kIOMUXC_GPR_SAI2MClk3Sel
enumerator kIOMUXC_GPR_SAI3MClk3Sel

enum _iomuxc_mqs_pwm_oversample_rate
    Values:
    enumerator kIOMUXC_MqsPwmOverSampleRate32
    enumerator kIOMUXC_MqsPwmOverSampleRate64

typedef enum _iomuxc_gpr_saimclk iomuxc_gpr_saimclk_t
typedef enum _iomuxc_mqs_pwm_oversample_rate iomuxc_mqs_pwm_oversample_rate_t

IOMUXC_GPIO_LPSR_00_FLEXCAN3_TX
IOMUXC_GPIO_LPSR_00_MIC_CLK
IOMUXC_GPIO_LPSR_00_MQS_RIGHT
IOMUXC_GPIO_LPSR_00_ARM_CM4_EVENT0
IOMUXC_GPIO_LPSR_00_GPIO_MUX6_IO00
IOMUXC_GPIO_LPSR_00_LPUART12_TXD
IOMUXC_GPIO_LPSR_00_SAI4_MCLK
IOMUXC_GPIO_LPSR_00_GPIO12_IO00
IOMUXC_GPIO_LPSR_01_FLEXCAN3_RX
IOMUXC_GPIO_LPSR_01_MIC_BITSTREAM0
IOMUXC_GPIO_LPSR_01_MQS_LEFT
IOMUXC_GPIO_LPSR_01_ARM_CM4_EVENT1
IOMUXC_GPIO_LPSR_01_GPIO_MUX6_IO01
IOMUXC_GPIO_LPSR_01_LPUART12_RXD
IOMUXC_GPIO_LPSR_01_GPIO12_IO01
IOMUXC_GPIO_LPSR_02_GPIO12_IO02
IOMUXC_GPIO_LPSR_02_SRC_BOOT_MODE00
IOMUXC_GPIO_LPSR_02_LPSPi5_SCK
IOMUXC_GPIO_LPSR_02_SAI4_TX_DATA
IOMUXC_GPIO_LPSR_02_MQS_RIGHT
IOMUXC_GPIO_LPSR_02_GPIO_MUX6_IO02
IOMUXC_GPIO_LPSR_03_SRC_BOOT_MODE01
```

IOMUXC_GPIO_LPSR_03_LPSPi5_PCS0
IOMUXC_GPIO_LPSR_03_SAI4_TX_SYNC
IOMUXC_GPIO_LPSR_03_MQS_LEFT
IOMUXC_GPIO_LPSR_03_GPIO_MUX6_IO03
IOMUXC_GPIO_LPSR_03_GPIO12_IO03
IOMUXC_GPIO_LPSR_04_LPI2C5_SDA
IOMUXC_GPIO_LPSR_04_LPSPi5_SOUT
IOMUXC_GPIO_LPSR_04_SAI4_TX_BCLK
IOMUXC_GPIO_LPSR_04_LPUART12_RTS_B
IOMUXC_GPIO_LPSR_04_GPIO_MUX6_IO04
IOMUXC_GPIO_LPSR_04_LPUART11_TXD
IOMUXC_GPIO_LPSR_04_GPIO12_IO04
IOMUXC_GPIO_LPSR_05_GPIO12_IO05
IOMUXC_GPIO_LPSR_05_LPI2C5_SCL
IOMUXC_GPIO_LPSR_05_LPSPi5_SIN
IOMUXC_GPIO_LPSR_05_SAI4_MCLK
IOMUXC_GPIO_LPSR_05_LPUART12_CTS_B
IOMUXC_GPIO_LPSR_05_GPIO_MUX6_IO05
IOMUXC_GPIO_LPSR_05_LPUART11_RXD
IOMUXC_GPIO_LPSR_05_NMI_GLUE_NMI
IOMUXC_GPIO_LPSR_06_LPI2C6_SDA
IOMUXC_GPIO_LPSR_06_SAI4_RX_DATA
IOMUXC_GPIO_LPSR_06_LPUART12_TXD
IOMUXC_GPIO_LPSR_06_LPSPi6_PCS3
IOMUXC_GPIO_LPSR_06_GPIO_MUX6_IO06
IOMUXC_GPIO_LPSR_06_FLEXCAN3_TX
IOMUXC_GPIO_LPSR_06_PIT2_TRIGGER3
IOMUXC_GPIO_LPSR_06_LPSPi5_PCS1
IOMUXC_GPIO_LPSR_06_GPIO12_IO06
IOMUXC_GPIO_LPSR_07_LPI2C6_SCL
IOMUXC_GPIO_LPSR_07_SAI4_RX_BCLK
IOMUXC_GPIO_LPSR_07_LPUART12_RXD
IOMUXC_GPIO_LPSR_07_LPSPi6_PCS2

IOMUXC_GPIO_LPSR_07_GPIO_MUX6_IO07
IOMUXC_GPIO_LPSR_07_FLEXCAN3_RX
IOMUXC_GPIO_LPSR_07_PIT2_TRIGGER2
IOMUXC_GPIO_LPSR_07_LPSPi5_PCS2
IOMUXC_GPIO_LPSR_07_GPIO12_IO07
IOMUXC_GPIO_LPSR_08_GPIO12_IO08
IOMUXC_GPIO_LPSR_08_LPUART11_TXD
IOMUXC_GPIO_LPSR_08_FLEXCAN3_TX
IOMUXC_GPIO_LPSR_08_SAI4_RX_SYNC
IOMUXC_GPIO_LPSR_08_MIC_CLK
IOMUXC_GPIO_LPSR_08_LPSPi6_PCS1
IOMUXC_GPIO_LPSR_08_GPIO_MUX6_IO08
IOMUXC_GPIO_LPSR_08_LPI2C5_SDA
IOMUXC_GPIO_LPSR_08_PIT2_TRIGGER1
IOMUXC_GPIO_LPSR_08_LPSPi5_PCS3
IOMUXC_GPIO_LPSR_09_GPIO12_IO09
IOMUXC_GPIO_LPSR_09_LPUART11_RXD
IOMUXC_GPIO_LPSR_09_FLEXCAN3_RX
IOMUXC_GPIO_LPSR_09_PIT2_TRIGGER0
IOMUXC_GPIO_LPSR_09_MIC_BITSTREAM0
IOMUXC_GPIO_LPSR_09_LPSPi6_PCS0
IOMUXC_GPIO_LPSR_09_GPIO_MUX6_IO09
IOMUXC_GPIO_LPSR_09_LPI2C5_SCL
IOMUXC_GPIO_LPSR_09_SAI4_TX_DATA
IOMUXC_GPIO_LPSR_10_GPIO12_IO10
IOMUXC_GPIO_LPSR_10_JTAG_MUX_TRSTB
IOMUXC_GPIO_LPSR_10_LPUART11_CTS_B
IOMUXC_GPIO_LPSR_10_LPI2C6_SDA
IOMUXC_GPIO_LPSR_10_MIC_BITSTREAM1
IOMUXC_GPIO_LPSR_10_LPSPi6_SCK
IOMUXC_GPIO_LPSR_10_GPIO_MUX6_IO10
IOMUXC_GPIO_LPSR_10_LPI2C5_SCLS
IOMUXC_GPIO_LPSR_10_SAI4_TX_SYNC

IOMUXC_GPIO_LPSR_10_LPUART12_TXD
IOMUXC_GPIO_LPSR_11_JTAG_MUX_TDO
IOMUXC_GPIO_LPSR_11_LPUART11_RTS_B
IOMUXC_GPIO_LPSR_11_LPI2C6_SCL
IOMUXC_GPIO_LPSR_11_MIC_BITSTREAM2
IOMUXC_GPIO_LPSR_11_LPSPi6_SOUT
IOMUXC_GPIO_LPSR_11_GPIO_MUX6_IO11
IOMUXC_GPIO_LPSR_11_LPI2C5_SDAS
IOMUXC_GPIO_LPSR_11_ARM_TRACE_SWO
IOMUXC_GPIO_LPSR_11_LPUART12_RXD
IOMUXC_GPIO_LPSR_11_GPIO12_IO11
IOMUXC_GPIO_LPSR_12_GPIO12_IO12
IOMUXC_GPIO_LPSR_12_JTAG_MUX_TDI
IOMUXC_GPIO_LPSR_12_PIT2_TRIGGER0
IOMUXC_GPIO_LPSR_12_MIC_BITSTREAM3
IOMUXC_GPIO_LPSR_12_LPSPi6_SIN
IOMUXC_GPIO_LPSR_12_GPIO_MUX6_IO12
IOMUXC_GPIO_LPSR_12_LPI2C5_HREQ
IOMUXC_GPIO_LPSR_12_SAI4_TX_BCLK
IOMUXC_GPIO_LPSR_12_LPSPi5_SCK
IOMUXC_GPIO_LPSR_13_GPIO12_IO13
IOMUXC_GPIO_LPSR_13_JTAG_MUX_MOD
IOMUXC_GPIO_LPSR_13_MIC_BITSTREAM1
IOMUXC_GPIO_LPSR_13_PIT2_TRIGGER1
IOMUXC_GPIO_LPSR_13_GPIO_MUX6_IO13
IOMUXC_GPIO_LPSR_13_SAI4_RX_DATA
IOMUXC_GPIO_LPSR_13_LPSPi5_PCS0
IOMUXC_GPIO_LPSR_14_JTAG_MUX_TCK
IOMUXC_GPIO_LPSR_14_MIC_BITSTREAM2
IOMUXC_GPIO_LPSR_14_PIT2_TRIGGER2
IOMUXC_GPIO_LPSR_14_GPIO_MUX6_IO14
IOMUXC_GPIO_LPSR_14_SAI4_RX_BCLK
IOMUXC_GPIO_LPSR_14_LPSPi5_SOUT

IOMUXC_GPIO_LPSR_14_GPIO12_IO14
IOMUXC_GPIO_LPSR_15_GPIO12_IO15
IOMUXC_GPIO_LPSR_15_JTAG_MUX_TMS
IOMUXC_GPIO_LPSR_15_MIC_BITSTREAM3
IOMUXC_GPIO_LPSR_15_PIT2_TRIGGER3
IOMUXC_GPIO_LPSR_15_GPIO_MUX6_IO15
IOMUXC_GPIO_LPSR_15_SAI4_RX_SYNC
IOMUXC_GPIO_LPSR_15_LPSPI5_SIN
IOMUXC_WAKEUP_DIG_GPIO13_IO00
IOMUXC_WAKEUP_DIG_NMI_GLUE_NMI
IOMUXC_PMIC_ON_REQ_DIG_SNVS_LP_PMIC_ON_REQ
IOMUXC_PMIC_ON_REQ_DIG_GPIO13_IO01
IOMUXC_PMIC_STBY_REQ_DIG_CCM_PMIC_VSTBY_REQ
IOMUXC_PMIC_STBY_REQ_DIG_GPIO13_IO02
IOMUXC_GPIO_SNVS_00_DIG_SNVS_TAMPER0
IOMUXC_GPIO_SNVS_00_DIG_GPIO13_IO03
IOMUXC_GPIO_SNVS_01_DIG_SNVS_TAMPER1
IOMUXC_GPIO_SNVS_01_DIG_GPIO13_IO04
IOMUXC_GPIO_SNVS_02_DIG_SNVS_TAMPER2
IOMUXC_GPIO_SNVS_02_DIG_GPIO13_IO05
IOMUXC_GPIO_SNVS_03_DIG_SNVS_TAMPER3
IOMUXC_GPIO_SNVS_03_DIG_GPIO13_IO06
IOMUXC_GPIO_SNVS_04_DIG_SNVS_TAMPER4
IOMUXC_GPIO_SNVS_04_DIG_GPIO13_IO07
IOMUXC_GPIO_SNVS_05_DIG_SNVS_TAMPER5
IOMUXC_GPIO_SNVS_05_DIG_GPIO13_IO08
IOMUXC_GPIO_SNVS_06_DIG_SNVS_TAMPER6
IOMUXC_GPIO_SNVS_06_DIG_GPIO13_IO09
IOMUXC_GPIO_SNVS_07_DIG_SNVS_TAMPER7
IOMUXC_GPIO_SNVS_07_DIG_GPIO13_IO10
IOMUXC_GPIO_SNVS_08_DIG_SNVS_TAMPER8
IOMUXC_GPIO_SNVS_08_DIG_GPIO13_IO11
IOMUXC_GPIO_SNVS_09_DIG_SNVS_TAMPER9

IOMUXC_GPIO_SNVS_09_DIG_GPIO13_IO12
IOMUXC_TEST_MODE_DIG
IOMUXC_POR_B_DIG
IOMUXC_ONOFF_DIG
IOMUXC_GPIO_EMC_B1_00_SEMC_DATA00
IOMUXC_GPIO_EMC_B1_00_FLEXPWM4_PWM0_A
IOMUXC_GPIO_EMC_B1_00_GPIO_MUX1_IO00
IOMUXC_GPIO_EMC_B1_00_FLEXIO1_D00
IOMUXC_GPIO_EMC_B1_00_GPIO7_IO00
IOMUXC_GPIO_EMC_B1_01_GPIO7_IO01
IOMUXC_GPIO_EMC_B1_01_SEMC_DATA01
IOMUXC_GPIO_EMC_B1_01_FLEXPWM4_PWM0_B
IOMUXC_GPIO_EMC_B1_01_GPIO_MUX1_IO01
IOMUXC_GPIO_EMC_B1_01_FLEXIO1_D01
IOMUXC_GPIO_EMC_B1_02_SEMC_DATA02
IOMUXC_GPIO_EMC_B1_02_FLEXPWM4_PWM1_A
IOMUXC_GPIO_EMC_B1_02_GPIO_MUX1_IO02
IOMUXC_GPIO_EMC_B1_02_FLEXIO1_D02
IOMUXC_GPIO_EMC_B1_02_GPIO7_IO02
IOMUXC_GPIO_EMC_B1_03_SEMC_DATA03
IOMUXC_GPIO_EMC_B1_03_FLEXPWM4_PWM1_B
IOMUXC_GPIO_EMC_B1_03_GPIO_MUX1_IO03
IOMUXC_GPIO_EMC_B1_03_FLEXIO1_D03
IOMUXC_GPIO_EMC_B1_03_GPIO7_IO03
IOMUXC_GPIO_EMC_B1_04_GPIO7_IO04
IOMUXC_GPIO_EMC_B1_04_SEMC_DATA04
IOMUXC_GPIO_EMC_B1_04_FLEXPWM4_PWM2_A
IOMUXC_GPIO_EMC_B1_04_GPIO_MUX1_IO04
IOMUXC_GPIO_EMC_B1_04_FLEXIO1_D04
IOMUXC_GPIO_EMC_B1_05_SEMC_DATA05
IOMUXC_GPIO_EMC_B1_05_FLEXPWM4_PWM2_B
IOMUXC_GPIO_EMC_B1_05_GPIO_MUX1_IO05
IOMUXC_GPIO_EMC_B1_05_FLEXIO1_D05

IOMUXC_GPIO_EMC_B1_05_GPIO7_IO05
IOMUXC_GPIO_EMC_B1_06_SEMC_DATA06
IOMUXC_GPIO_EMC_B1_06_FLEXPWM2_PWM0_A
IOMUXC_GPIO_EMC_B1_06_GPIO_MUX1_IO06
IOMUXC_GPIO_EMC_B1_06_FLEXIO1_D06
IOMUXC_GPIO_EMC_B1_06_GPIO7_IO06
IOMUXC_GPIO_EMC_B1_07_GPIO7_IO07
IOMUXC_GPIO_EMC_B1_07_SEMC_DATA07
IOMUXC_GPIO_EMC_B1_07_FLEXPWM2_PWM0_B
IOMUXC_GPIO_EMC_B1_07_GPIO_MUX1_IO07
IOMUXC_GPIO_EMC_B1_07_FLEXIO1_D07
IOMUXC_GPIO_EMC_B1_08_SEMC_DM00
IOMUXC_GPIO_EMC_B1_08_FLEXPWM2_PWM1_A
IOMUXC_GPIO_EMC_B1_08_GPIO_MUX1_IO08
IOMUXC_GPIO_EMC_B1_08_FLEXIO1_D08
IOMUXC_GPIO_EMC_B1_08_GPIO7_IO08
IOMUXC_GPIO_EMC_B1_09_SEMC_ADDR00
IOMUXC_GPIO_EMC_B1_09_FLEXPWM2_PWM1_B
IOMUXC_GPIO_EMC_B1_09_GPT5_CAPTURE1
IOMUXC_GPIO_EMC_B1_09_GPIO_MUX1_IO09
IOMUXC_GPIO_EMC_B1_09_FLEXIO1_D09
IOMUXC_GPIO_EMC_B1_09_GPIO7_IO09
IOMUXC_GPIO_EMC_B1_10_SEMC_ADDR01
IOMUXC_GPIO_EMC_B1_10_FLEXPWM2_PWM2_A
IOMUXC_GPIO_EMC_B1_10_GPT5_CAPTURE2
IOMUXC_GPIO_EMC_B1_10_GPIO_MUX1_IO10
IOMUXC_GPIO_EMC_B1_10_FLEXIO1_D10
IOMUXC_GPIO_EMC_B1_10_GPIO7_IO10
IOMUXC_GPIO_EMC_B1_11_GPIO7_IO11
IOMUXC_GPIO_EMC_B1_11_SEMC_ADDR02
IOMUXC_GPIO_EMC_B1_11_FLEXPWM2_PWM2_B
IOMUXC_GPIO_EMC_B1_11_GPT5_COMPARE1
IOMUXC_GPIO_EMC_B1_11_GPIO_MUX1_IO11

IOMUXC_GPIO_EMC_B1_11_FLEXIO1_D11
IOMUXC_GPIO_EMC_B1_12_SEMC_ADDR03
IOMUXC_GPIO_EMC_B1_12_XBAR1_INOUT04
IOMUXC_GPIO_EMC_B1_12_GPT5_COMPARE2
IOMUXC_GPIO_EMC_B1_12_GPIO_MUX1_IO12
IOMUXC_GPIO_EMC_B1_12_FLEXIO1_D12
IOMUXC_GPIO_EMC_B1_12_GPIO7_IO12
IOMUXC_GPIO_EMC_B1_13_SEMC_ADDR04
IOMUXC_GPIO_EMC_B1_13_XBAR1_INOUT05
IOMUXC_GPIO_EMC_B1_13_GPT5_COMPARE3
IOMUXC_GPIO_EMC_B1_13_GPIO_MUX1_IO13
IOMUXC_GPIO_EMC_B1_13_FLEXIO1_D13
IOMUXC_GPIO_EMC_B1_13_GPIO7_IO13
IOMUXC_GPIO_EMC_B1_14_GPIO7_IO14
IOMUXC_GPIO_EMC_B1_14_SEMC_ADDR05
IOMUXC_GPIO_EMC_B1_14_XBAR1_INOUT06
IOMUXC_GPIO_EMC_B1_14_GPT5_CLK
IOMUXC_GPIO_EMC_B1_14_GPIO_MUX1_IO14
IOMUXC_GPIO_EMC_B1_14_FLEXIO1_D14
IOMUXC_GPIO_EMC_B1_15_SEMC_ADDR06
IOMUXC_GPIO_EMC_B1_15_XBAR1_INOUT07
IOMUXC_GPIO_EMC_B1_15_GPIO_MUX1_IO15
IOMUXC_GPIO_EMC_B1_15_FLEXIO1_D15
IOMUXC_GPIO_EMC_B1_15_GPIO7_IO15
IOMUXC_GPIO_EMC_B1_16_SEMC_ADDR07
IOMUXC_GPIO_EMC_B1_16_XBAR1_INOUT08
IOMUXC_GPIO_EMC_B1_16_GPIO_MUX1_IO16
IOMUXC_GPIO_EMC_B1_16_FLEXIO1_D16
IOMUXC_GPIO_EMC_B1_16_GPIO7_IO16
IOMUXC_GPIO_EMC_B1_17_GPIO7_IO17
IOMUXC_GPIO_EMC_B1_17_SEMC_ADDR08
IOMUXC_GPIO_EMC_B1_17_FLEXPWM4_PWM3_A
IOMUXC_GPIO_EMC_B1_17_TMR1_TIMER0

IOMUXC_GPIO_EMC_B1_17_GPIO_MUX1_IO17
IOMUXC_GPIO_EMC_B1_17_FLEXIO1_D17
IOMUXC_GPIO_EMC_B1_18_SEMC_ADDR09
IOMUXC_GPIO_EMC_B1_18_FLEXPWM4_PWM3_B
IOMUXC_GPIO_EMC_B1_18_TMR2_TIMER0
IOMUXC_GPIO_EMC_B1_18_GPIO_MUX1_IO18
IOMUXC_GPIO_EMC_B1_18_FLEXIO1_D18
IOMUXC_GPIO_EMC_B1_18_GPIO7_IO18
IOMUXC_GPIO_EMC_B1_19_SEMC_ADDR11
IOMUXC_GPIO_EMC_B1_19_FLEXPWM2_PWM3_A
IOMUXC_GPIO_EMC_B1_19_TMR3_TIMER0
IOMUXC_GPIO_EMC_B1_19_GPIO_MUX1_IO19
IOMUXC_GPIO_EMC_B1_19_FLEXIO1_D19
IOMUXC_GPIO_EMC_B1_19_GPIO7_IO19
IOMUXC_GPIO_EMC_B1_20_SEMC_ADDR12
IOMUXC_GPIO_EMC_B1_20_FLEXPWM2_PWM3_B
IOMUXC_GPIO_EMC_B1_20_TMR4_TIMER0
IOMUXC_GPIO_EMC_B1_20_GPIO_MUX1_IO20
IOMUXC_GPIO_EMC_B1_20_FLEXIO1_D20
IOMUXC_GPIO_EMC_B1_20_GPIO7_IO20
IOMUXC_GPIO_EMC_B1_21_GPIO7_IO21
IOMUXC_GPIO_EMC_B1_21_SEMC_BA0
IOMUXC_GPIO_EMC_B1_21_FLEXPWM3_PWM3_A
IOMUXC_GPIO_EMC_B1_21_GPIO_MUX1_IO21
IOMUXC_GPIO_EMC_B1_21_FLEXIO1_D21
IOMUXC_GPIO_EMC_B1_22_GPIO7_IO22
IOMUXC_GPIO_EMC_B1_22_SEMC_BA1
IOMUXC_GPIO_EMC_B1_22_FLEXPWM3_PWM3_B
IOMUXC_GPIO_EMC_B1_22_GPIO_MUX1_IO22
IOMUXC_GPIO_EMC_B1_22_FLEXIO1_D22
IOMUXC_GPIO_EMC_B1_23_SEMC_ADDR10
IOMUXC_GPIO_EMC_B1_23_FLEXPWM1_PWM0_A
IOMUXC_GPIO_EMC_B1_23_GPIO_MUX1_IO23

IOMUXC_GPIO_EMC_B1_23_FLEXIO1_D23
IOMUXC_GPIO_EMC_B1_23_GPIO7_IO23
IOMUXC_GPIO_EMC_B1_24_GPIO7_IO24
IOMUXC_GPIO_EMC_B1_24_SEMC_CAS
IOMUXC_GPIO_EMC_B1_24_FLEXPWM1_PWM0_B
IOMUXC_GPIO_EMC_B1_24_GPIO_MUX1_IO24
IOMUXC_GPIO_EMC_B1_24_FLEXIO1_D24
IOMUXC_GPIO_EMC_B1_25_GPIO7_IO25
IOMUXC_GPIO_EMC_B1_25_SEMC_RAS
IOMUXC_GPIO_EMC_B1_25_FLEXPWM1_PWM1_A
IOMUXC_GPIO_EMC_B1_25_GPIO_MUX1_IO25
IOMUXC_GPIO_EMC_B1_25_FLEXIO1_D25
IOMUXC_GPIO_EMC_B1_26_SEMC_CLK
IOMUXC_GPIO_EMC_B1_26_FLEXPWM1_PWM1_B
IOMUXC_GPIO_EMC_B1_26_GPIO_MUX1_IO26
IOMUXC_GPIO_EMC_B1_26_FLEXIO1_D26
IOMUXC_GPIO_EMC_B1_26_GPIO7_IO26
IOMUXC_GPIO_EMC_B1_27_GPIO7_IO27
IOMUXC_GPIO_EMC_B1_27_SEMC_CKE
IOMUXC_GPIO_EMC_B1_27_FLEXPWM1_PWM2_A
IOMUXC_GPIO_EMC_B1_27_GPIO_MUX1_IO27
IOMUXC_GPIO_EMC_B1_27_FLEXIO1_D27
IOMUXC_GPIO_EMC_B1_28_GPIO7_IO28
IOMUXC_GPIO_EMC_B1_28_SEMC_WE
IOMUXC_GPIO_EMC_B1_28_FLEXPWM1_PWM2_B
IOMUXC_GPIO_EMC_B1_28_GPIO_MUX1_IO28
IOMUXC_GPIO_EMC_B1_28_FLEXIO1_D28
IOMUXC_GPIO_EMC_B1_29_SEMC_CS0
IOMUXC_GPIO_EMC_B1_29_FLEXPWM3_PWM0_A
IOMUXC_GPIO_EMC_B1_29_GPIO_MUX1_IO29
IOMUXC_GPIO_EMC_B1_29_FLEXIO1_D29
IOMUXC_GPIO_EMC_B1_29_GPIO7_IO29
IOMUXC_GPIO_EMC_B1_30_SEMC_DATA08

IOMUXC_GPIO_EMC_B1_30_FLEXPWM3_PWM0_B
IOMUXC_GPIO_EMC_B1_30_GPIO_MUX1_IO30
IOMUXC_GPIO_EMC_B1_30_FLEXIO1_D30
IOMUXC_GPIO_EMC_B1_30_GPIO7_IO30
IOMUXC_GPIO_EMC_B1_31_GPIO7_IO31
IOMUXC_GPIO_EMC_B1_31_SEMC_DATA09
IOMUXC_GPIO_EMC_B1_31_FLEXPWM3_PWM1_A
IOMUXC_GPIO_EMC_B1_31_GPIO_MUX1_IO31
IOMUXC_GPIO_EMC_B1_31_FLEXIO1_D31
IOMUXC_GPIO_EMC_B1_32_GPIO8_IO00
IOMUXC_GPIO_EMC_B1_32_SEMC_DATA10
IOMUXC_GPIO_EMC_B1_32_FLEXPWM3_PWM1_B
IOMUXC_GPIO_EMC_B1_32_GPIO_MUX2_IO00
IOMUXC_GPIO_EMC_B1_33_SEMC_DATA11
IOMUXC_GPIO_EMC_B1_33_FLEXPWM3_PWM2_A
IOMUXC_GPIO_EMC_B1_33_GPIO_MUX2_IO01
IOMUXC_GPIO_EMC_B1_33_GPIO8_IO01
IOMUXC_GPIO_EMC_B1_34_GPIO8_IO02
IOMUXC_GPIO_EMC_B1_34_SEMC_DATA12
IOMUXC_GPIO_EMC_B1_34_FLEXPWM3_PWM2_B
IOMUXC_GPIO_EMC_B1_34_GPIO_MUX2_IO02
IOMUXC_GPIO_EMC_B1_35_GPIO8_IO03
IOMUXC_GPIO_EMC_B1_35_SEMC_DATA13
IOMUXC_GPIO_EMC_B1_35_XBAR1_INOUT09
IOMUXC_GPIO_EMC_B1_35_GPIO_MUX2_IO03
IOMUXC_GPIO_EMC_B1_36_SEMC_DATA14
IOMUXC_GPIO_EMC_B1_36_XBAR1_INOUT10
IOMUXC_GPIO_EMC_B1_36_GPIO_MUX2_IO04
IOMUXC_GPIO_EMC_B1_36_GPIO8_IO04
IOMUXC_GPIO_EMC_B1_37_GPIO8_IO05
IOMUXC_GPIO_EMC_B1_37_SEMC_DATA15
IOMUXC_GPIO_EMC_B1_37_XBAR1_INOUT11
IOMUXC_GPIO_EMC_B1_37_GPIO_MUX2_IO05

IOMUXC_GPIO_EMC_B1_38_GPIO8_IO06
IOMUXC_GPIO_EMC_B1_38_SEMC_DM01
IOMUXC_GPIO_EMC_B1_38_FLEXPWM1_PWM3_A
IOMUXC_GPIO_EMC_B1_38_TMR1_TIMER1
IOMUXC_GPIO_EMC_B1_38_GPIO_MUX2_IO06
IOMUXC_GPIO_EMC_B1_39_SEMC_DQS
IOMUXC_GPIO_EMC_B1_39_FLEXPWM1_PWM3_B
IOMUXC_GPIO_EMC_B1_39_TMR2_TIMER1
IOMUXC_GPIO_EMC_B1_39_GPIO_MUX2_IO07
IOMUXC_GPIO_EMC_B1_39_GPIO8_IO07
IOMUXC_GPIO_EMC_B1_40_SEMC_RDY
IOMUXC_GPIO_EMC_B1_40_XBAR1_INOUT12
IOMUXC_GPIO_EMC_B1_40_MQS_RIGHT
IOMUXC_GPIO_EMC_B1_40_LPUART6_TXD
IOMUXC_GPIO_EMC_B1_40_GPIO_MUX2_IO08
IOMUXC_GPIO_EMC_B1_40_ENET_1G_MDC
IOMUXC_GPIO_EMC_B1_40_CCM_CLKO1
IOMUXC_GPIO_EMC_B1_40_GPIO8_IO08
IOMUXC_GPIO_EMC_B1_41_GPIO8_IO09
IOMUXC_GPIO_EMC_B1_41_SEMC_CSX00
IOMUXC_GPIO_EMC_B1_41_XBAR1_INOUT13
IOMUXC_GPIO_EMC_B1_41_MQS_LEFT
IOMUXC_GPIO_EMC_B1_41_LPUART6_RXD
IOMUXC_GPIO_EMC_B1_41_FLEXSPI2_B_DATA07
IOMUXC_GPIO_EMC_B1_41_GPIO_MUX2_IO09
IOMUXC_GPIO_EMC_B1_41_ENET_1G_MDIO
IOMUXC_GPIO_EMC_B1_41_CCM_CLKO2
IOMUXC_GPIO_EMC_B2_00_SEMC_DATA16
IOMUXC_GPIO_EMC_B2_00_CCM_ENET_REF_CLK_25M
IOMUXC_GPIO_EMC_B2_00_TMR3_TIMER1
IOMUXC_GPIO_EMC_B2_00_LPUART6_CTS_B
IOMUXC_GPIO_EMC_B2_00_FLEXSPI2_B_DATA06
IOMUXC_GPIO_EMC_B2_00_GPIO_MUX2_IO10

IOMUXC_GPIO_EMC_B2_00_XBAR1_INOUT20
IOMUXC_GPIO_EMC_B2_00_ENET_QOS_1588_EVENT1_OUT
IOMUXC_GPIO_EMC_B2_00_LPSP11_SCK
IOMUXC_GPIO_EMC_B2_00_LPI2C2_SCL
IOMUXC_GPIO_EMC_B2_00_GPIO8_IO10
IOMUXC_GPIO_EMC_B2_00_FLEXPWM3_PWM0_A
IOMUXC_GPIO_EMC_B2_01_SEMC_DATA17
IOMUXC_GPIO_EMC_B2_01_USDHC2_CD_B
IOMUXC_GPIO_EMC_B2_01_TMR4_TIMER1
IOMUXC_GPIO_EMC_B2_01_LPUART6_RTS_B
IOMUXC_GPIO_EMC_B2_01_FLEXSPI2_B_DATA05
IOMUXC_GPIO_EMC_B2_01_GPIO_MUX2_IO11
IOMUXC_GPIO_EMC_B2_01_XBAR1_INOUT21
IOMUXC_GPIO_EMC_B2_01_ENET_QOS_1588_EVENT1_IN
IOMUXC_GPIO_EMC_B2_01_LPSP11_PCS0
IOMUXC_GPIO_EMC_B2_01_LPI2C2_SDA
IOMUXC_GPIO_EMC_B2_01_GPIO8_IO11
IOMUXC_GPIO_EMC_B2_01_FLEXPWM3_PWM0_B
IOMUXC_GPIO_EMC_B2_02_SEMC_DATA18
IOMUXC_GPIO_EMC_B2_02_USDHC2_WP
IOMUXC_GPIO_EMC_B2_02_VIDEO_MUX_CSI_DATA23
IOMUXC_GPIO_EMC_B2_02_FLEXSPI2_B_DATA04
IOMUXC_GPIO_EMC_B2_02_GPIO_MUX2_IO12
IOMUXC_GPIO_EMC_B2_02_XBAR1_INOUT22
IOMUXC_GPIO_EMC_B2_02_ENET_QOS_1588_EVENT1_AUX_IN
IOMUXC_GPIO_EMC_B2_02_LPSP11_SOUT
IOMUXC_GPIO_EMC_B2_02_GPIO8_IO12
IOMUXC_GPIO_EMC_B2_02_FLEXPWM3_PWM1_A
IOMUXC_GPIO_EMC_B2_03_SEMC_DATA19
IOMUXC_GPIO_EMC_B2_03_USDHC2_VSELECT
IOMUXC_GPIO_EMC_B2_03_VIDEO_MUX_CSI_DATA22
IOMUXC_GPIO_EMC_B2_03_FLEXSPI2_B_DATA03
IOMUXC_GPIO_EMC_B2_03_GPIO_MUX2_IO13

IOMUXC_GPIO_EMC_B2_03_XBAR1_INOUT23
IOMUXC_GPIO_EMC_B2_03_ENET_1G_TX_DATA03
IOMUXC_GPIO_EMC_B2_03_LPSPI1_SIN
IOMUXC_GPIO_EMC_B2_03_GPIO8_IO13
IOMUXC_GPIO_EMC_B2_03_FLEXPWM3_PWM1_B
IOMUXC_GPIO_EMC_B2_04_SEMC_DATA20
IOMUXC_GPIO_EMC_B2_04_USDHC2_RESET_B
IOMUXC_GPIO_EMC_B2_04_SAI2_MCLK
IOMUXC_GPIO_EMC_B2_04_VIDEO_MUX_CSI_DATA21
IOMUXC_GPIO_EMC_B2_04_FLEXSPI2_B_DATA02
IOMUXC_GPIO_EMC_B2_04_GPIO_MUX2_IO14
IOMUXC_GPIO_EMC_B2_04_XBAR1_INOUT24
IOMUXC_GPIO_EMC_B2_04_ENET_1G_TX_DATA02
IOMUXC_GPIO_EMC_B2_04_LPSPI3_SCK
IOMUXC_GPIO_EMC_B2_04_GPIO8_IO14
IOMUXC_GPIO_EMC_B2_04_FLEXPWM3_PWM2_A
IOMUXC_GPIO_EMC_B2_05_SEMC_DATA21
IOMUXC_GPIO_EMC_B2_05_GPT3_CLK
IOMUXC_GPIO_EMC_B2_05_SAI2_RX_SYNC
IOMUXC_GPIO_EMC_B2_05_VIDEO_MUX_CSI_DATA20
IOMUXC_GPIO_EMC_B2_05_FLEXSPI2_B_DATA01
IOMUXC_GPIO_EMC_B2_05_GPIO_MUX2_IO15
IOMUXC_GPIO_EMC_B2_05_XBAR1_INOUT25
IOMUXC_GPIO_EMC_B2_05_ENET_1G_RX_CLK
IOMUXC_GPIO_EMC_B2_05_LPSPI3_PCS0
IOMUXC_GPIO_EMC_B2_05_PIT1_TRIGGER0
IOMUXC_GPIO_EMC_B2_05_GPIO8_IO15
IOMUXC_GPIO_EMC_B2_05_FLEXPWM3_PWM2_B
IOMUXC_GPIO_EMC_B2_06_SEMC_DATA22
IOMUXC_GPIO_EMC_B2_06_GPT3_CAPTURE1
IOMUXC_GPIO_EMC_B2_06_GPIO8_IO16
IOMUXC_GPIO_EMC_B2_06_SAI2_RX_BCLK
IOMUXC_GPIO_EMC_B2_06_FLEXPWM3_PWM3_A

IOMUXC_GPIO_EMC_B2_06_VIDEO_MUX_CSI_DATA19
IOMUXC_GPIO_EMC_B2_06_FLEXSPI2_B_DATA00
IOMUXC_GPIO_EMC_B2_06_GPIO_MUX2_IO16
IOMUXC_GPIO_EMC_B2_06_XBAR1_INOUT26
IOMUXC_GPIO_EMC_B2_06_ENET_1G_TX_ER
IOMUXC_GPIO_EMC_B2_06_LPSP13_SOUT
IOMUXC_GPIO_EMC_B2_06_PIT1_TRIGGER1
IOMUXC_GPIO_EMC_B2_07_SEMC_DATA23
IOMUXC_GPIO_EMC_B2_07_GPT3_CAPTURE2
IOMUXC_GPIO_EMC_B2_07_SAI2_RX_DATA
IOMUXC_GPIO_EMC_B2_07_VIDEO_MUX_CSI_DATA18
IOMUXC_GPIO_EMC_B2_07_FLEXSPI2_B_DQS
IOMUXC_GPIO_EMC_B2_07_GPIO_MUX2_IO17
IOMUXC_GPIO_EMC_B2_07_XBAR1_INOUT27
IOMUXC_GPIO_EMC_B2_07_ENET_1G_RX_DATA03
IOMUXC_GPIO_EMC_B2_07_LPSP13_SIN
IOMUXC_GPIO_EMC_B2_07_PIT1_TRIGGER2
IOMUXC_GPIO_EMC_B2_07_GPIO8_IO17
IOMUXC_GPIO_EMC_B2_07_FLEXPWM3_PWM3_B
IOMUXC_GPIO_EMC_B2_08_SEMC_DM02
IOMUXC_GPIO_EMC_B2_08_GPT3_COMPARE1
IOMUXC_GPIO_EMC_B2_08_SAI2_TX_DATA
IOMUXC_GPIO_EMC_B2_08_VIDEO_MUX_CSI_DATA17
IOMUXC_GPIO_EMC_B2_08_FLEXSPI2_B_SS0_B
IOMUXC_GPIO_EMC_B2_08_GPIO_MUX2_IO18
IOMUXC_GPIO_EMC_B2_08_XBAR1_INOUT28
IOMUXC_GPIO_EMC_B2_08_ENET_1G_RX_DATA02
IOMUXC_GPIO_EMC_B2_08_LPSP13_PCS1
IOMUXC_GPIO_EMC_B2_08_PIT1_TRIGGER3
IOMUXC_GPIO_EMC_B2_08_GPIO8_IO18
IOMUXC_GPIO_EMC_B2_09_GPIO8_IO19
IOMUXC_GPIO_EMC_B2_09_SEMC_DATA24
IOMUXC_GPIO_EMC_B2_09_GPT3_COMPARE2

IOMUXC_GPIO_EMC_B2_09_SAI2_TX_BCLK
IOMUXC_GPIO_EMC_B2_09_VIDEO_MUX_CSI_DATA16
IOMUXC_GPIO_EMC_B2_09_FLEXSPI2_B_SCLK
IOMUXC_GPIO_EMC_B2_09_GPIO_MUX2_IO19
IOMUXC_GPIO_EMC_B2_09_XBAR1_INOUT29
IOMUXC_GPIO_EMC_B2_09_ENET_1G_CRS
IOMUXC_GPIO_EMC_B2_09_LPSPI3_PCS2
IOMUXC_GPIO_EMC_B2_09_TMR1_TIMER0
IOMUXC_GPIO_EMC_B2_10_GPIO8_IO20
IOMUXC_GPIO_EMC_B2_10_SEMC_DATA25
IOMUXC_GPIO_EMC_B2_10_GPT3_COMPARE3
IOMUXC_GPIO_EMC_B2_10_SAI2_TX_SYNC
IOMUXC_GPIO_EMC_B2_10_VIDEO_MUX_CSI_FIELD
IOMUXC_GPIO_EMC_B2_10_FLEXSPI2_A_SCLK
IOMUXC_GPIO_EMC_B2_10_GPIO_MUX2_IO20
IOMUXC_GPIO_EMC_B2_10_XBAR1_INOUT30
IOMUXC_GPIO_EMC_B2_10_ENET_1G_COL
IOMUXC_GPIO_EMC_B2_10_LPSPI3_PCS3
IOMUXC_GPIO_EMC_B2_10_TMR1_TIMER1
IOMUXC_GPIO_EMC_B2_11_SEMC_DATA26
IOMUXC_GPIO_EMC_B2_11_SPDIF_IN
IOMUXC_GPIO_EMC_B2_11_ENET_1G_TX_DATA00
IOMUXC_GPIO_EMC_B2_11_SAI3_RX_SYNC
IOMUXC_GPIO_EMC_B2_11_FLEXSPI2_A_SS0_B
IOMUXC_GPIO_EMC_B2_11_GPIO_MUX2_IO21
IOMUXC_GPIO_EMC_B2_11_XBAR1_INOUT31
IOMUXC_GPIO_EMC_B2_11_EMVSIM1_IO
IOMUXC_GPIO_EMC_B2_11_TMR1_TIMER2
IOMUXC_GPIO_EMC_B2_11_GPIO8_IO21
IOMUXC_GPIO_EMC_B2_12_SEMC_DATA27
IOMUXC_GPIO_EMC_B2_12_SPDIF_OUT
IOMUXC_GPIO_EMC_B2_12_ENET_1G_TX_DATA01
IOMUXC_GPIO_EMC_B2_12_SAI3_RX_BCLK

IOMUXC_GPIO_EMC_B2_12_FLEXSPI2_A_DQS
IOMUXC_GPIO_EMC_B2_12_GPIO_MUX2_IO22
IOMUXC_GPIO_EMC_B2_12_XBAR1_INOUT32
IOMUXC_GPIO_EMC_B2_12_EMVSIM1_CLK
IOMUXC_GPIO_EMC_B2_12_TMR1_TIMER3
IOMUXC_GPIO_EMC_B2_12_GPIO8_IO22
IOMUXC_GPIO_EMC_B2_13_GPIO8_IO23
IOMUXC_GPIO_EMC_B2_13_SEMC_DATA28
IOMUXC_GPIO_EMC_B2_13_ENET_1G_TX_EN
IOMUXC_GPIO_EMC_B2_13_SAI3_RX_DATA
IOMUXC_GPIO_EMC_B2_13_FLEXSPI2_A_DATA00
IOMUXC_GPIO_EMC_B2_13_GPIO_MUX2_IO23
IOMUXC_GPIO_EMC_B2_13_XBAR1_INOUT33
IOMUXC_GPIO_EMC_B2_13_EMVSIM1_RST
IOMUXC_GPIO_EMC_B2_13_TMR2_TIMER0
IOMUXC_GPIO_EMC_B2_14_SEMC_DATA29
IOMUXC_GPIO_EMC_B2_14_ENET_1G_TX_CLK_IO
IOMUXC_GPIO_EMC_B2_14_SAI3_TX_DATA
IOMUXC_GPIO_EMC_B2_14_FLEXSPI2_A_DATA01
IOMUXC_GPIO_EMC_B2_14_GPIO_MUX2_IO24
IOMUXC_GPIO_EMC_B2_14_XBAR1_INOUT34
IOMUXC_GPIO_EMC_B2_14_SFA_ipp_do_atx_clk_under_test
IOMUXC_GPIO_EMC_B2_14_EMVSIM1_SVEN
IOMUXC_GPIO_EMC_B2_14_TMR2_TIMER1
IOMUXC_GPIO_EMC_B2_14_GPIO8_IO24
IOMUXC_GPIO_EMC_B2_15_SEMC_DATA30
IOMUXC_GPIO_EMC_B2_15_ENET_1G_RX_DATA00
IOMUXC_GPIO_EMC_B2_15_SAI3_TX_BCLK
IOMUXC_GPIO_EMC_B2_15_FLEXSPI2_A_DATA02
IOMUXC_GPIO_EMC_B2_15_GPIO_MUX2_IO25
IOMUXC_GPIO_EMC_B2_15_XBAR1_INOUT35
IOMUXC_GPIO_EMC_B2_15_EMVSIM1_PD
IOMUXC_GPIO_EMC_B2_15_TMR2_TIMER2

IOMUXC_GPIO_EMC_B2_15_GPIO8_IO25
IOMUXC_GPIO_EMC_B2_16_GPIO8_IO26
IOMUXC_GPIO_EMC_B2_16_SEMC_DATA31
IOMUXC_GPIO_EMC_B2_16_XBAR1_INOUT14
IOMUXC_GPIO_EMC_B2_16_ENET_1G_RX_DATA01
IOMUXC_GPIO_EMC_B2_16_SAI3_TX_SYNC
IOMUXC_GPIO_EMC_B2_16_FLEXSPI2_A_DATA03
IOMUXC_GPIO_EMC_B2_16_GPIO_MUX2_IO26
IOMUXC_GPIO_EMC_B2_16_EMVSIM1_POWER_FAIL
IOMUXC_GPIO_EMC_B2_16_TMR2_TIMER3
IOMUXC_GPIO_EMC_B2_17_SEMC_DM03
IOMUXC_GPIO_EMC_B2_17_XBAR1_INOUT15
IOMUXC_GPIO_EMC_B2_17_ENET_1G_RX_EN
IOMUXC_GPIO_EMC_B2_17_SAI3_MCLK
IOMUXC_GPIO_EMC_B2_17_FLEXSPI2_A_DATA04
IOMUXC_GPIO_EMC_B2_17_GPIO_MUX2_IO27
IOMUXC_GPIO_EMC_B2_17_WDOG1_ANY
IOMUXC_GPIO_EMC_B2_17_TMR3_TIMER0
IOMUXC_GPIO_EMC_B2_17_GPIO8_IO27
IOMUXC_GPIO_EMC_B2_18_SEMC_DQS4
IOMUXC_GPIO_EMC_B2_18_XBAR1_INOUT16
IOMUXC_GPIO_EMC_B2_18_ENET_1G_RX_ER
IOMUXC_GPIO_EMC_B2_18_EWM_OUT_B
IOMUXC_GPIO_EMC_B2_18_FLEXSPI2_A_DATA05
IOMUXC_GPIO_EMC_B2_18_GPIO_MUX2_IO28
IOMUXC_GPIO_EMC_B2_18_FLEXSPI1_A_DQS
IOMUXC_GPIO_EMC_B2_18_WDOG1_B
IOMUXC_GPIO_EMC_B2_18_TMR3_TIMER1
IOMUXC_GPIO_EMC_B2_18_GPIO8_IO28
IOMUXC_GPIO_EMC_B2_19_GPIO8_IO29
IOMUXC_GPIO_EMC_B2_19_SEMC_CLKX00
IOMUXC_GPIO_EMC_B2_19_ENET_MDC
IOMUXC_GPIO_EMC_B2_19_ENET_1G_MDC

IOMUXC_GPIO_EMC_B2_19_ENET_1G_REF_CLK
IOMUXC_GPIO_EMC_B2_19_FLEXSPI2_A_DATA06
IOMUXC_GPIO_EMC_B2_19_GPIO_MUX2_IO29
IOMUXC_GPIO_EMC_B2_19_ENET_QOS_MDC
IOMUXC_GPIO_EMC_B2_19_TMR3_TIMER2
IOMUXC_GPIO_EMC_B2_20_GPIO8_IO30
IOMUXC_GPIO_EMC_B2_20_SEMC_CLKX01
IOMUXC_GPIO_EMC_B2_20_ENET_MDIO
IOMUXC_GPIO_EMC_B2_20_ENET_1G_MDIO
IOMUXC_GPIO_EMC_B2_20_ENET_QOS_REF_CLK
IOMUXC_GPIO_EMC_B2_20_FLEXSPI2_A_DATA07
IOMUXC_GPIO_EMC_B2_20_GPIO_MUX2_IO30
IOMUXC_GPIO_EMC_B2_20_ENET_QOS_MDIO
IOMUXC_GPIO_EMC_B2_20_TMR3_TIMER3
IOMUXC_GPIO_AD_00_GPIO8_IO31
IOMUXC_GPIO_AD_00_EMVSIM1_IO
IOMUXC_GPIO_AD_00_FLEXCAN2_TX
IOMUXC_GPIO_AD_00_ENET_1G_1588_EVENT1_IN
IOMUXC_GPIO_AD_00_GPT2_CAPTURE1
IOMUXC_GPIO_AD_00_FLEXPWM1_PWM0_A
IOMUXC_GPIO_AD_00_GPIO_MUX2_IO31
IOMUXC_GPIO_AD_00_LPUART7_TXD
IOMUXC_GPIO_AD_00_FLEXIO2_D00
IOMUXC_GPIO_AD_00_FLEXSPI2_B_SS1_B
IOMUXC_GPIO_AD_01_GPIO9_IO00
IOMUXC_GPIO_AD_01_EMVSIM1_CLK
IOMUXC_GPIO_AD_01_FLEXCAN2_RX
IOMUXC_GPIO_AD_01_ENET_1G_1588_EVENT1_OUT
IOMUXC_GPIO_AD_01_GPT2_CAPTURE2
IOMUXC_GPIO_AD_01_FLEXPWM1_PWM0_B
IOMUXC_GPIO_AD_01_GPIO_MUX3_IO00
IOMUXC_GPIO_AD_01_LPUART7_RXD
IOMUXC_GPIO_AD_01_FLEXIO2_D01

IOMUXC_GPIO_AD_01_FLEXSPI2_A_SS1_B
IOMUXC_GPIO_AD_02_GPIO9_IO01
IOMUXC_GPIO_AD_02_EMVSIM1_RST
IOMUXC_GPIO_AD_02_LPUART7_CTS_B
IOMUXC_GPIO_AD_02_ENET_1G_1588_EVENT2_IN
IOMUXC_GPIO_AD_02_GPT2_COMPARE1
IOMUXC_GPIO_AD_02_FLEXPWM1_PWM1_A
IOMUXC_GPIO_AD_02_GPIO_MUX3_IO01
IOMUXC_GPIO_AD_02_LPUART8_TXD
IOMUXC_GPIO_AD_02_FLEXIO2_D02
IOMUXC_GPIO_AD_02_VIDEO_MUX_EXT_DCIC1
IOMUXC_GPIO_AD_03_GPIO9_IO02
IOMUXC_GPIO_AD_03_EMVSIM1_SVEN
IOMUXC_GPIO_AD_03_LPUART7_RTS_B
IOMUXC_GPIO_AD_03_ENET_1G_1588_EVENT2_OUT
IOMUXC_GPIO_AD_03_GPT2_COMPARE2
IOMUXC_GPIO_AD_03_FLEXPWM1_PWM1_B
IOMUXC_GPIO_AD_03_GPIO_MUX3_IO02
IOMUXC_GPIO_AD_03_LPUART8_RXD
IOMUXC_GPIO_AD_03_FLEXIO2_D03
IOMUXC_GPIO_AD_03_VIDEO_MUX_EXT_DCIC2
IOMUXC_GPIO_AD_04_EMVSIM1_PD
IOMUXC_GPIO_AD_04_LPUART8_CTS_B
IOMUXC_GPIO_AD_04_ENET_1G_1588_EVENT3_IN
IOMUXC_GPIO_AD_04_GPT2_COMPARE3
IOMUXC_GPIO_AD_04_FLEXPWM1_PWM2_A
IOMUXC_GPIO_AD_04_GPIO_MUX3_IO03
IOMUXC_GPIO_AD_04_WDOG1_B
IOMUXC_GPIO_AD_04_FLEXIO2_D04
IOMUXC_GPIO_AD_04_TMR4_TIMER0
IOMUXC_GPIO_AD_04_GPIO9_IO03
IOMUXC_GPIO_AD_05_EMVSIM1_POWER_FAIL
IOMUXC_GPIO_AD_05_LPUART8_RTS_B

IOMUXC_GPIO_AD_05_ENET_1G_1588_EVENT3_OUT
IOMUXC_GPIO_AD_05_GPT2_CLK
IOMUXC_GPIO_AD_05_FLEXPWM1_PWM2_B
IOMUXC_GPIO_AD_05_GPIO_MUX3_IO04
IOMUXC_GPIO_AD_05_WDOG2_B
IOMUXC_GPIO_AD_05_FLEXIO2_D05
IOMUXC_GPIO_AD_05_TMR4_TIMER1
IOMUXC_GPIO_AD_05_GPIO9_IO04
IOMUXC_GPIO_AD_06_USB_OTG2_OC
IOMUXC_GPIO_AD_06_FLEXCAN1_TX
IOMUXC_GPIO_AD_06_EMVSIM2_IO
IOMUXC_GPIO_AD_06_GPT3_CAPTURE1
IOMUXC_GPIO_AD_06_VIDEO_MUX_CSI_DATA15
IOMUXC_GPIO_AD_06_GPIO_MUX3_IO05
IOMUXC_GPIO_AD_06_ENET_1588_EVENT1_IN
IOMUXC_GPIO_AD_06_FLEXIO2_D06
IOMUXC_GPIO_AD_06_TMR4_TIMER2
IOMUXC_GPIO_AD_06_GPIO9_IO05
IOMUXC_GPIO_AD_06_FLEXPWM1_PWM0_X
IOMUXC_GPIO_AD_07_USB_OTG2_PWR
IOMUXC_GPIO_AD_07_FLEXCAN1_RX
IOMUXC_GPIO_AD_07_EMVSIM2_CLK
IOMUXC_GPIO_AD_07_GPT3_CAPTURE2
IOMUXC_GPIO_AD_07_VIDEO_MUX_CSI_DATA14
IOMUXC_GPIO_AD_07_GPIO_MUX3_IO06
IOMUXC_GPIO_AD_07_ENET_1588_EVENT1_OUT
IOMUXC_GPIO_AD_07_FLEXIO2_D07
IOMUXC_GPIO_AD_07_TMR4_TIMER3
IOMUXC_GPIO_AD_07_GPIO9_IO06
IOMUXC_GPIO_AD_07_FLEXPWM1_PWM1_X
IOMUXC_GPIO_AD_08_USBPHY2_OTG_ID
IOMUXC_GPIO_AD_08_LPI2C1_SCL
IOMUXC_GPIO_AD_08_EMVSIM2_RST

IOMUXC_GPIO_AD_08_GPT3_COMPARE1
IOMUXC_GPIO_AD_08_VIDEO_MUX_CSI_DATA13
IOMUXC_GPIO_AD_08_GPIO_MUX3_IO07
IOMUXC_GPIO_AD_08_ENET_1588_EVENT2_IN
IOMUXC_GPIO_AD_08_FLEXIO2_D08
IOMUXC_GPIO_AD_08_GPIO9_IO07
IOMUXC_GPIO_AD_08_FLEXPWM1_PWM2_X
IOMUXC_GPIO_AD_09_USBPHY1_OTG_ID
IOMUXC_GPIO_AD_09_LPI2C1_SDA
IOMUXC_GPIO_AD_09_EMVSIM2_SVEN
IOMUXC_GPIO_AD_09_GPT3_COMPARE2
IOMUXC_GPIO_AD_09_VIDEO_MUX_CSI_DATA12
IOMUXC_GPIO_AD_09_GPIO_MUX3_IO08
IOMUXC_GPIO_AD_09_ENET_1588_EVENT2_OUT
IOMUXC_GPIO_AD_09_FLEXIO2_D09
IOMUXC_GPIO_AD_09_GPIO9_IO08
IOMUXC_GPIO_AD_09_FLEXPWM1_PWM3_X
IOMUXC_GPIO_AD_10_USB_OTG1_PWR
IOMUXC_GPIO_AD_10_LPI2C1_SCLS
IOMUXC_GPIO_AD_10_EMVSIM2_PD
IOMUXC_GPIO_AD_10_GPT3_COMPARE3
IOMUXC_GPIO_AD_10_VIDEO_MUX_CSI_DATA11
IOMUXC_GPIO_AD_10_GPIO_MUX3_IO09
IOMUXC_GPIO_AD_10_ENET_1588_EVENT3_IN
IOMUXC_GPIO_AD_10_FLEXIO2_D10
IOMUXC_GPIO_AD_10_GPIO9_IO09
IOMUXC_GPIO_AD_10_FLEXPWM2_PWM0_X
IOMUXC_GPIO_AD_11_USB_OTG1_OC
IOMUXC_GPIO_AD_11_LPI2C1_SDAS
IOMUXC_GPIO_AD_11_EMVSIM2_POWER_FAIL
IOMUXC_GPIO_AD_11_GPT3_CLK
IOMUXC_GPIO_AD_11_VIDEO_MUX_CSI_DATA10
IOMUXC_GPIO_AD_11_GPIO_MUX3_IO10

IOMUXC_GPIO_AD_11_ENET_1588_EVENT3_OUT
IOMUXC_GPIO_AD_11_FLEXIO2_D11
IOMUXC_GPIO_AD_11_GPIO9_IO10
IOMUXC_GPIO_AD_11_FLEXPWM2_PWM1_X
IOMUXC_GPIO_AD_12_SPDIF_LOCK
IOMUXC_GPIO_AD_12_LPI2C1_HREQ
IOMUXC_GPIO_AD_12_GPT1_CAPTURE1
IOMUXC_GPIO_AD_12_FLEXSPI1_B_DATA03
IOMUXC_GPIO_AD_12_VIDEO_MUX_CSI_PIXCLK
IOMUXC_GPIO_AD_12_GPIO_MUX3_IO11
IOMUXC_GPIO_AD_12_ENET_TX_DATA03
IOMUXC_GPIO_AD_12_FLEXIO2_D12
IOMUXC_GPIO_AD_12_EWM_OUT_B
IOMUXC_GPIO_AD_12_GPIO9_IO11
IOMUXC_GPIO_AD_12_FLEXPWM2_PWM2_X
IOMUXC_GPIO_AD_13_SPDIF_SR_CLK
IOMUXC_GPIO_AD_13_PIT1_TRIGGER0
IOMUXC_GPIO_AD_13_GPT1_CAPTURE2
IOMUXC_GPIO_AD_13_FLEXSPI1_B_DATA02
IOMUXC_GPIO_AD_13_VIDEO_MUX_CSI_MCLK
IOMUXC_GPIO_AD_13_GPIO_MUX3_IO12
IOMUXC_GPIO_AD_13_ENET_TX_DATA02
IOMUXC_GPIO_AD_13_FLEXIO2_D13
IOMUXC_GPIO_AD_13_REF_CLK_32K
IOMUXC_GPIO_AD_13_GPIO9_IO12
IOMUXC_GPIO_AD_13_FLEXPWM2_PWM3_X
IOMUXC_GPIO_AD_14_SPDIF_EXT_CLK
IOMUXC_GPIO_AD_14_REF_CLK_24M
IOMUXC_GPIO_AD_14_GPT1_COMPARE1
IOMUXC_GPIO_AD_14_FLEXSPI1_B_DATA01
IOMUXC_GPIO_AD_14_VIDEO_MUX_CSI_VSYNC
IOMUXC_GPIO_AD_14_GPIO_MUX3_IO13
IOMUXC_GPIO_AD_14_ENET_RX_CLK

IOMUXC_GPIO_AD_14_FLEXIO2_D14
IOMUXC_GPIO_AD_14_CCM_ENET_REF_CLK_25M
IOMUXC_GPIO_AD_14_GPIO9_IO13
IOMUXC_GPIO_AD_14_FLEXPWM3_PWM0_X
IOMUXC_GPIO_AD_15_GPIO9_IO14
IOMUXC_GPIO_AD_15_FLEXPWM3_PWM1_X
IOMUXC_GPIO_AD_15_SPDIF_IN
IOMUXC_GPIO_AD_15_LPUART10_TXD
IOMUXC_GPIO_AD_15_GPT1_COMPARE2
IOMUXC_GPIO_AD_15_FLEXSPI1_B_DATA00
IOMUXC_GPIO_AD_15_VIDEO_MUX_CSI_HSYNC
IOMUXC_GPIO_AD_15_GPIO_MUX3_IO14
IOMUXC_GPIO_AD_15_ENET_TX_ER
IOMUXC_GPIO_AD_15_FLEXIO2_D15
IOMUXC_GPIO_AD_16_SPDIF_OUT
IOMUXC_GPIO_AD_16_LPUART10_RXD
IOMUXC_GPIO_AD_16_GPT1_COMPARE3
IOMUXC_GPIO_AD_16_FLEXSPI1_B_SCLK
IOMUXC_GPIO_AD_16_VIDEO_MUX_CSI_DATA09
IOMUXC_GPIO_AD_16_GPIO_MUX3_IO15
IOMUXC_GPIO_AD_16_ENET_RX_DATA03
IOMUXC_GPIO_AD_16_FLEXIO2_D16
IOMUXC_GPIO_AD_16_ENET_1G_MDC
IOMUXC_GPIO_AD_16_GPIO9_IO15
IOMUXC_GPIO_AD_16_FLEXPWM3_PWM2_X
IOMUXC_GPIO_AD_17_SAI1_MCLK
IOMUXC_GPIO_AD_17_ACMP1_OUT
IOMUXC_GPIO_AD_17_GPT1_CLK
IOMUXC_GPIO_AD_17_FLEXSPI1_A_DQS
IOMUXC_GPIO_AD_17_VIDEO_MUX_CSI_DATA08
IOMUXC_GPIO_AD_17_GPIO_MUX3_IO16
IOMUXC_GPIO_AD_17_ENET_RX_DATA02
IOMUXC_GPIO_AD_17_FLEXIO2_D17

IOMUXC_GPIO_AD_17_ENET_1G_MDIO
IOMUXC_GPIO_AD_17_GPIO9_IO16
IOMUXC_GPIO_AD_17_FLEXPWM3_PWM3_X
IOMUXC_GPIO_AD_18_GPIO9_IO17
IOMUXC_GPIO_AD_18_FLEXPWM4_PWM0_X
IOMUXC_GPIO_AD_18_SAI1_RX_SYNC
IOMUXC_GPIO_AD_18_ACMP2_OUT
IOMUXC_GPIO_AD_18_LPSPI1_PCS1
IOMUXC_GPIO_AD_18_FLEXSPI1_A_SS0_B
IOMUXC_GPIO_AD_18_VIDEO_MUX_CSI_DATA07
IOMUXC_GPIO_AD_18_GPIO_MUX3_IO17
IOMUXC_GPIO_AD_18_ENET_CRS
IOMUXC_GPIO_AD_18_FLEXIO2_D18
IOMUXC_GPIO_AD_18_LPI2C2_SCL
IOMUXC_GPIO_AD_19_SAI1_RX_BCLK
IOMUXC_GPIO_AD_19_ACMP3_OUT
IOMUXC_GPIO_AD_19_LPSPI1_PCS2
IOMUXC_GPIO_AD_19_FLEXSPI1_A_SCLK
IOMUXC_GPIO_AD_19_VIDEO_MUX_CSI_DATA06
IOMUXC_GPIO_AD_19_GPIO_MUX3_IO18
IOMUXC_GPIO_AD_19_ENET_COL
IOMUXC_GPIO_AD_19_FLEXIO2_D19
IOMUXC_GPIO_AD_19_LPI2C2_SDA
IOMUXC_GPIO_AD_19_GPIO9_IO18
IOMUXC_GPIO_AD_19_FLEXPWM4_PWM1_X
IOMUXC_GPIO_AD_20_SAI1_RX_DATA00
IOMUXC_GPIO_AD_20_ACMP4_OUT
IOMUXC_GPIO_AD_20_LPSPI1_PCS3
IOMUXC_GPIO_AD_20_FLEXSPI1_A_DATA00
IOMUXC_GPIO_AD_20_VIDEO_MUX_CSI_DATA05
IOMUXC_GPIO_AD_20_GPIO_MUX3_IO19
IOMUXC_GPIO_AD_20_KPP_ROW07
IOMUXC_GPIO_AD_20_FLEXIO2_D20

IOMUXC_GPIO_AD_20_ENET_QOS_1588_EVENT2_OUT
IOMUXC_GPIO_AD_20_GPIO9_IO19
IOMUXC_GPIO_AD_20_FLEXPWM4_PWM2_X
IOMUXC_GPIO_AD_21_SAI1_TX_DATA00
IOMUXC_GPIO_AD_21_LPSPI2_PCS1
IOMUXC_GPIO_AD_21_FLEXSPI1_A_DATA01
IOMUXC_GPIO_AD_21_VIDEO_MUX_CSI_DATA04
IOMUXC_GPIO_AD_21_GPIO_MUX3_IO20
IOMUXC_GPIO_AD_21_KPP_COL07
IOMUXC_GPIO_AD_21_FLEXIO2_D21
IOMUXC_GPIO_AD_21_ENET_QOS_1588_EVENT2_IN
IOMUXC_GPIO_AD_21_GPIO9_IO20
IOMUXC_GPIO_AD_21_FLEXPWM4_PWM3_X
IOMUXC_GPIO_AD_22_GPIO9_IO21
IOMUXC_GPIO_AD_22_SAI1_TX_BCLK
IOMUXC_GPIO_AD_22_LPSPI2_PCS2
IOMUXC_GPIO_AD_22_FLEXSPI1_A_DATA02
IOMUXC_GPIO_AD_22_VIDEO_MUX_CSI_DATA03
IOMUXC_GPIO_AD_22_GPIO_MUX3_IO21
IOMUXC_GPIO_AD_22_KPP_ROW06
IOMUXC_GPIO_AD_22_FLEXIO2_D22
IOMUXC_GPIO_AD_22_ENET_QOS_1588_EVENT3_OUT
IOMUXC_GPIO_AD_23_SAI1_TX_SYNC
IOMUXC_GPIO_AD_23_LPSPI2_PCS3
IOMUXC_GPIO_AD_23_FLEXSPI1_A_DATA03
IOMUXC_GPIO_AD_23_VIDEO_MUX_CSI_DATA02
IOMUXC_GPIO_AD_23_GPIO_MUX3_IO22
IOMUXC_GPIO_AD_23_KPP_COL06
IOMUXC_GPIO_AD_23_FLEXIO2_D23
IOMUXC_GPIO_AD_23_ENET_QOS_1588_EVENT3_IN
IOMUXC_GPIO_AD_23_GPIO9_IO22
IOMUXC_GPIO_AD_24_LPUART1_TXD
IOMUXC_GPIO_AD_24_LPSPI2_SCK

IOMUXC_GPIO_AD_24_VIDEO_MUX_CSI_DATA00
IOMUXC_GPIO_AD_24_ENET_RX_EN
IOMUXC_GPIO_AD_24_FLEXPWM2_PWM0_A
IOMUXC_GPIO_AD_24_GPIO_MUX3_IO23
IOMUXC_GPIO_AD_24_KPP_ROW05
IOMUXC_GPIO_AD_24_FLEXIO2_D24
IOMUXC_GPIO_AD_24_LPI2C4_SCL
IOMUXC_GPIO_AD_24_GPIO9_IO23
IOMUXC_GPIO_AD_25_GPIO9_IO24
IOMUXC_GPIO_AD_25_LPUART1_RXD
IOMUXC_GPIO_AD_25_LPSPI2_PCS0
IOMUXC_GPIO_AD_25_VIDEO_MUX_CSI_DATA01
IOMUXC_GPIO_AD_25_ENET_RX_ER
IOMUXC_GPIO_AD_25_FLEXPWM2_PWM0_B
IOMUXC_GPIO_AD_25_GPIO_MUX3_IO24
IOMUXC_GPIO_AD_25_KPP_COL05
IOMUXC_GPIO_AD_25_FLEXIO2_D25
IOMUXC_GPIO_AD_25_LPI2C4_SDA
IOMUXC_GPIO_AD_26_LPUART1_CTS_B
IOMUXC_GPIO_AD_26_LPSPI2_SOUT
IOMUXC_GPIO_AD_26_SEMC_CSX01
IOMUXC_GPIO_AD_26_ENET_RX_DATA00
IOMUXC_GPIO_AD_26_FLEXPWM2_PWM1_A
IOMUXC_GPIO_AD_26_GPIO_MUX3_IO25
IOMUXC_GPIO_AD_26_KPP_ROW04
IOMUXC_GPIO_AD_26_FLEXIO2_D26
IOMUXC_GPIO_AD_26_ENET_QOS_MDC
IOMUXC_GPIO_AD_26_GPIO9_IO25
IOMUXC_GPIO_AD_26_USDHC2_CD_B
IOMUXC_GPIO_AD_27_LPUART1_RTS_B
IOMUXC_GPIO_AD_27_LPSPI2_SIN
IOMUXC_GPIO_AD_27_SEMC_CSX02
IOMUXC_GPIO_AD_27_ENET_RX_DATA01

IOMUXC_GPIO_AD_27_FLEXPWM2_PWM1_B
IOMUXC_GPIO_AD_27_GPIO_MUX3_IO26
IOMUXC_GPIO_AD_27_KPP_COL04
IOMUXC_GPIO_AD_27_FLEXIO2_D27
IOMUXC_GPIO_AD_27_ENET_QOS_MDIO
IOMUXC_GPIO_AD_27_GPIO9_IO26
IOMUXC_GPIO_AD_27_USDHC2_WP
IOMUXC_GPIO_AD_28_GPIO9_IO27
IOMUXC_GPIO_AD_28_USDHC2_VSELECT
IOMUXC_GPIO_AD_28_LPSPI1_SCK
IOMUXC_GPIO_AD_28_LPUART5_TXD
IOMUXC_GPIO_AD_28_SEMC_CSX03
IOMUXC_GPIO_AD_28_ENET_TX_EN
IOMUXC_GPIO_AD_28_FLEXPWM2_PWM2_A
IOMUXC_GPIO_AD_28_GPIO_MUX3_IO27
IOMUXC_GPIO_AD_28_KPP_ROW03
IOMUXC_GPIO_AD_28_FLEXIO2_D28
IOMUXC_GPIO_AD_28_VIDEO_MUX_EXT_DCIC1
IOMUXC_GPIO_AD_29_LPSPI1_PCS0
IOMUXC_GPIO_AD_29_LPUART5_RXD
IOMUXC_GPIO_AD_29_ENET_REF_CLK
IOMUXC_GPIO_AD_29_ENET_TX_CLK
IOMUXC_GPIO_AD_29_FLEXPWM2_PWM2_B
IOMUXC_GPIO_AD_29_GPIO_MUX3_IO28
IOMUXC_GPIO_AD_29_KPP_COL03
IOMUXC_GPIO_AD_29_FLEXIO2_D29
IOMUXC_GPIO_AD_29_VIDEO_MUX_EXT_DCIC2
IOMUXC_GPIO_AD_29_GPIO9_IO28
IOMUXC_GPIO_AD_29_USDHC2_RESET_B
IOMUXC_GPIO_AD_30_LPSPI1_SOUT
IOMUXC_GPIO_AD_30_USB_OTG2_OC
IOMUXC_GPIO_AD_30_FLEXCAN2_TX
IOMUXC_GPIO_AD_30_ENET_TX_DATA00

IOMUXC_GPIO_AD_30_LPUART3_TXD
IOMUXC_GPIO_AD_30_GPIO_MUX3_IO29
IOMUXC_GPIO_AD_30_KPP_ROW02
IOMUXC_GPIO_AD_30_FLEXIO2_D30
IOMUXC_GPIO_AD_30_WDOG2_RESET_B_DEB
IOMUXC_GPIO_AD_30_GPIO9_IO29
IOMUXC_GPIO_AD_31_LPSPI1_SIN
IOMUXC_GPIO_AD_31_USB_OTG2_PWR
IOMUXC_GPIO_AD_31_FLEXCAN2_RX
IOMUXC_GPIO_AD_31_ENET_TX_DATA01
IOMUXC_GPIO_AD_31_LPUART3_RXD
IOMUXC_GPIO_AD_31_GPIO_MUX3_IO30
IOMUXC_GPIO_AD_31_KPP_COL02
IOMUXC_GPIO_AD_31_FLEXIO2_D31
IOMUXC_GPIO_AD_31_WDOG1_RESET_B_DEB
IOMUXC_GPIO_AD_31_GPIO9_IO30
IOMUXC_GPIO_AD_32_GPIO9_IO31
IOMUXC_GPIO_AD_32_LPI2C1_SCL
IOMUXC_GPIO_AD_32_USBPHY2_OTG_ID
IOMUXC_GPIO_AD_32_PGMC_PMIC_RDY
IOMUXC_GPIO_AD_32_ENET_MDC
IOMUXC_GPIO_AD_32_USDHC1_CD_B
IOMUXC_GPIO_AD_32_GPIO_MUX3_IO31
IOMUXC_GPIO_AD_32_KPP_ROW01
IOMUXC_GPIO_AD_32_LPUART10_TXD
IOMUXC_GPIO_AD_32_ENET_1G_MDC
IOMUXC_GPIO_AD_33_LPI2C1_SDA
IOMUXC_GPIO_AD_33_USBPHY1_OTG_ID
IOMUXC_GPIO_AD_33_XBAR1_INOUT17
IOMUXC_GPIO_AD_33_ENET_MDIO
IOMUXC_GPIO_AD_33_USDHC1_WP
IOMUXC_GPIO_AD_33_GPIO_MUX4_IO00
IOMUXC_GPIO_AD_33_KPP_COL01

IOMUXC_GPIO_AD_33_LPUART10_RXD
IOMUXC_GPIO_AD_33_ENET_1G_MDIO
IOMUXC_GPIO_AD_33_GPIO10_IO00
IOMUXC_GPIO_AD_34_ENET_1G_1588_EVENT0_IN
IOMUXC_GPIO_AD_34_USB_OTG1_PWR
IOMUXC_GPIO_AD_34_XBAR1_INOUT18
IOMUXC_GPIO_AD_34_ENET_1588_EVENT0_IN
IOMUXC_GPIO_AD_34_USDHC1_VSELECT
IOMUXC_GPIO_AD_34_GPIO_MUX4_IO01
IOMUXC_GPIO_AD_34_KPP_ROW00
IOMUXC_GPIO_AD_34_LPUART10_CTS_B
IOMUXC_GPIO_AD_34_WDOG1_ANY
IOMUXC_GPIO_AD_34_GPIO10_IO01
IOMUXC_GPIO_AD_35_GPIO10_IO02
IOMUXC_GPIO_AD_35_ENET_1G_1588_EVENT0_OUT
IOMUXC_GPIO_AD_35_USB_OTG1_OC
IOMUXC_GPIO_AD_35_XBAR1_INOUT19
IOMUXC_GPIO_AD_35_ENET_1588_EVENT0_OUT
IOMUXC_GPIO_AD_35_USDHC1_RESET_B
IOMUXC_GPIO_AD_35_GPIO_MUX4_IO02
IOMUXC_GPIO_AD_35_KPP_COL00
IOMUXC_GPIO_AD_35_LPUART10_RTS_B
IOMUXC_GPIO_AD_35_FLEXSPI1_B_SS1_B
IOMUXC_GPIO_SD_B1_00_USDHC1_CMD
IOMUXC_GPIO_SD_B1_00_XBAR1_INOUT20
IOMUXC_GPIO_SD_B1_00_GPT4_CAPTURE1
IOMUXC_GPIO_SD_B1_00_GPIO_MUX4_IO03
IOMUXC_GPIO_SD_B1_00_FLEXSPI2_A_SS0_B
IOMUXC_GPIO_SD_B1_00_KPP_ROW07
IOMUXC_GPIO_SD_B1_00_GPIO10_IO03
IOMUXC_GPIO_SD_B1_01_USDHC1_CLK
IOMUXC_GPIO_SD_B1_01_XBAR1_INOUT21
IOMUXC_GPIO_SD_B1_01_GPT4_CAPTURE2

IOMUXC_GPIO_SD_B1_01_GPIO_MUX4_IO04
IOMUXC_GPIO_SD_B1_01_FLEXSPI2_A_SCLK
IOMUXC_GPIO_SD_B1_01_KPP_COL07
IOMUXC_GPIO_SD_B1_01_GPIO10_IO04
IOMUXC_GPIO_SD_B1_02_GPIO10_IO05
IOMUXC_GPIO_SD_B1_02_USDHC1_DATA0
IOMUXC_GPIO_SD_B1_02_XBAR1_INOUT22
IOMUXC_GPIO_SD_B1_02_GPT4_COMPARE1
IOMUXC_GPIO_SD_B1_02_GPIO_MUX4_IO05
IOMUXC_GPIO_SD_B1_02_FLEXSPI2_A_DATA00
IOMUXC_GPIO_SD_B1_02_KPP_ROW06
IOMUXC_GPIO_SD_B1_02_FLEXSPI1_A_SS1_B
IOMUXC_GPIO_SD_B1_03_USDHC1_DATA1
IOMUXC_GPIO_SD_B1_03_XBAR1_INOUT23
IOMUXC_GPIO_SD_B1_03_GPT4_COMPARE2
IOMUXC_GPIO_SD_B1_03_GPIO_MUX4_IO06
IOMUXC_GPIO_SD_B1_03_FLEXSPI2_A_DATA01
IOMUXC_GPIO_SD_B1_03_KPP_COL06
IOMUXC_GPIO_SD_B1_03_FLEXSPI1_B_SS1_B
IOMUXC_GPIO_SD_B1_03_GPIO10_IO06
IOMUXC_GPIO_SD_B1_04_USDHC1_DATA2
IOMUXC_GPIO_SD_B1_04_XBAR1_INOUT24
IOMUXC_GPIO_SD_B1_04_GPT4_COMPARE3
IOMUXC_GPIO_SD_B1_04_GPIO_MUX4_IO07
IOMUXC_GPIO_SD_B1_04_FLEXSPI2_A_DATA02
IOMUXC_GPIO_SD_B1_04_FLEXSPI1_B_SS0_B
IOMUXC_GPIO_SD_B1_04_ENET_QOS_1588_EVENT2_AUX_IN
IOMUXC_GPIO_SD_B1_04_GPIO10_IO07
IOMUXC_GPIO_SD_B1_05_GPIO10_IO08
IOMUXC_GPIO_SD_B1_05_USDHC1_DATA3
IOMUXC_GPIO_SD_B1_05_XBAR1_INOUT25
IOMUXC_GPIO_SD_B1_05_GPT4_CLK
IOMUXC_GPIO_SD_B1_05_GPIO_MUX4_IO08

IOMUXC_GPIO_SD_B1_05_FLEXSPI2_A_DATA03
IOMUXC_GPIO_SD_B1_05_FLEXSPI1_B_DQS
IOMUXC_GPIO_SD_B1_05_ENET_QOS_1588_EVENT3_AUX_IN
IOMUXC_GPIO_SD_B2_00_GPIO10_IO09
IOMUXC_GPIO_SD_B2_00_USDHC2_DATA3
IOMUXC_GPIO_SD_B2_00_FLEXSPI1_B_DATA03
IOMUXC_GPIO_SD_B2_00_ENET_1G_RX_EN
IOMUXC_GPIO_SD_B2_00_LPUART9_TXD
IOMUXC_GPIO_SD_B2_00_LPSPI4_SCK
IOMUXC_GPIO_SD_B2_00_GPIO_MUX4_IO09
IOMUXC_GPIO_SD_B2_01_USDHC2_DATA2
IOMUXC_GPIO_SD_B2_01_FLEXSPI1_B_DATA02
IOMUXC_GPIO_SD_B2_01_ENET_1G_RX_CLK
IOMUXC_GPIO_SD_B2_01_LPUART9_RXD
IOMUXC_GPIO_SD_B2_01_LPSPI4_PCS0
IOMUXC_GPIO_SD_B2_01_GPIO_MUX4_IO10
IOMUXC_GPIO_SD_B2_01_GPIO10_IO10
IOMUXC_GPIO_SD_B2_02_GPIO10_IO11
IOMUXC_GPIO_SD_B2_02_USDHC2_DATA1
IOMUXC_GPIO_SD_B2_02_FLEXSPI1_B_DATA01
IOMUXC_GPIO_SD_B2_02_ENET_1G_RX_DATA00
IOMUXC_GPIO_SD_B2_02_LPUART9_CTS_B
IOMUXC_GPIO_SD_B2_02_LPSPI4_SOUT
IOMUXC_GPIO_SD_B2_02_GPIO_MUX4_IO11
IOMUXC_GPIO_SD_B2_03_GPIO10_IO12
IOMUXC_GPIO_SD_B2_03_USDHC2_DATA0
IOMUXC_GPIO_SD_B2_03_FLEXSPI1_B_DATA00
IOMUXC_GPIO_SD_B2_03_ENET_1G_RX_DATA01
IOMUXC_GPIO_SD_B2_03_LPUART9_RTS_B
IOMUXC_GPIO_SD_B2_03_LPSPI4_SIN
IOMUXC_GPIO_SD_B2_03_GPIO_MUX4_IO12
IOMUXC_GPIO_SD_B2_04_USDHC2_CLK
IOMUXC_GPIO_SD_B2_04_FLEXSPI1_B_SCLK

IOMUXC_GPIO_SD_B2_04_ENET_1G_RX_DATA02
IOMUXC_GPIO_SD_B2_04_FLEXSPI1_A_SS1_B
IOMUXC_GPIO_SD_B2_04_LPSPI4_PCS1
IOMUXC_GPIO_SD_B2_04_GPIO_MUX4_IO13
IOMUXC_GPIO_SD_B2_04_GPIO10_IO13
IOMUXC_GPIO_SD_B2_05_GPIO10_IO14
IOMUXC_GPIO_SD_B2_05_USDHC2_CMD
IOMUXC_GPIO_SD_B2_05_FLEXSPI1_A_DQS
IOMUXC_GPIO_SD_B2_05_ENET_1G_RX_DATA03
IOMUXC_GPIO_SD_B2_05_FLEXSPI1_B_SS0_B
IOMUXC_GPIO_SD_B2_05_LPSPI4_PCS2
IOMUXC_GPIO_SD_B2_05_GPIO_MUX4_IO14
IOMUXC_GPIO_SD_B2_06_GPIO10_IO15
IOMUXC_GPIO_SD_B2_06_USDHC2_RESET_B
IOMUXC_GPIO_SD_B2_06_FLEXSPI1_A_SS0_B
IOMUXC_GPIO_SD_B2_06_ENET_1G_TX_DATA03
IOMUXC_GPIO_SD_B2_06_LPSPI4_PCS3
IOMUXC_GPIO_SD_B2_06_GPT6_CAPTURE1
IOMUXC_GPIO_SD_B2_06_GPIO_MUX4_IO15
IOMUXC_GPIO_SD_B2_07_USDHC2_STROBE
IOMUXC_GPIO_SD_B2_07_FLEXSPI1_A_SCLK
IOMUXC_GPIO_SD_B2_07_ENET_1G_TX_DATA02
IOMUXC_GPIO_SD_B2_07_LPUART3_CTS_B
IOMUXC_GPIO_SD_B2_07_GPT6_CAPTURE2
IOMUXC_GPIO_SD_B2_07_GPIO_MUX4_IO16
IOMUXC_GPIO_SD_B2_07_LPSPI2_SCK
IOMUXC_GPIO_SD_B2_07_ENET_TX_ER
IOMUXC_GPIO_SD_B2_07_ENET_QOS_REF_CLK
IOMUXC_GPIO_SD_B2_07_GPIO10_IO16
IOMUXC_GPIO_SD_B2_08_GPIO10_IO17
IOMUXC_GPIO_SD_B2_08_USDHC2_DATA4
IOMUXC_GPIO_SD_B2_08_FLEXSPI1_A_DATA00
IOMUXC_GPIO_SD_B2_08_ENET_1G_TX_DATA01

IOMUXC_GPIO_SD_B2_08_LPUART3_RTS_B
IOMUXC_GPIO_SD_B2_08_GPT6_COMPARE1
IOMUXC_GPIO_SD_B2_08_GPIO_MUX4_IO17
IOMUXC_GPIO_SD_B2_08_LPSPI2_PCS0
IOMUXC_GPIO_SD_B2_09_GPIO10_IO18
IOMUXC_GPIO_SD_B2_09_USDHC2_DATA5
IOMUXC_GPIO_SD_B2_09_FLEXSPI1_A_DATA01
IOMUXC_GPIO_SD_B2_09_ENET_1G_TX_DATA00
IOMUXC_GPIO_SD_B2_09_LPUART5_CTS_B
IOMUXC_GPIO_SD_B2_09_GPT6_COMPARE2
IOMUXC_GPIO_SD_B2_09_GPIO_MUX4_IO18
IOMUXC_GPIO_SD_B2_09_LPSPI2_SOUT
IOMUXC_GPIO_SD_B2_10_GPIO10_IO19
IOMUXC_GPIO_SD_B2_10_USDHC2_DATA6
IOMUXC_GPIO_SD_B2_10_FLEXSPI1_A_DATA02
IOMUXC_GPIO_SD_B2_10_ENET_1G_TX_EN
IOMUXC_GPIO_SD_B2_10_LPUART5_RTS_B
IOMUXC_GPIO_SD_B2_10_GPT6_COMPARE3
IOMUXC_GPIO_SD_B2_10_GPIO_MUX4_IO19
IOMUXC_GPIO_SD_B2_10_LPSPI2_SIN
IOMUXC_GPIO_SD_B2_11_USDHC2_DATA7
IOMUXC_GPIO_SD_B2_11_FLEXSPI1_A_DATA03
IOMUXC_GPIO_SD_B2_11_ENET_1G_TX_CLK_IO
IOMUXC_GPIO_SD_B2_11_ENET_1G_REF_CLK
IOMUXC_GPIO_SD_B2_11_GPT6_CLK
IOMUXC_GPIO_SD_B2_11_GPIO_MUX4_IO20
IOMUXC_GPIO_SD_B2_11_LPSPI2_PCS1
IOMUXC_GPIO_SD_B2_11_GPIO10_IO20
IOMUXC_GPIO_DISP_B1_00_VIDEO_MUX_LCDIF_CLK
IOMUXC_GPIO_DISP_B1_00_ENET_1G_RX_EN
IOMUXC_GPIO_DISP_B1_00_TMR1_TIMER0
IOMUXC_GPIO_DISP_B1_00_XBAR1_INOUT26
IOMUXC_GPIO_DISP_B1_00_GPIO_MUX4_IO21

IOMUXC_GPIO_DISP_B1_00_ENET_QOS_RX_EN
IOMUXC_GPIO_DISP_B1_00_GPIO10_IO21
IOMUXC_GPIO_DISP_B1_01_VIDEO_MUX_LCDIF_ENABLE
IOMUXC_GPIO_DISP_B1_01_ENET_1G_RX_CLK
IOMUXC_GPIO_DISP_B1_01_ENET_1G_RX_ER
IOMUXC_GPIO_DISP_B1_01_TMR1_TIMER1
IOMUXC_GPIO_DISP_B1_01_XBAR1_INOUT27
IOMUXC_GPIO_DISP_B1_01_GPIO_MUX4_IO22
IOMUXC_GPIO_DISP_B1_01_ENET_QOS_RX_CLK
IOMUXC_GPIO_DISP_B1_01_ENET_QOS_RX_ER
IOMUXC_GPIO_DISP_B1_01_GPIO10_IO22
IOMUXC_GPIO_DISP_B1_02_GPIO10_IO23
IOMUXC_GPIO_DISP_B1_02_VIDEO_MUX_LCDIF_HSYNC
IOMUXC_GPIO_DISP_B1_02_ENET_1G_RX_DATA00
IOMUXC_GPIO_DISP_B1_02_LPI2C3_SCL
IOMUXC_GPIO_DISP_B1_02_TMR1_TIMER2
IOMUXC_GPIO_DISP_B1_02_XBAR1_INOUT28
IOMUXC_GPIO_DISP_B1_02_GPIO_MUX4_IO23
IOMUXC_GPIO_DISP_B1_02_ENET_QOS_RX_DATA00
IOMUXC_GPIO_DISP_B1_02_LPUART1_TXD
IOMUXC_GPIO_DISP_B1_03_VIDEO_MUX_LCDIF_VSYNC
IOMUXC_GPIO_DISP_B1_03_ENET_1G_RX_DATA01
IOMUXC_GPIO_DISP_B1_03_LPI2C3_SDA
IOMUXC_GPIO_DISP_B1_03_TMR2_TIMER0
IOMUXC_GPIO_DISP_B1_03_XBAR1_INOUT29
IOMUXC_GPIO_DISP_B1_03_GPIO_MUX4_IO24
IOMUXC_GPIO_DISP_B1_03_ENET_QOS_RX_DATA01
IOMUXC_GPIO_DISP_B1_03_LPUART1_RXD
IOMUXC_GPIO_DISP_B1_03_GPIO10_IO24
IOMUXC_GPIO_DISP_B1_04_VIDEO_MUX_LCDIF_DATA00
IOMUXC_GPIO_DISP_B1_04_ENET_1G_RX_DATA02
IOMUXC_GPIO_DISP_B1_04_LPUART4_RXD
IOMUXC_GPIO_DISP_B1_04_TMR2_TIMER1

IOMUXC_GPIO_DISP_B1_04_XBAR1_INOUT30
IOMUXC_GPIO_DISP_B1_04_GPIO_MUX4_IO25
IOMUXC_GPIO_DISP_B1_04_ENET_QOS_RX_DATA02
IOMUXC_GPIO_DISP_B1_04_LPSPI3_SCK
IOMUXC_GPIO_DISP_B1_04_GPIO10_IO25
IOMUXC_GPIO_DISP_B1_05_GPIO10_IO26
IOMUXC_GPIO_DISP_B1_05_VIDEO_MUX_LCDIF_DATA01
IOMUXC_GPIO_DISP_B1_05_ENET_1G_RX_DATA03
IOMUXC_GPIO_DISP_B1_05_LPUART4_CTS_B
IOMUXC_GPIO_DISP_B1_05_TMR2_TIMER2
IOMUXC_GPIO_DISP_B1_05_XBAR1_INOUT31
IOMUXC_GPIO_DISP_B1_05_GPIO_MUX4_IO26
IOMUXC_GPIO_DISP_B1_05_ENET_QOS_RX_DATA03
IOMUXC_GPIO_DISP_B1_05_LPSPI3_SIN
IOMUXC_GPIO_DISP_B1_06_VIDEO_MUX_LCDIF_DATA02
IOMUXC_GPIO_DISP_B1_06_ENET_1G_TX_DATA03
IOMUXC_GPIO_DISP_B1_06_LPUART4_TXD
IOMUXC_GPIO_DISP_B1_06_TMR3_TIMER0
IOMUXC_GPIO_DISP_B1_06_XBAR1_INOUT32
IOMUXC_GPIO_DISP_B1_06_GPIO_MUX4_IO27
IOMUXC_GPIO_DISP_B1_06_SRC_BT_CFG00
IOMUXC_GPIO_DISP_B1_06_ENET_QOS_TX_DATA03
IOMUXC_GPIO_DISP_B1_06_LPSPI3_SOUT
IOMUXC_GPIO_DISP_B1_06_GPIO10_IO27
IOMUXC_GPIO_DISP_B1_07_VIDEO_MUX_LCDIF_DATA03
IOMUXC_GPIO_DISP_B1_07_ENET_1G_TX_DATA02
IOMUXC_GPIO_DISP_B1_07_LPUART4_RTS_B
IOMUXC_GPIO_DISP_B1_07_TMR3_TIMER1
IOMUXC_GPIO_DISP_B1_07_XBAR1_INOUT33
IOMUXC_GPIO_DISP_B1_07_GPIO_MUX4_IO28
IOMUXC_GPIO_DISP_B1_07_SRC_BT_CFG01
IOMUXC_GPIO_DISP_B1_07_ENET_QOS_TX_DATA02
IOMUXC_GPIO_DISP_B1_07_LPSPI3_PCS0

IOMUXC_GPIO_DISP_B1_07_GPIO10_IO28
IOMUXC_GPIO_DISP_B1_08_GPIO10_IO29
IOMUXC_GPIO_DISP_B1_08_VIDEO_MUX_LCDIF_DATA04
IOMUXC_GPIO_DISP_B1_08_ENET_1G_TX_DATA01
IOMUXC_GPIO_DISP_B1_08_USDHC1_CD_B
IOMUXC_GPIO_DISP_B1_08_TMR3_TIMER2
IOMUXC_GPIO_DISP_B1_08_XBAR1_INOUT34
IOMUXC_GPIO_DISP_B1_08_GPIO_MUX4_IO29
IOMUXC_GPIO_DISP_B1_08_SRC_BT_CFG02
IOMUXC_GPIO_DISP_B1_08_ENET_QOS_TX_DATA01
IOMUXC_GPIO_DISP_B1_08_LPSPI3_PCS1
IOMUXC_GPIO_DISP_B1_09_VIDEO_MUX_LCDIF_DATA05
IOMUXC_GPIO_DISP_B1_09_ENET_1G_TX_DATA00
IOMUXC_GPIO_DISP_B1_09_USDHC1_WP
IOMUXC_GPIO_DISP_B1_09_TMR4_TIMER0
IOMUXC_GPIO_DISP_B1_09_XBAR1_INOUT35
IOMUXC_GPIO_DISP_B1_09_GPIO_MUX4_IO30
IOMUXC_GPIO_DISP_B1_09_SRC_BT_CFG03
IOMUXC_GPIO_DISP_B1_09_ENET_QOS_TX_DATA00
IOMUXC_GPIO_DISP_B1_09_LPSPI3_PCS2
IOMUXC_GPIO_DISP_B1_09_GPIO10_IO30
IOMUXC_GPIO_DISP_B1_10_VIDEO_MUX_LCDIF_DATA06
IOMUXC_GPIO_DISP_B1_10_ENET_1G_TX_EN
IOMUXC_GPIO_DISP_B1_10_USDHC1_RESET_B
IOMUXC_GPIO_DISP_B1_10_TMR4_TIMER1
IOMUXC_GPIO_DISP_B1_10_XBAR1_INOUT36
IOMUXC_GPIO_DISP_B1_10_GPIO_MUX4_IO31
IOMUXC_GPIO_DISP_B1_10_SRC_BT_CFG04
IOMUXC_GPIO_DISP_B1_10_ENET_QOS_TX_EN
IOMUXC_GPIO_DISP_B1_10_LPSPI3_PCS3
IOMUXC_GPIO_DISP_B1_10_GPIO10_IO31
IOMUXC_GPIO_DISP_B1_11_VIDEO_MUX_LCDIF_DATA07
IOMUXC_GPIO_DISP_B1_11_ENET_1G_TX_CLK_IO

IOMUXC_GPIO_DISP_B1_11_ENET_1G_REF_CLK
IOMUXC_GPIO_DISP_B1_11_TMR4_TIMER2
IOMUXC_GPIO_DISP_B1_11_XBAR1_INOUT37
IOMUXC_GPIO_DISP_B1_11_GPIO_MUX5_IO00
IOMUXC_GPIO_DISP_B1_11_SRC_BT_CFG05
IOMUXC_GPIO_DISP_B1_11_ENET_QOS_TX_CLK
IOMUXC_GPIO_DISP_B1_11_ENET_QOS_REF_CLK
IOMUXC_GPIO_DISP_B1_11_GPIO11_IO00
IOMUXC_GPIO_DISP_B2_00_GPIO11_IO01
IOMUXC_GPIO_DISP_B2_00_VIDEO_MUX_LCDIF_DATA08
IOMUXC_GPIO_DISP_B2_00_WDOG1_B
IOMUXC_GPIO_DISP_B2_00_MQS_RIGHT
IOMUXC_GPIO_DISP_B2_00_ENET_1G_TX_ER
IOMUXC_GPIO_DISP_B2_00_SAI1_TX_DATA03
IOMUXC_GPIO_DISP_B2_00_GPIO_MUX5_IO01
IOMUXC_GPIO_DISP_B2_00_SRC_BT_CFG06
IOMUXC_GPIO_DISP_B2_00_ENET_QOS_TX_ER
IOMUXC_GPIO_DISP_B2_01_VIDEO_MUX_LCDIF_DATA09
IOMUXC_GPIO_DISP_B2_01_USDHC1_VSELECT
IOMUXC_GPIO_DISP_B2_01_MQS_LEFT
IOMUXC_GPIO_DISP_B2_01_WDOG2_B
IOMUXC_GPIO_DISP_B2_01_SAI1_TX_DATA02
IOMUXC_GPIO_DISP_B2_01_GPIO_MUX5_IO02
IOMUXC_GPIO_DISP_B2_01_SRC_BT_CFG07
IOMUXC_GPIO_DISP_B2_01_EWM_OUT_B
IOMUXC_GPIO_DISP_B2_01_CCM_ENET_REF_CLK_25M
IOMUXC_GPIO_DISP_B2_01_GPIO11_IO02
IOMUXC_GPIO_DISP_B2_02_GPIO11_IO03
IOMUXC_GPIO_DISP_B2_02_VIDEO_MUX_LCDIF_DATA10
IOMUXC_GPIO_DISP_B2_02_ENET_TX_DATA00
IOMUXC_GPIO_DISP_B2_02_PIT1_TRIGGER3
IOMUXC_GPIO_DISP_B2_02_ARM_TRACE00
IOMUXC_GPIO_DISP_B2_02_SAI1_TX_DATA01

IOMUXC_GPIO_DISP_B2_02_GPIO_MUX5_IO03
IOMUXC_GPIO_DISP_B2_02_SRC_BT_CFG08
IOMUXC_GPIO_DISP_B2_02_ENET_QOS_TX_DATA00
IOMUXC_GPIO_DISP_B2_03_GPIO11_IO04
IOMUXC_GPIO_DISP_B2_03_VIDEO_MUX_LCDIF_DATA11
IOMUXC_GPIO_DISP_B2_03_ENET_TX_DATA01
IOMUXC_GPIO_DISP_B2_03_PIT1_TRIGGER2
IOMUXC_GPIO_DISP_B2_03_ARM_TRACE01
IOMUXC_GPIO_DISP_B2_03_SAI1_MCLK
IOMUXC_GPIO_DISP_B2_03_GPIO_MUX5_IO04
IOMUXC_GPIO_DISP_B2_03_SRC_BT_CFG09
IOMUXC_GPIO_DISP_B2_03_ENET_QOS_TX_DATA01
IOMUXC_GPIO_DISP_B2_04_VIDEO_MUX_LCDIF_DATA12
IOMUXC_GPIO_DISP_B2_04_ENET_TX_EN
IOMUXC_GPIO_DISP_B2_04_PIT1_TRIGGER1
IOMUXC_GPIO_DISP_B2_04_ARM_TRACE02
IOMUXC_GPIO_DISP_B2_04_SAI1_RX_SYNC
IOMUXC_GPIO_DISP_B2_04_GPIO_MUX5_IO05
IOMUXC_GPIO_DISP_B2_04_SRC_BT_CFG10
IOMUXC_GPIO_DISP_B2_04_ENET_QOS_TX_EN
IOMUXC_GPIO_DISP_B2_04_GPIO11_IO05
IOMUXC_GPIO_DISP_B2_05_GPIO11_IO06
IOMUXC_GPIO_DISP_B2_05_VIDEO_MUX_LCDIF_DATA13
IOMUXC_GPIO_DISP_B2_05_ENET_TX_CLK
IOMUXC_GPIO_DISP_B2_05_ENET_REF_CLK
IOMUXC_GPIO_DISP_B2_05_ARM_TRACE03
IOMUXC_GPIO_DISP_B2_05_SAI1_RX_BCLK
IOMUXC_GPIO_DISP_B2_05_GPIO_MUX5_IO06
IOMUXC_GPIO_DISP_B2_05_SRC_BT_CFG11
IOMUXC_GPIO_DISP_B2_05_ENET_QOS_TX_CLK
IOMUXC_GPIO_DISP_B2_06_GPIO11_IO07
IOMUXC_GPIO_DISP_B2_06_VIDEO_MUX_LCDIF_DATA14
IOMUXC_GPIO_DISP_B2_06_ENET_RX_DATA00

IOMUXC_GPIO_DISP_B2_06_LPUART7_TXD
IOMUXC_GPIO_DISP_B2_06_ARM_TRACE_CLK
IOMUXC_GPIO_DISP_B2_06_SAI1_RX_DATA00
IOMUXC_GPIO_DISP_B2_06_GPIO_MUX5_IO07
IOMUXC_GPIO_DISP_B2_06_ENET_QOS_RX_DATA00
IOMUXC_GPIO_DISP_B2_07_VIDEO_MUX_LCDIF_DATA15
IOMUXC_GPIO_DISP_B2_07_ENET_RX_DATA01
IOMUXC_GPIO_DISP_B2_07_LPUART7_RXD
IOMUXC_GPIO_DISP_B2_07_ARM_TRACE_SWO
IOMUXC_GPIO_DISP_B2_07_SAI1_TX_DATA00
IOMUXC_GPIO_DISP_B2_07_GPIO_MUX5_IO08
IOMUXC_GPIO_DISP_B2_07_ENET_QOS_RX_DATA01
IOMUXC_GPIO_DISP_B2_07_GPIO11_IO08
IOMUXC_GPIO_DISP_B2_08_GPIO11_IO09
IOMUXC_GPIO_DISP_B2_08_VIDEO_MUX_LCDIF_DATA16
IOMUXC_GPIO_DISP_B2_08_ENET_RX_EN
IOMUXC_GPIO_DISP_B2_08_LPUART8_TXD
IOMUXC_GPIO_DISP_B2_08_ARM_CM7_EVENTO
IOMUXC_GPIO_DISP_B2_08_SAI1_TX_BCLK
IOMUXC_GPIO_DISP_B2_08_GPIO_MUX5_IO09
IOMUXC_GPIO_DISP_B2_08_ENET_QOS_RX_EN
IOMUXC_GPIO_DISP_B2_08_LPUART1_TXD
IOMUXC_GPIO_DISP_B2_09_GPIO11_IO10
IOMUXC_GPIO_DISP_B2_09_VIDEO_MUX_LCDIF_DATA17
IOMUXC_GPIO_DISP_B2_09_ENET_RX_ER
IOMUXC_GPIO_DISP_B2_09_LPUART8_RXD
IOMUXC_GPIO_DISP_B2_09_ARM_CM7_EVENTI
IOMUXC_GPIO_DISP_B2_09_SAI1_TX_SYNC
IOMUXC_GPIO_DISP_B2_09_GPIO_MUX5_IO10
IOMUXC_GPIO_DISP_B2_09_ENET_QOS_RX_ER
IOMUXC_GPIO_DISP_B2_09_LPUART1_RXD
IOMUXC_GPIO_DISP_B2_10_GPIO11_IO11
IOMUXC_GPIO_DISP_B2_10_VIDEO_MUX_LCDIF_DATA18

IOMUXC_GPIO_DISP_B2_10_EMVSIM2_IO
IOMUXC_GPIO_DISP_B2_10_LPUART2_TXD
IOMUXC_GPIO_DISP_B2_10_WDOG2_RESET_B_DEB
IOMUXC_GPIO_DISP_B2_10_XBAR1_INOUT38
IOMUXC_GPIO_DISP_B2_10_GPIO_MUX5_IO11
IOMUXC_GPIO_DISP_B2_10_LPI2C3_SCL
IOMUXC_GPIO_DISP_B2_10_ENET_QOS_RX_ER
IOMUXC_GPIO_DISP_B2_10_SPDIF_IN
IOMUXC_GPIO_DISP_B2_11_VIDEO_MUX_LCDIF_DATA19
IOMUXC_GPIO_DISP_B2_11_EMVSIM2_CLK
IOMUXC_GPIO_DISP_B2_11_LPUART2_RXD
IOMUXC_GPIO_DISP_B2_11_WDOG1_RESET_B_DEB
IOMUXC_GPIO_DISP_B2_11_XBAR1_INOUT39
IOMUXC_GPIO_DISP_B2_11_GPIO_MUX5_IO12
IOMUXC_GPIO_DISP_B2_11_LPI2C3_SDA
IOMUXC_GPIO_DISP_B2_11_ENET_QOS_CRS
IOMUXC_GPIO_DISP_B2_11_SPDIF_OUT
IOMUXC_GPIO_DISP_B2_11_GPIO11_IO12
IOMUXC_GPIO_DISP_B2_12_GPIO11_IO13
IOMUXC_GPIO_DISP_B2_12_VIDEO_MUX_LCDIF_DATA20
IOMUXC_GPIO_DISP_B2_12_EMVSIM2_RST
IOMUXC_GPIO_DISP_B2_12_FLEXCAN1_TX
IOMUXC_GPIO_DISP_B2_12_LPUART2_CTS_B
IOMUXC_GPIO_DISP_B2_12_XBAR1_INOUT40
IOMUXC_GPIO_DISP_B2_12_GPIO_MUX5_IO13
IOMUXC_GPIO_DISP_B2_12_LPI2C4_SCL
IOMUXC_GPIO_DISP_B2_12_ENET_QOS_COL
IOMUXC_GPIO_DISP_B2_12_LPSPI4_SCK
IOMUXC_GPIO_DISP_B2_13_GPIO11_IO14
IOMUXC_GPIO_DISP_B2_13_VIDEO_MUX_LCDIF_DATA21
IOMUXC_GPIO_DISP_B2_13_EMVSIM2_SVEN
IOMUXC_GPIO_DISP_B2_13_FLEXCAN1_RX
IOMUXC_GPIO_DISP_B2_13_LPUART2_RTS_B

IOMUXC_GPIO_DISP_B2_13_ENET_REF_CLK
IOMUXC_GPIO_DISP_B2_13_GPIO_MUX5_IO14
IOMUXC_GPIO_DISP_B2_13_LPI2C4_SDA
IOMUXC_GPIO_DISP_B2_13_ENET_QOS_1588_EVENT0_OUT
IOMUXC_GPIO_DISP_B2_13_LPSP14_SIN
IOMUXC_GPIO_DISP_B2_14_GPIO_MUX5_IO15
IOMUXC_GPIO_DISP_B2_14_FLEXCAN1_TX
IOMUXC_GPIO_DISP_B2_14_ENET_QOS_1588_EVENT0_IN
IOMUXC_GPIO_DISP_B2_14_LPSP14_SOUT
IOMUXC_GPIO_DISP_B2_14_GPIO11_IO15
IOMUXC_GPIO_DISP_B2_14_VIDEO_MUX_LCDIF_DATA22
IOMUXC_GPIO_DISP_B2_14_EMVSIM2_PD
IOMUXC_GPIO_DISP_B2_14_WDOG2_B
IOMUXC_GPIO_DISP_B2_14_VIDEO_MUX_EXT_DCIC1
IOMUXC_GPIO_DISP_B2_14_ENET_1G_REF_CLK
IOMUXC_GPIO_DISP_B2_15_VIDEO_MUX_LCDIF_DATA23
IOMUXC_GPIO_DISP_B2_15_EMVSIM2_POWER_FAIL
IOMUXC_GPIO_DISP_B2_15_WDOG1_B
IOMUXC_GPIO_DISP_B2_15_VIDEO_MUX_EXT_DCIC2
IOMUXC_GPIO_DISP_B2_15_PIT1_TRIGGER0
IOMUXC_GPIO_DISP_B2_15_GPIO_MUX5_IO16
IOMUXC_GPIO_DISP_B2_15_FLEXCAN1_RX
IOMUXC_GPIO_DISP_B2_15_ENET_QOS_1588_EVENT0_AUX_IN
IOMUXC_GPIO_DISP_B2_15_LPSP14_PCS0
IOMUXC_GPIO_DISP_B2_15_GPIO11_IO16
IOMUXC_GPR_SAIMCLK_LOWBITMASK
IOMUXC_GPR_SAIMCLK_HIGHBITMASK

```
static inline void IOMUXC_SetPinMux(uint32_t muxRegister, uint32_t muxMode, uint32_t  
                                inputRegister, uint32_t inputDaisy, uint32_t  
                                configRegister, uint32_t inputOnfield)
```

Sets the IOMUXC pin mux mode.

This is an example to set the PTA6 as the lpuart0_tx:

```
IOMUXC_SetPinMux(IOMUXC_PTA6_LPUART0_TX, 0);
```

This is an example to set the PTA0 as GPIOA0:

```
IOMUXC_SetPinMux(IOMUXC_PTA0_GPIOA0, 0);
```

Note: The first five parameters can be filled with the pin function ID macros.

Parameters

- muxRegister – The pin mux register.
- muxMode – The pin mux mode.
- inputRegister – The select input register.
- inputDaisy – The input daisy.
- configRegister – The config register.
- inputOnfield – Software input on field.

```
static inline void IOMUXC_SetPinConfig(uint32_t muxRegister, uint32_t muxMode, uint32_t  
                                     inputRegister, uint32_t inputDaisy, uint32_t  
                                     configRegister, uint32_t configValue)
```

Sets the IOMUXC pin configuration.

This is an example to set pin configuration for IOMUXC_PTA3_LPI2C0_SCLS:

```
IOMUXC_SetPinConfig(IOMUXC_PTA3_LPI2C0_SCLS, IOMUXC_SW_PAD_CTL_PAD_PUS_  
↪ MASK | IOMUXC_SW_PAD_CTL_PAD_PUS(2U))
```

Note: The previous five parameters can be filled with the pin function ID macros.

Parameters

- muxRegister – The pin mux register.
- muxMode – The pin mux mode.
- inputRegister – The select input register.
- inputDaisy – The input daisy.
- configRegister – The config register.
- configValue – The pin config value.

```
static inline void IOMUXC_SetSaiMClkClockSource(IOMUXC_GPR_Type *base,  
                                                iomuxc_gpr_saimclk_t mclk, uint8_t clkSrc)
```

Sets IOMUXC general configuration for SAI MCLK selection.

Parameters

- base – The IOMUXC GPR base address.
- mclk – The SAI MCLK.
- clkSrc – The clock source. Take refer to register setting details for the clock source in RM.

```
static inline void IOMUXC_MQSEnterSoftwareReset(IOMUXC_GPR_Type *base, bool enable)
```

Enters or exit MQS software reset.

Parameters

- base – The IOMUXC GPR base address.

- enable – Enter or exit MQS software reset.

```
static inline void IOMUXC_MQSEnable(IOMUXC_GPR_Type *base, bool enable)
```

Enables or disables MQS.

Parameters

- base – The IOMUXC GPR base address.
- enable – Enable or disable the MQS.

```
static inline void IOMUXC_MQSConfig(IOMUXC_GPR_Type *base,
                                   iomuxc_mqs_pwm_oversample_rate_t rate, uint8_t
                                   divider)
```

Configure MQS PWM oversampling rate compared with mclk and divider ratio control for mclk from hmclk.

Parameters

- base – The IOMUXC GPR base address.
- rate – The MQS PWM oversampling rate, refer to “iomuxc_mqs_pwm_oversample_rate_t”.
- divider – The divider ratio control for mclk from hmclk. $mclk\ freq = 1 / (divider + 1) * hmclk\ freq$.

FSL_IOMUXC_DRIVER_VERSION

IOMUXC driver version 2.0.0.

FSL_COMPONENT_ID

2.61 Key_manager

FSL_KEYMGR_DRIVER_VERSION

Key Manager driver version. Version 2.0.2.

Current version: 2.0.2

Change log:

- Version 2.0.2
 - Fix MISRA-2012 issues
- Version 2.0.1
 - Fix MISRA-2012 issues
- Version 2.0.0
 - Initial version

```
enum __keymgr_lock
```

Values:

```
enumerator kKEYMGR_Unlock
```

```
enumerator kKEYMGR_Lock
```

```
enum __keymgr_allow
```

Values:

enumerator kKEYMGR_Disallow

enumerator kKEYMGR_Allow

enum _keymgr_slot

Values:

enumerator kKEYMGR_Slot0

enumerator kKEYMGR_Slot1

enumerator kKEYMGR_Slot2

enumerator kKEYMGR_Slot3

enumerator kKEYMGR_Slot4

typedef enum _keymgr_lock keymgr_lock_t

typedef enum _keymgr_allow keymgr_allow_t

typedef enum _keymgr_slot keymgr_slot_t

typedef struct _domain_slot_config domain_slot_config_t

Key Manager slot configuration structure.

status_t KEYMGR_MasterKeyControl(*KEY_MANAGER_Type* *base, *uint8_t* select, *keymgr_lock_t* lock)

Configures Master key settings.

This function configures Key Manager's setting for Master key.

Parameters

- base – Key Manager peripheral address.
- select – select source for Master key.
- lock – setting for lock Master key.

Returns

status of Master key control operation

status_t KEYMGR_OTFAD1KeyControl(*KEY_MANAGER_Type* *base, *uint8_t* select, *keymgr_lock_t* lock)

Configures OTFAD1 key settings.

This function configures Key Manager's setting for OTFAD1 key.

Parameters

- base – Key Manager peripheral address.
- select – select source for OTFAD1 key.
- lock – setting for lock OTFAD1 key.

Returns

status of OTFAD1 key control operation

status_t KEYMGR_OTFAD2KeyControl(*KEY_MANAGER_Type* *base, *uint8_t* select, *keymgr_lock_t* lock)

Configures OTFAD2 key settings.

This function configures Key Manager's setting for OTFAD2 key.

Parameters

- base – Key Manager peripheral address.

- select – select source for OTFAD2 key.
- lock – setting for lock OTFAD2 key.

Returns

status of OTFAD2 key control operation

void KEYMGR_IEEKeyReload(KEY_MANAGER_Type *base)

Restart load key signal for IEE.

This function generates Key Manager's restart signal for IEE key.

Parameters

- base – Key Manager peripheral address.

void KEYMGR_PUFKeyLock(KEY_MANAGER_Type *base, *keymgr_lock_t* lock)

Lock the key select from PUF.

This function locks selection of key for PUF.

Parameters

- base – Key Manager peripheral address.
- lock – Setting for selection of key for PUF.

status_t KEYMGR_SlotControl(KEY_MANAGER_Type *base, *domain_slot_config_t* *config,
keymgr_slot_t slot)

Configures Slot Domain control.

This function configures domain slot control which locks and allows writes.

Parameters

- base – Key Manager peripheral address.
- config – Pointer to slot configuration structure.
- slot – Select slot to be configured.

Returns

status of slot control operation

void KEYMGR_Init(KEY_MANAGER_Type *base)

Resets Key Manager module to factory default values.

This function performs hardware reset of Key Manager module.

Parameters

- base – Key Manager peripheral address.

status_t KEYMGR_GetDefaultConfig(*domain_slot_config_t* *config)

Sets the default configuration of Key manager slot.

This function initialize Key Manager slot config structure to default values.

Parameters

- config – Pointer to slot configuration structure.

KEYMGR_IEE_RELOAD

KEYMGR_SEL_OCOTP

KEYMGR_SEL_UDF

KEYMGR_SEL_PUF

keymgr_select_t

struct __domain_slot_config

#include <fsl_key_manager.h> Key Manager slot configuration structure.

Public Members

keymgr_lock_t lockControl

Lock control register of slot.

keymgr_allow_t allowUser

Allow user write access to domain control register or domain register.

keymgr_allow_t allowNonSecure

Allow non-secure write access to domain control register or domain register.

keymgr_lock_t lockList

Lock whitelist. SLOTx_CTRL[WHITE_LIST] cannot be changed.

uint8_t whiteList

Domains that on the Whitelist can change given slot. Each field represents one domain. Bit0~Bit3 represent DOMAIN0~DOMAIN3 respectively.

2.62 KPP: KeyPad Port Driver

void KPP_Init(KPP_Type *base, kpp_config_t *configure)

KPP initialize. This function ungates the KPP clock and initializes KPP. This function must be called before calling any other KPP driver functions.

Parameters

- base – KPP peripheral base address.
- configure – The KPP configuration structure pointer.

void KPP_Deinit(KPP_Type *base)

Deinitializes the KPP module and gates the clock. This function gates the KPP clock. As a result, the KPP module doesn't work after calling this function.

Parameters

- base – KPP peripheral base address.

static inline void KPP_EnableInterrupts(KPP_Type *base, uint16_t mask)

Enable the interrupt.

Parameters

- base – KPP peripheral base address.
- mask – KPP interrupts to enable. This is a logical OR of the enumeration :: kpp_interrupt_enable_t.

static inline void KPP_DisableInterrupts(KPP_Type *base, uint16_t mask)

Disable the interrupt.

Parameters

- base – KPP peripheral base address.
- mask – KPP interrupts to disable. This is a logical OR of the enumeration :: kpp_interrupt_enable_t.

```
static inline uint16_t KPP_GetStatusFlag(KPP_Type *base)
```

Gets the KPP interrupt event status.

Parameters

- base – KPP peripheral base address.

Returns

The status of the KPP. Application can use the enum type in the “kpp_interrupt_enable_t” to get the right status of the related event.

```
static inline void KPP_ClearStatusFlag(KPP_Type *base, uint16_t mask)
```

Clears KPP status flag.

Parameters

- base – KPP peripheral base address.
- mask – KPP mask to be cleared. This is a logical OR of the enumeration :: kpp_interrupt_enable_t.

```
static inline void KPP_SetSynchronizeChain(KPP_Type *base, uint16_t mask)
```

Set KPP synchronization chain.

Parameters

- base – KPP peripheral base address.
- mask – KPP mask to be cleared. This is a logical OR of the enumeration :: kpp_sync_operation_t.

```
status_t KPP_keyPressScanning(KPP_Type *base, uint8_t *data, uint32_t clockSrc_Hz)
```

Keypad press scanning.

This function will scanning all columns and rows. so all scanning data will be stored in the data pointer.

Parameters

- base – KPP peripheral base address.
- data – KPP key press scanning data. The data buffer should be prepared with length at least equal to KPP_KEYPAD_COLUMNNUM_MAX * KPP_KEYPAD_ROWNUM_MAX. the data pointer is recommended to be a array like uint8_t data[KPP_KEYPAD_COLUMNNUM_MAX]. for example the data[2] = 4, that means in column 1 row 2 has a key press event.
- clockSrc_Hz – Source clock.

Return values

kStatus_Success – kpp press scan succeed.

```
FSL_KPP_DRIVER_VERSION
```

KPP driver version.

```
enum _kpp_interrupt_enable
```

List of interrupts supported by the peripheral. This enumeration uses one-hot encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

Values:

```
enumerator kKPP_keyDepressInterrupt
```

Keypad depress interrupt source

```
enumerator kKPP_keyReleaseInterrupt
```

Keypad release interrupt source

enum `_kpp_sync_operation`

Lists of KPP synchronize chain operation.

Values:

enumerator `kKPP_ClearKeyDepressSyncChain`

Keypad depress interrupt status.

enumerator `kKPP_SetKeyReleasesSyncChain`

Keypad release interrupt status.

typedef enum `_kpp_interrupt_enable` `kpp_interrupt_enable_t`

List of interrupts supported by the peripheral. This enumeration uses one-hot encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

typedef enum `_kpp_sync_operation` `kpp_sync_operation_t`

Lists of KPP synchronize chain operation.

typedef struct `_kpp_config` `kpp_config_t`

Lists of KPP status.

`KPP_KEYPAD_COLUMNNUM_MAX`

`KPP_KEYPAD_ROWNUM_MAX`

struct `_kpp_config`

#include <fsl_kpp.h> Lists of KPP status.

Public Members

uint8_t `activeRow`

The row number: bit 7 ~ 0 represents the row 7 ~ 0.

uint8_t `activeColumn`

The column number: bit 7 ~ 0 represents the column 7 ~ 0.

uint16_t `interrupt`

KPP interrupt source. A logical OR of “`kpp_interrupt_enable_t`”.

2.63 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_UART

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 I2C status codes.

enumerator kStatusGroup_LPSCI
Group number for LPSCI status codes.

enumerator kStatusGroup_LPUART
Group number for LPUART status codes.

enumerator kStatusGroup_SPI
Group number for SPI status code.

enumerator kStatusGroup_XRDC
Group number for XRDC status code.

enumerator kStatusGroup_SEMA42
Group number for SEMA42 status code.

enumerator kStatusGroup_SDHC
Group number for SDHC status code

enumerator kStatusGroup_SDMMC
Group number for SDMMC status code

enumerator kStatusGroup_SAI
Group number for SAI status code

enumerator kStatusGroup_MCG
Group number for MCG status codes.

enumerator kStatusGroup_SCG
Group number for SCG status codes.

enumerator kStatusGroup_SDSPI
Group number for SDSPI status codes.

enumerator kStatusGroup_FLEXIO_I2S
Group number for FLEXIO I2S status codes

enumerator kStatusGroup_FLEXIO_MCULCD
Group number for FLEXIO LCD status codes

enumerator kStatusGroup_FLASHIAP
Group number for FLASHIAP status codes

enumerator kStatusGroup_FLEXCOMM_I2C
Group number for FLEXCOMM I2C status codes

enumerator kStatusGroup_I2S
Group number for I2S status codes

enumerator kStatusGroup_IUART
Group number for IUART status codes

enumerator kStatusGroup_CSI
Group number for CSI status codes

enumerator kStatusGroup_MIPI_DSI
Group number for MIPI DSI status codes

enumerator kStatusGroup_SDRAMC
Group number for SDRAMC status codes.

enumerator kStatusGroup_POWER
Group number for POWER status codes.

enumerator kStatusGroup_ENET
Group number for ENET status codes.

enumerator kStatusGroup_PHY
Group number for PHY status codes.

enumerator kStatusGroup_TRGMUX
Group number for TRGMUX status codes.

enumerator kStatusGroup_SMARTCARD
Group number for SMARTCARD status codes.

enumerator kStatusGroup_LMEM
Group number for LMEM status codes.

enumerator kStatusGroup_QSPI
Group number for QSPI status codes.

enumerator kStatusGroup_DMA
Group number for DMA status codes.

enumerator kStatusGroup_EDMA
Group number for EDMA status codes.

enumerator kStatusGroup_DMAMGR
Group number for DMAMGR status codes.

enumerator kStatusGroup_FLEXCAN
Group number for FlexCAN status codes.

enumerator kStatusGroup_LTC
Group number for LTC status codes.

enumerator kStatusGroup_FLEXIO_CAMERA
Group number for FLEXIO CAMERA status codes.

enumerator kStatusGroup_LPC_SPI
Group number for LPC_SPI status codes.

enumerator kStatusGroup_LPC_USART
Group number for LPC_USART status codes.

enumerator kStatusGroup_DMIC
Group number for DMIC status codes.

enumerator kStatusGroup_SDIF
Group number for SDIF status codes.

enumerator kStatusGroup_SPIFI
Group number for SPIFI status codes.

enumerator kStatusGroup_OTP
Group number for OTP status codes.

enumerator kStatusGroup_MCAN
Group number for MCAN status codes.

enumerator kStatusGroup_CAAM
Group number for CAAM status codes.

enumerator kStatusGroup_ECSPI
Group number for ECSPI status codes.

enumerator kStatusGroup_USDHC
Group number for USDHC status codes.

enumerator kStatusGroup_LPC_I2C
Group number for LPC_I2C status codes.

enumerator kStatusGroup_DCP
Group number for DCP status codes.

enumerator kStatusGroup_MSCAN
Group number for MSCAN status codes.

enumerator kStatusGroup_ESAI
Group number for ESAI status codes.

enumerator kStatusGroup_FLEXSPI
Group number for FLEXSPI status codes.

enumerator kStatusGroup_MMDC
Group number for MMDC status codes.

enumerator kStatusGroup_PDM
Group number for MIC status codes.

enumerator kStatusGroup_SDMA
Group number for SDMA status codes.

enumerator kStatusGroup_ICS
Group number for ICS status codes.

enumerator kStatusGroup_SPDIF
Group number for SPDIF status codes.

enumerator kStatusGroup_LPC_MINISPI
Group number for LPC_MINISPI status codes.

enumerator kStatusGroup_HASHCRYPT
Group number for Hashcrypt status codes

enumerator kStatusGroup_LPC_SPI_SSP
Group number for LPC_SPI_SSP status codes.

enumerator kStatusGroup_I3C
Group number for I3C status codes

enumerator kStatusGroup_LPC_I2C_1
Group number for LPC_I2C_1 status codes.

enumerator kStatusGroup_NOTIFIER
Group number for NOTIFIER status codes.

enumerator kStatusGroup_DebugConsole
Group number for debug console status codes.

enumerator kStatusGroup_SEMC
Group number for SEMC status codes.

enumerator kStatusGroup_ApplicationRangeStart
Starting number for application groups.

enumerator kStatusGroup_IAP
Group number for IAP status codes

enumerator kStatusGroup_SFA
Group number for SFA status codes

enumerator kStatusGroup_SPC
Group number for SPC status codes.

enumerator kStatusGroup_PUF
Group number for PUF status codes.

enumerator kStatusGroup_TOUCH_PANEL
Group number for touch panel status codes

enumerator kStatusGroup_VBAT
Group number for VBAT status codes

enumerator kStatusGroup_XSPI
Group number for XSPI status codes

enumerator kStatusGroup_PNGDEC
Group number for PNGDEC status codes

enumerator kStatusGroup_JPEGDEC
Group number for JPEGDEC status codes

enumerator kStatusGroup_AUDMIX
Group number for AUDMIX status codes

enumerator kStatusGroup_HAL_GPIO
Group number for HAL GPIO status codes.

enumerator kStatusGroup_HAL_UART
Group number for HAL UART status codes.

enumerator kStatusGroup_HAL_TIMER
Group number for HAL TIMER status codes.

enumerator kStatusGroup_HAL_SPI
Group number for HAL SPI status codes.

enumerator kStatusGroup_HAL_I2C
Group number for HAL I2C status codes.

enumerator kStatusGroup_HAL_FLASH
Group number for HAL FLASH status codes.

enumerator kStatusGroup_HAL_PWM
Group number for HAL PWM status codes.

enumerator kStatusGroup_HAL_RNG
Group number for HAL RNG status codes.

enumerator kStatusGroup_HAL_I2S
Group number for HAL I2S status codes.

enumerator kStatusGroup_HAL_ADC_SENSOR
Group number for HAL ADC SENSOR status codes.

enumerator kStatusGroup_TIMERMANAGER
Group number for TiMER MANAGER status codes.

enumerator kStatusGroup_SERIALMANAGER
Group number for SERIAL MANAGER status codes.

enumerator kStatusGroup_LED
Group number for LED status codes.

enumerator kStatusGroup_BUTTON
Group number for BUTTON status codes.

enumerator kStatusGroup_EXTERN_EEPROM
Group number for EXTERN EEPROM status codes.

enumerator kStatusGroup_SHELL
Group number for SHELL status codes.

enumerator kStatusGroup_MEM_MANAGER
Group number for MEM MANAGER status codes.

enumerator kStatusGroup_LIST
Group number for List status codes.

enumerator kStatusGroup_OSA
Group number for OSA status codes.

enumerator kStatusGroup_COMMON_TASK
Group number for Common task status codes.

enumerator kStatusGroup_MSG
Group number for messaging status codes.

enumerator kStatusGroup_SDK_OCOTP
Group number for OCOTP status codes.

enumerator kStatusGroup_SDK_FLEXSPINOR
Group number for FLEXSPINOR status codes.

enumerator kStatusGroup_CODEC
Group number for codec status codes.

enumerator kStatusGroup_ASRC
Group number for codec status ASRC.

enumerator kStatusGroup_OTFAD
Group number for codec status codes.

enumerator kStatusGroup_SDIO SLV
Group number for SDIO SLV status codes.

enumerator kStatusGroup_MECC
Group number for MECC status codes.

enumerator kStatusGroup_ENET_QOS
Group number for ENET_QOS status codes.

enumerator kStatusGroup_LOG
Group number for LOG status codes.

enumerator kStatusGroup_I3CBUS
Group number for I3CBUS status codes.

enumerator kStatusGroup_QSCI
Group number for QSCI status codes.

enumerator kStatusGroup_ELE MU
Group number for ELE MU status codes.

enumerator kStatusGroup_QUEUE DSPI
Group number for QSPI status codes.

enumerator kStatusGroup_POWER_MANAGER
Group number for POWER_MANAGER status codes.

enumerator kStatusGroup_IPED
Group number for IPED status codes.

enumerator kStatusGroup_ELS_PKC
Group number for ELS PKC status codes.

enumerator kStatusGroup_CSS_PKC
Group number for CSS PKC status codes.

enumerator kStatusGroup_HOSTIF
Group number for HOSTIF status codes.

enumerator kStatusGroup_CLIF
Group number for CLIF status codes.

enumerator kStatusGroup_BMA
Group number for BMA status codes.

enumerator kStatusGroup_NETC
Group number for NETC status codes.

enumerator kStatusGroup_ELE
Group number for ELE status codes.

enumerator kStatusGroup_GLIKEY
Group number for GLIKEY status codes.

enumerator kStatusGroup_AON_POWER

Group number for AON_POWER status codes.

enumerator kStatusGroup_AON_COMMON

Group number for AON_COMMON status codes.

enumerator kStatusGroup_ENDAT3

Group number for ENDAT3 status codes.

enumerator kStatusGroup_HIPERFACE

Group number for HIPERFACE status codes.

enumerator kStatusGroup_NPX

Group number for NPX status codes.

enumerator kStatusGroup_ELA_CSEC

Group number for ELA_CSEC status codes.

enumerator kStatusGroup_FLEXIO_T_FORMAT

Group number for T-format status codes.

enumerator kStatusGroup_FLEXIO_A_FORMAT

Group number for A-format status codes.

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.

2.64 LCDIFv2: LCD Interface v2

`void LCDIFV2_Init(LCDIFV2_Type *base)`

Initializes the LCDIF v2.

This function ungates the LCDIF v2 clock and release the peripheral reset.

Parameters

- base – LCDIF v2 peripheral base address.

`void LCDIFV2_Deinit(LCDIFV2_Type *base)`

Deinitializes the LCDIF peripheral.

Parameters

- base – LCDIF peripheral base address.

`void LCDIFV2_Reset(LCDIFV2_Type *base)`

Reset the LCDIF v2.

Parameters

- base – LCDIF peripheral base address.

`void LCDIFV2_DisplayGetDefaultConfig(lcdifv2_display_config_t *config)`

Gets the LCDIF display default configuration structure.

This function sets the configuration structure to default values. The default configuration is set to the following values.

```
config->panelWidth    = 0U;
config->panelHeight   = 0U;
config->hsw            = 3U;
config->hfp            = 3U;
config->hbp            = 3U;
config->vsw            = 3U;
config->vfp            = 3U;
config->vbp            = 3U;
config->polarityFlags = kLCDIFV2_VsyncActiveHigh | kLCDIFV2_HsyncActiveHigh | kLCDIFV2_
↪DataEnableActiveHigh |
                        kLCDIFV2_DriveDataOnRisingClkEdge | kLCDIFV2_DataActiveHigh;
config->lineOrder      = kLCDIFV2_LineOrderRGB;
```

Parameters

- config – Pointer to the LCDIF configuration structure.

`void LCDIFV2_SetDisplayConfig(LCDIFV2_Type *base, const lcdifv2_display_config_t *config)`

Set the LCDIF v2 display configurations.

Parameters

- base – LCDIF peripheral base address.
- config – Pointer to the LCDIF configuration structure.


```
static inline void LCDIFV2_EnableDisplay(LCDIFV2_Type *base, bool enable)
```

Enable or disable the display.

Parameters

- base – LCDIF peripheral base address.
- enable – Enable or disable.

```
static inline void LCDIFV2_EnableInterrupts(LCDIFV2_Type *base, uint8_t domain, uint32_t mask)
```

Enables LCDIF interrupt requests.

Parameters

- base – LCDIF peripheral base address.
- domain – CPU domain the interrupt signal routed to.
- mask – interrupt source, OR'ed value of `_lcdifv2_interrupt`.

```
static inline void LCDIFV2_DisableInterrupts(LCDIFV2_Type *base, uint8_t domain, uint32_t mask)
```

Disables LCDIF interrupt requests.

Parameters

- base – LCDIF peripheral base address.
- domain – CPU domain the interrupt signal routed to.
- mask – interrupt source, OR'ed value of `_lcdifv2_interrupt`.

```
static inline uint32_t LCDIFV2_GetInterruptStatus(LCDIFV2_Type *base, uint8_t domain)
```

Get LCDIF interrupt pending status.

Parameters

- base – LCDIF peripheral base address.
- domain – CPU domain the interrupt signal routed to.

Returns

Interrupt pending status, OR'ed value of `_lcdifv2_interrupt`.

```
static inline void LCDIFV2_ClearInterruptStatus(LCDIFV2_Type *base, uint8_t domain, uint32_t mask)
```

Clear LCDIF interrupt pending status.

Parameters

- base – LCDIF peripheral base address.
- domain – CPU domain the interrupt signal routed to.
- mask – of the flags to clear, OR'ed value of `_lcdifv2_interrupt`.

```
status_t LCDIFV2_SetLut(LCDIFV2_Type *base, uint8_t layerIndex, const uint32_t *lutData, uint16_t count, bool useShadowLoad)
```

Set the LUT data.

This function sets the specific layer LUT data, if `useShadowLoad` is true, call `LCDIFV2_TriggerLayerShadowLoad` after this function, the LUT will be loaded to the hardware during next vertical blanking period. If `useShadowLoad` is false, the LUT data is loaded to hardware directly.

Parameters

- base – LCDIF v2 peripheral base address.

- `layerIndex` – Which layer to set.
- `lutData` – The LUT data to load.
- `count` – Count of `lutData`.
- `useShadowLoad` – Use shadow load.

Return values

- `kStatus__Success` – Set success.
- `kStatus__Fail` – Previous LUT data is not loaded to hardware yet.

```
static inline void LCDIFV2_SetLayerSize(LCDIFV2_Type *base, uint8_t layerIndex, uint16_t width, uint16_t height)
```

Set the layer dimension.

Note: The layer width must be in multiples of the number of pixels that can be stored in 32 bits

Parameters

- `base` – LCDIFv2 peripheral base address.
- `layerIndex` – Layer `layerIndex`.
- `width` – Layer width in pixel.
- `height` – Layer height.

```
static inline void LCDIFV2_SetLayerOffset(LCDIFV2_Type *base, uint8_t layerIndex, uint16_t offsetX, uint16_t offsetY)
```

Set the layer position in output frame.

Parameters

- `base` – LCDIFv2 peripheral base address.
- `layerIndex` – Layer `layerIndex`.
- `offsetX` – Horizontal offset, start from 0.
- `offsetY` – Vertical offset, start from 0.

```
void LCDIFV2_SetLayerBufferConfig(LCDIFV2_Type *base, uint8_t layerIndex, const lcdifv2_buffer_config_t *config)
```

Set the layer source buffer configuration.

Parameters

- `base` – LCDIFv2 peripheral base address.
- `layerIndex` – Layer `layerIndex`.
- `config` – Pointer to the configuration.

```
static inline void LCDIFV2_SetLayerBufferAddr(LCDIFV2_Type *base, uint8_t layerIndex, uint32_t addr)
```

Set the layer source buffer address.

This function is used for fast runtime source buffer change.

Parameters

- `base` – LCDIFv2 peripheral base address.
- `layerIndex` – Layer `layerIndex`.

- `addr` – The new source buffer address passed to the layer, should be 64-bit aligned.

static inline void LCDIFV2_EnableLayer(LCDIFV2_Type *base, uint8_t layerIndex, bool enable)

Enable or disable the layer.

Parameters

- `base` – LCDIFv2 peripheral base address.
- `layerIndex` – Layer layerIndex.
- `enable` – Pass in true to enable, false to disable.

static inline void LCDIFV2_TriggerLayerShadowLoad(LCDIFV2_Type *base, uint8_t layerIndex)

Trigger the layer configuration shadow load.

The new layer configurations are written to the shadow registers first, When all configurations written finished, call this function, then shadowed control registers are updated to the active control registers on VSYNC of next frame.

Parameters

- `base` – LCDIFv2 peripheral base address.
- `layerIndex` – Layer layerIndex.

static inline void LCDIFV2_SetLayerBackGroundColor(LCDIFV2_Type *base, uint8_t layerIndex, uint32_t backgroundColor)

Set the layer back ground color.

The back ground color is used when layer not actived.

Parameters

- `base` – LCDIFv2 peripheral base address.
- `layerIndex` – Index of the layer.
- `backgroundColor` – Background color to use when this layer is not active.

void LCDIFV2_SetLayerBlendConfig(LCDIFV2_Type *base, uint8_t layerIndex, const *lcdifv2_blend_config_t* *config)

Set the layer alpha blend mode.

Parameters

- `base` – LCDIFv2 peripheral base address.
- `layerIndex` – Index of the CSC unit.
- `config` – Pointer to the blend configuration.

void LCDIFV2_SetCscMode(LCDIFV2_Type *base, uint8_t layerIndex, *lcdifv2_csc_mode_t* mode)

Set the color space conversion mode.

Supports YUV2RGB and YCbCr2RGB.

Parameters

- `base` – LCDIFv2 peripheral base address.
- `layerIndex` – Index of the layer.
- `mode` – The conversion mode.

status_t LCDIFV2_GetPorterDuffConfig(*lcdifv2_pd_blend_mode_t* mode, *lcdifv2_pd_layer_t* layer, *lcdifv2_blend_config_t* *config)

Get the blend configuration for Porter Duff blend.

This function gets the blend configuration for Porter Duff blend, config->pdFactorMode is set according to layer and mode, other blend configurations are set to:

```
config->pdAlphaMode = kLCDIFV2_PD_AlphaStraight;
config->pdColorMode = kLCDIFV2_PD_ColorStraight;
config->pdGlobalAlphaMode = kLCDIFV2_PD_LocalAlpha;
config->alphaMode = kLCDIFV2_AlphaPotterDuff;
```

This is the basic Porter Duff blend configuration, user still could modify the configurations after this function.

Parameters

- mode – Porter Duff blend mode.
- layer – The configuration for source layer or destination layer.
- config – Pointer to the configuration.

Return values

- kStatus_Success – Get the configuration successfully.
- kStatus_InvalidArgument – The argument is invalid.

status_t LCDIFV2_GetMultiLayerGlobalAlpha(const uint8_t blendedAlpha[], uint8_t globalAlpha[], uint8_t layerCount)

Get the global alpha values for multiple layer blend.

This function calculates the global alpha value for each layer based on the desired blended alpha.

When all layers use the global alpha, the relationship of blended alpha and global alpha of each layer is:

Layer 7: $ba_7 = ga_7$ Layer 6: $ba_6 = ga_6 * (1 - ga_7)$ Layer 5: $ba_5 = ga_5 * (1 - ga_6) * (1 - ga_7)$ Layer 4: $ba_4 = ga_4 * (1 - ga_5) * (1 - ga_6) * (1 - ga_7)$ Layer 3: $ba_3 = ga_3 * (1 - ga_4) * (1 - ga_5) * (1 - ga_6) * (1 - ga_7)$ Layer 2: $ba_2 = ga_2 * (1 - ga_3) * (1 - ga_4) * (1 - ga_5) * (1 - ga_6) * (1 - ga_7)$ Layer 1: $ba_1 = ga_1 * (1 - ga_2) * (1 - ga_3) * (1 - ga_4) * (1 - ga_5) * (1 - ga_6) * (1 - ga_7)$ Layer 0: $ba_0 = 1 * (1 - ga_1) * (1 - ga_2) * (1 - ga_3) * (1 - ga_4) * (1 - ga_5) * (1 - ga_6) * (1 - ga_7)$

Here ba_N is the blended alpha of layer N, ga_N is the global alpha configured to layer N.

This function calculates the global alpha based on the blended alpha. The blendedAlpha and globalAlpha are all arrays of size layerCount. The first layer is a background layer, so blendedAlpha[0] is useless, globalAlpha[0] is always 255.

Parameters

- blendedAlpha – **[in]** The desired blended alpha value, alpha range 0~255.
- globalAlpha – **[out]** Calculated global alpha set to each layer register.
- layerCount – **[in]** Total layer count.

Return values

- kStatus_Success – Get successfully.
- kStatus_InvalidArgument – The argument is invalid.

FSL_LCDIFV2_DRIVER_VERSION

LCDIF v2 driver version.

enum _lcdifv2_polarity_flags

LCDIF v2 signal polarity flags.

Values:

enumerator kLCDIFV2_VsyncActiveHigh

VSYNC active high.

enumerator kLCDIFV2_HsyncActiveHigh

HSYNC active high.

enumerator kLCDIFV2_DataEnableActiveHigh

Data enable line active high.

enumerator kLCDIFV2_DriveDataOnRisingClkEdge

Output data on rising clock edge, capture data on falling clock edge.

enumerator kLCDIFV2_DataActiveHigh

Data active high.

enumerator kLCDIFV2_VsyncActiveLow

VSYNC active low.

enumerator kLCDIFV2_HsyncActiveLow

HSYNC active low.

enumerator kLCDIFV2_DataEnableActiveLow

Data enable line active low.

enumerator kLCDIFV2_DriveDataOnFallingClkEdge

Output data on falling clock edge, capture data on rising clock edge.

enumerator kLCDIFV2_DataActiveLow

Data active high.

enum _lcdifv2_interrupt

The LCDIF v2 interrupts.

Values:

enumerator kLCDIFV2_Layer0FifoEmptyInterrupt

Layer 0 FIFO empty.

enumerator kLCDIFV2_Layer1FifoEmptyInterrupt

Layer 1 FIFO empty.

enumerator kLCDIFV2_Layer2FifoEmptyInterrupt

Layer 2 FIFO empty.

enumerator kLCDIFV2_Layer3FifoEmptyInterrupt

Layer 3 FIFO empty.

enumerator kLCDIFV2_Layer4FifoEmptyInterrupt

Layer 4 FIFO empty.

enumerator kLCDIFV2_Layer5FifoEmptyInterrupt

Layer 5 FIFO empty.

enumerator kLCDIFV2_Layer6FifoEmptyInterrupt

Layer 6 FIFO empty.

enumerator kLCDIFV2_Layer7FifoEmptyInterrupt

Layer 7 FIFO empty.

enumerator kLCDIFV2__Layer0DmaDoneInterrupt
Layer 0 DMA done.

enumerator kLCDIFV2__Layer1DmaDoneInterrupt
Layer 1 DMA done.

enumerator kLCDIFV2__Layer2DmaDoneInterrupt
Layer 2 DMA done.

enumerator kLCDIFV2__Layer3DmaDoneInterrupt
Layer 3 DMA done.

enumerator kLCDIFV2__Layer4DmaDoneInterrupt
Layer 4 DMA done.

enumerator kLCDIFV2__Layer5DmaDoneInterrupt
Layer 5 DMA done.

enumerator kLCDIFV2__Layer6DmaDoneInterrupt
Layer 6 DMA done.

enumerator kLCDIFV2__Layer7DmaDoneInterrupt
Layer 7 DMA done.

enumerator kLCDIFV2__Layer0DmaErrorInterrupt
Layer 0 DMA error.

enumerator kLCDIFV2__Layer1DmaErrorInterrupt
Layer 1 DMA error.

enumerator kLCDIFV2__Layer2DmaErrorInterrupt
Layer 2 DMA error.

enumerator kLCDIFV2__Layer3DmaErrorInterrupt
Layer 3 DMA error.

enumerator kLCDIFV2__Layer4DmaErrorInterrupt
Layer 4 DMA error.

enumerator kLCDIFV2__Layer5DmaErrorInterrupt
Layer 5 DMA error.

enumerator kLCDIFV2__Layer6DmaErrorInterrupt
Layer 6 DMA error.

enumerator kLCDIFV2__Layer7DmaErrorInterrupt
Layer 7 DMA error.

enumerator kLCDIFV2__VerticalBlankingInterrupt
Start of vertical blanking period.

enumerator kLCDIFV2__OutputUnderrunInterrupt
Output buffer underrun.

enumerator kLCDIFV2__VsyncEdgeInterrupt
Interrupt at VSYNC edge.

enum __lcdifv2__line__order
The LCDIF v2 output line order.

Values:

enumerator kLCDIFV2__LineOrderRGB
RGB

enumerator kLCDIFV2__LineOrderRBG
RBG

enumerator kLCDIFV2__LineOrderGBR
GBR

enumerator kLCDIFV2__LineOrderGRB
GRB

enumerator kLCDIFV2__LineOrderBRG
BRG

enumerator kLCDIFV2__LineOrderBGR
BGR

enum _lcdifv2__csc__mode
LCDIF v2 color space conversion mode.

Values:

enumerator kLCDIFV2__CscDisable
Disable the CSC.

enumerator kLCDIFV2__CscYUV2RGB
YUV to RGB.

enumerator kLCDIFV2__CscYCbCr2RGB
YCbCr to RGB.

enum _lcdifv2__pixel__format
LCDIF v2 pixel format.

Values:

enumerator kLCDIFV2__PixelFormatIndex1BPP
LUT index 1 bit.

enumerator kLCDIFV2__PixelFormatIndex2BPP
LUT index 2 bit.

enumerator kLCDIFV2__PixelFormatIndex4BPP
LUT index 4 bit.

enumerator kLCDIFV2__PixelFormatIndex8BPP
LUT index 8 bit.

enumerator kLCDIFV2__PixelFormatRGB565
RGB565, two pixels use 32 bits.

enumerator kLCDIFV2__PixelFormatARGB1555
ARGB1555, two pixels use 32 bits.

enumerator kLCDIFV2__PixelFormatARGB4444
ARGB4444, two pixels use 32 bits.

enumerator kLCDIFV2__PixelFormatUYVY
UYVY, only layer 0 and layer 1 support this.

enumerator kLCDIFV2__PixelFormatVYUY
VYUY, only layer 0 and layer 1 support this.

enumerator kLCDIFV2_PixelFormatYUYV
YUYV, only layer 0 and layer 1 support this.

enumerator kLCDIFV2_PixelFormatYVYU
YVYU, only layer 0 and layer 1 support this.

enumerator kLCDIFV2_PixelFormatRGB888
RGB888 packed, one pixel uses 24 bits.

enumerator kLCDIFV2_PixelFormatARGB8888
ARGB8888 unpacked, one pixel uses 32 bits.

enumerator kLCDIFV2_PixelFormatABGR8888
ABGR8888 unpacked, one pixel uses 32 bits.

enum _lcdifv2_alpha_mode
LCDIF v2 layer alpha blending mode.

Values:

enumerator kLCDIFV2_AlphaDisable
Disable alpha blend.

enumerator kLCDIFV2_AlphaOverride
Use the global alpha value, pixel defined alpha value is overridden.

enumerator kLCDIFV2_AlphaEmbedded
Use the pixel defined alpha value.

enumerator kLCDIFV2_AlphaPoterDuff
Use the PoterDuff alpha blending.

enum _lcdifv2_pd_alpha_mode
LCDIF v2 PoterDuff alpha mode.

Values:

enumerator kLCDIFV2_PD_AlphaStraight
Straight mode.

enumerator kLCDIFV2_PD_AlphaInversed
Inversed mode.

enum _lcdifv2_pd_color_mode
LCDIF v2 PoterDuff color mode.

Values:

enumerator kLCDIFV2_PD_ColorNoAlpha
Output color directly.

enumerator kLCDIFV2_PD_ColorWithAlpha
Output color multiples alpha.

enum _lcdifv2_pd_global_alpha_mode
LCDIF v2 PoterDuff global alpha mode.

Values:

enumerator kLCDIFV2_PD_GlobalAlpha
Use global alpha.

enumerator kLCDIFV2_PD_LocalAlpha
Use local alpha.

enumerator kLCDIFV2_PD_ScaledAlpha

Use scaled alpha.

enum _lcdifv2_pd_factor_mode

LCDIF v2 PorterDuff factor mode.

Values:

enumerator kLCDIFV2_PD_FactorOne

Use 1.

enumerator kLCDIFV2_PD_FactorZero

Use 0.

enumerator kLCDIFV2_PD_FactorStraightAlpha

Use straight alpha.

enumerator kLCDIFV2_PD_FactorInversedAlpha

Use inversed alpha.

enum _lcdifv2_pd_blend_mode

LCDIFv2 Porter Duff blend mode. Note: Don't change the enum item value.

Values:

enumerator kLCDIFV2_PD_Src

Source Only

enumerator kLCDIFV2_PD_Atop

Source Atop

enumerator kLCDIFV2_PD_Over

Source Over

enumerator kLCDIFV2_PD_In

Source In.

enumerator kLCDIFV2_PD_Out

Source Out.

enumerator kLCDIFV2_PD_Dst

Destination Only.

enumerator kLCDIFV2_PD_DstAtop

Destination Atop.

enumerator kLCDIFV2_PD_DstOver

Destination Over.

enumerator kLCDIFV2_PD_DstIn

Destination In.

enumerator kLCDIFV2_PD_DstOut

Destination Out.

enumerator kLCDIFV2_PD_Xor

XOR.

enumerator kLCDIFV2_PD_Clear

Clear.

enumerator kLCDIFV2_PD_Max

Used for boarder detection.

enum `_lcdifv2_pd_layer`

LCDIFv2 Porter Duff layer. Note: Don't change the enum item value.

Values:

enumerator `kLCDIFV2_PD_SrcLayer`

Source layer.

enumerator `kLCDIFV2_PD_DestLayer`

Destination layer.

enumerator `kLCDIFV2_PD_LayerMax`

Used for boarder detection.

typedef enum `_lcdifv2_line_order` `lcdifv2_line_order_t`

The LCDIF v2 output line order.

typedef struct `_lcdifv2_display_config` `lcdifv2_display_config_t`

LCDIF v2 display configure structure.

typedef enum `_lcdifv2_csc_mode` `lcdifv2_csc_mode_t`

LCDIF v2 color space conversion mode.

typedef enum `_lcdifv2_pixel_format` `lcdifv2_pixel_format_t`

LCDIF v2 pixel format.

typedef struct `_lcdifv2_buffer_config` `lcdifv2_buffer_config_t`

LCDIF v2 source buffer configuration.

typedef enum `_lcdifv2_alpha_mode` `lcdifv2_alpha_mode_t`

LCDIF v2 layer alpha blending mode.

typedef enum `_lcdifv2_pd_alpha_mode` `lcdifv2_pd_alpha_mode_t`

LCDIF v2 PoterDuff alpha mode.

typedef enum `_lcdifv2_pd_color_mode` `lcdifv2_pd_color_mode_t`

LCDIF v2 PoterDuff color mode.

typedef enum `_lcdifv2_pd_global_alpha_mode` `lcdifv2_pd_global_alpha_mode_t`

LCDIF v2 PoterDuff global alpha mode.

typedef enum `_lcdifv2_pd_factor_mode` `lcdifv2_pd_factor_mode_t`

LCDIF v2 PoterDuff factor mode.

typedef struct `_lcdifv2_blend_config` `lcdifv2_blend_config_t`

LCDIF v2 layer alpha blending configuration.

typedef enum `_lcdifv2_pd_blend_mode` `lcdifv2_pd_blend_mode_t`

LCDIFv2 Porter Duff blend mode. Note: Don't change the enum item value.

typedef enum `_lcdifv2_pd_layer` `lcdifv2_pd_layer_t`

LCDIFv2 Porter Duff layer. Note: Don't change the enum item value.

`LCDIFV2_LAYER_COUNT`

`LCDIFV2_LAYER_CSC_COUNT`

`LCDIFV2_ADDR_CPU_2_IP(addr)`

`LCDIFV2_MAKE_FIFO_EMPTY_INTERRUPT(layer)`

LCDIF v2 FIFO empty interrupt.

`LCDIFV2_MAKE_DMA_DONE_INTERRUPT(layer)`

LCDIF v2 DMA done interrupt.

LCDIFV2_MAKE_DMA_ERROR_INTERRUPT(layer)

LCDIF v2 DMA error interrupt.

LCDIFV2_LUT_ENTRY_NUM

struct _lcdifv2_display_config

#include <fsl_lcdifv2.h> LCDIF v2 display configure structure.

Public Members

uint16_t panelWidth

Display panel width, pixels per line.

uint16_t panelHeight

Display panel height, how many lines per panel.

uint8_t hsw

HSYNC pulse width.

uint8_t hfp

Horizontal front porch.

uint8_t hbp

Horizontal back porch.

uint8_t vsw

VSYNC pulse width.

uint8_t vfp

Vrtical front porch.

uint8_t vbp

Vertical back porch.

uint32_t polarityFlags

OR'ed value of _lcdifv2_polarity_flags, used to contol the signal polarity.

lcdifv2_line_order_t lineOrder

Line order.

struct _lcdifv2_buffer_config

#include <fsl_lcdifv2.h> LCDIF v2 source buffer configuration.

Public Members

uint16_t strideBytes

Number of bytes between two vertically adjacent pixels, suggest 64-bit aligned.

lcdifv2_pixel_format_t pixelFormat

Source buffer pixel format.

struct _lcdifv2_blend_config

#include <fsl_lcdifv2.h> LCDIF v2 layer alpha blending configuration.

Public Members

uint8_t globalAlpha

Global alpha value, only used when alphaMode is kLCDIFV2_AlphaOverride or kLCDIFV2_AlphaPoterDuff

lcdifv2_alpha_mode_t alphaMode

Alpha mode.

lcdifv2_pd_alpha_mode_t pdAlphaMode

PoterDuff alpha mode, only used when alphaMode is kLCDIFV2_AlphaPoterDuff

lcdifv2_pd_color_mode_t pdColorMode

PoterDuff color mode, only used when alphaMode is kLCDIFV2_AlphaPoterDuff

lcdifv2_pd_global_alpha_mode_t pdGlobalAlphaMode

PoterDuff global alpha mode, only used when alphaMode is kLCDIFV2_AlphaPoterDuff

lcdifv2_pd_factor_mode_t pdFactorMode

PoterDuff factor mode, only used when alphaMode is kLCDIFV2_AlphaPoterDuff

2.65 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

2.66 LPADC: 12-bit SAR Analog-to-Digital Converter Driver

enum _lpadc_status_flags

Define hardware flags of the module.

Values:

enumerator kLPADC_ResultFIFO0OverflowFlag

Indicates that more data has been written to the Result FIFO 0 than it can hold.

enumerator kLPADC_ResultFIFO0ReadyFlag

Indicates when the number of valid datawords in the result FIFO 0 is greater than the setting watermark level.

enumerator kLPADC_TriggerExceptionFlag

Indicates that a trigger exception event has occurred.

enumerator kLPADC_TriggerCompletionFlag

Indicates that a trigger completion event has occurred.

enumerator kLPADC_CalibrationReadyFlag

Indicates that the calibration process is done.

enumerator kLPADC_ActiveFlag

Indicates that the ADC is in active state.

enumerator kLPADC_ResultFIFOOverflowFlag

To compilitable with old version, do not recommend using this, please use kLPADC_ResultFIFO0OverflowFlag as instead.

enumerator kLPADC_ResultFIFOReadyFlag

To compilitable with old version, do not recommend using this, please use kLPADC_ResultFIFO0ReadyFlag as instead.

enum _lpadc_interrupt_enable

Define interrupt switchers of the module.

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

Values:

enumerator kLPADC_ResultFIFO0OverflowInterruptEnable

Configures ADC to generate overflow interrupt requests when FOF0 flag is asserted.

enumerator kLPADC_FIFO0WatermarkInterruptEnable

Configures ADC to generate watermark interrupt requests when RDY0 flag is asserted.

enumerator kLPADC_ResultFIFOOverflowInterruptEnable

To compilitable with old version, do not recommend using this, please use kLPADC_ResultFIFO0OverflowInterruptEnable as instead.

enumerator kLPADC_FIFOWatermarkInterruptEnable

To compilitable with old version, do not recommend using this, please use kLPADC_FIFO0WatermarkInterruptEnable as instead.

enumerator kLPADC_TriggerExceptionInterruptEnable

Configures ADC to generate trigger exception interrupt.

enumerator kLPADC_Trigger0CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 0 completion.

enumerator kLPADC_Trigger1CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 1 completion.

enumerator kLPADC_Trigger2CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 2 completion.

enumerator kLPADC_Trigger3CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 3 completion.

enumerator kLPADC_Trigger4CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 4 completion.

enumerator kLPADC_Trigger5CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 5 completion.

enumerator kLPADC_Trigger6CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 6 completion.

enumerator kLPADC_Trigger7CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 7 completion.

enumerator kLPADC_Trigger8CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 8 completion.

enumerator kLPADC_Trigger9CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 9 completion.

enumerator kLPADC_Trigger10CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 10 completion.

enumerator kLPADC_Trigger11CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 11 completion.

enumerator kLPADC_Trigger12CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 12 completion.

enumerator kLPADC_Trigger13CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 13 completion.

enumerator kLPADC_Trigger14CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 14 completion.

enumerator kLPADC_Trigger15CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 15 completion.

enum _lpadc_trigger_status_flags

The enumerator of lpadc trigger status flags, including interrupted flags and completed flags.

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

Values:

enumerator kLPADC_Trigger0InterruptedFlag
Trigger 0 is interrupted by a high priority exception.

enumerator kLPADC_Trigger1InterruptedFlag
Trigger 1 is interrupted by a high priority exception.

enumerator kLPADC_Trigger2InterruptedFlag
Trigger 2 is interrupted by a high priority exception.

enumerator kLPADC_Trigger3InterruptedFlag
Trigger 3 is interrupted by a high priority exception.

enumerator kLPADC_Trigger4InterruptedFlag
Trigger 4 is interrupted by a high priority exception.

enumerator kLPADC_Trigger5InterruptedFlag
Trigger 5 is interrupted by a high priority exception.

enumerator kLPADC_Trigger6InterruptedFlag
Trigger 6 is interrupted by a high priority exception.

enumerator kLPADC_Trigger7InterruptedFlag

Trigger 7 is interrupted by a high priority exception.

enumerator kLPADC_Trigger8InterruptedFlag

Trigger 8 is interrupted by a high priority exception.

enumerator kLPADC_Trigger9InterruptedFlag

Trigger 9 is interrupted by a high priority exception.

enumerator kLPADC_Trigger10InterruptedFlag

Trigger 10 is interrupted by a high priority exception.

enumerator kLPADC_Trigger11InterruptedFlag

Trigger 11 is interrupted by a high priority exception.

enumerator kLPADC_Trigger12InterruptedFlag

Trigger 12 is interrupted by a high priority exception.

enumerator kLPADC_Trigger13InterruptedFlag

Trigger 13 is interrupted by a high priority exception.

enumerator kLPADC_Trigger14InterruptedFlag

Trigger 14 is interrupted by a high priority exception.

enumerator kLPADC_Trigger15InterruptedFlag

Trigger 15 is interrupted by a high priority exception.

enumerator kLPADC_Trigger0CompletedFlag

Trigger 0 is completed and trigger 0 has enabled completion interrupts.

enumerator kLPADC_Trigger1CompletedFlag

Trigger 1 is completed and trigger 1 has enabled completion interrupts.

enumerator kLPADC_Trigger2CompletedFlag

Trigger 2 is completed and trigger 2 has enabled completion interrupts.

enumerator kLPADC_Trigger3CompletedFlag

Trigger 3 is completed and trigger 3 has enabled completion interrupts.

enumerator kLPADC_Trigger4CompletedFlag

Trigger 4 is completed and trigger 4 has enabled completion interrupts.

enumerator kLPADC_Trigger5CompletedFlag

Trigger 5 is completed and trigger 5 has enabled completion interrupts.

enumerator kLPADC_Trigger6CompletedFlag

Trigger 6 is completed and trigger 6 has enabled completion interrupts.

enumerator kLPADC_Trigger7CompletedFlag

Trigger 7 is completed and trigger 7 has enabled completion interrupts.

enumerator kLPADC_Trigger8CompletedFlag

Trigger 8 is completed and trigger 8 has enabled completion interrupts.

enumerator kLPADC_Trigger9CompletedFlag

Trigger 9 is completed and trigger 9 has enabled completion interrupts.

enumerator kLPADC_Trigger10CompletedFlag

Trigger 10 is completed and trigger 10 has enabled completion interrupts.

enumerator kLPADC_Trigger11CompletedFlag

Trigger 11 is completed and trigger 11 has enabled completion interrupts.

enumerator kLPADC_Trigger12CompletedFlag

Trigger 12 is completed and trigger 12 has enabled completion interrupts.

enumerator kLPADC_Trigger13CompletedFlag

Trigger 13 is completed and trigger 13 has enabled completion interrupts.

enumerator kLPADC_Trigger14CompletedFlag

Trigger 14 is completed and trigger 14 has enabled completion interrupts.

enumerator kLPADC_Trigger15CompletedFlag

Trigger 15 is completed and trigger 15 has enabled completion interrupts.

enum __lpadc_sample_scale_mode

Define enumeration of sample scale mode.

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

Values:

enumerator kLPADC_SamplePartScale

Use divided input voltage signal. (For scale select, please refer to the reference manual).

enumerator kLPADC_SampleFullScale

Full scale (Factor of 1).

enum __lpadc_sample_channel_mode

Define enumeration of channel sample mode.

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

Values:

enumerator kLPADC_SampleChannelSingleEndSideA

Single-end mode, only A-side channel is converted.

enumerator kLPADC_SampleChannelSingleEndSideB

Single-end mode, only B-side channel is converted.

enumerator kLPADC_SampleChannelDiffBothSideAB

Differential mode, the ADC result is (CHnA-CHnB).

enumerator kLPADC_SampleChannelDiffBothSideBA

Differential mode, the ADC result is (CHnB-CHnA).

enumerator kLPADC_SampleChannelDiffBothSide

Differential mode, the ADC result is (CHnA-CHnB).

enumerator kLPADC_SampleChannelDualSingleEndBothSide

Dual-Single-Ended Mode. Both A side and B side channels are converted independently.

enum __lpadc_hardware_average_mode

Define enumeration of hardware average selection.

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

Note: Some enumerator values are not available on some devices, mainly depends on the size of AVGS field in CMDH register.

Values:

enumerator kLPADC__HardwareAverageCount1

Single conversion.

enumerator kLPADC__HardwareAverageCount2

2 conversions averaged.

enumerator kLPADC__HardwareAverageCount4

4 conversions averaged.

enumerator kLPADC__HardwareAverageCount8

8 conversions averaged.

enumerator kLPADC__HardwareAverageCount16

16 conversions averaged.

enumerator kLPADC__HardwareAverageCount32

32 conversions averaged.

enumerator kLPADC__HardwareAverageCount64

64 conversions averaged.

enumerator kLPADC__HardwareAverageCount128

128 conversions averaged.

enum _lpadc_sample_time_mode

Define enumeration of sample time selection.

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

Values:

enumerator kLPADC__SampleTimeADCK3

3 ADCK cycles total sample time.

enumerator kLPADC__SampleTimeADCK5

5 ADCK cycles total sample time.

enumerator kLPADC__SampleTimeADCK7

7 ADCK cycles total sample time.

enumerator kLPADC__SampleTimeADCK11

11 ADCK cycles total sample time.

enumerator kLPADC__SampleTimeADCK19

19 ADCK cycles total sample time.

enumerator kLPADC__SampleTimeADCK35

35 ADCK cycles total sample time.

enumerator kLPADC__SampleTimeADCK67

69 ADCK cycles total sample time.

enumerator kLPADC__SampleTimeADCK131

131 ADCK cycles total sample time.

enum `_lpadc_hardware_compare_mode`

Define enumeration of hardware compare mode.

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

Values:

enumerator `kLPADC__HardwareCompareDisabled`

Compare disabled.

enumerator `kLPADC__HardwareCompareStoreOnTrue`

Compare enabled. Store on true.

enumerator `kLPADC__HardwareCompareRepeatUntilTrue`

Compare enabled. Repeat channel acquisition until true.

enum `_lpadc_conversion_resolution_mode`

Define enumeration of conversion resolution mode.

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to `lpadc_sample_channel_mode_t`

Values:

enumerator `kLPADC__ConversionResolutionStandard`

Standard resolution. Single-ended 12-bit conversion, Differential 13-bit conversion with 2's complement output.

enumerator `kLPADC__ConversionResolutionHigh`

High resolution. Single-ended 16-bit conversion; Differential 16-bit conversion with 2's complement output.

enum `_lpadc_conversion_average_mode`

Define enumeration of conversion averages mode.

Configure the conversion average number for auto-calibration.

Note: Some enumerator values are not available on some devices, mainly depends on the size of `CAL_AVGS` field in `CTRL` register.

Values:

enumerator `kLPADC__ConversionAverage1`

Single conversion.

enumerator `kLPADC__ConversionAverage2`

2 conversions averaged.

enumerator `kLPADC__ConversionAverage4`

4 conversions averaged.

enumerator `kLPADC__ConversionAverage8`

8 conversions averaged.

enumerator `kLPADC__ConversionAverage16`

16 conversions averaged.

enumerator `kLPADC__ConversionAverage32`

32 conversions averaged.

enumerator kLPADC_ConversionAverage64
64 conversions averaged.

enumerator kLPADC_ConversionAverage128
128 conversions averaged.

enumerator kLPADC_ConversionAverageMax

enum _lpadc_reference_voltage_mode

Define enumeration of reference voltage source.

For detail information, need to check the SoC's specification.

Values:

enumerator kLPADC_ReferenceVoltageAlt1
Option 1 setting.

enumerator kLPADC_ReferenceVoltageAlt2
Option 2 setting.

enumerator kLPADC_ReferenceVoltageAlt3
Option 3 setting.

enum _lpadc_power_level_mode

Define enumeration of power configuration.

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

Values:

enumerator kLPADC_PowerLevelAlt1
Lowest power setting.

enumerator kLPADC_PowerLevelAlt2
Next lowest power setting.

enumerator kLPADC_PowerLevelAlt3
...

enumerator kLPADC_PowerLevelAlt4
Highest power setting.

enum _lpadc_offset_calibration_mode

Define enumeration of offset calibration mode.

Values:

enumerator kLPADC_OffsetCalibration12bitMode
12 bit offset calibration mode.

enumerator kLPADC_OffsetCalibration16bitMode
16 bit offset calibration mode.

enum _lpadc_trigger_priority_policy

Define enumeration of trigger priority policy.

This selection controls how higher priority triggers are handled.

Note: **kLPADC_TriggerPriorityPreemptSubsequently** is not available on some devices, mainly depends on the size of TPRICTRL field in CFG register.

Values:

enumerator `kLPADC_ConvPreemptImmediatelyNotAutoResumed`

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion is not automatically resumed or restarted.

enumerator `kLPADC_ConvPreemptSoftlyNotAutoResumed`

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion is not resumed or restarted.

enumerator `kLPADC_ConvPreemptImmediatelyAutoRestarted`

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

enumerator `kLPADC_ConvPreemptSoftlyAutoRestarted`

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

enumerator `kLPADC_ConvPreemptImmediatelyAutoResumed`

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be resumed.

enumerator `kLPADC_ConvPreemptSoftlyAutoResumed`

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will be automatically be resumed.

enumerator `kLPADC_TriggerPriorityPreemptImmediately`

Legacy support is not recommended as it only ensures compatibility with older versions.

enumerator `kLPADC_TriggerPriorityPreemptSoftly`

Legacy support is not recommended as it only ensures compatibility with older versions.

enumerator `kLPADC_TriggerPriorityExceptionDisabled`

High priority trigger exception disabled.

typedef enum *lpadc_sample_scale_mode* `lpadc_sample_scale_mode_t`

Define enumeration of sample scale mode.

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

`typedef enum _lpadc_sample_channel_mode` `lpadc_sample_channel_mode_t`

Define enumeration of channel sample mode.

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

`typedef enum _lpadc_hardware_average_mode` `lpadc_hardware_average_mode_t`

Define enumeration of hardware average selection.

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

Note: Some enumerator values are not available on some devices, mainly depends on the size of AVGS field in CMDH register.

`typedef enum _lpadc_sample_time_mode` `lpadc_sample_time_mode_t`

Define enumeration of sample time selection.

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

`typedef enum _lpadc_hardware_compare_mode` `lpadc_hardware_compare_mode_t`

Define enumeration of hardware compare mode.

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

`typedef enum _lpadc_conversion_resolution_mode` `lpadc_conversion_resolution_mode_t`

Define enumeration of conversion resolution mode.

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to `lpadc_sample_channel_mode_t`

`typedef enum _lpadc_conversion_average_mode` `lpadc_conversion_average_mode_t`

Define enumeration of conversion averages mode.

Configure the conversion average number for auto-calibration.

Note: Some enumerator values are not available on some devices, mainly depends on the size of CAL_AVGS field in CTRL register.

`typedef enum _lpadc_reference_voltage_mode` `lpadc_reference_voltage_source_t`

Define enumeration of reference voltage source.

For detail information, need to check the SoC's specification.

`typedef enum _lpadc_power_level_mode` `lpadc_power_level_mode_t`

Define enumeration of power configuration.

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

`typedef enum _lpadc_offset_calibration_mode` `lpadc_offset_calibration_mode_t`

Define enumeration of offset calibration mode.

```
typedef enum _lpadc_trigger_priority_policy lpadc_trigger_priority_policy_t
```

Define enumeration of trigger priority policy.

This selection controls how higher priority triggers are handled.

Note: **kLPADC_TriggerPriorityPreemptSubsequently** is not available on some devices, mainly depends on the size of TPRICTRL field in CFG register.

```
typedef struct _lpadc_calibration_value lpadc_calibration_value_t
```

A structure of calibration value.

```
LPADC_CONVERSION_COMPLETE_TIMEOUT
```

Max loops to wait for LPADC conversion complete.

When doing calibration, driver will wait for the completion of conversion. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

```
LPADC_CALIBRATION_READY_TIMEOUT
```

Max loops to wait for LPADC calibration ready.

Before doing calibration, driver will wait for the calibration ready. This parameter defines how many loops to check the calibration ready. If defined as 0, driver will wait forever until ready.

```
LPADC_GAIN_CAL_READY_TIMEOUT
```

Max loops to wait for LPADC gain calibration GAIN_CAL ready.

Before doing calibration, driver will wait for the gain calibration GAIN_CAL ready. This parameter defines how many loops to check the gain calibration GAIN_CAL ready. If defined as 0, driver will wait forever until ready.

```
LPADC_GET_ACTIVE_COMMAND_STATUS(statusVal)
```

Define the MACRO function to get command status from status value.

The statusVal is the return value from LPADC_GetStatusFlags().

```
LPADC_GET_ACTIVE_TRIGGER_STATUE(statusVal)
```

Define the MACRO function to get trigger status from status value.

The statusVal is the return value from LPADC_GetStatusFlags().

```
void LPADC_Init(ADC_Type *base, const lpadc_config_t *config)
```

Initializes the LPADC module.

Parameters

- base – LPADC peripheral base address.
- config – Pointer to configuration structure. See “*lpadc_config_t*”.

```
void LPADC_GetDefaultConfig(lpadc_config_t *config)
```

Gets an available pre-defined settings for initial configuration.

This function initializes the converter configuration structure with an available settings. The default values are:

```
config->enableInDozeMode      = true;
config->enableAnalogPreliminary = false;
config->powerUpDelay           = 0x80;
config->referenceVoltageSource  = kLPADC_ReferenceVoltageAlt1;
config->powerLevelMode         = kLPADC_PowerLevelAlt1;
config->triggerPriorityPolicy   = kLPADC_TriggerPriorityPreemptImmediately;
```

(continues on next page)

(continued from previous page)

```

config->enableConvPause    = false;
config->convPauseDelay     = 0U;
config->FIFOWatermark      = 0U;

```

Parameters

- config – Pointer to configuration structure.

```
void LPADC_Deinit(ADC_Type *base)
```

De-initializes the LPADC module.

Parameters

- base – LPADC peripheral base address.

```
static inline void LPADC_Enable(ADC_Type *base, bool enable)
```

Switch on/off the LPADC module.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the module.

```
static inline void LPADC_DoResetFIFO(ADC_Type *base)
```

Do reset the conversion FIFO.

Parameters

- base – LPADC peripheral base address.

```
static inline void LPADC_DoResetConfig(ADC_Type *base)
```

Do reset the module's configuration.

Reset all ADC internal logic and registers, except the Control Register (ADCx_CTRL).

Parameters

- base – LPADC peripheral base address.

```
static inline uint32_t LPADC_GetStatusFlags(ADC_Type *base)
```

Get status flags.

Parameters

- base – LPADC peripheral base address.

Returns

status flags' mask. See to `_lpadc_status_flags`.

```
static inline void LPADC_ClearStatusFlags(ADC_Type *base, uint32_t mask)
```

Clear status flags.

Only the flags can be cleared by writing ADCx_STATUS register would be cleared by this API.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for flags to be cleared. See to `_lpadc_status_flags`.

```
static inline uint32_t LPADC_GetTriggerStatusFlags(ADC_Type *base)
```

Get trigger status flags to indicate which trigger sequences have been completed or interrupted by a high priority trigger exception.

Parameters

- base – LPADC peripheral base address.

Returns

The OR'ed value of `_lpadc_trigger_status_flags`.

```
static inline void LPADC_ClearTriggerStatusFlags(ADC_Type *base, uint32_t mask)
```

Clear trigger status flags.

Parameters

- base – LPADC peripheral base address.
- mask – The mask of trigger status flags to be cleared, should be the OR'ed value of `_lpadc_trigger_status_flags`.

```
static inline void LPADC_EnableInterrupts(ADC_Type *base, uint32_t mask)
```

Enable interrupts.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to `_lpadc_interrupt_enable`.

```
static inline void LPADC_DisableInterrupts(ADC_Type *base, uint32_t mask)
```

Disable interrupts.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to `_lpadc_interrupt_enable`.

```
static inline void LPADC_EnableFIFOWatermarkDMA(ADC_Type *base, bool enable)
```

Switch on/off the DMA trigger for FIFO watermark event.

Parameters

- base – LPADC peripheral base address.
- enable – Switcher to the event.

```
static inline uint32_t LPADC_GetConvResultCount(ADC_Type *base)
```

Get the count of result kept in conversion FIFO.

Parameters

- base – LPADC peripheral base address.

Returns

The count of result kept in conversion FIFO.

```
bool LPADC_GetConvResult(ADC_Type *base, lpadc_conv_result_t *result)
```

Get the result in conversion FIFO.

Parameters

- base – LPADC peripheral base address.
- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

Returns

Status whether FIFO entry is valid.

```
void LPADC_GetConvResultBlocking(ADC_Type *base, lpadc_conv_result_t *result)
```

Get the result in conversion FIFO using blocking method.

Parameters

- base – LPADC peripheral base address.

- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

```
void LPADC_SetConvTriggerConfig(ADC_Type *base, uint32_t triggerId, const
                                lpadc_conv_trigger_config_t *config)
```

Configure the conversion trigger source.

Each programmable trigger can launch the conversion command in command buffer.

Parameters

- base – LPADC peripheral base address.
- triggerId – ID for each trigger. Typically, the available value range is from 0.
- config – Pointer to configuration structure. See to `lpadc_conv_trigger_config_t`.

```
void LPADC_GetDefaultConvTriggerConfig(lpadc_conv_trigger_config_t *config)
```

Gets an available pre-defined settings for trigger's configuration.

This function initializes the trigger's configuration structure with an available settings. The default values are:

```
config->targetCommandId    = 0U;
config->delayPower         = 0U;
config->priority            = 0U;
config->channelAFIFOSelect  = 0U;
config->channelBFIFOSelect  = 0U;
config->enableHardwareTrigger = false;
```

Parameters

- config – Pointer to configuration structure.

```
static inline void LPADC_DoSoftwareTrigger(ADC_Type *base, uint32_t triggerIdMask)
```

Do software trigger to conversion command.

Parameters

- base – LPADC peripheral base address.
- triggerIdMask – Mask value for software trigger indexes, which count from zero.

```
static inline void LPADC_EnableHardwareTriggerCommandSelection(ADC_Type *base, uint32_t
                                                                triggerId, bool enable)
```

Enable hardware trigger command selection.

This function will use the hardware trigger command from ADC_ETC. The trigger command is then defined by ADC hardware trigger command selection field in ADC_ETC->TRIGx_CHAINy_z_n[CSEL].

Parameters

- base – LPADC peripheral base address.
- triggerId – ID for each trigger. Typically, the available value range is from 0.
- enable – True to enable or false to disable.

```
void LPADC_SetConvCommandConfig(ADC_Type *base, uint32_t commandId, const
                                lpadc_conv_command_config_t *config)
```


Configure conversion command.

Note: The number of compare value register on different chips is different, that is mean in some chips, some command buffers do not have the compare functionality.

Parameters

- `base` – LPADC peripheral base address.
- `commandId` – ID for command in command buffer. Typically, the available value range is 1 - 15.
- `config` – Pointer to configuration structure. See to `lpadc_conv_command_config_t`.

`void LPADC_GetDefaultConvCommandConfig(lpadc_conv_command_config_t *config)`

Gets an available pre-defined settings for conversion command's configuration.

This function initializes the conversion command's configuration structure with an available settings. The default values are:

```
config->sampleScaleMode      = kLPADC_SampleFullScale;
config->channelBScaleMode    = kLPADC_SampleFullScale;
config->sampleChannelMode    = kLPADC_SampleChannelSingleEndSideA;
config->channelNumber        = 0U;
config->channelBNumber       = 0U;
config->chainedNextCommandNumber = 0U;
config->enableAutoChannelIncrement = false;
config->loopCount            = 0U;
config->hardwareAverageMode    = kLPADC_HardwareAverageCount1;
config->sampleTimeMode        = kLPADC_SampleTimeADCK3;
config->hardwareCompareMode    = kLPADC_HardwareCompareDisabled;
config->hardwareCompareValueHigh = 0U;
config->hardwareCompareValueLow  = 0U;
config->conversionResolutionMode = kLPADC_ConversionResolutionStandard;
config->enableWaitTrigger      = false;
config->enableChannelB        = false;
```

Parameters

- `config` – Pointer to configuration structure.

`void LPADC_EnableCalibration(ADC_Type *base, bool enable)`

Enable the calibration function.

When CALOFS is set, the ADC is configured to perform a calibration function anytime the ADC executes a conversion. Any channel selected is ignored and the value returned in the RESFIFO is a signed value between -31 and 31. -32 is not a valid and is never a returned value. Software should copy the lower 6- bits of the conversion result stored in the RESFIFO after a completed calibration conversion to the OFSTRIM field. The OFSTRIM field is used in normal operation for offset correction.

Parameters

- `base` – LPADC peripheral base address.
- `enable` – switcher to the calibration function.

`static inline void LPADC_SetOffsetValue(ADC_Type *base, uint32_t value)`

Set proper offset value to trim ADC.

To minimize the offset during normal operation, software should read the conversion result from the RESFIFO calibration operation and write the lower 6 bits to the OFSTRIM register.

Parameters

- base – LPADC peripheral base address.
- value – Setting offset value.

status_t LPADC_DoAutoCalibration(ADC_Type *base)

Do auto calibration.

Calibration function should be executed before using converter in application. It used the software trigger and a dummy conversion, get the offset and write them into the OFSTRIM register. It called some of functional API including: -LPADC_EnableCalibration(...) -LPADC_LPADC_SetOffsetValue(...) -LPADC_SetConvCommandConfig(...) -LPADC_SetConvTriggerConfig(...)

Parameters

- base – LPADC peripheral base address.
- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

static inline void LPADC_EnableOffsetCalibration(ADC_Type *base, bool enable)

Enable the offset calibration function.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the calibration function.

static inline void LPADC_SetOffsetCalibrationMode(ADC_Type *base,
lpadc_offset_calibration_mode_t mode)

Set offset calibration mode.

Parameters

- base – LPADC peripheral base address.
- mode – set offset calibration mode.see to lpadc_offset_calibration_mode_t .

status_t LPADC_DoOffsetCalibration(ADC_Type *base)

Do offset calibration.

Parameters

- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

void LPADC_PrepareAutoCalibration(ADC_Type *base)

Prepare auto calibration, LPADC_FinishAutoCalibration has to be called before using the LPADC. LPADC_DoAutoCalibration has been split in two API to avoid to be stuck too long in the function.

Parameters

- base – LPADC peripheral base address.

status_t LPADC_FinishAutoCalibration(ADC_Type *base)

Finish auto calibration start with LPADC_PrepareAutoCalibration.

Note: This feature is used for LPADC with CTRL[CALOFSMODE].

Parameters

- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

void LPADC_GetCalibrationValue(ADC_Type *base, *lpadc_calibration_value_t* *ptrCalibrationValue)

Get calibration value into the memory which is defined by invoker.

Note: Please note the ADC will be disabled temporary.

Note: This function should be used after finish calibration.

Parameters

- base – LPADC peripheral base address.
- ptrCalibrationValue – Pointer to *lpadc_calibration_value_t* structure, this memory block should be always powered on even in low power modes.

status_t LPADC_SetCalibrationValue(ADC_Type *base, const *lpadc_calibration_value_t* *ptrCalibrationValue)

Set calibration value into ADC calibration registers.

Note: Please note the ADC will be disabled temporary.

Parameters

- base – LPADC peripheral base address.
- ptrCalibrationValue – Pointer to *lpadc_calibration_value_t* structure which contains ADC's calibration value.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

FSL_LPADC_DRIVER_VERSION

LPADC driver version 2.9.4.

struct *lpadc_config_t*

#include <fsl_lpadc.h> LPADC global configuration.

This structure would used to keep the settings for initialization.

Public Members

bool enableInternalClock

Enables the internally generated clock source. The clock source is used in clock selection logic at the chip level and is optionally used for the ADC clock source.

bool enableVref1LowVoltage

If voltage reference option1 input is below 1.8V, it should be “true”. If voltage reference option1 input is above 1.8V, it should be “false”.

bool enableInDozeMode

Control system transition to Stop and Wait power modes while ADC is converting. When enabled in Doze mode, immediate entries to Wait or Stop are allowed. When disabled, the ADC will wait for the current averaging iteration/FIFO storage to complete before acknowledging stop or wait mode entry.

lpadc_conversion_average_mode_t conversionAverageMode

Auto-Calibration Averages.

bool enableAnalogPreliminary

ADC analog circuits are pre-enabled and ready to execute conversions without startup delays(at the cost of higher DC current consumption).

uint32_t powerUpDelay

When the analog circuits are not pre-enabled, the ADC analog circuits are only powered while the ADC is active and there is a counted delay defined by this field after an initial trigger transitions the ADC from its Idle state to allow time for the analog circuits to stabilize. The startup delay count of (powerUpDelay * 4) ADCK cycles must result in a longer delay than the analog startup time.

lpadc_reference_voltage_source_t referenceVoltageSource

Selects the voltage reference high used for conversions.

lpadc_power_level_mode_t powerLevelMode

Power Configuration Selection.

lpadc_trigger_priority_policy_t triggerPriorityPolicy

Control how higher priority triggers are handled, see to *lpadc_trigger_priority_policy_t*.

bool enableConvPause

Enables the ADC pausing function. When enabled, a programmable delay is inserted during command execution sequencing between LOOP iterations, between commands in a sequence, and between conversions when command is executing in “Compare Until True” configuration.

uint32_t convPauseDelay

Controls the duration of pausing during command execution sequencing. The pause delay is a count of (convPauseDelay*4) ADCK cycles. Only available when ADC pausing function is enabled. The available value range is in 9-bit.

uint32_t FIFOWatermark

FIFOWatermark is a programmable threshold setting. When the number of datawords stored in the ADC Result FIFO is greater than the value in this field, the ready flag would be asserted to indicate stored data has reached the programmable threshold.

struct lpadc_conv_command_config_t

#include <fsl_lpadc.h> Define structure to keep the configuration for conversion command.

Public Members

lpadc_sample_scale_mode_t sampleScaleMode

Sample scale mode.

lpadc_sample_scale_mode_t channelBScaleMode

Alternate channel B Scale mode.

lpadc_sample_channel_mode_t sampleChannelMode

Channel sample mode.

uint32_t channelNumber

Channel number; select the channel or channel pair.

uint32_t channelBNumber

Alternate Channel B number; select the channel.

uint32_t chainedNextCommandNumber

Selects the next command to be executed after this command completes. 1-15 is available, 0 is to terminate the chain after this command.

bool enableAutoChannelIncrement

Loop with increment: when disabled, the “loopCount” field selects the number of times the selected channel is converted consecutively; when enabled, the “loopCount” field defines how many consecutive channels are converted as part of the command execution.

uint32_t loopCount

Selects how many times this command executes before finish and transition to the next command or Idle state. Command executes LOOP+1 times. 0-15 is available.

lpadc_hardware_average_mode_t hardwareAverageMode

Hardware average selection.

lpadc_sample_time_mode_t sampleTimeMode

Sample time selection.

lpadc_hardware_compare_mode_t hardwareCompareMode

Hardware compare selection.

uint32_t hardwareCompareValueHigh

Compare Value High. The available value range is in 16-bit.

uint32_t hardwareCompareValueLow

Compare Value Low. The available value range is in 16-bit.

lpadc_conversion_resolution_mode_t conversionResolutionMode

Conversion resolution mode.

bool enableWaitTrigger

Wait for trigger assertion before execution: when disabled, this command will be automatically executed; when enabled, the active trigger must be asserted again before executing this command.

struct *lpadc_conv_trigger_config_t*

#include <fsl_lpadc.h> Define structure to keep the configuration for conversion trigger.

Public Members

uint32_t targetCommandId

Select the command from command buffer to execute upon detect of the associated trigger event.

uint32_t delayPower

Select the trigger delay duration to wait at the start of servicing a trigger event. When this field is clear, then no delay is incurred. When this field is set to a non-zero value, the duration for the delay is $2^{\text{delayPower}}$ ADCK cycles. The available value range is 4-bit.

uint32_t priority

Sets the priority of the associated trigger source. If two or more triggers have the same priority level setting, the lower order trigger event has the higher priority. The lower value for this field is for the higher priority, the available value range is 1-bit.

bool enableHardwareTrigger

Enable hardware trigger source to initiate conversion on the rising edge of the input trigger source or not. The software trigger is always available.

struct lpadc_conv_result_t

#include <fsl_lpadc.h> Define the structure to keep the conversion result.

Public Members

uint32_t commandIdSource

Indicate the command buffer being executed that generated this result.

uint32_t loopCountIndex

Indicate the loop count value during command execution that generated this result.

uint32_t triggerIdSource

Indicate the trigger source that initiated a conversion and generated this result.

uint16_t convValue

Data result.

struct __lpadc_calibration_value

#include <fsl_lpadc.h> A structure of calibration value.

2.67 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.

2.68 LPI2C Master Driver

```
void LPI2C_MasterGetDefaultConfig(lpi2c_master_config_t *masterConfig)
```

Provides a default configuration for the LPI2C master peripheral.

This function provides the following default configuration for the LPI2C master peripheral:

```
masterConfig->enableMaster      = true;
masterConfig->debugEnable       = false;
masterConfig->ignoreAck         = false;
masterConfig->pinConfig         = kLPI2C_2PinOpenDrain;
masterConfig->baudRate_Hz       = 100000U;
masterConfig->busIdleTimeout_ns = 0;
masterConfig->pinLowTimeout_ns  = 0;
masterConfig->sdaGlitchFilterWidth_ns = 0;
masterConfig->sclGlitchFilterWidth_ns = 0;
masterConfig->hostRequest.enable = false;
masterConfig->hostRequest.source  = kLPI2C_HostRequestExternalPin;
masterConfig->hostRequest.polarity = kLPI2C_HostRequestPinActiveHigh;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with LPI2C_MasterInit().

Parameters

- masterConfig – **[out]** User provided configuration structure for default values. Refer to *lpi2c_master_config_t*.

```
void LPI2C_MasterInit(LPI2C_Type *base, const lpi2c_master_config_t *masterConfig, uint32_t
    sourceClock_Hz)
```

Initializes the LPI2C master peripheral.

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- base – The LPI2C peripheral base address.
- masterConfig – User provided peripheral configuration. Use LPI2C_MasterGetDefaultConfig() to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

```
void LPI2C_MasterDeinit(LPI2C_Type *base)
```

Deinitializes the LPI2C master peripheral.

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

```
void LPI2C_MasterConfigureDataMatch(LPI2C_Type *base, const lpi2c_data_match_config_t
    *matchConfig)
```

Configures LPI2C master data match feature.

Parameters

- base – The LPI2C peripheral base address.
- matchConfig – Settings for the data match feature.

status_t LPI2C_MasterCheckAndClearError(LPI2C_Type *base, uint32_t status)

Convert provided flags to status code, and clear any errors if present.

Parameters

- base – The LPI2C peripheral base address.
- status – Current status flags value that will be checked.

Return values

- kStatus_Success –
- kStatus_LPI2C_PinLowTimeout –
- kStatus_LPI2C_ArbitrationLost –
- kStatus_LPI2C_Nak –
- kStatus_LPI2C_FifoError –

status_t LPI2C_CheckForBusyBus(LPI2C_Type *base)

Make sure the bus isn't already busy.

A busy bus is allowed if we are the one driving it.

Parameters

- base – The LPI2C peripheral base address.

Return values

- kStatus_Success –
- kStatus_LPI2C_Busy –

static inline void LPI2C_MasterReset(LPI2C_Type *base)

Performs a software reset.

Restores the LPI2C master peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_MasterEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as master.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as master.

static inline uint32_t LPI2C_MasterGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C master status flags.

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_lpi2c_master_flags`

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_MasterClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C master status flag state.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

Attempts to clear other flags has no effect.

See also:

`_lpi2c_master_flags`.

Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of `_lpi2c_master_flags` enumerators OR'd together. You may pass the result of a previous call to `LPI2C_MasterGetStatusFlags()`.

static inline void LPI2C_MasterEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Enables the LPI2C master interrupt requests.

All flags except `kLPI2C_MasterBusyFlag` and `kLPI2C_MasterBusBusyFlag` can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See `_lpi2c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline void LPI2C_MasterDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Disables the LPI2C master interrupt requests.

All flags except `kLPI2C_MasterBusyFlag` and `kLPI2C_MasterBusBusyFlag` can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See `_lpi2c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t LPI2C_MasterGetEnabledInterrupts(LPI2C_Type *base)

Returns the set of currently enabled LPI2C master interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of `_lpi2c_master_flags` enumerators OR'd together to indicate the set of enabled interrupts.

static inline void LPI2C_MasterEnableDMA(LPI2C_Type *base, bool enableTx, bool enableRx)
Enables or disables LPI2C master DMA requests.

Parameters

- `base` – The LPI2C peripheral base address.
- `enableTx` – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- `enableRx` – Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t LPI2C_MasterGetTxFifoAddress(LPI2C_Type *base)
Gets LPI2C master transmit data register address for DMA transfer.

Parameters

- `base` – The LPI2C peripheral base address.

Returns

The LPI2C Master Transmit Data Register address.

static inline uint32_t LPI2C_MasterGetRxFifoAddress(LPI2C_Type *base)
Gets LPI2C master receive data register address for DMA transfer.

Parameters

- `base` – The LPI2C peripheral base address.

Returns

The LPI2C Master Receive Data Register address.

static inline void LPI2C_MasterSetWatermarks(LPI2C_Type *base, size_t txWords, size_t rxWords)
Sets the watermarks for LPI2C master FIFOs.

Parameters

- `base` – The LPI2C peripheral base address.
- `txWords` – Transmit FIFO watermark value in words. The `kLPI2C_MasterTxReadyFlag` flag is set whenever the number of words in the transmit FIFO is equal or less than `txWords`. Writing a value equal or greater than the FIFO size is truncated.
- `rxWords` – Receive FIFO watermark value in words. The `kLPI2C_MasterRxReadyFlag` flag is set whenever the number of words in the receive FIFO is greater than `rxWords`. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPI2C_MasterGetFifoCounts(LPI2C_Type *base, size_t *rxCount, size_t *txCount)
Gets the current number of words in the LPI2C master FIFOs.

Parameters

- `base` – The LPI2C peripheral base address.
- `txCount` – **[out]** Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.

```
void LPI2C_MasterSetBaudRate(LPI2C_Type *base, uint32_t sourceClock_Hz, uint32_t
                             baudRate_Hz)
```

Sets the I2C bus frequency for master transactions.

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Note: Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

Parameters

- `base` – The LPI2C peripheral base address.
- `sourceClock_Hz` – LPI2C functional clock frequency in Hertz.
- `baudRate_Hz` – Requested bus frequency in Hertz.

```
static inline bool LPI2C_MasterGetBusIdleState(LPI2C_Type *base)
```

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- `base` – The LPI2C peripheral base address.

Return values

- `true` – Bus is busy.
- `false` – Bus is idle.

```
status_t LPI2C_MasterStart(LPI2C_Type *base, uint8_t address, lpi2c_direction_t dir)
```

Sends a START signal and slave address on the I2C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- `base` – The LPI2C peripheral base address.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kLPI2C_Read` or `kLPI2C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- `kStatus_Success` – START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.

```
static inline status_t LPI2C_MasterRepeatedStart(LPI2C_Type *base, uint8_t address,
                                                  lpi2c_direction_t dir)
```

Sends a repeated START signal and slave address on the I2C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like `LPI2C_MasterStart()`, it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- `base` – The LPI2C peripheral base address.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kLPI2C_Read` or `kLPI2C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- `kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.

status_t LPI2C_MasterSend(LPI2C_Type *base, void *txBuff, size_t txSize)

Performs a polling send transfer on the I2C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns `kStatus_LPI2C_Nak`.

Parameters

- `base` – The LPI2C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.

Return values

- `kStatus_Success` – Data was sent successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or over run.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- `base` – The LPI2C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.

Return values

- `kStatus_Success` – Data was received successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.

- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

`status_t LPI2C_MasterStop(LPI2C_Type *base)`

Sends a STOP signal on the I2C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- `base` – The LPI2C peripheral base address.

Return values

- `kStatus_Success` – The STOP signal was successfully sent on the bus and the transaction terminated.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

`status_t LPI2C_MasterTransferBlocking(LPI2C_Type *base, lpi2c_master_transfer_t *transfer)`

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- `base` – The LPI2C peripheral base address.
- `transfer` – Pointer to the transfer structure.

Return values

- `kStatus_Success` – Data was received successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

`void LPI2C_MasterTransferCreateHandle(LPI2C_Type *base, lpi2c_master_handle_t *handle, lpi2c_master_transfer_callback_t callback, void *userData)`

Creates a new handle for the LPI2C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `LPI2C_MasterTransferAbort()` API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_MasterTransferNonBlocking(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_t* *transfer)

Performs a non-blocking transaction on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t LPI2C_MasterTransferGetCount(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *size_t* *count)

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void LPI2C_MasterTransferAbort(LPI2C_Type *base, *lpi2c_master_handle_t* *handle)

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

void LPI2C_MasterTransferHandleIRQ(LPI2C_Type *base, void *lpi2cMasterHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The LPI2C peripheral base address.
- lpi2cMasterHandle – Pointer to the LPI2C master driver handle.

enum _lpi2c_master_flags

LPI2C master peripheral flags.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_MasterTxReadyFlag

Transmit data flag

enumerator kLPI2C_MasterRxReadyFlag

Receive data flag

enumerator kLPI2C_MasterEndOfPacketFlag

End Packet flag

enumerator kLPI2C_MasterStopDetectFlag

Stop detect flag

enumerator kLPI2C_MasterNackDetectFlag

NACK detect flag

enumerator kLPI2C_MasterArbitrationLostFlag

Arbitration lost flag

enumerator kLPI2C_MasterFifoErrFlag

FIFO error flag

enumerator kLPI2C_MasterPinLowTimeoutFlag

Pin low timeout flag

enumerator kLPI2C_MasterDataMatchFlag

Data match flag

enumerator kLPI2C_MasterBusyFlag

Master busy flag

enumerator kLPI2C_MasterBusBusyFlag

Bus busy flag

enumerator kLPI2C_MasterClearFlags

All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_MasterIrqFlags

IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_MasterErrorFlags

Errors to check for.

enum _lpi2c_direction

Direction of master and slave transfers.

Values:

enumerator kLPI2C_Write

Master transmit.

enumerator kLPI2C_Read

Master receive.

enum _lpi2c_master_pin_config

LPI2C pin configuration.

Values:

enumerator kLPI2C_2PinOpenDrain

LPI2C Configured for 2-pin open drain mode

enumerator kLPI2C_2PinOutputOnly

LPI2C Configured for 2-pin output only mode (ultra-fast mode)

enumerator kLPI2C_2PinPushPull

LPI2C Configured for 2-pin push-pull mode

enumerator kLPI2C_4PinPushPull

LPI2C Configured for 4-pin push-pull mode

enumerator kLPI2C_2PinOpenDrainWithSeparateSlave

LPI2C Configured for 2-pin open drain mode with separate LPI2C slave

enumerator kLPI2C_2PinOutputOnlyWithSeparateSlave

LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave

enumerator kLPI2C_2PinPushPullWithSeparateSlave

LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave

enumerator kLPI2C_4PinPushPullWithInvertedOutput

LPI2C Configured for 4-pin push-pull mode(inverted outputs)

enum _lpi2c_host_request_source

LPI2C master host request selection.

Values:

enumerator kLPI2C_HostRequestExternalPin

Select the LPI2C_HREQ pin as the host request input

enumerator kLPI2C_HostRequestInputTrigger

Select the input trigger as the host request input

enum _lpi2c_host_request_polarity

LPI2C master host request pin polarity configuration.

Values:

enumerator kLPI2C_HostRequestPinActiveLow

Configure the LPI2C_HREQ pin active low

enumerator kLPI2C_HostRequestPinActiveHigh

Configure the LPI2C_HREQ pin active high

enum _lpi2c_data_match_config_mode

LPI2C master data match configuration modes.

Values:

enumerator kLPI2C_MatchDisabled

LPI2C Match Disabled

enumerator kLPI2C_1stWordEqualsM0OrM1

LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1

enumerator kLPI2C_AnyWordEqualsM0OrM1

LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1

enumerator kLPI2C_1stWordEqualsM0And2ndWordEqualsM1

LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1

enumerator kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1

LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1

enumerator kLPI2C_1stWordAndM1EqualsM0AndM1

LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1

enumerator kLPI2C_AnyWordAndM1EqualsM0AndM1

LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1

enum _lpi2c_master_transfer_flags

Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the `_lpi2c_master_transfer::flags` field.

Values:

enumerator kLPI2C_TransferDefaultFlag

Transfer starts with a start signal, stops with a stop signal.

enumerator kLPI2C_TransferNoStartFlag

Don't send a start condition, address, and sub address

enumerator kLPI2C_TransferRepeatedStartFlag

Send a repeated start condition

enumerator kLPI2C_TransferNoStopFlag

Don't send a stop condition.

```
typedef enum _lpi2c_direction lpi2c_direction_t
```

Direction of master and slave transfers.

```
typedef enum _lpi2c_master_pin_config lpi2c_master_pin_config_t
```

LPI2C pin configuration.

```
typedef enum _lpi2c_host_request_source lpi2c_host_request_source_t
```

LPI2C master host request selection.

```
typedef enum _lpi2c_host_request_polarity lpi2c_host_request_polarity_t
```

LPI2C master host request pin polarity configuration.

```
typedef struct _lpi2c_master_config lpi2c_master_config_t
```

Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

```
typedef enum _lpi2c_data_match_config_mode lpi2c_data_match_config_mode_t
```

LPI2C master data match configuration modes.

```
typedef struct _lpi2c_match_config lpi2c_data_match_config_t
```

LPI2C master data match configuration structure.

```
typedef struct _lpi2c_master_transfer lpi2c_master_transfer_t
```

LPI2C master descriptor of the transfer.

```
typedef struct _lpi2c_master_handle lpi2c_master_handle_t
```

LPI2C master handle of the transfer.

```
typedef void (*lpi2c_master_transfer_callback_t)(LPI2C_Type *base, lpi2c_master_handle_t  
*handle, status_t completionStatus, void *userData)
```

Master completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C_MasterTransferCreateHandle().

Param base

The LPI2C peripheral base address.

Param handle

Pointer to the LPI2C master driver handle.

Param completionStatus

Either kStatus_Success or an error code describing how the transfer completed.

Param userData

Arbitrary pointer-sized value passed from the application.

```
typedef void (*lpi2c_master_isr_t)(LPI2C_Type *base, void *handle)
```

Typedef for master interrupt handler, used internally for LPI2C master interrupt and EDMA transactional APIs.

```
struct _lpi2c_master_config
```

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableMaster

Whether to enable master mode.

bool enableDoze

Whether master is enabled in doze mode.

bool debugEnable

Enable transfers to continue when halted in debug mode.

bool ignoreAck

Whether to ignore ACK/NACK.

lpi2c_master_pin_config_t pinConfig

The pin configuration option.

uint32_t baudRate_Hz

Desired baud rate in Hertz.

uint32_t busIdleTimeout_ns

Bus idle timeout in nanoseconds. Set to 0 to disable.

uint32_t pinLowTimeout_ns

Pin low timeout in nanoseconds. Set to 0 to disable.

uint8_t sdaGlitchFilterWidth_ns

Width in nanoseconds of glitch filter on SDA pin. Set to 0 to disable.

uint8_t sclGlitchFilterWidth_ns

Width in nanoseconds of glitch filter on SCL pin. Set to 0 to disable.

struct *_lpi2c_master_config* hostRequest

Host request options.

struct *_lpi2c_match_config*

#include <fsl_lpi2c.h> LPI2C master data match configuration structure.

Public Members

lpi2c_data_match_config_mode_t matchMode

Data match configuration setting.

bool rxDataMatchOnly

When set to true, received data is ignored until a successful match.

uint32_t match0

Match value 0.

uint32_t match1

Match value 1.

struct *_lpi2c_master_transfer*

#include <fsl_lpi2c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the LPI2C_MasterTransferNonBlocking() API.

Public Members

uint32_t flags

Bit mask of options for the transfer. See enumeration `_lpi2c_master_transfer_flags` for available options. Set to 0 or `kLPI2C_TransferDefaultFlag` for normal transfers.

uint16_t slaveAddress

The 7-bit slave address.

lpi2c_direction_t direction

Either `kLPI2C_Read` or `kLPI2C_Write`.

uint32_t subaddress

Sub address. Transferred MSB first.

size_t subaddressSize

Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data

Pointer to data to transfer.

size_t dataSize

Number of bytes to transfer.

struct `_lpi2c_master_handle`

#include <fsl_lpi2c.h> Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state

Transfer state machine current state.

uint16_t remainingBytes

Remaining byte count in current state.

uint8_t *buf

Buffer pointer for current state.

uint16_t commandBuffer[6]

LPI2C command sequence. When all 6 command words are used: `Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word]`

lpi2c_master_transfer_t transfer

Copy of the current transfer info.

lpi2c_master_transfer_callback_t completionCallback

Callback function pointer.

void *userData

Application data passed to callback.

struct hostRequest

Public Members

bool enable

Enable host request.

lpi2c_host_request_source_t source

Host request source.

lpi2c_host_request_polarity_t polarity

Host request pin polarity.

2.69 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.

2.70 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;
```

(continues on next page)

(continued from previous page)

```

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 enableRxDatasCLStall 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.

2.71 LPSPI: Low Power Serial Peripheral Interface

2.72 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 kLPSPIClockPolarityActiveHigh

CPOL=0. Active-high LPSPIClock (idles low)

enumerator kLPSPIClockPolarityActiveLow

CPOL=1. Active-low LPSPIClock (idles high)

enum _lpspi_clock_phase

LPSPIClock phase configuration.

Values:

enumerator kLPSPIClockPhaseFirstEdge

CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.

enumerator kLPSPIClockPhaseSecondEdge

CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

enum _lpspi_shift_direction

LPSPIData shifter direction options.

Values:

enumerator kLPSPIMsbFirst

Data transfers start with most significant bit.

enumerator kLPSPILsbFirst

Data transfers start with least significant bit.

enum _lpspi_host_request_select

LPSPIClock Host Request select configuration.

Values:

enumerator kLPSPIClockHostReqExtPin

Host Request is an ext pin.

enumerator kLPSPIClockHostReqInternalTrigger

Host Request is an internal trigger.

enum _lpspi_match_config

LPSPIMatch configuration options.

Values:

enumerator kLPSPIMatchDisabled

LPSPIMatch Disabled.

enumerator kLPSPIMatch1stWordEqualsM0orM1

LPSPIMatch Enabled.

enumerator kLPSPIMatchAnyWordEqualsM0orM1

LPSPIMatch Enabled.

enumerator kLPSPIMatch1stWordEqualsM0and2ndWordEqualsM1

LPSPIMatch Enabled.

enumerator kLPSPIMatchAnyWordEqualsM0andNxtWordEqualsM1

LPSPIMatch 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 flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
- ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPi_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi_MasterByteSwap flag.
- iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPi_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi_MasterByteSwap flag.

enum _lpspi_transfer_config_flag_for_slave

Use this enumeration for LPSPi slave transfer configFlags.

Values:

enumerator kLPSPi_SlavePcs0

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS0 signal

enumerator kLPSPi_SlavePcs1

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS1 signal

enumerator kLPSPi_SlavePcs2

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS2 signal

enumerator kLPSPi_SlavePcs3

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS3 signal

enumerator kLPSPi_SlaveByteSwap

Is slave swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

- i. If you set bitPerFrame = 8 , no matter the kLPSPi_SlaveByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
- ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPi_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi_SlaveByteSwap flag.
- iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPi_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi_SlaveByteSwap flag.

enum `_lpspi_transfer_state`

LPSPI transfer state, which is used for LPSPI transactional API state machine.

Values:

enumerator `kLPSPi_Idle`

Nothing in the transmitter/receiver.

enumerator `kLPSPi_Busy`

Transfer queue is not finished.

enumerator `kLPSPi_Error`

Transfer error.

typedef enum `_lpspi_master_slave_mode` `lpspi_master_slave_mode_t`

LPSPI master or slave mode configuration.

typedef enum `_lpspi_which_pcs_config` `lpspi_which_pcs_t`

LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

typedef enum `_lpspi_pcs_polarity_config` `lpspi_pcs_polarity_config_t`

LPSPI Peripheral Chip Select (PCS) Polarity configuration.

typedef enum `_lpspi_clock_polarity` `lpspi_clock_polarity_t`

LPSPI clock polarity configuration.

typedef enum `_lpspi_clock_phase` `lpspi_clock_phase_t`

LPSPI clock phase configuration.

typedef enum `_lpspi_shift_direction` `lpspi_shift_direction_t`

LPSPI data shifter direction options.

typedef enum `_lpspi_host_request_select` `lpspi_host_request_select_t`

LPSPI Host Request select configuration.

typedef enum `_lpspi_match_config` `lpspi_match_config_t`

LPSPI Match configuration options.

typedef enum `_lpspi_pin_config` `lpspi_pin_config_t`

LPSPI pin (SDO and SDI) configuration.

typedef enum `_lpspi_data_out_config` `lpspi_data_out_config_t`

LPSPI data output configuration.

typedef enum `_lpspi_transfer_width` `lpspi_transfer_width_t`

LPSPI transfer width configuration.

typedef enum `_lpspi_delay_type` `lpspi_delay_type_t`

LPSPI delay type selection.

typedef struct `_lpspi_master_config` `lpspi_master_config_t`

LPSPI master configuration structure.

typedef struct `_lpspi_slave_config` `lpspi_slave_config_t`

LPSPI slave configuration structure.

typedef struct `_lpspi_master_handle` `lpspi_master_handle_t`

Forward declaration of the `_lpspi_master_handle` typedefs.

typedef struct `_lpspi_slave_handle` `lpspi_slave_handle_t`

Forward declaration of the `_lpspi_slave_handle` typedefs.

```
typedef void (*lpspi_master_transfer_callback_t)(LPSPI_Type *base, lpspi_master_handle_t
*handle, status_t status, void *userData)
```

Master completion callback function pointer type.

Param base

LPSPI peripheral address.

Param handle

Pointer to the handle for the LPSPI master.

Param status

Success or error code describing whether the transfer is completed.

Param userData

Arbitrary pointer-dataSized value passed from the application.

```
typedef void (*lpspi_slave_transfer_callback_t)(LPSPI_Type *base, lpspi_slave_handle_t *handle,
status_t status, void *userData)
```

Slave completion callback function pointer type.

Param base

LPSPI peripheral address.

Param handle

Pointer to the handle for the LPSPI slave.

Param status

Success or error code describing whether the transfer is completed.

Param userData

Arbitrary pointer-dataSized value passed from the application.

```
typedef struct _lpspi_transfer lpspi_transfer_t
```

LPSPI master/slave transfer structure.

```
volatile uint8_t g_lpspiDummyData[]
```

Global variable for dummy data value setting.

```
LPSPI_DUMMY_DATA
```

LPSPI dummy data if no Tx data.

Dummy data used for tx if there is not txData.

```
SPI_RETRY_TIMES
```

Retry times for waiting flag.

```
LPSPI_MASTER_PCS_SHIFT
```

LPSPI master PCS shift macro , internal used.

```
LPSPI_MASTER_PCS_MASK
```

LPSPI master PCS shift macro , internal used.

```
LPSPI_SLAVE_PCS_SHIFT
```

LPSPI slave PCS shift macro , internal used.

```
LPSPI_SLAVE_PCS_MASK
```

LPSPI slave PCS shift macro , internal used.

```
struct _lpspi_master_config
```

#include <fsl_lpspi.h> LPSPI master configuration structure.

Public Members

uint32_t baudRate

Baud Rate for LPSPI.

uint32_t bitsPerFrame

Bits per frame, minimum 8, maximum 4096.

lpspi_clock_polarity_t cpol

Clock polarity.

lpspi_clock_phase_t cpha

Clock phase.

lpspi_shift_direction_t direction

MSB or LSB data shift direction.

uint32_t pcsToSckDelayInNanoSec

PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32_t lastSckToPcsDelayInNanoSec

Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32_t betweenTransferDelayInNanoSec

After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

lpspi_which_pcs_t whichPcs

Desired Peripheral Chip Select (PCS).

lpspi_pcs_polarity_config_t pcsActiveHighOrLow

Desired PCS active high or low

lpspi_pin_config_t pinCfg

Configures which pins are used for input and output data during single bit transfers.

lpspi_data_out_config_t dataOutConfig

Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

bool enableInputDelay

Enable master to sample the input data on a delayed SCK. This can help improve slave setup time. Refer to device data sheet for specific time length.

struct __lpspi_slave_config

#include <fsl_lpspi.h> LPSPI slave configuration structure.

Public Members

uint32_t bitsPerFrame

Bits per frame, minimum 8, maximum 4096.

lpspi_clock_polarity_t cpol

Clock polarity.

lpspi_clock_phase_t cpha

Clock phase.

lpspi_shift_direction_t direction

MSB or LSB data shift direction.

lpspi_which_pcs_t whichPcs

Desired Peripheral Chip Select (pcs)

lpspi_pcs_polarity_config_t pcsActiveHighOrLow

Desired PCS active high or low

lpspi_pin_config_t pinCfg

Configures which pins are used for input and output data during single bit transfers.

lpspi_data_out_config_t dataOutConfig

Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

struct *_lpspi_transfer*

#include <fsl_lpspi.h> LPSPI master/slave transfer structure.

Public Members

const uint8_t *txData

Send buffer.

uint8_t *rxData

Receive buffer.

volatile size_t dataSize

Transfer bytes.

uint32_t configFlags

Transfer transfer configuration flags. Set from *_lpspi_transfer_config_flag_for_master* if the transfer is used for master or *_lpspi_transfer_config_flag_for_slave* enumeration if the transfer is used for slave.

struct *_lpspi_master_handle*

#include <fsl_lpspi.h> LPSPI master transfer handle structure used for transactional API.

Public Members

volatile bool isPcsContinuous

Is PCS continuous in transfer.

volatile bool writeTcrInIsr

A flag that whether should write TCR in ISR.

volatile bool isByteSwap

A flag that whether should byte swap.

volatile bool isTxMask

A flag that whether TCR[TXMSK] is set.

volatile uint16_t bytesPerFrame

Number of bytes in each frame

volatile uint16_t frameSize

Backup of TCR[FRAMESZ]

volatile uint8_t fifoSize

FIFO dataSize.

volatile uint8_t rxWatermark

Rx watermark.

`volatile uint8_t bytesEachWrite`
Bytes for each write TDR.

`volatile uint8_t bytesEachRead`
Bytes for each read RDR.

`const uint8_t *volatile txData`
Send buffer.

`uint8_t *volatile rxData`
Receive buffer.

`volatile size_t txRemainingByteCount`
Number of bytes remaining to send.

`volatile size_t rxRemainingByteCount`
Number of bytes remaining to receive.

`volatile uint32_t writeRegRemainingTimes`
Write TDR register remaining times.

`volatile uint32_t readRegRemainingTimes`
Read RDR register remaining times.

`uint32_t totalByteCount`
Number of transfer bytes

`uint32_t txBuffIfNull`
Used if the txData is NULL.

`volatile uint8_t state`
LPSPI transfer state , `_lpspi_transfer_state`.

`lpspi_master_transfer_callback_t callback`
Completion callback.

`void *userData`
Callback user data.

`struct _lpspi_slave_handle`
#include <fsl_lpspi.h> LPSPI slave transfer handle structure used for transactional API.

Public Members

`volatile bool isByteSwap`
A flag that whether should byte swap.

`volatile uint8_t fifoSize`
FIFO dataSize.

`volatile uint8_t rxWatermark`
Rx watermark.

`volatile uint8_t bytesEachWrite`
Bytes for each write TDR.

`volatile uint8_t bytesEachRead`
Bytes for each read RDR.

`const uint8_t *volatile txData`
Send buffer.

uint8_t *volatile rxData
Receive buffer.

volatile size_t txRemainingByteCount
Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
Read RDR register remaining times.

uint32_t totalByteCount
Number of transfer bytes

volatile uint8_t state
LPSPI transfer state , _lpspi_transfer_state.

volatile uint32_t errorCount
Error count for slave transfer.

lpspi_slave_transfer_callback_t callback
Completion callback.

void *userData
Callback user data.

2.73 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

2.74 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

2.75 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;
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 __unnamed76__`

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 __unnamed78__`

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 __unnamed80__`

Public Members

`uint8_t *volatile rxData`
Address of remaining data to receive.

`uint16_t *volatile rxData16`
Address of remaining data to receive.

`union __unnamed82__`

Public Members

uint8_t *rxRingBuffer

Start address of the receiver ring buffer.

uint16_t *rxRingBuffer16

Start address of the receiver ring buffer.

2.76 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

2.77 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.

2.78 MECC: internal error correction code

`void MECC_Init(MECC_Type *base, mecc_config_t *config)`

MECC module initialization function.

Parameters

- `base` – MECC base address.
- `config` – pointer to the MECC configuration structure.

`void MECC_Deinit(MECC_Type *base)`

Deinitializes the MECC.

Parameters

- `base` – MECC base address.

`void MECC_GetDefaultConfig(mecc_config_t *config)`

Sets the MECC configuration structure to default values.

Parameters

- `config` – pointer to the MECC configuration structure.

`static inline uint32_t MECC_GetStatusFlags(MECC_Type *base)`

Gets MECC status flags.

Parameters

- `base` – MECC peripheral base address.

Returns

MECC status flags.

`static inline void MECC_ClearStatusFlags(MECC_Type *base, uint32_t mask)`

MECC module clear interrupt status.

Parameters

- `base` – MECC base address.
- `mask` – status to clear.

`static inline void MECC_EnableInterruptStatus(MECC_Type *base, uint32_t mask)`

MECC module enable interrupt status.

Parameters

- `base` – MECC base address.
- `mask` – status to enable.

`static inline void MECC_DisableInterruptStatus(MECC_Type *base, uint32_t mask)`

MECC module disable interrupt status.

Parameters

- `base` – MECC base address.
- `mask` – status to disable.

`static inline void MECC_EnableInterrupts(MECC_Type *base, uint32_t mask)`

MECC module enable interrupt.

Parameters

- `base` – MECC base address.
- `mask` – The interrupts to enable.

```
static inline void MECC_DisableInterrupts(MECC_Type *base, uint32_t mask)
```

MECC module disable interrupt.

Parameters

- base – MECC base address.
- mask – The interrupts to disable.

```
status_t MECC_ErrorInjection(MECC_Type *base, uint32_t lowerrordata, uint32_t  
                             higherrordata, uint8_t eccdata, uint8_t banknumber)
```

MECC module error injection.

Bank0: ocram_base_address+0x20*i Bank1: ocram_base_address+0x20*i+0x8 Bank2:
ocram_base_address+0x20*i+0x10 Bank3: ocram_base_address+0x20*i+0x18 i =
0,1,2,3,4,....

Parameters

- base – MECC base address.
- lowerrordata – low 32 bits data.
- higherrordata – high 32 bits data.
- eccdata – ecc code.
- banknumber – ocram bank number.

Return values

kStatus_Success. –

```
status_t MECC_GetSingleErrorInfo(MECC_Type *base, mecc_single_error_info_t *info, uint8_t  
                                banknumber)
```

MECC module get single error information.

Bank0: ocram_base_address+0x20*i Bank1: ocram_base_address+0x20*i+0x8 Bank2:
ocram_base_address+0x20*i+0x10 Bank3: ocram_base_address+0x20*i+0x18 i =
0,1,2,3,4,....

Parameters

- base – MECC base address.
- info – single error information.
- banknumber – ocram bank number.

Return values

- kStatus_Success. –
- kStatus_MECC_BankMiss. –

```
status_t MECC_GetMultiErrorInfo(MECC_Type *base, mecc_multi_error_info_t *info, uint8_t  
                                banknumber)
```

MECC module get multiple error information.

Bank0: ocram_base_address+0x20*i Bank1: ocram_base_address+0x20*i+0x8 Bank2:
ocram_base_address+0x20*i+0x10 Bank3: ocram_base_address+0x20*i+0x18 i =
0,1,2,3,4,....

Parameters

- base – MECC base address.

- info – multiple error information.
- banknumber – ocrum bank number.

Return values

- kStatus_Success. –
- kStatus_MECC_BankMiss. –

```
static inline uint32_t MECC_GetPendingFlags(MECC_Type *base)
```

Get pending flags for OCRAM wait and pipeline enable.

Parameters

- base – MECC base address.

Returns

Pending flags, should be the OR'ed value of mecc_pending_flag_t.

FSL_MECC_DRIVER_VERSION

Driver version 2.1.0.

Error codes for the MECC driver.

Values:

```
enumerator kStatus_MECC_BankMiss
```

Ocrum bank miss

MECC interrupt configuration structure, default settings all disabled.

This structure contains the settings for all of the MECC interrupt configurations.

Values:

```
enumerator kMECC_SingleError0InterruptEnable
```

Single Bit Error On Ocrum Bank0 interrupt enable.

```
enumerator kMECC_SingleError1InterruptEnable
```

Single Bit Error On Ocrum Bank1 interrupt enable

```
enumerator kMECC_SingleError2InterruptEnable
```

Single Bit Error On Ocrum Bank2 interrupt enable

```
enumerator kMECC_SingleError3InterruptEnable
```

Single Bit Error On Ocrum Bank3 interrupt enable

```
enumerator kMECC_MultiError0InterruptEnable
```

Multiple Bits Error On Ocrum Bank0 interrupt enable

```
enumerator kMECC_MultiError1InterruptEnable
```

Multiple Bits Error On Ocrum Bank1 interrupt enable

```
enumerator kMECC_MultiError2InterruptEnable
```

Multiple Bits Error On Ocrum Bank2 interrupt enable

```
enumerator kMECC_MultiError3InterruptEnable
```

Multiple Bits Error On Ocrum Bank3 interrupt enable

```
enumerator kMECC_StrobeError0InterruptEnable
```

AXI Strobe Error On Ocrum Bank0 interrupt enable

```
enumerator kMECC_StrobeError1InterruptEnable
```

AXI Strobe Error On Ocrum Bank1 interrupt enable

enumerator kMECC_StrobeError2InterruptEnable
AXI Strobe Error On Ocram Bank2 interrupt enable

enumerator kMECC_StrobeError3InterruptEnable
AXI Strobe Error On Ocram Bank3 interrupt enable

enumerator kMECC_AccessError0InterruptEnable
Ocram Access Error On Bank0 interrupt enable

enumerator kMECC_AccessError1InterruptEnable
Ocram Access Error On Bank1 interrupt enable

enumerator kMECC_AccessError2InterruptEnable
Ocram Access Error On Bank2 interrupt enable

enumerator kMECC_AccessError3InterruptEnable
Ocram Access Error On Bank3 interrupt enable

enumerator kMECC_AllInterruptsEnable
all interrupts enable

MECC interrupt status configuration structure, default settings all disabled.

This structure contains the settings for all of the MECC interrupt status configurations.

Values:

enumerator kMECC_SingleError0InterruptStatusEnable
Single Bit Error On Ocram Bank0 interrupt status enable.

enumerator kMECC_SingleError1InterruptStatusEnable
Single Bit Error On Ocram Bank1 interrupt status enable

enumerator kMECC_SingleError2InterruptStatusEnable
Single Bit Error On Ocram Bank2 interrupt status enable

enumerator kMECC_SingleError3InterruptStatusEnable
Single Bit Error On Ocram Bank3 interrupt status enable

enumerator kMECC_MultiError0InterruptStatusEnable
Multiple Bits Error On Ocram Bank0 interrupt status enable

enumerator kMECC_MultiError1InterruptStatusEnable
Multiple Bits Error On Ocram Bank1 interrupt status enable

enumerator kMECC_MultiError2InterruptStatusEnable
Multiple Bits Error On Ocram Bank2 interrupt status enable

enumerator kMECC_MultiError3InterruptStatusEnable
Multiple Bits Error On Ocram Bank3 interrupt status enable

enumerator kMECC_StrobeError0InterruptStatusEnable
AXI Strobe Error On Ocram Bank0 interrupt status enable

enumerator kMECC_StrobeError1InterruptStatusEnable
AXI Strobe Error On Ocram Bank1 interrupt status enable

enumerator kMECC_StrobeError2InterruptStatusEnable
AXI Strobe Error On Ocram Bank2 interrupt status enable

enumerator kMECC_StrobeError3InterruptStatusEnable
AXI Strobe Error On Ocram Bank3 interrupt status enable

enumerator kMECC_AccessError0InterruptStatusEnable
Ocram Access Error On Bank0 interrupt status enable

enumerator kMECC_AccessError1InterruptStatusEnable
Ocram Access Error On Bank1 interrupt status enable

enumerator kMECC_AccessError2InterruptStatusEnable
Ocram Access Error On Bank2 interrupt status enable

enumerator kMECC_AccessError3InterruptStatusEnable
Ocram Access Error On Bank3 interrupt status enable

enumerator kMECC_AllInterruptsStatusEnable
all interrupts enable

MECC status flags.

This provides constants for the MECC status flags for use in the MECC functions.

Values:

enumerator kMECC_SingleError0InterruptFlag
Single Bit Error On Ocram Bank0 interrupt flag

enumerator kMECC_SingleError1InterruptFlag
Single Bit Error On Ocram Bank1 interrupt flag

enumerator kMECC_SingleError2InterruptFlag
Single Bit Error On Ocram Bank2 interrupt flag

enumerator kMECC_SingleError3InterruptFlag
Single Bit Error On Ocram Bank3 interrupt flag

enumerator kMECC_MultiError0InterruptFlag
Multiple Bits Error On Ocram Bank0 interrupt flag

enumerator kMECC_MultiError1InterruptFlag
Multiple Bits Error On Ocram Bank1 interrupt flag

enumerator kMECC_MultiError2InterruptFlag
Multiple Bits Error On Ocram Bank2 interrupt flag

enumerator kMECC_MultiError3InterruptFlag
Multiple Bits Error On Ocram Bank3 interrupt flag

enumerator kMECC_StrobeError0InterruptFlag
AXI Strobe Error On Ocram Bank0 interrupt flag

enumerator kMECC_StrobeError1InterruptFlag
AXI Strobe Error On Ocram Bank1 interrupt flag

enumerator kMECC_StrobeError2InterruptFlag
AXI Strobe Error On Ocram Bank2 interrupt flag

enumerator kMECC_StrobeError3InterruptFlag
AXI Strobe Error On Ocram Bank3 interrupt flag

enumerator kMECC_AccessError0InterruptFlag
Ocram Access Error On Bank0 interrupt flag

enumerator kMECC_AccessError1InterruptFlag
Ocram Access Error On Bank1 interrupt flag

enumerator kMECC_AccessError2InterruptFlag
Ocram Access Error On Bank2 interrupt flag

enumerator kMECC_AccessError3InterruptFlag
Ocram Access Error On Bank3 interrupt flag

enumerator kMECC_AllInterruptsFlag
all interrupts interrupt flag

MECC ocram bank number.

Values:

enumerator kMECC_OcramBank0
ocram bank number 0: ocram_base_address+0x20*i

enumerator kMECC_OcramBank1
ocram bank number 1: ocram_base_address+0x20*i+0x8

enumerator kMECC_OcramBank2
ocram bank number 2: ocram_base_address+0x20*i+0x10

enumerator kMECC_OcramBank3
ocram bank number 3: ocram_base_address+0x20*i+0x18

enum _mecc_pending_flag

Pending flags for OCRAM wait and pipeline enable. .

Values:

enumerator kMECC_ReadDataWaitPendingFlag
Indicate an update pending status for read data wait.

enumerator kMECC_ReadAddrPipelinePendingFlag
Indicate an update pending status for read address pipeline.

enumerator kMECC_WriteDataPipelinePendingFlag
Indicate an update pending status for write data pipeline.

enumerator kMECC_WriteAddrPipelinePendingFlag
Indicate an update pending status for write address pipeline.

enumerator kMECC_AllPendingFlags
Indicate all pending status flags.

typedef struct *_mecc_config* mecc_config_t
MECC user configuration.

Note: Ocram1StartAddress, Ocram1EndAddress, Ocram2StartAddress, Ocram2EndAddress are removed since 2.1.0 version; This changes will cause compile error for applications which use MECC driver before 2.1.0 version; To resolve compile error please use *startAddress* and *endAddress* as instead.

typedef struct *_mecc_single_error_info* mecc_single_error_info_t
MECC ocram single error information, including single error address, ECC code, error data and error bit position.

typedef struct *_mecc_multi_error_info* mecc_multi_error_info_t
MECC ocram multiple error information, including multiple error address, ECC code, error data.

```
struct _mecc_config
#include <fsl_mecc.h> MECC user configuration.
```

Note: Ocram1StartAddress, Ocram1EndAddress, Ocram2StartAddress, Ocram2EndAddress are removed since 2.1.0 version; This changes will cause compile error for applications which use MECC driver before 2.1.0 version; To resolve compile error please use *startAddress* and *endAddress* as instead.

Public Members

bool enableMecc
Enable the MECC function.

uint32_t startAddress
Start address of corresponding OCRAM memory region to enable ECC.

uint32_t endAddress
end address of corresponding OCRAM memory region to enable ECC.

bool enableReadDataWait
uint32_t Ocram1StartAddress; Ocram 1 start address, deprecated since 2.1.0
uint32_t Ocram1EndAddress; Ocram 1 end address, deprecated since 2.1.0
uint32_t Ocram2StartAddress; Ocram 2 start address, deprecated since 2.1.0.
uint32_t Ocram2EndAddress; Ocram 2 end address, deprecated since 2.1.0 If enabled, 1-cycle wait state will be inserted into each read access:

- **true** Enable read data wait;
- **false** Disable read data wait.

bool enableReadAddrPipeline
If enabled, the read address will be registered before can be applied to memory cell:

- **true** Enable Read address pipeline;
- **false** Disable Read address pipeline.

bool enableWriteDataPipeline
If enabled, the write data will be registered before can be applied to memory cell:

- **true** Enable write data pipeline;
- **false** Disable write data pipeline.

bool enableWriteAddrPipeline
If enabled, write address will be registered before can be applied to memory cell:

- **true** Enable write address pipeline;
- **false** Disable write address pipeline.

```
struct _mecc_single_error_info
#include <fsl_mecc.h> MECC ocram single error information, including single error address,
ECC code, error data and error bit position.
```

Public Members

uint32_t singleErrorAddress
Single error address on Ocram bank n

uint32_t singleErrorDataLow

Single error low 32 bits uncorrected read data on Ocram bank n

uint32_t singleErrorDataHigh

Single error high 32 bits uncorrected read data on Ocram bank n

uint32_t singleErrorPosLow

Single error bit position of low 32 bits read data on Ocram bank n

uint32_t singleErrorPosHigh

Single error bit position of high 32 bits read data on Ocram bank n

uint8_t singleErrorEccCode

Single error ECC code on Ocram bank n

struct _mecc_multi_error_info

#include <fsl_mecc.h> MECC ocram multiple error information, including multiple error address, ECC code, error data.

Public Members

uint32_t multiErrorAddress

Multiple error address on Ocram bank n

uint32_t multiErrorDataLow

Multiple error low 32 bits read data on Ocram bank n

uint32_t multiErrorDataHigh

Multiple error high 32 bits read data on Ocram bank n

uint8_t multiErrorEccCode

Multiple error ECC code on Ocram bank n

2.79 MIPI DSI Driver

void DSI_Init(const *MIPI_DSI_Type* *base, const *dsi_config_t* *config)

Initializes an MIPI DSI host with the user configuration.

This function initializes the MIPI DSI host with the configuration, it should be called first before other MIPI DSI driver functions.

Parameters

- base – MIPI DSI host peripheral base address.
- config – Pointer to a user-defined configuration structure.

void DSI_Deinit(const *MIPI_DSI_Type* *base)

Deinitializes an MIPI DSI host.

This function should be called after all other MIPI DSI driver functions.

Parameters

- base – MIPI DSI host peripheral base address.

void DSI_GetDefaultConfig(*dsi_config_t* *config)

Get the default configuration to initialize the MIPI DSI host.

The default value is:

```

config->numLanes = 4;
config->enableNonContinuousHsClk = false;
config->enableTxUlps = false;
config->autoInsertEoTp = true;
config->numExtraEoTp = 0;
config->htxTo_ByteClk = 0;
config->lrxHostTo_ByteClk = 0;
config->btaTo_ByteClk = 0;

```

Parameters

- config – Pointer to a user-defined configuration structure.

```
void DSI_SetDpiConfig(const MIPI_DSI_Type *base, const dsi_dpi_config_t *config, uint8_t
numLanes, uint32_t dpiPixelClkFreq_Hz, uint32_t dsiHsBitClkFreq_Hz)
```

Configure the DPI interface core.

This function sets the DPI interface configuration, it should be used in video mode.

Parameters

- base – MIPI DSI host peripheral base address.
- config – Pointer to the DPI interface configuration.
- numLanes – Lane number, should be same with the setting in *dsi_dpi_config_t*.
- dpiPixelClkFreq_Hz – The DPI pixel clock frequency in Hz.
- dsiHsBitClkFreq_Hz – The DSI high speed bit clock frequency in Hz. It is the same with DPHY PLL output.

```
uint32_t DSI_InitDphy(const MIPI_DSI_Type *base, const dsi_dphy_config_t *config, uint32_t
refClkFreq_Hz)
```

Initializes the D-PHY.

This function configures the D-PHY timing and setups the D-PHY PLL based on user configuration. The configuration structure could be got by the function *DSI_GetDphyDefaultConfig*.

For some platforms there is not dedicated D-PHY PLL, indicated by the macro *FSL_FEATURE_MIPI_DSI_NO_DPHY_PLL*. For these platforms, the *refClkFreq_Hz* is useless.

Parameters

- base – MIPI DSI host peripheral base address.
- config – Pointer to the D-PHY configuration.
- refClkFreq_Hz – The REFCLK frequency in Hz.

Returns

The actual D-PHY PLL output frequency. If could not configure the PLL to the target frequency, the return value is 0.

```
void DSI_DeinitDphy(const MIPI_DSI_Type *base)
```

Deinitializes the D-PHY.

Power down the D-PHY PLL and shut down D-PHY.

Parameters

- base – MIPI DSI host peripheral base address.

```
void DSI_GetDphyDefaultConfig(dsi_dphy_config_t *config, uint32_t txHsBitClk_Hz, uint32_t
txEscClk_Hz)
```

Get the default D-PHY configuration.

Gets the default D-PHY configuration, the timing parameters are set according to D-PHY specification. User could use the configuration directly, or change some parameters according to the special device.

Parameters

- config – Pointer to the D-PHY configuration.
- txHsBitClk_Hz – High speed bit clock in Hz.
- txEscClk_Hz – Esc clock in Hz.

```
static inline void DSI_EnableInterrupts(const 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(const 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(const 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_SetApbPacketControl(const MIPI_DSI_Type *base, uint16_t wordCount, uint8_t virtualChannel, dsi_tx_data_type_t dataType, uint8_t flags)
```

Configure the APB packet to send.

This function configures the next APB packet transfer. After configuration, the packet transfer could be started with function DSI_SendApbPacket. If the packet is long packet, Use DSI_WriteApbTxPayload to fill the payload before start transfer.

Parameters

- base – MIPI DSI host peripheral base address.
- wordCount – For long packet, this is the byte count of the payload. For short packet, this is (data1 « 8) | data0.
- virtualChannel – Virtual channel.

- `dataType` – The packet data type, (DI).
- `flags` – The transfer control flags, see `_dsi_transfer_flags`.

```
void DSI_WriteApbTxPayload(const MIPI_DSI_Type *base, const uint8_t *payload, uint16_t  
                           payloadSize)
```

Fill the long APB packet payload.

Write the long packet payload to TX FIFO.

Parameters

- `base` – MIPI DSI host peripheral base address.
- `payload` – Pointer to the payload.
- `payloadSize` – Payload size in byte.

```
void DSI_WriteApbTxPayloadExt(const MIPI_DSI_Type *base, const uint8_t *payload, uint16_t  
                              payloadSize, bool sendDcsCmd, uint8_t dcsCmd)
```

Extended function to fill the payload to TX FIFO.

Write the long packet payload to TX FIFO. This function could be used in two ways

- a. Include the DCS command in parameter `payload`. In this case, the DCS command is the first byte of `payload`. The parameter `sendDcsCmd` is set to false, the `dcsCmd` is not used. This function is the same as `DSI_WriteApbTxPayload` when used in this way.
- b. The DCS command is not in parameter `payload`, but specified by parameter `dcsCmd`. In this case, the parameter `sendDcsCmd` is set to true, the `dcsCmd` is the DCS command to send. The `payload` is sent after `dcsCmd`.

Parameters

- `base` – MIPI DSI host peripheral base address.
- `payload` – Pointer to the payload.
- `payloadSize` – Payload size in byte.
- `sendDcsCmd` – If set to true, the DCS command is specified by `dcsCmd`, otherwise the DCS command is included in the `payload`.
- `dcsCmd` – The DCS command to send, only used when `sendDcsCmd` is true.

```
void DSI_ReadApbRxPayload(const MIPI_DSI_Type *base, uint8_t *payload, uint16_t  
                           payloadSize)
```

Read the long APB packet payload.

Read the long packet payload from RX FIFO. This function reads directly but does not check the RX FIFO status. Upper layer should make sure there are available data.

Parameters

- `base` – MIPI DSI host peripheral base address.
- `payload` – Pointer to the payload.
- `payloadSize` – Payload size in byte.

```
static inline void DSI_SendApbPacket(const MIPI_DSI_Type *base)
```

Trigger the controller to send out APB packet.

Send the packet set by `DSI_SetApbPacketControl`.

Parameters

- `base` – MIPI DSI host peripheral base address.

static inline uint32_t DSI_GetApbStatus(const *MIPI_DSI_Type* *base)

Get the APB status.

The return value is OR'ed value of _dsi_apb_status.

Parameters

- base – MIPI DSI host peripheral base address.

Returns

The APB status.

static inline uint32_t DSI_GetRxErrorStatus(const *MIPI_DSI_Type* *base)

Get the error status during data transfer.

The return value is OR'ed value of _dsi_rx_error_status.

Parameters

- base – MIPI DSI host peripheral base address.

Returns

The error status.

static inline uint8_t DSI_GetEccRxErrorPosition(uint32_t rxErrorStatus)

Get the one-bit RX ECC error position.

When one-bit ECC RX error detected using DSI_GetRxErrorStatus, this function could be used to get the error bit position.

```
uint8_t eccErrorPos;
uint32_t rxErrorStatus = DSI_GetRxErrorStatus(MIPI_DSI);
if (kDSI_RxErrorEccOneBit & rxErrorStatus)
{
    eccErrorPos = DSI_GetEccRxErrorPosition(rxErrorStatus);
}
```

Parameters

- rxErrorStatus – The error status returned by DSI_GetRxErrorStatus.

Returns

The 1-bit ECC error position.

static inline uint32_t DSI_GetAndClearHostStatus(const *MIPI_DSI_Type* *base)

Get and clear the DSI host status.

The host status are returned as mask value of _dsi_host_status.

Parameters

- base – MIPI DSI host peripheral base address.

Returns

The DSI host status.

static inline uint32_t DSI_GetRxPacketHeader(const *MIPI_DSI_Type* *base)

Get the RX packet header.

Parameters

- base – MIPI DSI host peripheral base address.

Returns

The RX packet header.

static inline *dsi_rx_data_type_t* DSI_GetRxPacketType(uint32_t rxPktHeader)

Extract the RX packet type from the packet header.

Extract the RX packet type from the packet header get by DSI_GetRxPacketHeader.

Parameters

- rxPktHeader – The RX packet header get by DSI_GetRxPacketHeader.

Returns

The RX packet type.

static inline uint16_t DSI_GetRxPacketWordCount(uint32_t rxPktHeader)

Extract the RX packet word count from the packet header.

Extract the RX packet word count from the packet header get by DSI_GetRxPacketHeader.

Parameters

- rxPktHeader – The RX packet header get by DSI_GetRxPacketHeader.

Returns

For long packet, return the payload word count (byte). For short packet, return the (data0 « 8) | data1.

static inline uint8_t DSI_GetRxPacketVirtualChannel(uint32_t rxPktHeader)

Extract the RX packet virtual channel from the packet header.

Extract the RX packet virtual channel from the packet header get by DSI_GetRxPacketHeader.

Parameters

- rxPktHeader – The RX packet header get by DSI_GetRxPacketHeader.

Returns

The virtual channel.

status_t DSI_TransferBlocking(const *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.

When using this API to read data, the actually read data count could be got from xfer->rxDataSize.

Parameters

- base – MIPI DSI host peripheral base address.
- xfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Data transfer finished with no error.
- kStatus_Timeout – Transfer failed because of timeout.
- kStatus_DSI_RxDataError – RX data error, user could use DSI_GetRxErrorStatus to check the error details.
- kStatus_DSI_ErrorReportReceived – Error Report packet received, user could use DSI_GetAndClearHostStatus to check the error report status.
- kStatus_DSI_NotSupported – Transfer format not supported.
- kStatus_DSI_Fail – Transfer failed for other reasons.

```
status_t DSI_TransferCreateHandle(const MIPI_DSI_Type *base, dsi_handle_t *handle,  
                                dsi_callback_t callback, void *userData)
```

Create the MIPI DSI handle.

This function initializes the MIPI DSI handle which can be used for other transactional APIs.

Parameters

- base – MIPI DSI host peripheral base address.
- handle – Handle pointer.
- callback – Callback function.
- userData – User data.

```
status_t DSI_TransferNonBlocking(const MIPI_DSI_Type *base, dsi_handle_t *handle,  
                                dsi_transfer_t *xfer)
```

APB data transfer using interrupt method.

Perform APB data transfer using interrupt method, when transfer finished, upper layer could be informed through callback function.

When using this API to read data, the actually read data count could be got from handle->xfer->rxDataSize after read finished.

Parameters

- base – MIPI DSI host peripheral base address.
- handle – pointer to dsi_handle_t structure which stores the transfer state.
- xfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Data transfer started successfully.
- kStatus_DSI_Busy – Failed to start transfer because DSI is busy with previous transfer.
- kStatus_DSI_NotSupported – Transfer format not supported.

```
void DSI_TransferAbort(const MIPI_DSI_Type *base, dsi_handle_t *handle)
```

Abort current APB data transfer.

Parameters

- base – MIPI DSI host peripheral base address.
- handle – pointer to dsi_handle_t structure which stores the transfer state.

```
void DSI_TransferHandleIRQ(const MIPI_DSI_Type *base, dsi_handle_t *handle)
```

Interrupt handler for the DSI.

Parameters

- base – MIPI DSI host peripheral base address.
- handle – pointer to dsi_handle_t structure which stores the transfer state.

FSL_MIPI_DSI_DRIVER_VERSION

Error codes for the MIPI DSI driver.

Values:

enumerator kStatus_DSI_Busy

DSI is busy.

enumerator kStatus_DSI_RxDataError

Read data error.

enumerator kStatus_DSI_ErrorReportReceived

Error report package received.

enumerator kStatus_DSI_NotSupported

The transfer type not supported.

enum _dsi_dpi_color_coding

MIPI DPI interface color coding.

Values:

enumerator kDSI_Dpi16BitConfig1

16-bit configuration 1. RGB565: XXXXXXXX_RRRRRGGG_GGGBBBBB.

enumerator kDSI_Dpi16BitConfig2

16-bit configuration 2. RGB565: XXXRRRRR_XXGGGGGG_XXXBBBBB.

enumerator kDSI_Dpi16BitConfig3

16-bit configuration 3. RGB565: XXRRRRRX_XXGGGGGG_XXBBBBBX.

enumerator kDSI_Dpi18BitConfig1

18-bit configuration 1. RGB666: XXXXXXRR_RRRRGGGG_GGBBBBBB.

enumerator kDSI_Dpi18BitConfig2

18-bit configuration 2. RGB666: XXRRRRRR_XXGGGGGG_XXBBBBBB.

enumerator kDSI_Dpi24Bit

24-bit.

enum _dsi_dpi_pixel_packet

MIPI DSI pixel packet type send through DPI interface.

Values:

enumerator kDSI_PixelPacket16Bit

16 bit RGB565.

enumerator kDSI_PixelPacket18Bit

18 bit RGB666 packed.

enumerator kDSI_PixelPacket18BitLoosely

18 bit RGB666 loosely packed into three bytes.

enumerator kDSI_PixelPacket24Bit

24 bit RGB888, each pixel uses three bytes.

_dsi_dpi_polarity_flag DPI signal polarity.

Values:

enumerator kDSI_DpiVsyncActiveLow

VSYNC active low.

enumerator kDSI_DpiHsyncActiveLow

HSYNC active low.

enumerator kDSI_DpiVsyncActiveHigh

VSYNC active high.

enumerator kDSI_DpiHsyncActiveHigh
HSYNC active high.

enum _dsi_dpi_video_mode
DPI video mode.

Values:

enumerator kDSI_DpiNonBurstWithSyncPulse
Non-Burst mode with Sync Pulses.

enumerator kDSI_DpiNonBurstWithSyncEvent
Non-Burst mode with Sync Events.

enumerator kDSI_DpiBurst
Burst mode.

enum _dsi_dpi_bllp_mode
Behavior in BLLP (Blanking or Low-Power Interval).

Values:

enumerator kDSI_DpiBllpLowPower
LP mode used in BLLP periods.

enumerator kDSI_DpiBllpBlanking
Blanking packets used in BLLP periods.

enumerator kDSI_DpiBllpNull
Null packets used in BLLP periods.

_dsi_apb_status Status of APB to packet interface.

Values:

enumerator kDSI_ApbNotIdle
State machine not idle

enumerator kDSI_ApbTxDone
Tx packet done

enumerator kDSI_ApbRxControl
DPHY direction 0 - tx had control, 1 - rx has control

enumerator kDSI_ApbTxOverflow
TX fifo overflow

enumerator kDSI_ApbTxUnderflow
TX fifo underflow

enumerator kDSI_ApbRxOverflow
RX fifo overflow

enumerator kDSI_ApbRxUnderflow
RX fifo underflow

enumerator kDSI_ApbRxHeaderReceived
RX packet header has been received

enumerator kDSI_ApbRxPacketReceived
All RX packet payload data has been received

`_dsi_rx_error_status` Host receive error status.

Values:

enumerator `kDSI_RxErrorEccOneBit`

ECC single bit error detected.

enumerator `kDSI_RxErrorEccMultiBit`

ECC multi bit error detected.

enumerator `kDSI_RxErrorCrc`

CRC error detected.

enumerator `kDSI_RxErrorHtxTo`

High Speed forward TX timeout detected.

enumerator `kDSI_RxErrorLrxTo`

Reverse Low power data receive timeout detected.

enumerator `kDSI_RxErrorBtaTo`

BTA timeout detected.

enum `_dsi_host_status`

DSI host controller status (`status_out`)

Values:

enumerator `kDSI_HostSoTError`

SoT error from peripheral error report.

enumerator `kDSI_HostSoTSyncError`

SoT Sync error from peripheral error report.

enumerator `kDSI_HostEoTSyncError`

EoT Sync error from peripheral error report.

enumerator `kDSI_HostEscEntryCmdError`

Escape Mode Entry Command Error from peripheral error report.

enumerator `kDSI_HostLpTxSyncError`

Low-power transmit Sync Error from peripheral error report.

enumerator `kDSI_HostPeriphToError`

Peripheral timeout error from peripheral error report.

enumerator `kDSI_HostFalseControlError`

False control error from peripheral error report.

enumerator `kDSI_HostContentionDetected`

Contention detected from peripheral error report.

enumerator `kDSI_HostEccErrorOneBit`

Single bit ECC error (corrected) from peripheral error report.

enumerator `kDSI_HostEccErrorMultiBit`

Multi bit ECC error (not corrected) from peripheral error report.

enumerator `kDSI_HostChecksumError`

Checksum error from peripheral error report.

enumerator `kDSI_HostInvalidDataType`

DSI data type not recognized.

enumerator kDSI_HostInvalidVcId
DSI VC ID invalid.

enumerator kDSI_HostInvalidTxLength
Invalid transmission length.

enumerator kDSI_HostProtocolViolation
DSI protocol violation.

enumerator kDSI_HostResetTriggerReceived
Reset trigger received.

enumerator kDSI_HostTearTriggerReceived
Tear effect trigger receive.

enumerator kDSI_HostAckTriggerReceived
Acknowledge trigger message received.

_dsi_interrupt DSI interrupt.

Values:

enumerator kDSI_InterruptGroup1ApbNotIdle
State machine not idle

enumerator kDSI_InterruptGroup1ApbTxDone
Tx packet done

enumerator kDSI_InterruptGroup1ApbRxControl
DPHY direction 0 - tx control, 1 - rx control

enumerator kDSI_InterruptGroup1ApbTxOverflow
TX fifo overflow

enumerator kDSI_InterruptGroup1ApbTxUnderflow
TX fifo underflow

enumerator kDSI_InterruptGroup1ApbRxOverflow
RX fifo overflow

enumerator kDSI_InterruptGroup1ApbRxUnderflow
RX fifo underflow

enumerator kDSI_InterruptGroup1ApbRxHeaderReceived
RX packet header has been received

enumerator kDSI_InterruptGroup1ApbRxPacketReceived
All RX packet payload data has been received

enumerator kDSI_InterruptGroup1SoTError
SoT error from peripheral error report.

enumerator kDSI_InterruptGroup1SoTSyncError
SoT Sync error from peripheral error report.

enumerator kDSI_InterruptGroup1EoTSyncError
EoT Sync error from peripheral error report.

enumerator kDSI_InterruptGroup1EscEntryCmdError
Escape Mode Entry Command Error from peripheral error report.

enumerator kDSI_InterruptGroup1LpTxSyncError
Low-power transmit Sync Error from peripheral error report.

enumerator kDSI_InterruptGroup1PeriphToError
Peripheral timeout error from peripheral error report.

enumerator kDSI_InterruptGroup1FalseControlError
False control error from peripheral error report.

enumerator kDSI_InterruptGroup1ContentionDetected
Contention detected from peripheral error report.

enumerator kDSI_InterruptGroup1EccErrorOneBit
Single bit ECC error (corrected) from peripheral error report.

enumerator kDSI_InterruptGroup1EccErrorMultiBit
Multi bit ECC error (not corrected) from peripheral error report.

enumerator kDSI_InterruptGroup1ChecksumError
Checksum error from peripheral error report.

enumerator kDSI_InterruptGroup1InvalidDataType
DSI data type not recognized.

enumerator kDSI_InterruptGroup1InvalidVcId
DSI VC ID invalid.

enumerator kDSI_InterruptGroup1InvalidTxLength
Invalid transmission length.

enumerator kDSI_InterruptGroup1ProtocalViolation
DSI protocol violation.

enumerator kDSI_InterruptGroup1ResetTriggerReceived
Reset trigger received.

enumerator kDSI_InterruptGroup1TearTriggerReceived
Tear effect trigger receive.

enumerator kDSI_InterruptGroup1AckTriggerReceived
Acknowledge trigger message received.

enumerator kDSI_InterruptGroup1HtxTo
High speed TX timeout.

enumerator kDSI_InterruptGroup1LrxTo
Low power RX timeout.

enumerator kDSI_InterruptGroup1BtaTo
Host BTA timeout.

enumerator kDSI_InterruptGroup2EccOneBit
Sinle bit ECC error.

enumerator kDSI_InterruptGroup2EccMultiBit
Multi bit ECC error.

enumerator kDSI_InterruptGroup2CrcError
CRC error.

enum _dsi_tx_data_type
DSI TX data type.

Values:

enumerator kDSI_TxDataVsyncStart
V Sync start.

enumerator kDSI_TxDataVsyncEnd
V Sync end.

enumerator kDSI_TxDataHsyncStart
H Sync start.

enumerator kDSI_TxDataHsyncEnd
H Sync end.

enumerator kDSI_TxDataEoTp
End of transmission packet.

enumerator kDSI_TxDataCmOff
Color mode off.

enumerator kDSI_TxDataCmOn
Color mode on.

enumerator kDSI_TxDataShutDownPeriph
Shut down peripheral.

enumerator kDSI_TxDataTurnOnPeriph
Turn on peripheral.

enumerator kDSI_TxDataGenShortWrNoParam
Generic Short WRITE, no parameters.

enumerator kDSI_TxDataGenShortWrOneParam
Generic Short WRITE, one parameter.

enumerator kDSI_TxDataGenShortWrTwoParam
Generic Short WRITE, two parameter.

enumerator kDSI_TxDataGenShortRdNoParam
Generic Short READ, no parameters.

enumerator kDSI_TxDataGenShortRdOneParam
Generic Short READ, one parameter.

enumerator kDSI_TxDataGenShortRdTwoParam
Generic Short READ, two parameter.

enumerator kDSI_TxDataDcsShortWrNoParam
DCS Short WRITE, no parameters.

enumerator kDSI_TxDataDcsShortWrOneParam
DCS Short WRITE, one parameter.

enumerator kDSI_TxDataDcsShortRdNoParam
DCS Short READ, no parameters.

enumerator kDSI_TxDataSetMaxReturnPktSize
Set the Maximum Return Packet Size.

enumerator kDSI_TxDataNull
Null Packet, no data.

enumerator kDSI_TxDataBlanking
Blanking Packet, no data.

enumerator kDSI_TxDataGenLongWr

Generic long write.

enumerator kDSI_TxDataDcsLongWr

DCS Long Write/write_LUT Command Packet.

enumerator kDSI_TxDataLooselyPackedPixel20BitYCbCr

Loosely Packed Pixel Stream, 20-bit YCbCr, 4:2:2 Format.

enumerator kDSI_TxDataPackedPixel24BitYCbCr

Packed Pixel Stream, 24-bit YCbCr, 4:2:2 Format.

enumerator kDSI_TxDataPackedPixel16BitYCbCr

Packed Pixel Stream, 16-bit YCbCr, 4:2:2 Format.

enumerator kDSI_TxDataPackedPixel30BitRGB

Packed Pixel Stream, 30-bit RGB, 10-10-10 Format.

enumerator kDSI_TxDataPackedPixel36BitRGB

Packed Pixel Stream, 36-bit RGB, 12-12-12 Format.

enumerator kDSI_TxDataPackedPixel12BitYCrCb

Packed Pixel Stream, 12-bit YCbCr, 4:2:0 Format.

enumerator kDSI_TxDataPackedPixel16BitRGB

Packed Pixel Stream, 16-bit RGB, 5-6-5 Format.

enumerator kDSI_TxDataPackedPixel18BitRGB

Packed Pixel Stream, 18-bit RGB, 6-6-6 Format.

enumerator kDSI_TxDataLooselyPackedPixel18BitRGB

Loosely Packed Pixel Stream, 18-bit RGB, 6-6-6 Format.

enumerator kDSI_TxDataPackedPixel24BitRGB

Packed Pixel Stream, 24-bit RGB, 8-8-8 Format.

enum _dsi_rx_data_type

DSI RX data type.

Values:

enumerator kDSI_RxDataAckAndErrorReport

Acknowledge and Error Report

enumerator kDSI_RxDataEoTp

End of Transmission packet.

enumerator kDSI_RxDataGenShortRdResponseOneByte

Generic Short READ Response, 1 byte returned.

enumerator kDSI_RxDataGenShortRdResponseTwoByte

Generic Short READ Response, 2 byte returned.

enumerator kDSI_RxDataGenLongRdResponse

Generic Long READ Response.

enumerator kDSI_RxDataDcsLongRdResponse

DCS Long READ Response.

enumerator kDSI_RxDataDcsShortRdResponseOneByte

DCS Short READ Response, 1 byte returned.

enumerator `kDSI_RxDataDcsShortRdResponseTwoByte`
DCS Short READ Response, 2 byte returned.

`_dsi_transfer_flags` DSI transfer control flags.

Values:

enumerator `kDSI_TransferUseHighSpeed`
Use high speed mode or not.

enumerator `kDSI_TransferPerformBTA`
Perform BTA or not.

typedef struct `_dsi_config` `dsi_config_t`
MIPI DSI controller configuration.

typedef enum `_dsi_dpi_color_coding` `dsi_dpi_color_coding_t`
MIPI DPI interface color coding.

typedef enum `_dsi_dpi_pixel_packet` `dsi_dpi_pixel_packet_t`
MIPI DSI pixel packet type send through DPI interface.

typedef enum `_dsi_dpi_video_mode` `dsi_dpi_video_mode_t`
DPI video mode.

typedef enum `_dsi_dpi_bllp_mode` `dsi_dpi_bllp_mode_t`
Behavior in BLLP (Blanking or Low-Power Interval).

typedef struct `_dsi_dpi_config` `dsi_dpi_config_t`
MIPI DSI controller DPI interface configuration.

typedef struct `_dsi_dphy_config` `dsi_dphy_config_t`
MIPI DSI D-PHY configuration.

typedef enum `_dsi_tx_data_type` `dsi_tx_data_type_t`
DSI TX data type.

typedef enum `_dsi_rx_data_type` `dsi_rx_data_type_t`
DSI RX data type.

typedef struct `_dsi_transfer` `dsi_transfer_t`
Structure for the data transfer.

typedef struct `_dsi_handle` `dsi_handle_t`
MIPI DSI transfer handle.

typedef void (*`dsi_callback_t`)(const *MIPI_DSI_Type* *base, *dsi_handle_t* *handle, *status_t* status, void *userData)

MIPI DSI callback for finished transfer.

When transfer finished, one of these status values will be passed to the user:

- `kStatus_Success` Data transfer finished with no error.
- `kStatus_Timeout` Transfer failed because of timeout.
- `kStatus_DSI_RxDataError` RX data error, user could use `DSI_GetRxErrorStatus` to check the error details.
- `kStatus_DSI_ErrorReportReceived` Error Report packet received, user could use `DSI_GetAndClearHostStatus` to check the error report status.
- `kStatus_Fail` Transfer failed for other reasons.

FSL_DSI_TX_MAX_PAYLOAD_BYTE

FSL_DSI_RX_MAX_PAYLOAD_BYTE

struct MIPI_DSI_Type

#include <fsl_mipi_dsi.h> MIPI DSI structure definition.

Public Members

DSI_HOST_Type *host

Pointer to HOST registers.

DSI_HOST_APB_PKT_IF_Type *apb

Pointer to APB registers.

DSI_HOST_DPI_INTFC_Type *dpi

Pointer to DPI registers.

DSI_HOST_NXP_FDSOI28_DPHY_INTFC_Type *dphy

Pointer to DPHY registers.

struct __dsi_config

#include <fsl_mipi_dsi.h> MIPI DSI controller configuration.

Public Members

uint8_t numLanes

Number of lanes.

bool enableNonContinuousHsClk

In enabled, the high speed clock will enter low power mode between transmissions.

bool enableTxUlps

Enable the TX ULPS.

bool autoInsertEoTp

Insert an EoTp short package when switching from HS to LP.

uint8_t numExtraEoTp

How many extra EoTp to send after the end of a packet.

uint32_t htxTo_ByteClk

HS TX timeout count (HTX_TO) in byte clock.

uint32_t lrxHostTo_ByteClk

LP RX host timeout count (LRX-H_TO) in byte clock.

uint32_t btaTo_ByteClk

Bus turn around timeout count (TA_TO) in byte clock.

struct __dsi_dpi_config

#include <fsl_mipi_dsi.h> MIPI DSI controller DPI interface configuration.

Public Members

uint16_t pixelPayloadSize

Maximum number of pixels that should be sent as one DSI packet. Recommended that the line size (in pixels) is evenly divisible by this parameter.

dsi_dpi_color_coding_t dpiColorCoding

DPI color coding.

dsi_dpi_pixel_packet_t pixelPacket

Pixel packet format.

dsi_dpi_video_mode_t videoMode

Video mode.

dsi_dpi_bllp_mode_t bllpMode

Behavior in BLLP.

uint8_t polarityFlags

OR'ed value of *_dsi_dpi_polarity_flag* controls signal polarity.

uint16_t hfp

Horizontal front porch, in dpi pixel clock.

uint16_t hbp

Horizontal back porch, in dpi pixel clock.

uint16_t hsw

Horizontal sync width, in dpi pixel clock.

uint8_t vfp

Number of lines in vertical front porch.

uint8_t vbp

Number of lines in vertical back porch.

uint16_t panelHeight

Line number in vertical active area.

uint8_t virtualChannel

Virtual channel.

struct *_dsi_dphy_config*

#include <fsl_mipi_dsi.h> MIPI DSI D-PHY configuration.

Public Members

uint32_t txHsBitClk_Hz

The generated HS TX bit clock in Hz.

uint8_t tClkPre_ByteClk

TLPX + TCLK-PREPARE + TCLK-ZERO + TCLK-PRE in byte clock. Set how long the controller will wait after enabling clock lane for HS before enabling data lanes for HS.

uint8_t tClkPost_ByteClk

TCLK-POST + T_CLK-TRAIL in byte clock. Set how long the controller will wait before putting clock lane into LP mode after data lanes detected in stop state.

uint8_t tHsExit_ByteClk

THS-EXIT in byte clock. Set how long the controller will wait after the clock lane has been put into LP mode before enabling clock lane for HS again.

uint32_t tWakeup_EscClk

Number of *clk_esc* clock periods to keep a clock or data lane in Mark-1 state after exiting ULPS.

uint8_t tHsPrepare_HalfEscClk

THS-PREPARE in clk_esc/2. Set how long to drive the LP-00 state before HS transmissions, available values are 2, 3, 4, 5.

uint8_t tClkPrepare_HalfEscClk

TCLK-PREPARE in clk_esc/2. Set how long to drive the LP-00 state before HS transmissions, available values are 2, 3.

uint8_t tHsZero_ByteClk

THS-ZERO in clk_byte. Set how long that controller drives data lane HS-0 state before transmit the Sync sequence. Available values are 6, 7, ..., 37.

uint8_t tClkZero_ByteClk

TCLK-ZERO in clk_byte. Set how long that controller drives clock lane HS-0 state before transmit the Sync sequence. Available values are 3, 4, ..., 66.

uint8_t tHsTrail_ByteClk

THS-TRAIL + 4*UI in clk_byte. Set the time of the flipped differential state after last payload data bit of HS transmission burst. Available values are 0, 1, ..., 15.

uint8_t tClkTrail_ByteClk

TCLK-TRAIL + 4*UI in clk_byte. Set the time of the flipped differential state after last payload data bit of HS transmission burst. Available values are 0, 1, ..., 15.

struct _dsi_transfer

#include <fsl_mipi_dsi.h> Structure for the data transfer.

Public Members

uint8_t virtualChannel

Virtual channel.

dsi_tx_data_type_t txDataType

TX data type.

uint8_t flags

Flags to control the transfer; see _dsi_transfer_flags.

const uint8_t *txData

The TX data buffer.

uint8_t *rxData

The 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.

struct _dsi_handle

#include <fsl_mipi_dsi.h> MIPI DSI transfer handle structure.

Public Members

volatile bool isBusy
MIPI DSI is busy with APB data transfer.

dsi_transfer_t xfer
Transfer information.

dsi_callback_t callback
DSI callback

void *userData
Callback parameter

const *MIPI_DSI_Type* *dsi
Pointer to MIPI DSI peripheral.

2.80 MIPI_DSI: MIPI DSI Host Controller

2.81 MU: Messaging Unit

void MU_Init(MU_Type *base)
Initializes the MU module.
This function enables the MU clock only.

Parameters

- base – MU peripheral base address.

void MU_Deinit(MU_Type *base)
De-initializes the MU module.
This function disables the MU clock only.

Parameters

- base – MU peripheral base address.

static inline void MU_SendMsgNonBlocking(MU_Type *base, uint32_t regIndex, uint32_t msg)
Writes a message to the TX register.

This function writes a message to the specific TX register. It does not check whether the TX register is empty or not. The upper layer should make sure the TX register is empty before calling this function. This function can be used in ISR for better performance.

```
while (!(kMU_Tx0EmptyFlag & MU_GetStatusFlags(base))) { } Wait for TX0 register empty.  
MU_SendMsgNonBlocking(base, kMU_MsgReg0, MSG_VAL); Write message to the TX0 register.
```

Parameters

- base – MU peripheral base address.
- regIndex – TX register index, see *mu_msg_reg_index_t*.
- msg – Message to send.

status_t MU_SendMsg(MU_Type *base, uint32_t regIndex, uint32_t msg)
Blocks to send a message.

This function waits until the TX register is empty and sends the message. If *MU_BUSY_POLL_COUNT* is defined and non-zero, the function will timeout after the specified number of polling iterations and returns *kStatus_Timeout*.

Parameters

- base – MU peripheral base address.
- regIndex – MU message register, see mu_msg_reg_index_t.
- msg – Message to send.

Return values

- kStatus_Success – Message sent successfully.
- kStatus_Timeout – Timeout occurred while waiting for TX register to be empty.

Returns

status_t

static inline uint32_t MU_ReceiveMsgNonBlocking(MU_Type *base, uint32_t regIndex)

Reads a message from the RX register.

This function reads a message from the specific RX register. It does not check whether the RX register is full or not. The upper layer should make sure the RX register is full before calling this function. This function can be used in ISR for better performance.

```
uint32_t msg;
while (!(kMU_Rx0FullFlag & MU_GetStatusFlags(base)))
{
    } Wait for the RX0 register full.

msg = MU_ReceiveMsgNonBlocking(base, kMU_MsgReg0); Read message from RX0 register.
```

Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see mu_msg_reg_index_t.

Returns

The received message.

status_t MU_ReceiveMsgTimeout(MU_Type *base, uint32_t regIndex, uint32_t *readValue)

Blocks to receive a message with timeout protection.

This function waits until the RX register is full and receives the message. If MU_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

This function provides the same blocking behavior as MU_ReceiveMsg() but with additional timeout protection to prevent system hangs if the other core becomes unresponsive or if hardware issues occur.

Note: Both MU_ReceiveMsg() and MU_ReceiveMsgTimeout() are blocking functions. The difference is that this function includes timeout protection while MU_ReceiveMsg() waits indefinitely.

Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see mu_msg_reg_index_t.
- readValue – Pointer to store the received message.

Return values

- `kStatus_Success` – Message received successfully.
- `kStatus_InvalidArgument` – Invalid `readValue` pointer.
- `kStatus_Timeout` – Timeout occurred while waiting for RX register to be full.

Returns`status_t``uint32_t MU_ReceiveMsg(MU_Type *base, uint32_t regIndex)`

Blocks to receive a message (infinite wait, no timeout protection).

This function waits until the RX register is full and receives the message. This function will wait indefinitely until a message is received.

Note: Both `MU_ReceiveMsg()` and `MU_ReceiveMsgTimeout()` are blocking functions. The difference is that `MU_ReceiveMsgTimeout()` includes timeout protection while this function waits indefinitely.

Warning: This function does not include timeout protection and may cause system hangs if the other core becomes unresponsive. For applications requiring timeout protection, use `MU_ReceiveMsgTimeout()` instead.

Parameters

- `base` – MU peripheral base address.
- `regIndex` – RX register index, see `mu_msg_reg_index_t`.

Returns

The received message.

`static inline void MU_SetFlagsNonBlocking(MU_Type *base, uint32_t flags)`

Sets the 3-bit MU flags reflect on the other MU side.

This function sets the 3-bit MU flags directly. Every time the 3-bit MU flags are changed, the status flag `kMU_FlagsUpdatingFlag` asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag `kMU_FlagsUpdatingFlag` is cleared by hardware. During the flags updating period, the flags cannot be changed. The upper layer should make sure the status flag `kMU_FlagsUpdatingFlag` is cleared before calling this function.

```
while (kMU_FlagsUpdatingFlag & MU_GetStatusFlags(base))
{
    } Wait for previous MU flags updating.
    MU_SetFlagsNonBlocking(base, 0U); Set the mU flags.
```

Parameters

- `base` – MU peripheral base address.
- `flags` – The 3-bit MU flags to set.

`status_t MU_SetFlags(MU_Type *base, uint32_t flags)`

Blocks setting the 3-bit MU flags reflect on the other MU side.

This function blocks setting the 3-bit MU flags. Every time the 3-bit MU flags are changed, the status flag `kMU_FlagsUpdatingFlag` asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag `kMU_FlagsUpdatingFlag` is cleared by hardware. During the flags updating period, the flags cannot be changed. This

function waits for the MU status flag kMU_FlagsUpdatingFlag cleared and sets the 3-bit MU flags.

If MU_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

Parameters

- base – MU peripheral base address.
- flags – The 3-bit MU flags to set.

Return values

- kStatus_Success – Flags were set successfully.
- kStatus_Timeout – Timeout occurred while waiting for flags to update.

Returns

status_t

static inline uint32_t MU_GetFlags(MU_Type *base)

Gets the current value of the 3-bit MU flags set by the other side.

This function gets the current 3-bit MU flags on the current side.

Parameters

- base – MU peripheral base address.

Returns

flags Current value of the 3-bit flags.

static inline uint32_t MU_GetStatusFlags(MU_Type *base)

Gets the MU status flags.

This function returns the bit mask of the MU status flags. See _mu_status_flags.

```
uint32_t flags;
flags = MU_GetStatusFlags(base); Get all status flags.
if (kMU_Tx0EmptyFlag & flags)
{
    The TX0 register is empty. Message can be sent.
    MU_SendMsgNonBlocking(base, kMU_MsgReg0, MSG0_VAL);
}
if (kMU_Tx1EmptyFlag & flags)
{
    The TX1 register is empty. Message can be sent.
    MU_SendMsgNonBlocking(base, kMU_MsgReg1, MSG1_VAL);
}
```

Parameters

- base – MU peripheral base address.

Returns

Bit mask of the MU status flags, see _mu_status_flags.

static inline uint32_t MU_GetRxStatusFlags(MU_Type *base)

Return the RX status flags.

This function return the RX status flags. Note: RFn bits of SR[27-24](mu status register) are mapped in reverse numerical order: RF0 -> SR[27] RF1 -> SR[26] RF2 -> SR[25] RF3 -> SR[24]

```
status_reg = MU_GetRxStatusFlags(base);
```

Parameters

- base – MU peripheral base address.

Returns

MU RX status

```
static inline uint32_t MU_GetInterruptsPending(MU_Type *base)
```

Gets the MU IRQ pending status of enabled interrupts.

This function returns the bit mask of the pending MU IRQs of enabled interrupts. Only these flags are checked. kMU_Tx0EmptyFlag kMU_Tx1EmptyFlag
kMU_Tx2EmptyFlag kMU_Tx3EmptyFlag kMU_Rx0FullFlag kMU_Rx1FullFlag
kMU_Rx2FullFlag kMU_Rx3FullFlag kMU_GenInt0Flag kMU_GenInt1Flag
kMU_GenInt2Flag kMU_GenInt3Flag

Parameters

- base – MU peripheral base address.

Returns

Bit mask of the MU IRQs pending.

```
static inline void MU_ClearStatusFlags(MU_Type *base, uint32_t mask)
```

Clears the specific MU status flags.

This function clears the specific MU status flags. The flags to clear should be passed in as bit mask. See `_mu_status_flags`.

Clear general interrupt 0 and general interrupt 1 pending flags.
MU_ClearStatusFlags(base, kMU_GenInt0Flag | kMU_GenInt1Flag);

Parameters

- base – MU peripheral base address.
- mask – Bit mask of the MU status flags. See `_mu_status_flags`. The following flags are cleared by hardware, this function could not clear them.
 - kMU_Tx0EmptyFlag
 - kMU_Tx1EmptyFlag
 - kMU_Tx2EmptyFlag
 - kMU_Tx3EmptyFlag
 - kMU_Rx0FullFlag
 - kMU_Rx1FullFlag
 - kMU_Rx2FullFlag
 - kMU_Rx3FullFlag
 - kMU_EventPendingFlag
 - kMU_FlagsUpdatingFlag
 - kMU_OtherSideInResetFlag

```
static inline void MU_EnableInterrupts(MU_Type *base, uint32_t mask)
```

Enables the specific MU interrupts.

This function enables the specific MU interrupts. The interrupts to enable should be passed in as bit mask. See `_mu_interrupt_enable`.

Enable general interrupt 0 and TX0 empty interrupt.
MU_EnableInterrupts(base, kMU_GenInt0InterruptEnable | kMU_Tx0EmptyInterruptEnable);

Parameters

- base – MU peripheral base address.
- mask – Bit mask of the MU interrupts. See `_mu_interrupt_enable`.

`static inline void MU_DisableInterrupts(MU_Type *base, uint32_t mask)`

Disables the specific MU interrupts.

This function disables the specific MU interrupts. The interrupts to disable should be passed in as bit mask. See `_mu_interrupt_enable`.

Disable general interrupt 0 and TX0 empty interrupt.
`MU_DisableInterrupts(base, kMU_GenInt0InterruptEnable | kMU_Tx0EmptyInterruptEnable);`

Parameters

- base – MU peripheral base address.
- mask – Bit mask of the MU interrupts. See `_mu_interrupt_enable`.

`status_t MU_TriggerInterrupts(MU_Type *base, uint32_t mask)`

Triggers interrupts to the other core.

This function triggers the specific interrupts to the other core. The interrupts to trigger are passed in as bit mask. See `_mu_interrupt_trigger`. The MU should not trigger an interrupt to the other core when the previous interrupt has not been processed by the other core. This function checks whether the previous interrupts have been processed. If not, it returns an error.

```
if (kStatus_Success != MU_TriggerInterrupts(base, kMU_GenInt0InterruptTrigger | kMU_
↪ GenInt2InterruptTrigger))
{
    Previous general purpose interrupt 0 or general purpose interrupt 2
    has not been processed by the other core.
}
```

Parameters

- base – MU peripheral base address.
- mask – Bit mask of the interrupts to trigger. See `_mu_interrupt_trigger`.

Return values

- `kStatus_Success` – Interrupts have been triggered successfully.
- `kStatus_Fail` – Previous interrupts have not been accepted.

`static inline void MU_MaskHardwareReset(MU_Type *base, bool mask)`

Mask hardware reset by the other core.

The other core could call `MU_HardwareResetOtherCore()` to reset current core. To mask the reset, call this function and pass in true.

Parameters

- base – MU peripheral base address.
- mask – Pass true to mask the hardware reset, pass false to unmask it.

`FSL_MU_DRIVER_VERSION`

MU driver version.

`enum _mu_status_flags`

MU status flags.

Values:

enumerator kMU__Tx0EmptyFlag

TX0 empty.

enumerator kMU__Tx1EmptyFlag

TX1 empty.

enumerator kMU__Tx2EmptyFlag

TX2 empty.

enumerator kMU__Tx3EmptyFlag

TX3 empty.

enumerator kMU__Rx0FullFlag

RX0 full.

enumerator kMU__Rx1FullFlag

RX1 full.

enumerator kMU__Rx2FullFlag

RX2 full.

enumerator kMU__Rx3FullFlag

RX3 full.

enumerator kMU__GenInt0Flag

General purpose interrupt 0 pending.

enumerator kMU__GenInt1Flag

General purpose interrupt 1 pending.

enumerator kMU__GenInt2Flag

General purpose interrupt 2 pending.

enumerator kMU__GenInt3Flag

General purpose interrupt 3 pending.

enumerator kMU__EventPendingFlag

MU event pending.

enumerator kMU__FlagsUpdatingFlag

MU flags update is on-going.

enumerator kMU__ResetAssertInterruptFlag

The other core reset assert interrupt pending.

enumerator kMU__ResetDeassertInterruptFlag

The other core reset de-assert interrupt pending.

enumerator kMU__OtherSideInResetFlag

The other side is in reset.

enumerator kMU__MuResetInterruptFlag

The other side initializes MU reset.

enumerator kMU__HardwareResetInterruptFlag

Current side has been hardware reset by the other side.

enum __mu_interrupt_enable

MU interrupt source to enable.

Values:

enumerator kMU_Tx0EmptyInterruptEnable
TX0 empty.

enumerator kMU_Tx1EmptyInterruptEnable
TX1 empty.

enumerator kMU_Tx2EmptyInterruptEnable
TX2 empty.

enumerator kMU_Tx3EmptyInterruptEnable
TX3 empty.

enumerator kMU_Rx0FullInterruptEnable
RX0 full.

enumerator kMU_Rx1FullInterruptEnable
RX1 full.

enumerator kMU_Rx2FullInterruptEnable
RX2 full.

enumerator kMU_Rx3FullInterruptEnable
RX3 full.

enumerator kMU_GenInt0InterruptEnable
General purpose interrupt 0.

enumerator kMU_GenInt1InterruptEnable
General purpose interrupt 1.

enumerator kMU_GenInt2InterruptEnable
General purpose interrupt 2.

enumerator kMU_GenInt3InterruptEnable
General purpose interrupt 3.

enumerator kMU_ResetAssertInterruptEnable
The other core reset assert interrupt.

enumerator kMU_ResetDeassertInterruptEnable
The other core reset de-assert interrupt.

enumerator kMU_MuResetInterruptEnable
The other side initializes MU reset. The interrupt is ORed with the general purpose interrupt 3. The general purpose interrupt 3 is issued when the other side set the MU reset and this interrupt is enabled.

enumerator kMU_HardwareResetInterruptEnable
Current side has been hardware reset by the other side.

enum _mu_interrupt_trigger
MU interrupt that could be triggered to the other core.

Values:

enumerator kMU_GenInt0InterruptTrigger
General purpose interrupt 0.

enumerator kMU_GenInt1InterruptTrigger
General purpose interrupt 1.

enumerator kMU_GenInt2InterruptTrigger
General purpose interrupt 2.

enumerator kMU_GenInt3InterruptTrigger
General purpose interrupt 3.

enum _mu_msg_reg_index
MU message register.

Values:

enumerator kMU_MsgReg0
enumerator kMU_MsgReg1
enumerator kMU_MsgReg2
enumerator kMU_MsgReg3

typedef enum _mu_msg_reg_index mu_msg_reg_index_t
MU message register.

MU_CR_NMI_MASK

MU_BUSY_POLL_COUNT

Maximum polling iterations for MU waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the MU code before timing out and returning an error.

It applies to all waiting loops in MU driver, such as waiting for TX register to be empty or waiting for RX register to be full.

This is a count of loop iterations, not a time-based value.

If defined as 0, polling loops will continue indefinitely until their exit condition is met, which could potentially cause the system to hang if a core becomes unresponsive.

MU_GET_CORE_FLAG(flags)

MU_GET_STAT_FLAG(flags)

MU_GET_TX_FLAG(flags)

MU_GET_RX_FLAG(flags)

MU_GET_GI_FLAG(flags)

2.82 OCOTP: On Chip One-Time Programmable controller.

FSL_OCOTP_DRIVER_VERSION

OCOTP driver version.

_ocotp_status Error codes for the OCOTP driver.

Values:

enumerator kStatus_OCOTP_AccessError
eFuse and shadow register access error.
enumerator kStatus_OCOTP_CrcFail
CRC check failed.
enumerator kStatus_OCOTP_ReloadError
Error happens during reload shadow register.

enumerator kStatus_OCOTP_ProgramFail

Fuse programming failed.

enumerator kStatus_OCOTP_Locked

Fuse is locked and cannot be programmed.

typedef struct *_ocotp_timing* ocotp_timing_t

OCOTP timing structure. Note that, these value are used for calcalating the read/write timings. And the values should satisfy below rules:

Tsp_rd=(WAIT+1)/ipg_clk_freq should be $\geq 150\text{ns}$; Tsp_pgm=(RELAX+1)/ipg_clk_freq should be $\geq 100\text{ns}$; Trd = ((STROBE_READ+1)- 2*(RELAX_READ+1)) /ipg_clk_freq, The Trd is required to be larger than 40 ns. Tpgm = ((STROBE_PROG+1)- 2*(RELAX_PROG+1)) /ipg_clk_freq; The Tpgm should be configured within the range of $9000\text{ ns} < \text{Tpgm} < 11000\text{ ns}$;

void OCOTP_Init(OCOTP_Type *base, uint32_t srcClock_Hz)

Initializes OCOTP controller.

Parameters

- base – OCOTP peripheral base address.
- srcClock_Hz – source clock frequency in unit of Hz. When the macro FSL_FEATURE_OCOTP_HAS_TIMING_CTRL is defined as 0, this parameter is not used, application could pass in 0 in this case.

void OCOTP_Deinit(OCOTP_Type *base)

De-initializes OCOTP controller.

Return values

kStatus_Success – upon successful execution, error status otherwise.

static inline bool OCOTP_CheckBusyStatus(OCOTP_Type *base)

Checking the BUSY bit in CTRL register. Checking this BUSY bit will help confirm if the OCOTP controller is ready for access.

Parameters

- base – OCOTP peripheral base address.

Return values

true – for bit set and false for cleared.

static inline bool OCOTP_CheckErrorStatus(OCOTP_Type *base)

Checking the ERROR bit in CTRL register.

Parameters

- base – OCOTP peripheral base address.

Return values

true – for bit set and false for cleared.

static inline void OCOTP_ClearErrorStatus(OCOTP_Type *base)

Clear the error bit if this bit is set.

Parameters

- base – OCOTP peripheral base address.

status_t OCOTP_ReloadShadowRegister(OCOTP_Type *base)

Reload the shadow register. This function will help reload the shadow register without resetting the OCOTP module. Please make sure the OCOTP has been initialized before calling this API.

Parameters

- base – OCOTP peripheral base address.

Return values

- kStatus__Success – Reload success.
- kStatus_OCOTP_ReloadError – Reload failed.

uint32_t OCOTP_ReadFuseShadowRegister(OCOTP_Type *base, uint32_t address)

Read the fuse shadow register with the fuse address.

Deprecated:

Use OCOTP_ReadFuseShadowRegisterExt instead of this function.

Parameters

- base – OCOTP peripheral base address.
- address – the fuse address to be read from.

Returns

The read out data.

status_t OCOTP_ReadFuseShadowRegisterExt(OCOTP_Type *base, uint32_t address, uint32_t *data, uint8_t fuseWords)

Read the fuse shadow register from the fuse address.

This function reads fuse from address, how many words to read is specified by the parameter fuseWords. This function could read at most OCOTP_READ_FUSE_DATA_COUNT fuse word one time.

Parameters

- base – OCOTP peripheral base address.
- address – the fuse address to be read from.
- data – Data array to save the readout fuse value.
- fuseWords – How many words to read.

Return values

- kStatus__Success – Read success.
- kStatus__Fail – Error occurs during read.

status_t OCOTP_WriteFuseShadowRegister(OCOTP_Type *base, uint32_t address, uint32_t data)

Write the fuse shadow register with the fuse address and data. Please make sure the write address is not locked while calling this API.

Parameters

- base – OCOTP peripheral base address.
- address – the fuse address to be written.
- data – the value will be written to fuse address.

Return values

write – status, kStatus__Success for success and kStatus__Fail for failed.

status_t OCOTP_WriteFuseShadowRegisterWithLock(OCOTP_Type *base, uint32_t address, uint32_t data, bool lock)

Write the fuse shadow register and lock it.

Please make sure the write address is not locked while calling this API.

Some OCOTP controller supports ECC mode and redundancy mode (see reference manual for more details). OCOTP controller will auto select ECC or redundancy mode to program the fuse word according to fuse map definition. In ECC mode, the 32 fuse bits in one word can only be written once. In redundancy mode, the word can be written more than once as long as they are different fuse bits. Set parameter `lock` as true to force use ECC mode.

Parameters

- `base` – OCOTP peripheral base address.
- `address` – The fuse address to be written.
- `data` – The value will be written to fuse address.
- `lock` – Lock or unlock write fuse shadow register operation.

Return values

- `kStatus__Success` – Program and reload success.
- `kStatus_OCOTP_Locked` – The eFuse word is locked and cannot be programmed.
- `kStatus_OCOTP_ProgramFail` – eFuse word programming failed.
- `kStatus_OCOTP_ReloadError` – eFuse word programming success, but error happens during reload the values.
- `kStatus_OCOTP_AccessError` – Cannot access eFuse word.

```
static inline uint32_t OCOTP_GetVersion(OCOTP_Type *base)
```

Get the OCOTP controller version from the register.

Parameters

- `base` – OCOTP peripheral base address.

Return values

`return` – the version value.

```
OCOTP_READ_FUSE_DATA_COUNT
```

```
struct __ocotp_timing
```

#include <fsl_ocotp.h> OCOTP timing structure. Note that, these value are used for calculating the read/write timings. And the values should satisfy below rules:

$Tsp\_rd = (WAIT + 1) / ipg\_clk\_freq$ should be $\geq 150ns$; $Tsp\_pgm = (RELAX + 1) / ipg\_clk\_freq$ should be $\geq 100ns$; $Trd = ((STROBE\_READ + 1) - 2 * (RELAX\_READ + 1)) / ipg\_clk\_freq$, The Trd is required to be larger than 40 ns. $Tpgm = ((STROBE\_PROG + 1) - 2 * (RELAX\_PROG + 1)) / ipg\_clk\_freq$; The $Tpgm$ should be configured within the range of $9000\ ns < Tpgm < 11000\ ns$;

Public Members

```
uint32_t wait
```

Wait time value to fill in the TIMING register.

```
uint32_t relax
```

Relax time value to fill in the TIMING register.

```
uint32_t strobe_prog
```

Strobe program time value to fill in the TIMING register.

```
uint32_t strobe_read
```

Strobe read time value to fill in the TIMING register.

2.83 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

2.84 PDM: Microphone Interface

2.85 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 medium

enumerator kPDM_QualityModeHigh
quality mode high

enumerator kPDM_QualityModeLow
quality mode low

enumerator kPDM_QualityModeVeryLow0
quality mode very low0

enumerator kPDM_QualityModeVeryLow1
quality mode very low1

enumerator kPDM_QualityModeVeryLow2
quality mode very low2

enum __pdm_quality_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_removal pdm_dc_removal_t
    PDM DC removal configurations.
typedef enum _pdm_df_quality_mode pdm_df_quality_mode_t
    PDM decimation filter quality mode.
typedef enum _pdm_df_output_gain pdm_df_output_gain_t
    PDM decimation filter output gain.
typedef struct _pdm_channel_config pdm_channel_config_t
    PDM channel configurations.
typedef struct _pdm_config pdm_config_t
    PDM user configuration structure.
typedef enum _pdm_hwvad_hpf_config pdm_hwvad_hpf_config_t
    High pass filter configuration cut-off frequency.
typedef enum _pdm_hwvad_filter_status pdm_hwvad_filter_status_t
    HWVAD internal filter status.
typedef struct _pdm_hwvad_config pdm_hwvad_config_t
    PDM voice activity detector user configuration structure.
typedef struct _pdm_hwvad_noise_filter pdm_hwvad_noise_filter_t
    PDM voice activity detector noise filter user configuration structure.
typedef enum _pdm_hwvad_zcd_result pdm_hwvad_zcd_result_t
    PDM voice activity detector zero cross detector result.
typedef struct _pdm_hwvad_zero_cross_detector pdm_hwvad_zero_cross_detector_t
    PDM voice activity detector zero cross detector configuration structure.
typedef struct _pdm_transfer pdm_transfer_t
    PDM SDMA transfer structure.
typedef struct _pdm_handle pdm_handle_t
    PDM handle.
typedef void (*pdm_transfer_callback_t)(PDM_Type *base, pdm_handle_t *handle, status_t
status, void *userData)
    PDM transfer callback prototype.
typedef void (*pdm_hwvad_callback_t)(status_t status, void *userData)
    PDM HWVAD callback prototype.
typedef struct _pdm_hwvad_notification pdm_hwvad_notification_t
    PDM HWVAD notification structure.
PDM_XFER_QUEUE_SIZE
    PDM XFER QUEUE SIZE.
struct _pdm_channel_config
    #include <fsl_pdm.h> PDM channel configurations.

```


Public Members

pdm_dc_remover_t cutOffFreq
DC remover cut off frequency

pdm_df_output_gain_t gain
Decimation Filter Output Gain

struct __pdm_config
#include <fsl_pdm.h> PDM user configuration structure.

Public Members

bool enableDoze
This module will enter disable/low leakage mode if DOZEN is active with ipg_doze is asserted

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

2.86 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 function support dynamic scatter gather and static scatter gather, a. for the dynamic scatter gather case: Application should call `PDM_TransferReceiveEDMA` function continuously to make sure new receive request is submit before the previous one finish. b. for the static scatter gather case: Application should use the link transfer feature and make sure a loop link transfer is provided, such as:

```
pdm_edma_transfer_t pdmXfer[2] =
{
    {
        .data = s_buffer,
        .dataSize = BUFFER_SIZE,
        .linkTransfer = &pdmXfer[1],
    },

    {
        .data = &s_buffer[BUFFER_SIZE],
    },
}
```

(continues on next page)

(continued from previous page)

```

        .dataSize = BUFFER_SIZE,
        .linkTransfer = &pdmXfer[0]
    },
};

```

- b. Multi channel case: This function support receive multi pdm channel data, for example, if two channel is requested,

```

PDM_TransferSetChannelConfigEDMA(DEMO_PDM, &s_pdmRxHandle_0, DEMO_PDM_
↪ENABLE_CHANNEL_0, &channelConfig);
PDM_TransferSetChannelConfigEDMA(DEMO_PDM, &s_pdmRxHandle_0, DEMO_PDM_
↪ENABLE_CHANNEL_1, &channelConfig);
PDM_TransferReceiveEDMA(DEMO_PDM, &s_pdmRxHandle_0, pdmXfer);

```

The output data will be formatted as below if handle->interleaveType =

Note: This interface returns immediately after the transfer initiates. Call the PDM_GetReceiveRemainingBytes to poll the transfer status and check whether the PDM transfer is finished.

void PDM_TransferTerminateReceiveEDMA(PDM_Type *base, *pdm_edma_handle_t* *handle)

Terminate all PDM receive.

This function will clear all transfer slots buffered in the pdm queue. If users only want to abort the current transfer slot, please call PDM_TransferAbortReceiveEDMA.

Parameters

- base – PDM base pointer.
- handle – PDM eDMA handle pointer.

void PDM_TransferAbortReceiveEDMA(PDM_Type *base, *pdm_edma_handle_t* *handle)

Aborts a PDM receive using eDMA.

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call PDM_TransferTerminateReceiveEDMA.

Parameters

- base – PDM base pointer
- handle – PDM eDMA handle pointer.

status_t PDM_TransferGetReceiveCountEDMA(PDM_Type *base, *pdm_edma_handle_t* *handle, *size_t* *count)

Gets byte count received by PDM.

Parameters

- base – PDM base pointer
- handle – PDM eDMA handle pointer.
- count – Bytes count received by PDM.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

FSL_PDM_EDMA_DRIVER_VERSION

Version 2.6.5

enum __pdm_edma_multi_channel_interleave

pdm multi channel interleave type

Values:

enumerator kPDM_EDMAMultiChannelInterleavePerChannelSample

enumerator kPDM_EDMAMultiChannelInterleavePerChannelBlock

typedef struct *_pdm_edma_handle* pdm_edma_handle_t

PDM edma handler.

typedef enum *_pdm_edma_multi_channel_interleave* pdm_edma_multi_channel_interleave_t

pdm multi channel interleave type

typedef struct *_pdm_edma_transfer* pdm_edma_transfer_t

PDM edma transfer.

typedef void (*pdm_edma_callback_t)(PDM_Type *base, *pdm_edma_handle_t* *handle, *status_t* status, void *userData)

PDM eDMA transfer callback function for finish and error.

MCUX_SDK_PDM_EDMA_PDM_ENABLE_INTERNAL

the PDM enable position When calling PDM_TransferReceiveEDMA

struct __pdm_edma_transfer

#include <fsl_pdm_edma.h> PDM edma transfer.

Public Members

volatile uint8_t *data

Data start address to transfer.

volatile size_t dataSize

Total Transfer bytes size.

struct *_pdm_edma_transfer* *linkTransfer

linked transfer configurations

struct __pdm_edma_handle

#include <fsl_pdm_edma.h> PDM DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *dmaHandle

DMA handler for PDM send

uint8_t count

The transfer data count in a DMA request

uint32_t receivedBytes

total transfer count

uint32_t state

Internal state for PDM eDMA transfer

pdm_edma_callback_t callback
Callback for users while transfer finish or error occurs

bool isLoopTransfer
loop transfer

void *userData
User callback parameter

edma_tcd_t *tcd
TCD pool for eDMA transfer.

uint32_t tcdNum
TCD number

uint32_t tcdUser
Index for user to queue transfer.

uint32_t tcdDriver
Index for driver to get the transfer data and size

volatile uint32_t tcdUsedNum
Index for user to queue transfer.

pdm_edma_multi_channel_interleave_t interleaveType
multi channel transfer interleave type

uint8_t endChannel
The last enabled channel

uint8_t channelNums
total channel numbers

2.87 PIT: Periodic Interrupt Timer

void PIT_Init(PIT_Type *base, const *pit_config_t* *config)

Ungates the PIT clock, enables the PIT module, and configures the peripheral for basic operations.

Note: This API should be called at the beginning of the application using the PIT driver.

Parameters

- base – PIT peripheral base address
- config – Pointer to the user's PIT config structure

void PIT_Deinit(PIT_Type *base)

Gates the PIT clock and disables the PIT module.

Parameters

- base – PIT peripheral base address

static inline void PIT_GetDefaultConfig(*pit_config_t* *config)

Fills in the PIT configuration structure with the default settings.

The default values are as follows.

```
config->enableRunInDebug = false;
```

Parameters

- `config` – Pointer to the configuration structure.

```
static inline void PIT_SetTimerChainMode(PIT_Type *base, pit_chnl_t channel, bool enable)
```

Enables or disables chaining a timer with the previous timer.

When a timer has a chain mode enabled, it only counts after the previous timer has expired. If the timer `n-1` has counted down to 0, counter `n` decrements the value by one. Each timer is 32-bits, which allows the developers to chain timers together and form a longer timer (64-bits and larger). The first timer (timer 0) can't be chained to any other timer.

Parameters

- `base` – PIT peripheral base address
- `channel` – Timer channel number which is chained with the previous timer
- `enable` – Enable or disable chain. `true`: Current timer is chained with the previous timer. `false`: Timer doesn't chain with other timers.

```
static inline void PIT_EnableInterrupts(PIT_Type *base, pit_chnl_t channel, uint32_t mask)
```

Enables the selected PIT interrupts.

Parameters

- `base` – PIT peripheral base address
- `channel` – Timer channel number
- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `pit_interrupt_enable_t`

```
static inline void PIT_DisableInterrupts(PIT_Type *base, pit_chnl_t channel, uint32_t mask)
```

Disables the selected PIT interrupts.

Parameters

- `base` – PIT peripheral base address
- `channel` – Timer channel number
- `mask` – The interrupts to disable. This is a logical OR of members of the enumeration `pit_interrupt_enable_t`

```
static inline uint32_t PIT_GetEnabledInterrupts(PIT_Type *base, pit_chnl_t channel)
```

Gets the enabled PIT interrupts.

Parameters

- `base` – PIT peripheral base address
- `channel` – Timer channel number

Returns

The enabled interrupts. This is the logical OR of members of the enumeration `pit_interrupt_enable_t`

```
static inline uint32_t PIT_GetStatusFlags(PIT_Type *base, pit_chnl_t channel)
```

Gets the PIT status flags.

Parameters

- `base` – PIT peripheral base address
- `channel` – Timer channel number

Returns

The status flags. This is the logical OR of members of the enumeration `pit_status_flags_t`

```
static inline void PIT_ClearStatusFlags(PIT_Type *base, pit_chnl_t channel, uint32_t mask)
```

Clears the PIT status flags.

Parameters

- `base` – PIT peripheral base address
- `channel` – Timer channel number
- `mask` – The status flags to clear. This is a logical OR of members of the enumeration `pit_status_flags_t`

```
static inline void PIT_SetTimerPeriod(PIT_Type *base, pit_chnl_t channel, uint32_t count)
```

Sets the timer period in units of count.

Timers begin counting from the value set by this function until it reaches 0, then it generates an interrupt and load 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: Users can call the utility macros provided in `fsl_common.h` to convert to ticks.

Parameters

- `base` – PIT peripheral base address
- `channel` – Timer channel number
- `count` – Timer period in units of ticks

```
static inline uint32_t PIT_GetCurrentTimerCount(PIT_Type *base, pit_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: Users can call the utility macros provided in `fsl_common.h` to convert ticks to usec or msec.

Parameters

- `base` – PIT peripheral base address
- `channel` – Timer channel number

Returns

Current timer counting value in ticks

```
static inline void PIT_StartTimer(PIT_Type *base, pit_chnl_t channel)
```

Starts the timer counting.

After calling this function, timers load period value, count down to 0 and then load the respective start value again. Each time a timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

- `base` – PIT peripheral base address
- `channel` – Timer channel number.

```
static inline void PIT_StopTimer(PIT_Type *base, pit_chnl_t channel)
```

Stops the timer counting.

This function stops every timer counting. Timers reload their periods respectively after the next time they call the PIT_DRV_StartTimer.

Parameters

- *base* – PIT peripheral base address
- *channel* – Timer channel number.

```
FSL_PIT_DRIVER_VERSION
```

PIT Driver Version 2.2.0.

```
enum __pit_chnl
```

List of PIT channels.

Note: Actual number of available channels is SoC dependent

Values:

```
enumerator kPIT_Chnl_0
```

PIT channel number 0

```
enumerator kPIT_Chnl_1
```

PIT channel number 1

```
enumerator kPIT_Chnl_2
```

PIT channel number 2

```
enumerator kPIT_Chnl_3
```

PIT channel number 3

```
enum __pit_interrupt_enable
```

List of PIT interrupts.

Values:

```
enumerator kPIT_TimerInterruptEnable
```

Timer interrupt enable

```
enum __pit_status_flags
```

List of PIT status flags.

Values:

```
enumerator kPIT_TimerFlag
```

Timer flag

```
typedef enum __pit_chnl pit_chnl_t
```

List of PIT channels.

Note: Actual number of available channels is SoC dependent

```
typedef enum __pit_interrupt_enable pit_interrupt_enable_t
```

List of PIT interrupts.

```
typedef enum __pit_status_flags pit_status_flags_t
```

List of PIT status flags.

`typedef struct _pit_config pit_config_t`

PIT configuration structure.

This structure holds the configuration settings for the PIT peripheral. To initialize this structure to reasonable defaults, call the `PIT_GetDefaultConfig()` function and pass a pointer to your config structure instance.

The configuration structure can be made constant so it resides in flash.

`uint64_t PIT_GetLifetimeTimerCount(PIT_Type *base)`

Reads the current lifetime counter value.

The lifetime timer is a 64-bit timer which chains timer 0 and timer 1 together. Timer 0 and 1 are chained by calling the `PIT_SetTimerChainMode` before using this timer. The period of lifetime timer is equal to the “period of timer 0 * period of timer 1”. For the 64-bit value, the higher 32-bit has the value of timer 1, and the lower 32-bit has the value of timer 0.

Parameters

- `base` – PIT peripheral base address

Returns

Current lifetime timer value

`struct _pit_config`

`#include <fsl_pit.h>` PIT configuration structure.

This structure holds the configuration settings for the PIT peripheral. To initialize this structure to reasonable defaults, call the `PIT_GetDefaultConfig()` function and pass a pointer to your config structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

`bool enableRunInDebug`

true: Timers run in debug mode; false: Timers stop in debug mode

2.88 PUF: Physical Unclonable Function

`FSL_PUF_DRIVER_VERSION`

PUF driver version. Version 2.2.0.

Current version: 2.2.0

Change log:

- 2.0.0
 - Initial version.
- 2.0.1
 - Fixed `puf_wait_usec` function optimization issue.
- 2.0.2
 - Add PUF configuration structure and support for PUF SRAM controller. Remove magic constants.
- 2.0.3
 - Fix MISRA C-2012 issue.
- 2.1.0

- Align driver with PUF SRAM controller registers on LPCXpresso55s16.
 - Update initialization logic .
- 2.1.1
 - Fix ARMGCC build warning .
- 2.1.2
 - Update: Add automatic big to little endian swap for user (pre-shared) keys destined to secret hardware bus (PUF key index 0).
- 2.1.3
 - Fix MISRA C-2012 issue.
- 2.1.4
 - Replace register uint32_t ticksCount with volatile uint32_t ticksCount in puf_wait_usec() to prevent optimization out delay loop.
- 2.1.5
 - Use common SDK delay in puf_wait_usec()
- 2.1.6
 - Changed wait time in PUF_Init(), when initialization fails it will try PUF_Powercycle() with shorter time. If this shorter time will also fail, initialization will be tried with worst case time as before.
- 2.2.0
- Add support for kPUF_KeySlot4.
- Add new PUF_ClearKey() function, that clears a desired PUF internal HW key register.

enum __puf_key_index_register

Values:

```
enumerator kPUF_KeyIndex_00
enumerator kPUF_KeyIndex_01
enumerator kPUF_KeyIndex_02
enumerator kPUF_KeyIndex_03
enumerator kPUF_KeyIndex_04
enumerator kPUF_KeyIndex_05
enumerator kPUF_KeyIndex_06
enumerator kPUF_KeyIndex_07
enumerator kPUF_KeyIndex_08
enumerator kPUF_KeyIndex_09
enumerator kPUF_KeyIndex_10
enumerator kPUF_KeyIndex_11
enumerator kPUF_KeyIndex_12
enumerator kPUF_KeyIndex_13
enumerator kPUF_KeyIndex_14
```

```
    enumerator kPUF__KeyIndex_15
enum _puf_min_max
    Values:
    enumerator kPUF__KeySizeMin
    enumerator kPUF__KeySizeMax
    enumerator kPUF__KeyIndexMax
enum _puf_key_slot
    PUF key slot.
    Values:
    enumerator kPUF__KeySlot0
        PUF key slot 0
    enumerator kPUF__KeySlot1
        PUF key slot 1

    PUF status return codes.
    Values:
    enumerator kStatus__EnrollNotAllowed
    enumerator kStatus__StartNotAllowed

typedef enum _puf_key_index_register puf_key_index_register_t
typedef enum _puf_min_max puf_min_max_t
typedef enum _puf_key_slot puf_key_slot_t
    PUF key slot.
PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(x)
    Get Key Code size in bytes from key size in bytes at compile time.
PUF_MIN_KEY_CODE_SIZE
PUF_ACTIVATION_CODE_SIZE
KEystore_PUF_DISCHARGE_TIME_FIRST_TRY_MS
KEystore_PUF_DISCHARGE_TIME_MAX_MS
struct puf_config_t
    #include <fsl_puf.h>
```

2.89 PWM: Pulse Width Modulator

```
status_t PWM_Init(PWM_Type *base, pwm_submodule_t subModule, const pwm_config_t
    *config)
```

Ungates the PWM submodule clock and configures the peripheral for basic operation.

This API should be called at the beginning of the application using the PWM driver. When user select PWMX, user must choose edge aligned output, because there are some limitation on center aligned PWMX output. When output PWMX in center aligned mode, VAL1 register controls both PWM period and PWMX duty cycle, PWMA and PWMB output will

be corrupted. But edge aligned PWMX output do not have such limit. In master reload counter initialization mode, PWM period is depended by period of set LDOK in submodule 0 because this operation will reload register. Submodule 0 counter initialization cannot be master sync or master reload.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- config – Pointer to user's PWM config structure.

Returns

kStatus_Success means success; else failed.

void PWM_Deinit(PWM_Type *base, *pwm_submodule_t* subModule)

Gate the PWM submodule clock.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to deinitialize

void PWM_GetDefaultConfig(*pwm_config_t* *config)

Fill in the PWM config struct with the default settings.

The default values are:

```
config->enableDebugMode = false;
config->enableWait = false;
config->reloadSelect = kPWM_LocalReload;
config->clockSource = kPWM_BusClock;
config->prescale = kPWM_Prescale_Divide_1;
config->initializationControl = kPWM_Initialize_LocalSync;
config->forceTrigger = kPWM_Force_Local;
config->reloadFrequency = kPWM_LoadEveryOpportunity;
config->reloadLogic = kPWM_ReloadImmediate;
config->pairOperation = kPWM_Independent;
```

Parameters

- config – Pointer to user's PWM config structure.

status_t PWM_SetupPwm(PWM_Type *base, *pwm_submodule_t* subModule, const *pwm_signal_param_t* *chnlParams, uint8_t numOfChnls, *pwm_mode_t* mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz)

Sets up the PWM signals for a PWM submodule.

The function initializes the submodule according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements. If the dead time insertion logic is enabled, the pulse period is reduced by the dead time period specified by the user. When user select PWMX, user must choose edge aligned output, because there are some limitation on center aligned PWMX output. Due to edge aligned PWMX is negative true signal, need to configure PWMX active low true level to get correct duty cycle. The half cycle point will not be exactly in the middle of the PWM cycle when PWMX enabled.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- 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. Array size should not be more than 3 as each submodule has 3 pins to output PWM.
- mode – PWM operation mode, options available in enumeration `pwm_mode_t`
- pwmFreq_Hz – PWM signal frequency in Hz
- srcClock_Hz – PWM source clock of correspond submodule in Hz. If source clock of submodule1,2,3 is from submodule0 AUX_CLK, its source clock is submodule0 source clock divided with submodule0 prescaler value instead of submodule0 source clock.

Returns

Returns `kStatus_Fail` if there was error setting up the signal; `kStatus_Success` otherwise

```
status_t PWM_SetupPwmPhaseShift(PWM_Type *base, pwm_submodule_t subModule,  
                                pwm_channels_t pwmChannel, uint32_t pwmFreq_Hz,  
                                uint32_t srcClock_Hz, uint8_t shiftvalue, bool doSync)
```

Set PWM phase shift for PWM channel running on channel PWM_A, PWM_B which with 50% duty cycle.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- pwmFreq_Hz – PWM signal frequency in Hz
- srcClock_Hz – PWM main counter clock in Hz.
- shiftvalue – Phase shift value, range in 0 ~ 50
- doSync – true: Set LDOK bit for the submodule list; false: LDOK bit don't set, need to call `PWM_SetPwmLdok` to sync update.

Returns

Returns `kStatus_Fail` if there was error setting up the signal; `kStatus_Success` otherwise

```
void PWM_UpdatePwmDutycycle(PWM_Type *base, pwm_submodule_t subModule,  
                             pwm_channels_t pwmSignal, pwm_mode_t currPwmMode,  
                             uint8_t dutyCyclePercent)
```

Updates the PWM signal's dutycycle.

The function updates the PWM dutycycle to the new value that is passed in. If the dead time insertion logic is enabled then the pulse period is reduced by the dead time period specified by the user.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmSignal – Signal (PWM A, PWM B, PWM X) to update
- currPwmMode – 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)

```
void PWM_UpdatePwmDutycycleHighAccuracy(PWM_Type *base, pwm_submodule_t subModule,  
                                         pwm_channels_t pwmSignal, pwm_mode_t  
                                         currPwmMode, uint16_t dutyCycle)
```

Updates the PWM signal's dutycycle with 16-bit accuracy.

The function updates the PWM dutycycle to the new value that is passed in. If the dead time insertion logic is enabled then the pulse period is reduced by the dead time period specified by the user.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmSignal – Signal (PWM A, PWM B, PWM X) to update
- currPwmMode – The current PWM mode set during PWM setup
- dutyCycle – New PWM pulse width, value should be between 0 to 65535
0=inactive signal(0% duty cycle)... 65535=active signal (100% duty cycle)

```
void PWM_UpdatePwmPeriodAndDutycycle(PWM_Type *base, pwm_submodule_t subModule,  
                                       pwm_channels_t pwmSignal, pwm_mode_t  
                                       currPwmMode, uint16_t pulseCnt, uint16_t  
                                       dutyCycle)
```

Update the PWM signal's period and dutycycle for a PWM submodule.

The function updates PWM signal period generated by a specific submodule according to the parameters passed in by the user. This function can also set dutycycle whether you want to keep original dutycycle or update new dutycycle. Call this function in local sync control mode because PWM period is depended by

INIT and VAL1 register of each submodule. In master sync initialization control mode, call this function to update INIT and VAL1 register of all submodule because PWM period is depended by INIT and VAL1 register in submodule0. If the dead time insertion logic is enabled, the pulse period is reduced by the dead time period specified by the user. PWM signal will not be generated if its period is less than dead time duration.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmSignal – Signal (PWM A or PWM B) to update
- currPwmMode – The current PWM mode set during PWM setup, options available in enumeration `pwm_mode_t`
- pulseCnt – New PWM period, value should be between 0 to 65535 0=minimum PWM period... 65535=maximum PWM period
- dutyCycle – New PWM pulse width of channel, value should be between 0 to 65535 0=inactive signal(0% duty cycle)... 65535=active signal (100% duty cycle) You can keep original duty cycle or update new duty cycle

```
static inline void PWM_EnableInterrupts(PWM_Type *base, pwm_submodule_t subModule,  
                                         uint32_t mask)
```

Enables the selected PWM interrupts.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure

- mask – The interrupts to enable. This is a logical OR of members of the enumeration `pwm_interrupt_enable_t`

```
static inline void PWM_DisableInterrupts(PWM_Type *base, pwm_submodule_t subModule,  
                                         uint32_t mask)
```

Disables the selected PWM interrupts.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- mask – The interrupts to enable. This is a logical OR of members of the enumeration `pwm_interrupt_enable_t`

```
static inline uint32_t PWM_GetEnabledInterrupts(PWM_Type *base, pwm_submodule_t  
                                                subModule)
```

Gets the enabled PWM interrupts.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure

Returns

The enabled interrupts. This is the logical OR of members of the enumeration `pwm_interrupt_enable_t`

```
static inline void PWM_DMAFIFOWatermarkControl(PWM_Type *base, pwm_submodule_t  
                                                subModule, pwm_watermark_control_t  
                                                pwm_watermark_control)
```

Capture DMA Enable Source Select.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- `pwm_watermark_control` – PWM FIFO watermark and control

```
static inline void PWM_DMACaptureSourceSelect(PWM_Type *base, pwm_submodule_t  
                                              subModule, pwm_dma_source_select_t  
                                              pwm_dma_source_select)
```

Capture DMA Enable Source Select.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- `pwm_dma_source_select` – PWM capture DMA enable source select

```
static inline void PWM_EnableDMACapture(PWM_Type *base, pwm_submodule_t subModule,  
                                         uint16_t mask, bool activate)
```

Enables or disables the selected PWM DMA Capture read request.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- mask – The DMA to enable or disable. This is a logical OR of members of the enumeration `pwm_dma_enable_t`
- activate – true: Enable DMA read request; false: Disable DMA read request

```
static inline void PWM_EnableDMAWrite(PWM_Type *base, pwm_submodule_t subModule, bool
                                     activate)
```

Enables or disables the PWM DMA write request.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- activate – true: Enable DMA write request; false: Disable DMA write request

```
static inline uint32_t PWM_GetStatusFlags(PWM_Type *base, pwm_submodule_t subModule)
```

Gets the PWM status flags.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure

Returns

The status flags. This is the logical OR of members of the enumeration `pwm_status_flags_t`

```
static inline void PWM_ClearStatusFlags(PWM_Type *base, pwm_submodule_t subModule,
                                       uint32_t mask)
```

Clears the PWM status flags.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- mask – The status flags to clear. This is a logical OR of members of the enumeration `pwm_status_flags_t`

```
static inline void PWM_StartTimer(PWM_Type *base, uint8_t subModulesToStart)
```

Starts the PWM counter for a single or multiple submodules.

Sets the Run bit which enables the clocks to the PWM submodule. This function can start multiple submodules at the same time.

Parameters

- base – PWM peripheral base address
- subModulesToStart – PWM submodules to start. This is a logical OR of members of the enumeration `pwm_module_control_t`

```
static inline void PWM_StopTimer(PWM_Type *base, uint8_t subModulesToStop)
```

Stops the PWM counter for a single or multiple submodules.

Clears the Run bit which resets the submodule's counter. This function can stop multiple submodules at the same time.

Parameters

- base – PWM peripheral base address
- subModulesToStop – PWM submodules to stop. This is a logical OR of members of the enumeration `pwm_module_control_t`

FSL_PWM_DRIVER_VERSION

Version 2.9.1

enum `_pwm_submodule`

List of PWM submodules.

Values:

enumerator `kPWM_Module_0`

Submodule 0

enumerator `kPWM_Module_1`

Submodule 1

enumerator `kPWM_Module_2`

Submodule 2

enum `_pwm_channels`

List of PWM channels in each module.

Values:

enumerator `kPWM_PwmB`

enumerator `kPWM_PwmA`

enumerator `kPWM_PwmX`

enum `_pwm_value_register`

List of PWM value registers.

Values:

enumerator `kPWM_ValueRegister_0`

PWM Value0 register

enumerator `kPWM_ValueRegister_1`

PWM Value1 register

enumerator `kPWM_ValueRegister_2`

PWM Value2 register

enumerator `kPWM_ValueRegister_3`

PWM Value3 register

enumerator `kPWM_ValueRegister_4`

PWM Value4 register

enumerator `kPWM_ValueRegister_5`

PWM Value5 register

enum `_pwm_value_register_mask`

List of PWM value registers mask.

Values:

enumerator `kPWM_ValueRegisterMask_0`

PWM Value0 register mask

enumerator `kPWM_ValueRegisterMask_1`

PWM Value1 register mask

enumerator `kPWM_ValueRegisterMask_2`

PWM Value2 register mask

enumerator `kPWM_ValueRegisterMask_3`

PWM Value3 register mask

enumerator kPWM_ValueRegisterMask_4

PWM Value4 register mask

enumerator kPWM_ValueRegisterMask_5

PWM Value5 register mask

enum __pwm_clock_source

PWM clock source selection.

Values:

enumerator kPWM_BusClock

Device specific IPBus clock, refer reference manual for frequency

enumerator kPWM_ExternalClock

EXT_CLK is used as the clock

enumerator kPWM_Submodule0Clock

Clock of the submodule 0 (AUX_CLK) is used as the source clock

enum __pwm_clock_prescale

PWM prescaler factor selection for clock source.

Values:

enumerator kPWM_Prescale_Divide_1

PWM clock frequency = fclk/1

enumerator kPWM_Prescale_Divide_2

PWM clock frequency = fclk/2

enumerator kPWM_Prescale_Divide_4

PWM clock frequency = fclk/4

enumerator kPWM_Prescale_Divide_8

PWM clock frequency = fclk/8

enumerator kPWM_Prescale_Divide_16

PWM clock frequency = fclk/16

enumerator kPWM_Prescale_Divide_32

PWM clock frequency = fclk/32

enumerator kPWM_Prescale_Divide_64

PWM clock frequency = fclk/64

enumerator kPWM_Prescale_Divide_128

PWM clock frequency = fclk/128

enum __pwm_force_output_trigger

Options that can trigger a PWM FORCE_OUT.

Values:

enumerator kPWM_Force_Local

The local force signal, CTRL2[FORCE], from the submodule is used to force updates

enumerator kPWM_Force_Master

The master force signal from submodule 0 is used to force updates

enumerator kPWM_Force_LocalReload

The local reload signal from this submodule is used to force updates without regard to the state of LDOK

enumerator kPWM_Force_MasterReload

The master reload signal from submodule 0 is used to force updates if LDOK is set

enumerator kPWM_Force_LocalSync

The local sync signal from this submodule is used to force updates

enumerator kPWM_Force_MasterSync

The master sync signal from submodule0 is used to force updates

enumerator kPWM_Force_External

The external force signal, EXT_FORCE, from outside the PWM module causes updates

enumerator kPWM_Force_ExternalSync

The external sync signal, EXT_SYNC, from outside the PWM module causes updates

enum _pwm_output_state

PWM channel output status.

Values:

enumerator kPWM_HighState

The output state of PWM channel is high

enumerator kPWM_LowState

The output state of PWM channel is low

enumerator kPWM_NormalState

The output state of PWM channel is normal

enumerator kPWM_InvertState

The output state of PWM channel is invert

enumerator kPWM_MaskState

The output state of PWM channel is mask

enum _pwm_init_source

PWM counter initialization options.

Values:

enumerator kPWM_Initialize_LocalSync

Local sync causes initialization

enumerator kPWM_Initialize_MasterReload

Master reload from submodule 0 causes initialization

enumerator kPWM_Initialize_MasterSync

Master sync from submodule 0 causes initialization

enumerator kPWM_Initialize_ExtSync

EXT_SYNC causes initialization

enum _pwm_load_frequency

PWM load frequency selection.

Values:

enumerator kPWM_LoadEveryOportunity

Every PWM opportunity

enumerator kPWM_LoadEvery2Oportunity

Every 2 PWM opportunities

enumerator kPWM_LoadEvery3Opportunity

Every 3 PWM opportunities

enumerator kPWM_LoadEvery4Opportunity

Every 4 PWM opportunities

enumerator kPWM_LoadEvery5Opportunity

Every 5 PWM opportunities

enumerator kPWM_LoadEvery6Opportunity

Every 6 PWM opportunities

enumerator kPWM_LoadEvery7Opportunity

Every 7 PWM opportunities

enumerator kPWM_LoadEvery8Opportunity

Every 8 PWM opportunities

enumerator kPWM_LoadEvery9Opportunity

Every 9 PWM opportunities

enumerator kPWM_LoadEvery10Opportunity

Every 10 PWM opportunities

enumerator kPWM_LoadEvery11Opportunity

Every 11 PWM opportunities

enumerator kPWM_LoadEvery12Opportunity

Every 12 PWM opportunities

enumerator kPWM_LoadEvery13Opportunity

Every 13 PWM opportunities

enumerator kPWM_LoadEvery14Opportunity

Every 14 PWM opportunities

enumerator kPWM_LoadEvery15Opportunity

Every 15 PWM opportunities

enumerator kPWM_LoadEvery16Opportunity

Every 16 PWM opportunities

enum __pwm_fault_input

List of PWM fault selections.

Values:

enumerator kPWM_Fault_0

Fault 0 input pin

enumerator kPWM_Fault_1

Fault 1 input pin

enumerator kPWM_Fault_2

Fault 2 input pin

enumerator kPWM_Fault_3

Fault 3 input pin

enum __pwm_fault_disable

List of PWM fault disable mapping selections.

Values:

enumerator kPWM_FaultDisable_0

Fault 0 disable mapping

enumerator kPWM_FaultDisable_1

Fault 1 disable mapping

enumerator kPWM_FaultDisable_2

Fault 2 disable mapping

enumerator kPWM_FaultDisable_3

Fault 3 disable mapping

enum __pwm_fault_channels

List of PWM fault channels.

Values:

enumerator kPWM_faultchannel_0

enum __pwm_input_capture_edge

PWM capture edge select.

Values:

enumerator kPWM_Disable

Disabled

enumerator kPWM_FallingEdge

Capture on falling edge only

enumerator kPWM_RisingEdge

Capture on rising edge only

enumerator kPWM_RiseAndFallEdge

Capture on rising or falling edge

enum __pwm_force_signal

PWM output options when a FORCE_OUT signal is asserted.

Values:

enumerator kPWM_UsePwm

Generated PWM signal is used by the deadtime logic.

enumerator kPWM_InvertedPwm

Inverted PWM signal is used by the deadtime logic.

enumerator kPWM_SoftwareControl

Software controlled value is used by the deadtime logic.

enumerator kPWM_UseExternal

PWM_EXT_A signal is used by the deadtime logic.

enum __pwm_chnl_pair_operation

Options available for the PWM A & B pair operation.

Values:

enumerator kPWM_Independent

PWM A & PWM B operate as 2 independent channels

enumerator kPWM_ComplementaryPwmA

PWM A & PWM B are complementary channels, PWM A generates the signal

enumerator kPWM_ComplementaryPwmB

PWM A & PWM B are complementary channels, PWM B generates the signal

enum __pwm_register_reload

Options available on how to load the buffered-registers with new values.

Values:

enumerator kPWM_ReloadImmediate

Buffered-registers get loaded with new values as soon as LDOK bit is set

enumerator kPWM_ReloadPwmHalfCycle

Registers loaded on a PWM half cycle

enumerator kPWM_ReloadPwmFullCycle

Registers loaded on a PWM full cycle

enumerator kPWM_ReloadPwmHalfAndFullCycle

Registers loaded on a PWM half & full cycle

enum __pwm_fault_recovery_mode

Options available on how to re-enable the PWM output when recovering from a fault.

Values:

enumerator kPWM_NoRecovery

PWM output will stay inactive

enumerator kPWM_RecoverHalfCycle

PWM output re-enabled at the first half cycle

enumerator kPWM_RecoverFullCycle

PWM output re-enabled at the first full cycle

enumerator kPWM_RecoverHalfAndFullCycle

PWM output re-enabled at the first half or full cycle

enum __pwm_interrupt_enable

List of PWM interrupt options.

Values:

enumerator kPWM_CompareVal0InterruptEnable

PWM VAL0 compare interrupt

enumerator kPWM_CompareVal1InterruptEnable

PWM VAL1 compare interrupt

enumerator kPWM_CompareVal2InterruptEnable

PWM VAL2 compare interrupt

enumerator kPWM_CompareVal3InterruptEnable

PWM VAL3 compare interrupt

enumerator kPWM_CompareVal4InterruptEnable

PWM VAL4 compare interrupt

enumerator kPWM_CompareVal5InterruptEnable

PWM VAL5 compare interrupt

enumerator kPWM_CaptureX0InterruptEnable

PWM capture X0 interrupt

enumerator kPWM__CaptureX1InterruptEnable

PWM capture X1 interrupt

enumerator kPWM__CaptureB0InterruptEnable

PWM capture B0 interrupt

enumerator kPWM__CaptureB1InterruptEnable

PWM capture B1 interrupt

enumerator kPWM__CaptureA0InterruptEnable

PWM capture A0 interrupt

enumerator kPWM__CaptureA1InterruptEnable

PWM capture A1 interrupt

enumerator kPWM__ReloadInterruptEnable

PWM reload interrupt

enumerator kPWM__ReloadErrorInterruptEnable

PWM reload error interrupt

enumerator kPWM__Fault0InterruptEnable

PWM fault 0 interrupt

enumerator kPWM__Fault1InterruptEnable

PWM fault 1 interrupt

enumerator kPWM__Fault2InterruptEnable

PWM fault 2 interrupt

enumerator kPWM__Fault3InterruptEnable

PWM fault 3 interrupt

enum _pwm_status_flags

List of PWM status flags.

Values:

enumerator kPWM__CompareVal0Flag

PWM VAL0 compare flag

enumerator kPWM__CompareVal1Flag

PWM VAL1 compare flag

enumerator kPWM__CompareVal2Flag

PWM VAL2 compare flag

enumerator kPWM__CompareVal3Flag

PWM VAL3 compare flag

enumerator kPWM__CompareVal4Flag

PWM VAL4 compare flag

enumerator kPWM__CompareVal5Flag

PWM VAL5 compare flag

enumerator kPWM__CaptureX0Flag

PWM capture X0 flag

enumerator kPWM__CaptureX1Flag

PWM capture X1 flag

enumerator kPWM__CaptureB0Flag

PWM capture B0 flag

enumerator kPWM__CaptureB1Flag

PWM capture B1 flag

enumerator kPWM__CaptureA0Flag

PWM capture A0 flag

enumerator kPWM__CaptureA1Flag

PWM capture A1 flag

enumerator kPWM__ReloadFlag

PWM reload flag

enumerator kPWM__ReloadErrorFlag

PWM reload error flag

enumerator kPWM__RegUpdatedFlag

PWM registers updated flag

enumerator kPWM__Fault0Flag

PWM fault 0 flag

enumerator kPWM__Fault1Flag

PWM fault 1 flag

enumerator kPWM__Fault2Flag

PWM fault 2 flag

enumerator kPWM__Fault3Flag

PWM fault 3 flag

enum _pwm_dma_enable

List of PWM DMA options.

Values:

enumerator kPWM__CaptureX0DMAEnable

PWM capture X0 DMA

enumerator kPWM__CaptureX1DMAEnable

PWM capture X1 DMA

enumerator kPWM__CaptureB0DMAEnable

PWM capture B0 DMA

enumerator kPWM__CaptureB1DMAEnable

PWM capture B1 DMA

enumerator kPWM__CaptureA0DMAEnable

PWM capture A0 DMA

enumerator kPWM__CaptureA1DMAEnable

PWM capture A1 DMA

enum _pwm_dma_source_select

List of PWM capture DMA enable source select.

Values:

enumerator kPWM__DMARequestDisable

Read DMA requests disabled

enumerator kPWM_DMAWatermarksEnable

Exceeding a FIFO watermark sets the DMA read request

enumerator kPWM_DMALocalSync

A local sync (VAL1 matches counter) sets the read DMA request

enumerator kPWM_DMALocalReload

A local reload (STS[RF] being set) sets the read DMA request

enum __pwm_watermark_control

PWM FIFO Watermark AND Control.

Values:

enumerator kPWM_FIFOWatermarksOR

Selected FIFO watermarks are OR'ed together

enumerator kPWM_FIFOWatermarksAND

Selected FIFO watermarks are AND'ed together

enum __pwm_mode

PWM operation mode.

Values:

enumerator kPWM_SignedCenterAligned

Signed center-aligned

enumerator kPWM_CenterAligned

Unsigned cente-aligned

enumerator kPWM_SignedEdgeAligned

Signed edge-aligned

enumerator kPWM_EdgeAligned

Unsigned edge-aligned

enum __pwm_level_select

PWM output pulse mode, high-true or low-true.

Values:

enumerator kPWM_HighTrue

High level represents “on” or “active” state

enumerator kPWM_LowTrue

Low level represents “on” or “active” state

enum __pwm_fault_state

PWM output fault status.

Values:

enumerator kPWM_PwmFaultState0

Output is forced to logic 0 state prior to consideration of output polarity control.

enumerator kPWM_PwmFaultState1

Output is forced to logic 1 state prior to consideration of output polarity control.

enumerator kPWM_PwmFaultState2

Output is tristated.

enumerator kPWM_PwmFaultState3

Output is tristated.

enum `_pwm_reload_source_select`

PWM reload source select.

Values:

enumerator `kPWM_LocalReload`

The local reload signal is used to reload registers

enumerator `kPWM_MasterReload`

The master reload signal (from submodule 0) is used to reload

enum `_pwm_fault_clear`

PWM fault clearing options.

Values:

enumerator `kPWM_Automatic`

Automatic fault clearing

enumerator `kPWM_ManualNormal`

Manual fault clearing with no fault safety mode

enumerator `kPWM_ManualSafety`

Manual fault clearing with fault safety mode

enum `_pwm_module_control`

Options for submodule master control operation.

Values:

enumerator `kPWM_Control_Module_0`

Control submodule 0's start/stop,buffer reload operation

enumerator `kPWM_Control_Module_1`

Control submodule 1's start/stop,buffer reload operation

enumerator `kPWM_Control_Module_2`

Control submodule 2's start/stop,buffer reload operation

enumerator `kPWM_Control_Module_3`

Control submodule 3's start/stop,buffer reload operation

typedef enum `_pwm_submodule` `pwm_submodule_t`

List of PWM submodules.

typedef enum `_pwm_channels` `pwm_channels_t`

List of PWM channels in each module.

typedef enum `_pwm_value_register` `pwm_value_register_t`

List of PWM value registers.

typedef enum `_pwm_clock_source` `pwm_clock_source_t`

PWM clock source selection.

typedef enum `_pwm_clock_prescale` `pwm_clock_prescale_t`

PWM prescaler factor selection for clock source.

typedef enum `_pwm_force_output_trigger` `pwm_force_output_trigger_t`

Options that can trigger a PWM FORCE_OUT.

typedef enum `_pwm_output_state` `pwm_output_state_t`

PWM channel output status.

`typedef enum _pwm_init_source` `pwm_init_source_t`
PWM counter initialization options.

`typedef enum _pwm_load_frequency` `pwm_load_frequency_t`
PWM load frequency selection.

`typedef enum _pwm_fault_input` `pwm_fault_input_t`
List of PWM fault selections.

`typedef enum _pwm_fault_disable` `pwm_fault_disable_t`
List of PWM fault disable mapping selections.

`typedef enum _pwm_fault_channels` `pwm_fault_channels_t`
List of PWM fault channels.

`typedef enum _pwm_input_capture_edge` `pwm_input_capture_edge_t`
PWM capture edge select.

`typedef enum _pwm_force_signal` `pwm_force_signal_t`
PWM output options when a FORCE_OUT signal is asserted.

`typedef enum _pwm_chnl_pair_operation` `pwm_chnl_pair_operation_t`
Options available for the PWM A & B pair operation.

`typedef enum _pwm_register_reload` `pwm_register_reload_t`
Options available on how to load the buffered-registers with new values.

`typedef enum _pwm_fault_recovery_mode` `pwm_fault_recovery_mode_t`
Options available on how to re-enable the PWM output when recovering from a fault.

`typedef enum _pwm_interrupt_enable` `pwm_interrupt_enable_t`
List of PWM interrupt options.

`typedef enum _pwm_status_flags` `pwm_status_flags_t`
List of PWM status flags.

`typedef enum _pwm_dma_enable` `pwm_dma_enable_t`
List of PWM DMA options.

`typedef enum _pwm_dma_source_select` `pwm_dma_source_select_t`
List of PWM capture DMA enable source select.

`typedef enum _pwm_watermark_control` `pwm_watermark_control_t`
PWM FIFO Watermark AND Control.

`typedef enum _pwm_mode` `pwm_mode_t`
PWM operation mode.

`typedef enum _pwm_level_select` `pwm_level_select_t`
PWM output pulse mode, high-true or low-true.

`typedef enum _pwm_fault_state` `pwm_fault_state_t`
PWM output fault status.

`typedef enum _pwm_reload_source_select` `pwm_reload_source_select_t`
PWM reload source select.

`typedef enum _pwm_fault_clear` `pwm_fault_clear_t`
PWM fault clearing options.

`typedef enum _pwm_module_control` `pwm_module_control_t`
Options for submodule master control operation.

```
typedef struct _pwm_signal_param pwm_signal_param_t
```

Structure for the user to define the PWM signal characteristics.

```
typedef struct _pwm_config pwm_config_t
```

PWM config structure.

This structure holds the configuration settings for the PWM peripheral. To initialize this structure to reasonable defaults, call the PWM_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

```
typedef struct _pwm_fault_input_filter_param pwm_fault_input_filter_param_t
```

Structure for the user to configure the fault input filter.

```
typedef struct _pwm_fault_param pwm_fault_param_t
```

Structure is used to hold the parameters to configure a PWM fault.

```
typedef struct _pwm_input_capture_param pwm_input_capture_param_t
```

Structure is used to hold parameters to configure the capture capability of a signal pin.

```
void PWM_SetupInputCapture(PWM_Type *base, pwm_submodule_t subModule,  
                           pwm_channels_t pwmChannel, const  
                           pwm_input_capture_param_t *inputCaptureParams)
```

Sets up the PWM input capture.

Each PWM submodule has 3 pins that can be configured for use as input capture pins. This function sets up the capture parameters for each pin and enables the pin for input capture operation.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – Channel in the submodule to setup
- inputCaptureParams – Parameters passed in to set up the input pin

```
void PWM_SetupFaultInputFilter(PWM_Type *base, const pwm_fault_input_filter_param_t  
                              *faultInputFilterParams)
```

Sets up the PWM fault channel 0 input filter.

Parameters

- base – PWM peripheral base address
- faultInputFilterParams – Parameters passed in to set up the fault input filter.

```
void PWM_SetupFaultInputFilterExt(PWM_Type *base, pwm_fault_channels_t faultChannel,  
                                 const pwm_fault_input_filter_param_t  
                                 *faultInputFilterParams)
```

Sets up the PWM fault input filter.

Parameters

- base – PWM peripheral base address
- faultChannel – PWM fault channel to configure.
- faultInputFilterParams – Parameters passed in to set up the fault input filter.

```
void PWM_SetupFaults(PWM_Type *base, pwm_fault_input_t faultNum, const  
                    pwm_fault_param_t *faultParams)
```

Sets up the PWM fault channel 0 protection.

Parameters

- base – PWM peripheral base address
- faultNum – PWM fault to configure.
- faultParams – Pointer to the PWM fault config structure

```
void PWM_SetupFaultsExt(PWM_Type *base, pwm_fault_channels_t faultChannel,  
                       pwm_fault_input_t faultNum, const pwm_fault_param_t  
                       *faultParams)
```

Sets up the PWM fault protection.

Parameters

- base – PWM peripheral base address
- faultChannel – PWM fault channel to configure.
- faultNum – PWM fault to configure.
- faultParams – Pointer to the PWM fault config structure

```
void PWM_FaultDefaultConfig(pwm_fault_param_t *config)
```

Fill in the PWM fault config struct with the default settings.

The default values are:

```
config->faultClearingMode = kPWM_Automatic;  
config->faultLevel = false;  
config->enableCombinationalPath = true;  
config->recoverMode = kPWM_NoRecovery;
```

Parameters

- config – Pointer to user's PWM fault config structure.

```
void PWM_SetupForceSignal(PWM_Type *base, pwm_submodule_t subModule, pwm_channels_t  
                         pwmChannel, pwm_force_signal_t mode)
```

Selects the signal to output on a PWM pin when a FORCE_OUT signal is asserted.

The user specifies which channel to configure by supplying the submodule number and whether to modify PWM A or PWM B within that submodule.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – Channel to configure
- mode – Signal to output when a FORCE_OUT is triggered

```
static inline void PWM_SetVALxValue(PWM_Type *base, pwm_submodule_t subModule,  
                                   pwm_value_register_t valueRegister, uint16_t value)
```

Set the PWM VALx registers.

This function allows the user to write value into VAL registers directly. And it will destroying the PWM clock period set by the PWM_SetupPwm()/PWM_SetupPwmPhaseShift() functions. Due to VALx registers are bufferd, the new value will not active uless call PWM_SetPwmLdok() and the reload point is reached.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- valueRegister – VALx register that will be written new value

- `value` – Value that will be write into VALx register

```
static inline uint16_t PWM_GetVALxValue(PWM_Type *base, pwm_submodule_t subModule,  
                                       pwm_value_register_t valueRegister)
```

Get the PWM VALx registers.

Parameters

- `base` – PWM peripheral base address
- `subModule` – PWM submodule to configure
- `valueRegister` – VALx register that will be read value

Returns

The VALx register value

```
static inline void PWM_OutputTriggerEnable(PWM_Type *base, pwm_submodule_t subModule,  
                                          pwm_value_register_t valueRegister, bool activate)
```

Enables or disables the PWM output trigger.

This function allows the user to enable or disable the PWM trigger. The PWM has 2 triggers. Trigger 0 is activated when the counter matches VAL 0, VAL 2, or VAL 4 register. Trigger 1 is activated when the counter matches VAL 1, VAL 3, or VAL 5 register.

Parameters

- `base` – PWM peripheral base address
- `subModule` – PWM submodule to configure
- `valueRegister` – Value register that will activate the trigger
- `activate` – true: Enable the trigger; false: Disable the trigger

```
static inline void PWM_ActivateOutputTrigger(PWM_Type *base, pwm_submodule_t subModule,  
                                             uint16_t valueRegisterMask)
```

Enables the PWM output trigger.

This function allows the user to enable one or more (VAL0-5) PWM trigger.

Parameters

- `base` – PWM peripheral base address
- `subModule` – PWM submodule to configure
- `valueRegisterMask` – Value register mask that will activate one or more (VAL0-5) trigger enumeration `_pwm_value_register_mask`

```
static inline void PWM_DeactivateOutputTrigger(PWM_Type *base, pwm_submodule_t  
                                              subModule, uint16_t valueRegisterMask)
```

Disables the PWM output trigger.

This function allows the user to disables one or more (VAL0-5) PWM trigger.

Parameters

- `base` – PWM peripheral base address
- `subModule` – PWM submodule to configure
- `valueRegisterMask` – Value register mask that will Deactivate one or more (VAL0-5) trigger enumeration `_pwm_value_register_mask`

```
static inline void PWM_SetupSwCtrlOut(PWM_Type *base, pwm_submodule_t subModule,  
                                      pwm_channels_t pwmChannel, bool value)
```

Sets the software control output for a pin to high or low.

The user specifies which channel to modify by supplying the submodule number and whether to modify PWM A or PWM B within that submodule.

Parameters

- `base` – PWM peripheral base address
- `subModule` – PWM submodule to configure
- `pwmChannel` – Channel to configure
- `value` – `true`: Supply a logic 1, `false`: Supply a logic 0.

```
static inline void PWM_SetPwmLdok(PWM_Type *base, uint8_t subModulesToUpdate, bool value)
```

Sets or clears the PWM LDOK bit on a single or multiple submodules.

Set LDOK bit to load buffered values into CTRL[PRSC] and the INIT, FRACVAL and VAL registers. The values are loaded immediately if `kPWM_ReloadImmediate` option was chosen during config. Else the values are loaded at the next PWM reload point. This function can issue the load command to multiple submodules at the same time.

Parameters

- `base` – PWM peripheral base address
- `subModulesToUpdate` – PWM submodules to update with buffered values. This is a logical OR of members of the enumeration `pwm_module_control_t`
- `value` – `true`: Set LDOK bit for the submodule list; `false`: Clear LDOK bit

```
static inline void PWM_SetPwmFaultState(PWM_Type *base, pwm_submodule_t subModule, pwm_channels_t pwmChannel, pwm_fault_state_t faultState)
```

Set PWM output fault status.

These bits determine the fault state for the PWM_A output in fault conditions and STOP mode. It may also define the output state in WAIT and DEBUG modes depending on the settings of CTRL2[WAITEN] and CTRL2[DBGEN]. This function can update PWM output fault status.

Parameters

- `base` – PWM peripheral base address
- `subModule` – PWM submodule to configure
- `pwmChannel` – Channel to configure
- `faultState` – PWM output fault status

```
static inline void PWM_SetupFaultDisableMap(PWM_Type *base, pwm_submodule_t subModule, pwm_channels_t pwmChannel, pwm_fault_channels_t pwm_fault_channels, uint16_t value)
```

Set PWM fault disable mapping.

Each of the four bits of this read/write field is one-to-one associated with the four FAULTx inputs of fault channel 0/1. The PWM output will be turned off if there is a logic 1 on an FAULTx input and a 1 in the corresponding bit of this field. A reset sets all bits in this field.

Parameters

- `base` – PWM peripheral base address
- `subModule` – PWM submodule to configure
- `pwmChannel` – PWM channel to configure

- `pwm_fault_channels` – PWM fault channel to configure
- `value` – Fault disable mapping mask value enumeration
`pwm_fault_disable_t`

```
static inline void PWM_OutputEnable(PWM_Type *base, pwm_channels_t pwmChannel,  
                                   pwm_submodule_t subModule)
```

Set PWM output enable.

This feature allows the user to enable the PWM Output. Recommend to invoke this API after PWM and fault configuration. But invoke this API before configure MCTRL register is okay, such as set LDOK or start timer.

Parameters

- `base` – PWM peripheral base address
- `pwmChannel` – PWM channel to configure
- `subModule` – PWM submodule to configure

```
static inline void PWM_OutputDisable(PWM_Type *base, pwm_channels_t pwmChannel,  
                                     pwm_submodule_t subModule)
```

Set PWM output disable.

This feature allows the user to disable the PWM output. Recommend to invoke this API after PWM and fault configuration. But invoke this API before configure MCTRL register is okay, such as set LDOK or start timer.

Parameters

- `base` – PWM peripheral base address
- `pwmChannel` – PWM channel to configure
- `subModule` – PWM submodule to configure

```
uint8_t PWM_GetPwmChannelState(PWM_Type *base, pwm_submodule_t subModule,  
                               pwm_channels_t pwmChannel)
```

Get the dutycycle value.

Parameters

- `base` – PWM peripheral base address
- `subModule` – PWM submodule to configure
- `pwmChannel` – PWM channel to configure

Returns

Current channel dutycycle value.

```
status_t PWM_SetOutputToIdle(PWM_Type *base, pwm_channels_t pwmChannel,  
                             pwm_submodule_t subModule, bool idleStatus)
```

Set PWM output in idle status (high or low).

Note: This API should call after `PWM_SetupPwm()` APIs, and PWMX submodule is not supported.

Parameters

- `base` – PWM peripheral base address
- `pwmChannel` – PWM channel to configure
- `subModule` – PWM submodule to configure

- `idleStatus` – True: PWM output is high in idle status; false: PWM output is low in idle status.

Returns

`kStatus_Fail` if there was error setting up the signal; `kStatus_Success` if set output idle success

```
void PWM_SetClockMode(PWM_Type *base, pwm_submodule_t subModule,  
                     pwm_clock_prescale_t prescaler)
```

Set the pwm submodule prescaler.

Parameters

- `base` – PWM peripheral base address
- `subModule` – PWM submodule to configure
- `prescaler` – Set prescaler value

```
void PWM_SetPwmForceOutputToZero(PWM_Type *base, pwm_submodule_t subModule,  
                                 pwm_channels_t pwmChannel, bool forcetozero)
```

This function enables-disables the forcing of the output of a given eFlexPwm channel to logic 0.

Parameters

- `base` – PWM peripheral base address
- `pwmChannel` – PWM channel to configure
- `subModule` – PWM submodule to configure
- `forcetozero` – True: Enable the pwm force output to zero; False: Disable the pwm output resumes normal function.

```
void PWM_SetChannelOutput(PWM_Type *base, pwm_submodule_t subModule,  
                          pwm_channels_t pwmChannel, pwm_output_state_t outputstate)
```

This function set the output state of the PWM pin as requested for the current cycle.

Parameters

- `base` – PWM peripheral base address
- `subModule` – PWM submodule to configure
- `pwmChannel` – PWM channel to configure
- `outputstate` – Set pwm output state, see `pwm_output_state_t`.

```
status_t PWM_SetPhaseDelay(PWM_Type *base, pwm_channels_t pwmChannel,  
                           pwm_submodule_t subModule, uint16_t delayCycles)
```

This function set the phase delay from the master sync signal of submodule 0.

Parameters

- `base` – PWM peripheral base address
- `subModule` – PWM submodule to configure
- `pwmChannel` – PWM channel to configure
- `delayCycles` – Number of cycles delayed from submodule 0.

Returns

`kStatus_Fail` if the number of delay cycles is set larger than the period defined in submodule 0; `kStatus_Success` if set phase delay success

```
static inline void PWM_SetFilterSampleCount(PWM_Type *base, pwm_channels_t pwmChannel,
                                           pwm_submodule_t subModule, uint8_t
                                           filterSampleCount)
```

This function set the number of consecutive samples that must agree prior to the input filter.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- filterSampleCount – Number of consecutive samples.

```
static inline void PWM_SetFilterSamplePeriod(PWM_Type *base, pwm_channels_t pwmChannel,
                                           pwm_submodule_t subModule, uint8_t
                                           filterSamplePeriod)
```

This function set the sampling period of the fault pin input filter.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- filterSamplePeriod – Sampling period of input filter.

PWM_SUBMODULE_SWCONTROL_WIDTH

Number of bits per submodule for software output control

PWM_SUBMODULE_CHANNEL

Submodule channels include PWMA, PWMB, PWMX.

```
struct __pwm_signal_param
```

#include <fsl_pwm.h> Structure for the user to define the PWM signal characteristics.

Public Members

pwm_channels_t pwmChannel

PWM channel being configured; PWM A or PWM B

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)

pwm_level_select_t level

PWM output active level select

uint16_t deadtimeValue

The deadtime value; only used if channel pair is operating in complementary mode

pwm_fault_state_t faultState

PWM output fault status

bool pwmchannelenable

Enable PWM output

struct __pwm_config

#include <fsl_pwm.h> PWM config structure.

This structure holds the configuration settings for the PWM peripheral. To initialize this structure to reasonable defaults, call the PWM_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

Public Members

bool enableDebugMode

true: PWM continues to run in debug mode; false: PWM is paused in debug mode

pwm_init_source_t initializationControl

Option to initialize the counter

pwm_clock_source_t clockSource

Clock source for the counter

pwm_clock_prescale_t prescale

Pre-scaler to divide down the clock

pwm_chnl_pair_operation_t pairOperation

Channel pair in independent or complementary mode

pwm_register_reload_t reloadLogic

PWM Reload logic setup

pwm_reload_source_select_t reloadSelect

Reload source select

pwm_load_frequency_t reloadFrequency

Specifies when to reload, used when user's choice is not immediate reload

pwm_force_output_trigger_t forceTrigger

Specify which signal will trigger a FORCE_OUT

struct __pwm_fault_input_filter_param

#include <fsl_pwm.h> Structure for the user to configure the fault input filter.

Public Members

uint8_t faultFilterCount

Fault filter count

uint8_t faultFilterPeriod

Fault filter period; value of 0 will bypass the filter

bool faultGlitchStretch

Fault Glitch Stretch Enable: A logic 1 means that input fault signals will be stretched to at least 2 IPBus clock cycles

struct __pwm_fault_param

#include <fsl_pwm.h> Structure is used to hold the parameters to configure a PWM fault.

Public Members

pwm_fault_clear_t faultClearingMode

Fault clearing mode to use

bool faultLevel

true: Logic 1 indicates fault; false: Logic 0 indicates fault

bool enableCombinationalPath

true: Combinational Path from fault input is enabled; false: No combination path is available

pwm_fault_recovery_mode_t recoverMode

Specify when to re-enable the PWM output

struct *_pwm_input_capture_param*

#include <fsl_pwm.h> Structure is used to hold parameters to configure the capture capability of a signal pin.

Public Members

bool captureInputSel

true: Use the edge counter signal as source false: Use the raw input signal from the pin as source

uint8_t edgeCompareValue

Compare value, used only if edge counter is used as source

pwm_input_capture_edge_t edge0

Specify which edge causes a capture for input circuitry 0

pwm_input_capture_edge_t edge1

Specify which edge causes a capture for input circuitry 1

bool enableOneShotCapture

true: Use one-shot capture mode; false: Use free-running capture mode

uint8_t fifoWatermark

Watermark level for capture FIFO. The capture flags in the status register will set if the word count in the FIFO is greater than this watermark level

2.90 PXP: Pixel Pipeline

void PXP_Init(PXP_Type *base)

Initialize the PXP.

This function enables the PXP peripheral clock, and resets the PXP registers to default status.

Parameters

- base – PXP peripheral base address.

void PXP_Deinit(PXP_Type *base)

De-initialize the PXP.

This function disables the PXP peripheral clock.

Parameters

- base – PXP peripheral base address.

void PXP_Reset(PXP_Type *base)

Reset the PXP.

This function resets the PXP peripheral registers to default status.

Parameters

- base – PXP peripheral base address.

void PXP_ResetControl(PXP_Type *base)

Reset the PXP and the control register to initialized state.

Parameters

- base – PXP peripheral base address.

static inline void PXP_Start(PXP_Type *base)

Start process.

Start PXP process using current configuration.

Parameters

- base – PXP peripheral base address.

static inline void PXP_EnableLcdHandShake(PXP_Type *base, bool enable)

Enable or disable LCD hand shake.

Parameters

- base – PXP peripheral base address.
- enable – True to enable, false to disable.

static inline void PXP_EnableContinuousRun(PXP_Type *base, bool enable)

Enable or disable continuous run.

If continuous run not enabled, PXP_Start starts the PXP process. When completed, PXP enters idle mode and flag kPXP_CompleteFlag asserts.

If continuous run enabled, the PXP will repeat based on the current configuration register settings.

Parameters

- base – PXP peripheral base address.
- enable – True to enable, false to disable.

static inline void PXP_SetProcessBlockSize(PXP_Type *base, *pxp_block_size_t* size)

Set the PXP processing block size.

This function chooses the pixel block size that PXP using during process. Larger block size means better performance, but be careful that when PXP is rotating, the output must be divisible by the block size selected.

Parameters

- base – PXP peripheral base address.
- size – The pixel block size.

static inline void PXP_EnableProcessEngine(PXP_Type *base, uint32_t mask, bool enable)

Enables or disables PXP engines in the process flow.

Parameters

- base – PXP peripheral base address.
- mask – The engines to enable. Logical OR of *pxp_process_engine_name_t*.

- enable – true to enable, false to disable.

static inline uint32_t PXP_GetStatusFlags(PXP_Type *base)

Gets PXP status flags.

This function gets all PXP status flags. The flags are returned as the logical OR value of the enumerators `_pxp_flags`. To check a specific status, compare the return value with enumerators in `_pxp_flags`. For example, to check whether the PXP has completed process, use like this:

```
if (kPXP_CompleteFlag & PXP_GetStatusFlags(PXP))
{
    ...
}
```

Parameters

- base – PXP peripheral base address.

Returns

PXP status flags which are OR'ed by the enumerators in the `_pxp_flags`.

static inline void PXP_ClearStatusFlags(PXP_Type *base, uint32_t statusMask)

Clears status flags with the provided mask.

This function clears PXP status flags with a provided mask.

Parameters

- base – PXP peripheral base address.
- statusMask – The status flags to be cleared; it is logical OR value of `_pxp_flags`.

static inline uint8_t PXP_GetAxiErrorId(PXP_Type *base, uint8_t axiIndex)

Gets the AXI ID of the failing bus operation.

Parameters

- base – PXP peripheral base address.
- axiIndex – Which AXI to get
 - 0: AXI0
 - 1: AXI1

Returns

The AXI ID of the failing bus operation.

static inline void PXP_EnableInterrupts(PXP_Type *base, uint32_t mask)

Enables PXP interrupts according to the provided mask.

This function enables the PXP interrupts according to the provided mask. The mask is a logical OR of enumeration members. See `_pxp_interrupt_enable`. For example, to enable PXP process complete interrupt and command loaded interrupt, do the following.

```
PXP_EnableInterrupts(PXP, kPXP_CommandLoadInterruptEnable | kPXP_
    ↪ CompleteInterruptEnable);
```

Parameters

- base – PXP peripheral base address.
- mask – The interrupts to enable. Logical OR of `_pxp_interrupt_enable`.


```
static inline void PXP_DisableInterrupts(PXP_Type *base, uint32_t mask)
```

Disables PXP interrupts according to the provided mask.

This function disables the PXP interrupts according to the provided mask. The mask is a logical OR of enumeration members. See `_pxp_interrupt_enable`.

Parameters

- `base` – PXP peripheral base address.
- `mask` – The interrupts to disable. Logical OR of `_pxp_interrupt_enable`.

```
void PXP_SetAlphaSurfaceBufferConfig(PXP_Type *base, const pxp_as_buffer_config_t *config)
```

Set the alpha surface input buffer configuration.

Parameters

- `base` – PXP peripheral base address.
- `config` – Pointer to the configuration.

```
void PXP_SetAlphaSurfaceBlendConfig(PXP_Type *base, const pxp_as_blend_config_t *config)
```

Set the alpha surface blending configuration.

Parameters

- `base` – PXP peripheral base address.
- `config` – Pointer to the configuration structure.

```
void PXP_SetAlphaSurfaceBlendSecondaryConfig(PXP_Type *base, const  
                                              pxp_as_blend_secondary_config_t *config)
```

Set the alpha surface blending configuration for the secondary engine.

Parameters

- `base` – PXP peripheral base address.
- `config` – Pointer to the configuration structure.

```
void PXP_SetAlphaSurfaceOverlayColorKey(PXP_Type *base, uint8_t num, uint32_t colorKeyLow,  
                                         uint32_t colorKeyHigh)
```

Set the alpha surface overlay color key.

If a pixel in the current overlay image with a color that falls in the range from the `colorKeyLow` to `colorKeyHigh` range, it will use the process surface pixel value for that location. If no PS image is present or if the PS image also matches its colorkey range, the PS background color is used.

Note: Colorkey operations are higher priority than alpha or ROP operations

Parameters

- `base` – PXP peripheral base address.
- `num` – instance number. 0 for alpha engine A, 1 for alpha engine B.
- `colorKeyLow` – Color key low range.
- `colorKeyHigh` – Color key high range.

```
static inline void PXP_EnableAlphaSurfaceOverlayColorKey(PXP_Type *base, uint32_t num, bool  
                                                         enable)
```

Enable or disable the alpha surface color key.

Parameters

- base – PXP peripheral base address.
- num – instance number. 0 for alpha engine A, 1 for alpha engine B.
- enable – True to enable, false to disable.

```
void PXP_SetAlphaSurfacePosition(PXP_Type *base, uint16_t upperLeftX, uint16_t upperLeftY,  
                                uint16_t lowerRightX, uint16_t lowerRightY)
```

Set the alpha surface position in output buffer.

Parameters

- base – PXP peripheral base address.
- upperLeftX – X of the upper left corner.
- upperLeftY – Y of the upper left corner.
- lowerRightX – X of the lower right corner.
- lowerRightY – Y of the lower right corner.

```
static inline void PXP_SetProcessSurfaceBackGroundColor(PXP_Type *base, uint8_t num,  
                                                         uint32_t backGroundColor)
```

Set the back ground color of PS.

Parameters

- base – PXP peripheral base address.
- num – instance number. 0 for alpha engine A, 1 for alpha engine B.
- backGroundColor – Pixel value of the background color.

```
void PXP_SetProcessSurfaceBufferConfig(PXP_Type *base, const pxp_ps_buffer_config_t *config)
```

Set the process surface input buffer configuration.

Parameters

- base – PXP peripheral base address.
- config – Pointer to the configuration.

```
void PXP_SetProcessSurfaceScaler(PXP_Type *base, uint16_t inputWidth, uint16_t inputHeight,  
                                uint16_t outputWidth, uint16_t outputHeight)
```

Set the process surface scaler configuration.

The valid down scale fact is $1/(2^{12}) \sim 16$.

Parameters

- base – PXP peripheral base address.
- inputWidth – Input image width.
- inputHeight – Input image height.
- outputWidth – Output image width.
- outputHeight – Output image height.

```
void PXP_SetProcessSurfacePosition(PXP_Type *base, uint16_t upperLeftX, uint16_t upperLeftY,  
                                   uint16_t lowerRightX, uint16_t lowerRightY)
```

Set the process surface position in output buffer.

Parameters

- base – PXP peripheral base address.
- upperLeftX – X of the upper left corner.
- upperLeftY – Y of the upper left corner.

- lowerRightX – X of the lower right corner.
- lowerRightY – Y of the lower right corner.

```
void PXP_SetProcessSurfaceColorKey(PXP_Type *base, uint8_t num, uint32_t colorKeyLow,
                                   uint32_t colorKeyHigh)
```

Set the process surface color key.

If the PS image matches colorkey range, the PS background color is output. Set colorKeyLow to 0xFFFFFFFF and p colorKeyHigh to 0 will disable the colorkeying.

Parameters

- base – PXP peripheral base address.
- num – instance number. 0 for alpha engine A, 1 for alpha engine B.
- colorKeyLow – Color key low range.
- colorKeyHigh – Color key high range.

```
static inline void PXP_SetProcessSurfaceYUVFormat(PXP_Type *base, pxp_ps_yuv_format_t
                                                  format)
```

Set the process surface input pixel format YUV or YCbCr.

If process surface input pixel format is YUV and CSC1 is not enabled, in other words, the process surface output pixel format is also YUV, then this function should be called to set whether input pixel format is YUV or YCbCr.

Parameters

- base – PXP peripheral base address.
- format – The YUV format.

```
void PXP_SetOutputBufferConfig(PXP_Type *base, const pxp_output_buffer_config_t *config)
```

Set the PXP output buffer configuration.

Parameters

- base – PXP peripheral base address.
- config – Pointer to the configuration.

```
static inline void PXP_SetOverwrittenAlphaValue(PXP_Type *base, uint8_t alpha)
```

Set the global overwritten alpha value.

If global overwritten alpha is enabled, the alpha component in output buffer pixels will be overwritten, otherwise the computed alpha value is used.

Parameters

- base – PXP peripheral base address.
- alpha – The alpha value.

```
static inline void PXP_EnableOverWrittenAlpha(PXP_Type *base, bool enable)
```

Enable or disable the global overwritten alpha value.

If global overwritten alpha is enabled, the alpha component in output buffer pixels will be overwritten, otherwise the computed alpha value is used.

Parameters

- base – PXP peripheral base address.
- enable – True to enable, false to disable.

```
static inline void PXP_SetRotateConfig(PXP_Type *base, pxp_rotate_position_t position,  
                                       pxp_rotate_degree_t degree, pxp_flip_mode_t flipMode)
```

Set the rotation configuration.

The PXP could rotate the process surface or the output buffer. There are two PXP versions:

- Version 1: Only has one rotate sub module, the output buffer and process surface share the same rotate sub module, which means the process surface and output buffer could not be rotate at the same time. When pass in `kPXP_RotateOutputBuffer`, the process surface could not use the rotate, Also when pass in `kPXP_RotateProcessSurface`, output buffer could not use the rotate.
- Version 2: Has two separate rotate sub modules, the output buffer and process surface could configure the rotation independently.

Upper layer could use the macro `PXP_SHARE_ROTATE` to check which version is. `PXP_SHARE_ROTATE=1` means version 1.

Note: This function is different depends on the macro `PXP_SHARE_ROTATE`.

Parameters

- base – PXP peripheral base address.
- position – Rotate process surface or output buffer.
- degree – Rotate degree.
- flipMode – Flip mode.

```
void PXP_BuildRect(PXP_Type *base, pxp_output_pixel_format_t outFormat, uint32_t value,  
                  uint16_t width, uint16_t height, uint16_t pitch, uint32_t outAddr)
```

Build a solid rectangle of given pixel value.

Parameters

- base – PXP peripheral base address.
- outFormat – output pixel format.
- value – The value of the pixel to be filled in the rectangle in ARGB8888 format.
- width – width of the rectangle.
- height – height of the rectangle.
- pitch – output pitch in byte.
- outAddr – address of the memory to store the rectangle.

```
void PXP_SetNextCommand(PXP_Type *base, void *commandAddr)
```

Set the next command.

The PXP supports a primitive ability to queue up one operation while the current operation is running. Workflow:

- a. Prepare the PXP register values except STAT, CSCCOEFn, NEXT in the memory in the order they appear in the register map.
- b. Call this function sets the new operation to PXP.
- c. There are two methods to check whether the PXP has loaded the new operation. The first method is using `PXP_IsNextCommandPending`. If there is new operation not loaded by the PXP, this function returns true. The second method is checking the flag

kPXP_CommandLoadFlag, if command loaded, this flag asserts. User could enable interrupt kPXP_CommandLoadInterruptEnable to get the loaded signal in interrupt way.

d. When command loaded by PXP, a new command could be set using this function.

```
uint32_t pxp_command1[48];
uint32_t pxp_command2[48];

pxp_command1[0] = ...;
pxp_command1[1] = ...;
...
pxp_command2[0] = ...;
pxp_command2[1] = ...;
...

while (PXP_IsNextCommandPending(PXP))
{
}

PXP_SetNextCommand(PXP, pxp_command1);

while (PXP_IsNextCommandPending(PXP))
{
}

PXP_SetNextCommand(PXP, pxp_command2);
```

Parameters

- base – PXP peripheral base address.
- commandAddr – Address of the new command.

static inline bool PXP_IsNextCommandPending(PXP_Type *base)

Check whether the next command is pending.

Parameters

- base – UART peripheral base address.

Returns

True is pending, false is not.

static inline void PXP_CancelNextCommand(PXP_Type *base)

Cancel command set by PXP_SetNextCommand.

Parameters

- base – UART peripheral base address.

void PXP_SetCsc1Mode(PXP_Type *base, *pxp_csc1_mode_t* mode)

Set the CSC1 mode.

The CSC1 module receives scaled YUV/YCbCr444 pixels from the scale engine and converts the pixels to the RGB888 color space. It could only be used by process surface.

Parameters

- base – PXP peripheral base address.
- mode – The conversion mode.

static inline void PXP_EnableCsc1(PXP_Type *base, bool enable)

Enable or disable the CSC1.

Parameters

- base – PXP peripheral base address.

- enable – True to enable, false to disable.

```
void PXP_SetInternalRamData(PXP_Type *base, pxp_ram_t ram, uint16_t bytesNum, uint8_t *data, uint16_t memStartAddr)
```

Write data to the PXP internal memory.

Parameters

- base – PXP peripheral base address.
- ram – Which internal memory to write.
- bytesNum – How many bytes to write.
- data – Pointer to the data to write.
- memStartAddr – The start address in the internal memory to write the data.

```
void PXP_SetDitherFinalLutData(PXP_Type *base, const pxp_dither_final_lut_data_t *data)
```

Set the dither final LUT data.

The dither final LUT is only applicable to dither engine 0. It takes the bits[7:4] of the output pixel and looks up and 8 bit value from the 16 value LUT to generate the final output pixel to the next process module.

Parameters

- base – PXP peripheral base address.
- data – Pointer to the LUT data to set.

```
static inline void PXP_SetDitherConfig(PXP_Type *base, const pxp_dither_config_t *config)
```

Set the configuration for the dither block.

If the pre-dither LUT, post-dither LUT or ordered dither is used, please call PXP_SetInternalRamData to set the LUT data to internal memory.

If the final LUT is used, please call PXP_SetDitherFinalLutData to set the LUT data.

Note: When using ordered dithering, please set the PXP process block size same with the ordered dithering matrix size using function PXP_SetProcessBlockSize.

Parameters

- base – PXP peripheral base address.
- config – Pointer to the configuration.

```
void PXP_EnableDither(PXP_Type *base, bool enable)
```

Enable or disable dither engine in the PXP process path.

After the initialize function PXP_Init, the dither engine is disabled and not use in the PXP processing path. This function enables the dither engine and routes the dither engine output to the output buffer. When the dither engine is enabled using this function, PXP_SetDitherConfig must be called to configure dither engine correctly, otherwise there is not output to the output buffer.

Parameters

- base – PXP peripheral base address.
- enable – Pass in true to enable, false to disable.

```
void PXP_SetPorterDuffConfig(PXP_Type *base, uint8_t num, const pxp_porter_duff_config_t *config)
```

Set the Porter Duff configuration for one of the alpha process engine.

Parameters

- base – PXP peripheral base address.
- num – instance number.
- config – Pointer to the configuration.

status_t PXP_GetPorterDuffConfigExt(*pxp_porter_duff_blend_mode_t* mode,
pxp_porter_duff_config_t *config, uint8_t
dstGlobalAlphaMode, uint8_t dstAlphaMode, uint8_t
dstColorMode, uint8_t srcGlobalAlphaMode, uint8_t
srcAlphaMode, uint8_t srcColorMode, uint8_t
dstGlobalAlpha, uint8_t srcGlobalAlpha)

Get the Porter Duff configuration.

The FactorMode are selected based on blend mode, the other values are set based on input parameters. These values could be modified after calling this function. This function is extened PXP_GetPorterDuffConfig.

Parameters

- mode – The blend mode.
- config – Pointer to the configuration.
- dstGlobalAlphaMode – Destination layer (or PS, s0) global alpha mode, see *pxp_porter_duff_global_alpha_mode*
- dstAlphaMode – Destination layer (or PS, s0) alpha mode, see *pxp_porter_duff_alpha_mode*.
- dstColorMode – Destination layer (or PS, s0) color mode, see *pxp_porter_duff_color_mode*.
- srcGlobalAlphaMode – Source layer (or AS, s1) global alpha mode, see *pxp_porter_duff_global_alpha_mode*
- srcAlphaMode – Source layer (or AS, s1) alpha mode, see *pxp_porter_duff_alpha_mode*.
- srcColorMode – Source layer (or AS, s1) color mode, see *pxp_porter_duff_color_mode*.
- dstGlobalAlpha – Destination layer (or PS, s0) global alpha value, 0~255
- srcGlobalAlpha – Source layer (or AS, s1) global alpha value, 0~255

Return values

- kStatus_Success – Successfully get the configuratoin.
- kStatus_InvalidArgument – The blend mode not supported.

static inline *status_t* PXP_GetPorterDuffConfig(*pxp_porter_duff_blend_mode_t* mode,
pxp_porter_duff_config_t *config)

Get the Porter Duff configuration by blend mode.

The FactorMode are selected based on blend mode, the AlphaMode are set to kPXP_PorterDuffAlphaStraight, the ColorMode are set to kPXP_PorterDuffColorWithAlpha, the GlobalAlphaMode are set to kPXP_PorterDuffLocalAlpha. These values could be modified after calling this function.

Parameters

- mode – The blend mode.
- config – Pointer to the configuration.

Return values

- kStatus_Success – Successfully get the configuratoin.
- kStatus_InvalidArgument – The blend mode not supported.

FSL_PXP_DRIVER_VERSION

enum __pxp_interrupt_enable

PXP interrupts to enable.

Values:

enumerator kPXP_CompleteInterruptEnable

PXP process completed. bit 1

enumerator kPXP_CommandLoadInterruptEnable

Interrupt to show that the command set by PXP_SetNextCommand has been loaded.
bit 2

enumerator kPXP_CompressDoneInterruptEnable

Compress done interrupt enable. bit 15

enumerator kPXP_InputFetchCh0InterruptEnable

Input fetch channel 0 completed. bit 16

enumerator kPXP_InputFetchCh1InterruptEnable

Input fetch channel 1 completed. bit 17

enumerator kPXP_InputStoreCh0InterruptEnable

Input store channel 0 completed. bit 18

enumerator kPXP_InputStoreCh1InterruptEnable

Input store channel 1 completed. bit 19

enumerator kPXP_DitherFetchCh0InterruptEnable

Dither fetch channel 0 completed. bit 20

enumerator kPXP_DitherFetchCh1InterruptEnable

Dither fetch channel 1 completed. bit 21

enumerator kPXP_DitherStoreCh0InterruptEnable

Dither store channle 0 completed. bit 22

enumerator kPXP_DitherStoreCh1InterruptEnable

Dither store channle 1 completed. bit 23

enumerator kPXP_WfeaStoreCh0InterruptEnable

WFE-A store channel 0 completed. bit 24

enumerator kPXP_WfeaStoreCh1InterruptEnable

WFE-A store channel 1 completed. bit 25

enumerator kPXP_WfebStoreCh0InterruptEnable

WFE-B store channel 0 completed. bit 26

enumerator kPXP_WfebStoreCh1InterruptEnable

WFE-B store channel 1 completed. bit 27

enumerator kPXP_InputStoreInterruptEnable

Input store completed. bit 28

enumerator kPXP_DitherStoreInterruptEnable

Dither store completed. bit 29

enumerator kPXP_WfeaStoreInterruptEnable
WFE-A store completed. bit 30

enumerator kPXP_WfebStoreInterruptEnable
WFE-B store completed. bit 31

enum __pxp_flags
PXP status flags.

Note: These enumerations are meant to be OR'd together to form a bit mask.

Values:

enumerator kPXP_CompleteFlag
PXP process completed. bit 0

enumerator kPXP_Axi0WriteErrorFlag
PXP encountered an AXI write error and processing has been terminated. bit 1

enumerator kPXP_Axi0ReadErrorFlag
PXP encountered an AXI read error and processing has been terminated. bit 2

enumerator kPXP_CommandLoadFlag
The command set by PXP_SetNextCommand has been loaded, could set new command.
bit 3

enumerator kPXP_CompressDoneFlag
Compress done. bit 15

enumerator kPXP_InputFetchCh0CompleteFlag
Input fetch channel 0 completed. bit 16

enumerator kPXP_InputFetchCh1CompleteFlag
Input fetch channel 1 completed. bit 17

enumerator kPXP_InputStoreCh0CompleteFlag
Input store channel 0 completed. bit 18

enumerator kPXP_InputStoreCh1CompleteFlag
Input store channel 1 completed. bit 19

enumerator kPXP_DitherFetchCh0CompleteFlag
Dither fetch channel 0 completed. bit 20

enumerator kPXP_DitherFetchCh1CompleteFlag
Dither fetch channel 1 completed. bit 21

enumerator kPXP_DitherStoreCh0CompleteFlag
Dither store channel 0 completed. bit 22

enumerator kPXP_DitherStoreCh1CompleteFlag
Dither store channel 1 completed. bit 23

enumerator kPXP_WfeaStoreCh0CompleteFlag
WFE-A store channel 0 completed. bit 24

enumerator kPXP_WfeaStoreCh1CompleteFlag
WFE-A store channel 1 completed. bit 25

enumerator kPXP_WfebStoreCh0CompleteFlag
WFE-B store channel 0 completed. bit 26

enumerator kPXP_WfebStoreCh1CompleteFlag

WFE-B store channel 1 completed. bit 27

enumerator kPXP_InputStoreCompleteFlag

Input store completed. bit 28

enumerator kPXP_DitherStoreCompleteFlag

Dither store completed. bit 29

enumerator kPXP_WfeaStoreCompleteFlag

WFE-A store completed. bit 30

enumerator kPXP_WfebStoreCompleteFlag

WFE-B store completed. bit 31

enum __pxp_flip_mode

PXP output flip mode.

Values:

enumerator kPXP_FlipDisable

Flip disable.

enumerator kPXP_FlipHorizontal

Horizontal flip.

enumerator kPXP_FlipVertical

Vertical flip.

enumerator kPXP_FlipBoth

Flip both directions.

enum __pxp_rotate_position

PXP rotate mode.

Values:

enumerator kPXP_RotateOutputBuffer

Rotate the output buffer.

enumerator kPXP_RotateProcessSurface

Rotate the process surface. Cannot be used together with flip, scale, or decimation function.

enum __pxp_rotate_degree

PXP rotate degree.

Values:

enumerator kPXP_Rotate0

Clock wise rotate 0 deg.

enumerator kPXP_Rotate90

Clock wise rotate 90 deg.

enumerator kPXP_Rotate180

Clock wise rotate 180 deg.

enumerator kPXP_Rotate270

Clock wise rotate 270 deg.

enum __pxp_interlaced_output_mode

PXP interlaced output mode.

Values:

enumerator kPXP_OutputProgressive

All data written in progressive format to output buffer 0.

enumerator kPXP_OutputField0

Only write field 0 data to output buffer 0.

enumerator kPXP_OutputField1

Only write field 1 data to output buffer 0.

enumerator kPXP_OutputInterlaced

Field 0 write to buffer 0, field 1 write to buffer 1.

enum __pxp_output_pixel_format

PXP output buffer format.

Values:

enumerator kPXP_OutputPixelFormatARGB8888

32-bit pixels with alpha.

enumerator kPXP_OutputPixelFormatRGB888

32-bit pixels without alpha (unpacked 24-bit format)

enumerator kPXP_OutputPixelFormatRGB888P

24-bit pixels without alpha (packed 24-bit format)

enumerator kPXP_OutputPixelFormatARGB1555

16-bit pixels with alpha.

enumerator kPXP_OutputPixelFormatARGB4444

16-bit pixels with alpha.

enumerator kPXP_OutputPixelFormatRGB555

16-bit pixels without alpha.

enumerator kPXP_OutputPixelFormatRGB444

16-bit pixels without alpha.

enumerator kPXP_OutputPixelFormatRGB565

16-bit pixels without alpha.

enumerator kPXP_OutputPixelFormatYUV1P444

32-bit pixels (1-plane XYUV unpacked).

enumerator kPXP_OutputPixelFormatUYVY1P422

16-bit pixels (1-plane U0,Y0,V0,Y1 interleaved bytes)

enumerator kPXP_OutputPixelFormatVYUY1P422

16-bit pixels (1-plane V0,Y0,U0,Y1 interleaved bytes)

enumerator kPXP_OutputPixelFormatY8

8-bit monochrome pixels (1-plane Y luma output)

enumerator kPXP_OutputPixelFormatY4

4-bit monochrome pixels (1-plane Y luma, 4 bit truncation)

enumerator kPXP_OutputPixelFormatYUV2P422

16-bit pixels (2-plane UV interleaved bytes)

enumerator kPXP_OutputPixelFormatYUV2P420

16-bit pixels (2-plane UV)

enumerator kPXP__OutputPixelFormatYVU2P422

16-bit pixels (2-plane VU interleaved bytes)

enumerator kPXP__OutputPixelFormatYVU2P420

16-bit pixels (2-plane VU)

enum __pxp_ps_pixel_format

PXP process surface buffer pixel format.

Values:

enumerator kPXP__PsPixelFormatRGB888

32-bit pixels without alpha (unpacked 24-bit format)

enumerator kPXP__PsPixelFormatRGB555

16-bit pixels without alpha.

enumerator kPXP__PsPixelFormatRGB444

16-bit pixels without alpha.

enumerator kPXP__PsPixelFormatRGB565

16-bit pixels without alpha.

enumerator kPXP__PsPixelFormatYUV1P444

32-bit pixels (1-plane XYUV unpacked).

enumerator kPXP__PsPixelFormatUYVY1P422

16-bit pixels (1-plane U0,Y0,V0,Y1 interleaved bytes)

enumerator kPXP__PsPixelFormatVYUY1P422

16-bit pixels (1-plane V0,Y0,U0,Y1 interleaved bytes)

enumerator kPXP__PsPixelFormatY8

8-bit monochrome pixels (1-plane Y luma output)

enumerator kPXP__PsPixelFormatY4

4-bit monochrome pixels (1-plane Y luma, 4 bit truncation)

enumerator kPXP__PsPixelFormatYUV2P422

16-bit pixels (2-plane UV interleaved bytes)

enumerator kPXP__PsPixelFormatYUV2P420

16-bit pixels (2-plane UV)

enumerator kPXP__PsPixelFormatYVU2P422

16-bit pixels (2-plane VU interleaved bytes)

enumerator kPXP__PsPixelFormatYVU2P420

16-bit pixels (2-plane VU)

enumerator kPXP__PsPixelFormatYVU422

16-bit pixels (3-plane)

enumerator kPXP__PsPixelFormatYVU420

16-bit pixels (3-plane)

enum __pxp_ps_yuv_format

PXP process surface buffer YUV format.

Values:

enumerator kPXP__PsYUVFormatYUV

YUV format.

enumerator kPXP_PsYUVFormatYCbCr
YCbCr format.

enum _pxp_as_pixel_format

PXP alpha surface buffer pixel format.

Values:

enumerator kPXP_AsPixelFormatARGB8888
32-bit pixels with alpha.

enumerator kPXP_AsPixelFormatRGB888
32-bit pixels without alpha (unpacked 24-bit format)

enumerator kPXP_AsPixelFormatARGB1555
16-bit pixels with alpha.

enumerator kPXP_AsPixelFormatARGB4444
16-bit pixels with alpha.

enumerator kPXP_AsPixelFormatRGB555
16-bit pixels without alpha.

enumerator kPXP_AsPixelFormatRGB444
16-bit pixels without alpha.

enumerator kPXP_AsPixelFormatRGB565
16-bit pixels without alpha.

enum _pxp_alpha_mode

PXP alpha mode during blending.

Values:

enumerator kPXP_AlphaEmbedded
The alpha surface pixel alpha value will be used for blend.

enumerator kPXP_AlphaOverride
The user defined alpha value will be used for blend directly.

enumerator kPXP_AlphaMultiply
The alpha surface pixel alpha value scaled the user defined alpha value will be used for blend, for example, pixel alpha set to 200, user defined alpha set to 100, then the result alpha is $200 * 100 / 255$.

enumerator kPXP_AlphaRop
Raster operation.

enum _pxp_rop_mode

PXP ROP mode during blending.

Explanation:

- AS: Alpha surface
- PS: Process surface
- nAS: Alpha surface NOT value
- nPS: Process surface NOT value

Values:

enumerator kPXP_RopMaskAs
AS AND PS.

enumerator kPXP_RopMaskNotAs
nAS AND PS.

enumerator kPXP_RopMaskAsNot
AS AND nPS.

enumerator kPXP_RopMergeAs
AS OR PS.

enumerator kPXP_RopMergeNotAs
nAS OR PS.

enumerator kPXP_RopMergeAsNot
AS OR nPS.

enumerator kPXP_RopNotCopyAs
nAS.

enumerator kPXP_RopNot
nPS.

enumerator kPXP_RopNotMaskAs
AS NAND PS.

enumerator kPXP_RopNotMergeAs
AS NOR PS.

enumerator kPXP_RopXorAs
AS XOR PS.

enumerator kPXP_RopNotXorAs
AS XNOR PS.

enum _pxp_block_size
PXP process block size.

Values:

enumerator kPXP_BlockSize8
Process 8x8 pixel blocks.

enumerator kPXP_BlockSize16
Process 16x16 pixel blocks.

enum _pxp_csc1_mode
PXP CSC1 mode.

Values:

enumerator kPXP_Csc1YUV2RGB
YUV to RGB.

enumerator kPXP_Csc1YCbCr2RGB
YCbCr to RGB.

enum _pxp_csc2_mode
PXP CSC2 mode.

Values:

enumerator kPXP_Csc2YUV2RGB
YUV to RGB.

enumerator kPXP_Csc2YCbCr2RGB
YCbCr to RGB.

enumerator kPXP_Csc2RGB2YUV
RGB to YUV.

enumerator kPXP_Csc2RGB2YCbCr
RGB to YCbCr.

enum __pxp_ram
PXP internal memory.

Values:

enumerator kPXP_RamDither0Lut
Dither 0 LUT memory.

enumerator kPXP_RamDither1Lut
Dither 1 LUT memory.

enumerator kPXP_RamDither2Lut
Dither 2 LUT memory.

enumerator kPXP_RamDither0Err0
Dither 0 ERR0 memory.

enumerator kPXP_RamDither0Err1
Dither 0 ERR1 memory.

enumerator kPXP_RamAluA
ALU A instr memory.

enumerator kPXP_RamAluB
ALU B instr memory.

enumerator kPXP_WfeAFetch
WFE-A fetch memory.

enumerator kPXP_WfeBFetch
WFE-B fetch memory.

enum __pxp_dither_mode
PXP dither mode.

Values:

enumerator kPXP_DitherPassThrough
Pass through, no dither.

enumerator kPXP_DitherFloydSteinberg
Floyd-Steinberg. For dither engine 0 only.

enumerator kPXP_DitherAtkinson
Atkinson. For dither engine 0 only.

enumerator kPXP_DitherOrdered
Ordered dither.

enumerator kPXP_DitherQuantOnly
No dithering, only quantization.

enumerator kPXP_DitherSierra
Sierra. For dither engine 0 only.

enum _pxp_dither_lut_mode

PXP dither LUT mode.

Values:

enumerator kPXP_DitherLutOff

The LUT memory is not used for LUT, could be used as ordered dither index matrix.

enumerator kPXP_DitherLutPreDither

Use LUT at the pre-dither stage, The pre-dither LUT could only be used in Floyd mode or Atkinson mode, which are not supported by current PXP module.

enumerator kPXP_DitherLutPostDither

Use LUT at the post-dither stage.

enum _pxp_dither_matrix_size

PXP dither matrix size.

Values:

enumerator kPXP_DitherMatrix4

The dither index matrix is 4x4.

enumerator kPXP_DitherMatrix8

The dither index matrix is 8x8.

enumerator kPXP_DitherMatrix16

The dither index matrix is 16x16.

Porter Duff factor mode. .

Values:

enumerator kPXP_PorterDuffFactorOne

Use 1.

enumerator kPXP_PorterDuffFactorZero

Use 0.

enumerator kPXP_PorterDuffFactorStraight

Use straight alpha.

enumerator kPXP_PorterDuffFactorInversed

Use inversed alpha.

Porter Duff global alpha mode. .

Values:

enumerator kPXP_PorterDuffGlobalAlpha

Use global alpha.

enumerator kPXP_PorterDuffLocalAlpha

Use local alpha in each pixel.

enumerator kPXP_PorterDuffScaledAlpha

Use global alpha * local alpha.

Porter Duff alpha mode. .

Values:

enumerator kPXP_PorterDuffAlphaStraight
Use straight alpha, $s0\_alpha' = s0\_alpha$.

enumerator kPXP_PorterDuffAlphaInversed
Use inversed alpha, $s0\_alpha' = 0xFF - s0\_alpha$.

Porter Duff color mode. .

Values:

enumerator kPXP_PorterDuffColorStraight

Deprecated:

Use kPXP_PorterDuffColorNoAlpha.

enumerator kPXP_PorterDuffColorInversed

Deprecated:

Use kPXP_PorterDuffColorWithAlpha.

enumerator kPXP_PorterDuffColorNoAlpha
 $s0\_pixel' = s0\_pixel$.

enumerator kPXP_PorterDuffColorWithAlpha
 $s0\_pixel' = s0\_pixel * s0\_alpha$.

enum _pxp_porter_duff_blend_mode

PXP Porter Duff blend mode. Note: don't change the enum item value.

Values:

enumerator kPXP_PorterDuffSrc
Source Only

enumerator kPXP_PorterDuffAtop
Source Atop

enumerator kPXP_PorterDuffOver
Source Over

enumerator kPXP_PorterDuffIn
Source In.

enumerator kPXP_PorterDuffOut
Source Out.

enumerator kPXP_PorterDuffDst
Destination Only.

enumerator kPXP_PorterDuffDstAtop
Destination Atop.

enumerator kPXP_PorterDuffDstOver
Destination Over.

enumerator kPXP_PorterDuffDstIn
Destination In.

enumerator kPXP_PorterDuffDstOut
Destination Out.

enumerator kPXP_PorterDuffXor
XOR.

enumerator kPXP__PorterDuffClear
Clear.

enumerator kPXP__PorterDuffMax

enum __pxp__process__engine__name
PXP process engine enumeration.

Values:

enumerator kPXP__PsAsOutEngine

enumerator kPXP__DitherEngine

enumerator kPXP__WfeaEngine

enumerator kPXP__WfebEngine

enumerator kPXP__InputFetchStoreEngine

enumerator kPXP__Alpha1Engine

enumerator kPXP__Csc2Engine

enumerator kPXP__LutEngine

enumerator kPXP__Rotate0Engine

enumerator kPXP__Rotate1Engine

enum __pxp__fetch__engine__name
PXP fetch engine enumeration.

There are actually 4 fetch engine implemented, the others are WFE-A fetch engine and WFE-B fetch engine, whose registers are reserved from developer.

Values:

enumerator kPXP__FetchInput

enumerator kPXP__FetchDither

enum __pxp__fetch__interface__mode
PXP fetch engine interface mode with the upstream store engine.

Values:

enumerator kPXP__FetchModeNormal

enumerator kPXP__FetchModeHandshake

enumerator kPXP__FetchModeBypass

enum __pxp__scanline__burst
PXP fetch/store engine burst length for scanline mode.

Values:

enumerator kPXP__Scanline8bytes

enumerator kPXP__Scanline16bytes

enumerator kPXP__Scanline32bytes

enumerator kPXP__Scanline64bytes

enum `_pxp_activeBits`

PXP fetch/store engine input/output active bits configuration.

Since fetch engine is 64-bit input and 32-bit output per channel, need to configure both channels to use 64-bit input mode. And expand configuration will have no effect.

Values:

enumerator `kPXP_Active8Bits`

enumerator `kPXP_Active16Bits`

enumerator `kPXP_Active32Bits`

enumerator `kPXP_Active64Bits`

enum `_pxp_fetch_output_word_order`

PXP fetch engine output word order when using 2 channels for 64-bit mode.

Values:

enumerator `kPXP_FetchOutputChannel1channel0`

In 64bit mode, channel 1 output high byte.

enumerator `kPXP_FetchOutputChannel0channel1`

In 64bit mode, channel 0 output high byte.

enum `_pxp_fetch_pixel_format`

PXP fetch engine input pixel format.

Values:

enumerator `kPXP_FetchFormatRGB565`

enumerator `kPXP_FetchFormatRGB555`

enumerator `kPXP_FetchFormatARGB1555`

enumerator `kPXP_FetchFormatRGB444`

enumerator `kPXP_FetchFormatARGB4444`

enumerator `kPXP_FetchFormatYUYVOrYVYU`

enumerator `kPXP_FetchFormatUYVYOrVYUY`

enumerator `kPXP_FetchFormatYUV422_2P`

enum `_pxp_store_engine_name`

PXP store engine enumeration.

There are actually 4 store engine implemented, the others are WFE-A store engine and WFE-B store engine, whose registers are reserved from developer.

Values:

enumerator `kPXP_StoreInput`

enumerator `kPXP_StoreDither`

enum `_pxp_store_interface_mode`

PXP store engine interface mode with the downstream fetch engine.

Values:

enumerator kPXP_StoreModeBypass

Store engine output the input data, after the shift function directly to the downstream Fetch Engine.

enumerator kPXP_StoreModeNormal

Store engine stores the input data to memory.

enumerator kPXP_StoreModeHandshake

Downstream fetch engine fetch data per scanline from memory using buffer sharing with store engine.

enumerator kPXP_StoreModeDual

Store engine outputs data directly to downstream fetch engine(Bypass) but also storing it to memory at the same time.

enum __pxp_store_yuv_mode

PXP store engine YUV output mode.

Values:

enumerator kPXP_StoreYUVDisable

Do not output YUV pixel format.

enumerator kPXP_StoreYUVPlane1

Use channel to output YUV422_1p pixel format, need to use shift operation to make sure each pixel component in its proper position: 64-bits of pixel data format and each 32 bits as {Y0, U0, Y1, V0}.

enumerator kPXP_StoreYUVPlane2

Use channel to output YUV422_2p pixel format, need to use shift operation to make sure each pixel component in its proper position: channel 0 {Y0,Y1}, channel 1 {U0,V0}.

enum __pxp_cfa_input_format

PXP pre-dither CFA engine input pixel format.

Values:

enumerator kPXP_CfaRGB888

enumerator kPXP_CfaRGB444

enum __pxp_histogram_mask_condition

PXP histogram mask condition.

Values:

enumerator kPXP_HistogramMaskEqual

Value that equal to value0 will pass the mask operation.

enumerator kPXP_HistogramMaskNotEqual

Value that not equal to value0 will pass the mask operation.

enumerator kPXP_HistogramMaskIn

Value that within the range of value0-value1 will pass the mask operation.

enumerator kPXP_HistogramMaskOut

Value that without the range of value0-value1 will pass the mask operation.

enum __pxp_histogram_flags

PXP Histogram operation result flags.

Values:

enumerator kPXP_Histogram2levelMatch

enumerator kPXP_Histogram4levelMatch

enumerator kPXP_Histogram8levelMatch

enumerator kPXP_Histogram16levelMatch

enumerator kPXP_Histogram32levelMatch

typedef enum *_pxp_flip_mode* pxp_flip_mode_t
PXP output flip mode.

typedef enum *_pxp_rotate_position* pxp_rotate_position_t
PXP rotate mode.

typedef enum *_pxp_rotate_degree* pxp_rotate_degree_t
PXP rotate degree.

typedef enum *_pxp_interlaced_output_mode* pxp_interlaced_output_mode_t
PXP interlaced output mode.

typedef enum *_pxp_output_pixel_format* pxp_output_pixel_format_t
PXP output buffer format.

typedef struct *_pxp_output_buffer_config* pxp_output_buffer_config_t
PXP output buffer configuration.

typedef enum *_pxp_ps_pixel_format* pxp_ps_pixel_format_t
PXP process surface buffer pixel format.

typedef enum *_pxp_ps_yuv_format* pxp_ps_yuv_format_t
PXP process surface buffer YUV format.

typedef struct *_pxp_ps_buffer_config* pxp_ps_buffer_config_t
PXP process surface buffer configuration.

typedef enum *_pxp_as_pixel_format* pxp_as_pixel_format_t
PXP alpha surface buffer pixel format.

typedef struct *_pxp_as_buffer_config* pxp_as_buffer_config_t
PXP alphas surface buffer configuration.

typedef enum *_pxp_alpha_mode* pxp_alpha_mode_t
PXP alpha mode during blending.

typedef enum *_pxp_rop_mode* pxp_rop_mode_t
PXP ROP mode during blending.

Explanation:

- AS: Alpha surface
- PS: Process surface
- nAS: Alpha surface NOT value
- nPS: Process surface NOT value

typedef struct *_pxp_as_blend_config* pxp_as_blend_config_t
PXP alpha surface blending configuration.

typedef struct *_pxp_as_blend_secondary_config* pxp_as_blend_secondary_config_t
PXP secondary alpha surface blending engine configuration.

typedef enum *_pxp_block_size* pxp_block_size_t
PXP process block size.

typedef enum *_pxp_csc1_mode* pxp_csc1_mode_t
 PXP CSC1 mode.

typedef enum *_pxp_csc2_mode* pxp_csc2_mode_t
 PXP CSC2 mode.

typedef struct *_pxp_csc2_config* pxp_csc2_config_t
 PXP CSC2 configuration.

Converting from YUV/YCbCr color spaces to the RGB color space uses the following equation structure:

$$R = A1(Y+D1) + A2(U+D2) + A3(V+D3) \quad G = B1(Y+D1) + B2(U+D2) + B3(V+D3) \quad B = C1(Y+D1) + C2(U+D2) + C3(V+D3)$$

Converting from the RGB color space to YUV/YCbCr color spaces uses the following equation structure:

$$Y = A1*R + A2*G + A3*B + D1 \quad U = B1*R + B2*G + B3*B + D2 \quad V = C1*R + C2*G + C3*B + D3$$

typedef enum *_pxp_ram* pxp_ram_t
 PXP internal memory.

typedef struct *_pxp_dither_final_lut_data* pxp_dither_final_lut_data_t
 PXP dither final LUT data.

typedef struct *_pxp_dither_config* pxp_dither_config_t
 PXP dither configuration.

typedef enum *_pxp_porter_duff_blend_mode* pxp_porter_duff_blend_mode_t
 PXP Porter Duff blend mode. Note: don't change the enum item value.

typedef struct *_pxp_pic_copy_config* pxp_pic_copy_config_t
 PXP Porter Duff blend mode. Note: don't change the enum item value.

typedef enum *_pxp_process_engine_name* pxp_process_engine_name_t
 PXP process engine enumeration.

typedef enum *_pxp_fetch_engine_name* pxp_fetch_engine_name_t
 PXP fetch engine enumeration.

There are actually 4 fetch engine implemented, the others are WFE-A fetch engine and WFE-B fetch engine, whose registers are reserved from developer.

typedef enum *_pxp_fetch_interface_mode* pxp_fetch_interface_mode_t
 PXP fetch engine interface mode with the upstream store engine.

typedef enum *_pxp_scanline_burst* pxp_scanline_burst_t
 PXP fetch/store engine burst length for scanline mode.

typedef struct *_pxp_block_format_config* pxp_block_config_t
 PXP fetch engine block configuration.

typedef enum *_pxp_activeBits* pxp_active_bits_t
 PXP fetch/store engine input/output active bits configuration.

Since fetch engine is 64-bit input and 32-bit output per channel, need to configure both channels to use 64-bit input mode. And expand configuration will have no effect.

typedef enum *_pxp_fetch_output_word_order* pxp_fetch_output_word_order_t
 PXP fetch engine output word order when using 2 channels for 64-bit mode.

`typedef struct _pxp_fetch_shift_component pxp_fetch_shift_component_t`
PXP fetch engine shift component configuration.

Fetch engine can divided each word into 4 components and shift them.

`typedef struct _pxp_fetch_shift_config pxp_fetch_shift_config_t`
PXP fetch engine shift configuration.

Fetch engine can divided each word into 4 components and shift them. For example, to change YUV444 to YVU444, U and V positions need to be shifted: OFFSET0=8, OFFSET1=0, OFFSET2=16, OFFSET3=24, WIDTH0/1/2/3=8

`typedef enum _pxp_fetch_pixel_format pxp_fetch_pixel_format_t`
PXP fetch engine input pixel format.

`typedef struct _pxp_fetch_engine_config pxp_fetch_engine_config_t`
PXP fetch engine configuration for one of the channel.

`typedef enum _pxp_store_engine_name pxp_store_engine_name_t`
PXP store engine enumeration.

There are actually 4 store engine implemented, the others are WFE-A store engine and WFE-B store engine, whose registers are reserved from developer.

`typedef enum _pxp_store_interface_mode pxp_store_interface_mode_t`
PXP store engine interface mode with the downstream fetch engine.

`typedef enum _pxp_store_yuv_mode pxp_store_yuv_mode_t`
PXP store engine YUV output mode.

`typedef struct _pxp_store_shift_config pxp_store_shift_config_t`
Shift configuration for PXP store engine.

`typedef struct _pxp_store_engine_config pxp_store_engine_config_t`
PXP store engine configuration for one of the channel.

`typedef enum _pxp_cfa_input_format pxp_cfa_input_format_t`
PXP pre-dither CFA engine input pixel format.

`typedef struct _pxp_cfa_config pxp_cfa_config_t`
PXP pre-dither CFA engine configuration.

`typedef enum _pxp_histogram_mask_condition pxp_histogram_mask_condition_t`
PXP histogram mask condition.

`typedef struct _pxp_histogram_config pxp_histogram_config_t`
PXP Histogram configuration.

`typedef struct _pxp_histogram_mask_result pxp_histogram_mask_result_t`
PXP Histogram mask result.

`typedef struct _pxp_wfea_engine_config pxp_wfea_engine_config_t`
PXP WFE-A engine configuration.

PXP_LUT_TABLE_BYTE

PXP_INTERNAL_RAM_LUT_BYTE

PXP_SHARE_ROTATE

PXP_USE_PATH

PXP_COMBINE_BYTE_TO_WORD(dataAddr)

`struct _pxp_output_buffer_config`
`#include <fsl_pxp.h>` PXP output buffer configuration.

Public Members

pxp_output_pixel_format_t pixelFormat

Output buffer pixel format.

pxp_interlaced_output_mode_t interlacedMode

Interlaced output mode.

uint32_t buffer0Addr

Output buffer 0 address.

uint32_t buffer1Addr

Output buffer 1 address, used for UV data in YUV 2-plane mode, or field 1 in output interlaced mode.

uint16_t pitchBytes

Number of bytes between two vertically adjacent pixels.

uint16_t width

Pixels per line.

uint16_t height

How many lines in output buffer.

struct __pxp_ps_buffer_config

#include <fsl_pxp.h> PXP process surface buffer configuration.

Public Members

pxp_ps_pixel_format_t pixelFormat

PS buffer pixel format.

bool swapByte

For each 16 bit word, set true to swap the two bytes.

uint32_t bufferAddr

Input buffer address for the first panel.

uint32_t bufferAddrU

Input buffer address for the second panel.

uint32_t bufferAddrV

Input buffer address for the third panel.

uint16_t pitchBytes

Number of bytes between two vertically adjacent pixels.

struct __pxp_as_buffer_config

#include <fsl_pxp.h> PXP alphas surface buffer configuration.

Public Members

pxp_as_pixel_format_t pixelFormat

AS buffer pixel format.

uint32_t bufferAddr

Input buffer address.

uint16_t pitchBytes

Number of bytes between two vertically adjacent pixels.

struct __pxp_as_blend_config
#include <fsl_pxp.h> PXP alpha surface blending configuration.

Public Members

uint8_t alpha
User defined alpha value, only used when alphaMode is kPXP_AlphaOverride or kPXP_AlphaRop.

bool invertAlpha
Set true to invert the alpha.

pxp_alpha_mode_t alphaMode
Alpha mode.

pxp_rop_mode_t ropMode
ROP mode, only valid when alphaMode is kPXP_AlphaRop.

struct __pxp_as_blend_secondary_config
#include <fsl_pxp.h> PXP secondary alpha surface blending engine configuration.

Public Members

bool invertAlpha
Set true to invert the alpha.

bool ropEnable
Enable rop mode.

pxp_rop_mode_t ropMode
ROP mode, only valid when ropEnable is true.

struct __pxp_csc2_config
#include <fsl_pxp.h> PXP CSC2 configuration.

Converting from YUV/YCbCr color spaces to the RGB color space uses the following equation structure:

$$R = A1(Y+D1) + A2(U+D2) + A3(V+D3) \quad G = B1(Y+D1) + B2(U+D2) + B3(V+D3) \quad B = C1(Y+D1) + C2(U+D2) + C3(V+D3)$$

Converting from the RGB color space to YUV/YCbCr color spaces uses the following equation structure:

$$Y = A1*R + A2*G + A3*B + D1 \quad U = B1*R + B2*G + B3*B + D2 \quad V = C1*R + C2*G + C3*B + D3$$

Public Members

pxp_csc2_mode_t mode
Conversion mode.

float A1
A1.

float A2
A2.

float A3
A3.

float B1
B1.

float B2
B2.

float B3
B3.

float C1
C1.

float C2
C2.

float C3
C3.

int16_t D1
D1.

int16_t D2
D2.

int16_t D3
D3.

struct _pxp_dither_final_lut_data
#include <fsl_pxp.h> PXP dither final LUT data.

Public Members

uint32_t data_3_0
Data 3 to data 0. Data 0 is the least significant byte.

uint32_t data_7_4
Data 7 to data 4. Data 4 is the least significant byte.

uint32_t data_11_8
Data 11 to data 8. Data 8 is the least significant byte.

uint32_t data_15_12
Data 15 to data 12. Data 12 is the least significant byte.

struct _pxp_dither_config
#include <fsl_pxp.h> PXP dither configuration.

Public Members

uint32_t enableDither0
Enable dither engine 0 or not, set 1 to enable, 0 to disable.

uint32_t enableDither1
Enable dither engine 1 or not, set 1 to enable, 0 to disable.

uint32_t enableDither2
Enable dither engine 2 or not, set 1 to enable, 0 to disable.

uint32_t ditherMode0
Dither mode for dither engine 0. See _pxp_dither_mode.

uint32_t ditherMode1

Dither mode for dither engine 1. See `_pxp_dither_mode`.

uint32_t ditherMode2

Dither mode for dither engine 2. See `_pxp_dither_mode`.

uint32_t quantBitNum

Number of bits quantize down to, the valid value is 1~7.

uint32_t lutMode

How to use the memory LUT, see `_pxp_dither_lut_mode`. This must be set to `kPXP_DitherLutOff` if any dither engine uses `kPXP_DitherOrdered` mode.

uint32_t idxMatrixSize0

Size of index matrix used for dither for dither engine 0, see `_pxp_dither_matrix_size`.

uint32_t idxMatrixSize1

Size of index matrix used for dither for dither engine 1, see `_pxp_dither_matrix_size`.

uint32_t idxMatrixSize2

Size of index matrix used for dither for dither engine 2, see `_pxp_dither_matrix_size`.

uint32_t enableFinalLut

Enable the final LUT, set 1 to enable, 0 to disable.

struct pxp_porter_duff_config_t

#include <fsl_pxp.h> PXP Porter Duff configuration.

Public Members

uint32_t enable

Enable or disable Porter Duff.

uint32_t srcFactorMode

Source layer (or AS, s1) factor mode, see `pxp_porter_duff_factor_mode`.

uint32_t dstGlobalAlphaMode

Destination layer (or PS, s0) global alpha mode, see `pxp_porter_duff_global_alpha_mode`.

uint32_t dstAlphaMode

Destination layer (or PS, s0) alpha mode, see `pxp_porter_duff_alpha_mode`.

uint32_t dstColorMode

Destination layer (or PS, s0) color mode, see `pxp_porter_duff_color_mode`.

uint32_t dstFactorMode

Destination layer (or PS, s0) factor mode, see `pxp_porter_duff_factor_mode`.

uint32_t srcGlobalAlphaMode

Source layer (or AS, s1) global alpha mode, see `pxp_porter_duff_global_alpha_mode`.

uint32_t srcAlphaMode

Source layer (or AS, s1) alpha mode, see `pxp_porter_duff_alpha_mode`.

uint32_t srcColorMode

Source layer (or AS, s1) color mode, see `pxp_porter_duff_color_mode`.

uint32_t dstGlobalAlpha

Destination layer (or PS, s0) global alpha value, 0~255.

uint32_t srcGlobalAlpha

Source layer (or AS, s1) global alpha value, 0~255.

struct __pxp_pic_copy_config

#include <fsl_pxp.h> PXP Porter Duff blend mode. Note: don't change the enum item value.

Public Members

uint32_t srcPicBaseAddr

Source picture base address.

uint16_t srcPitchBytes

Pitch of the source buffer.

uint16_t srcOffsetX

Copy position in source picture.

uint16_t srcOffsetY

Copy position in source picture.

uint32_t destPicBaseAddr

Destination picture base address.

uint16_t destPitchBytes

Pitch of the destination buffer.

uint16_t destOffsetX

Copy position in destination picture.

uint16_t destOffsetY

Copy position in destination picture.

uint16_t width

Pixel number each line to copy.

uint16_t height

Lines to copy.

pxp_as_pixel_format_t pixelFormat

Buffer pixel format.

struct __pxp_block_format_config

#include <fsl_pxp.h> PXP fetch engine block configuration.

Public Members

bool enableblock

Enable to use block mode instead of scanline mode. Note: 1.Make sure to enable if rotate or flip mode is enabled. 2.Block mode cannot work on 64bpp data stream where activeBits = 64. 3. If LUT processing is in the path between the fetch and store engind, block mode must be enabled.

bool blockSize16

Enable to use 16*16 block, otherwise it will be 8*8 block.

pxp_scanline_burst_t burstLength

When using scanline mode, configure this for burst length.

struct __pxp_fetch_shift_component

#include <fsl_pxp.h> PXP fetch engine shift component configuration.

Fetch engine can divided each word into 4 components and shift them.

struct __pxp_fetch_shift_config

#include <fsl_pxp.h> PXP fetch engine shift configuration.

Fetch engine can divided each word into 4 components and shift them. For example, to change YUV444 to YVU444, U and V positions need to be shifted: OFFSET0=8, OFFSET1=0, OFFSET2=16, OFFSET3=24, WIDTH0/1/2/3=8

struct __pxp_fetch_engine_config

#include <fsl_pxp.h> PXP fetch engine configuration for one of the channel.

Public Members

bool channelEnable

Enable channel.

uint32_t inputBaseAddr0

The input base address. Used for Y plane input when pixel format is YUV422_2p.

uint32_t inputBaseAddr1

Must configure this for UV plane when input pixel format is YUV422_2p.

uint16_t totalHeight

Total height for the actual fetch size.

uint16_t totalWidth

Total width for the actual fetch size.

uint16_t pitchBytes

Channel input pitch

uint16_t ulcX

X coordinate of upper left coordinate in pixels of the active area of the total input memory

uint16_t ulcY

Y coordinate of upper left coordinate in pixels of the active area of the total input memory

uint16_t lrcX

X coordinate of Lower right coordinate in pixels of the active area of the total input memory

uint16_t lrcY

Y coordinate of Lower right coordinate in pixels of the active area of the total input memory

uint32_t backGroundColor

Pixel value of the background color for the space outside the active area.

pxp_fetch_interface_mode_t interface

Interface mode, normal/bypass/handshake

pxp_active_bits_t activeBits

Input active bits.

pxp_fetch_pixel_format_t pixelFormat

Input pixel fetch format

bool expandEnable

If enabled, input pixel will be expanded to ARGB8888, RGB888 or YUV444 of 32-bit format at the output.

pxp_flip_mode_t flipMode

Flip the fetched input.

pxp_rotate_degree_t rotateDegree

Rotate the fetched input.

pxp_block_config_t fetchFormat

Block mode configuration. Make sure to enable block if rotate or flip mode is enabled.

pxp_fetch_shift_config_t shiftConfig

Shift operation configuration.

pxp_fetch_output_word_order_t wordOrder

Output word order when using 2 channels for 64-bit mode.

struct __pxp_store_shift_config

#include <fsl_pxp.h> Shift configuration for PXP store engine.

Public Members

bool shiftBypass

Bypass the data shift

uint64_t *pDataShiftMask

Pointer to mask0~mask7 to mask the 64-bit of output data, data is masked first then shifted according to width.

uint8_t *pDataShiftWidth

Pointer to width0~width7. Bit 7 is for shifted direction, 0 to right. Bit0~5 is for shift width.

uint8_t *pFlagShiftMask

Pointer to mask0~mask7 to mask the 8-bit of output flag, flag is masked first then shifted according to width.

uint8_t *pFlagShiftWidth

Pointer to width0~width7. Bit 6 is for shifted direction, 0 to right. Bit0~5 is for shift width.

struct __pxp_store_engine_config

#include <fsl_pxp.h> PXP store engine configuration for one of the channel.

Public Members

bool channelEnable

Enable channel.

uint32_t outputBaseAddr0

The channel 0 output address if using 2 channels. If using 1 channel(must be channel 0) and YUV422_2p output format, is for Y plane address.

uint32_t outputBaseAddr1

The channel 1 output address if using 2 channels. If using 1 channel(must be channel 0) and YUV422_2p output format, is for UV plane address.

`uint16_t totalHeight`
Total height for the actual store size.

`uint16_t totalWidth`
Total width for the actual store size.

`uint16_t pitchBytes`
Channel input pitch

`pxp_store_interface_mode_t interface`
Interface mode, normal/bypass/handshake/dual. Make sure 2 channels use the same mode if both enabled.

`pxp_active_bits_t activeBits`
Output active bits.

`pxp_store_yuv_mode_t yuvMode`
Whether to output YUV pixel format.

`bool useFixedData`
Whether to use fixed value for the output data. Can be used to write fixed value to specific memory location for memory initialization.

`uint32_t fixedData`
The value of the fixed data.

`bool packInSelect`
When enabled, channel 0 will select low 32 bit shift out data to pack while channel i select high 32 bit, otherwise all 64-bit of data will be selected.

`pxp_block_config_t storeFormat`
The format to store data, block or otherwise.

`pxp_store_shift_config_t shiftConfig`
Shift operation configuration.

`struct _pxp_cfa_config`
`#include <fsl_pxp.h>` PXP pre-dither CFA engine configuration.

Public Members

`bool bypass`
Bypass the CFA process

`pxp_cfa_input_format_t pixelInFormat`
The pixel input format for CFA.

`uint8_t arrayWidth`
CFA array vertical size in pixels, min 3 max 15.

`uint8_t arrayHeight`
CFA array horizontal size in pixels, min 3 max 15.

`uint16_t totalHeight`
Total height for the buffer size, make sure it is aligned with the dither fetch engine and dither engine.

`uint16_t totalWidth`
Total width for the buffer size, make sure it is aligned with the dither fetch engine and dither engine.

uint32_t *cfaValue

Pointer to the value for the CFA array. 2-bit per component: 00-R,01-G,10-B,11-W. For a 4x4 array, 32 bits are need.

struct __pxp_histogram_config

#include <fsl_pxp.h> PXP Histogram configuration.

Public Members

bool enable

Enable histogram process.

uint8_t *pParamValue

Pointer to the 62(2+4+8+16+32) byte of param value for 2-level, 4-level.....32-level parameters. Only low 5-bit of each byte is valid.

uint8_t lutValueOffset

The starting bit position of the LUT value.

uint8_t lutValueWidth

The bit width of the LUT value, should be no more than 6 bits since only 63 LUTs are supported.

bool enableMask

Enable mask operation.

uint8_t maskValue0

Value 0 for the condition judgement.

uint8_t maskValue1

Value 1 for the condition judgement.

uint8_t maskOffset

The starting bit position of the field to be checked against mask condition.

uint8_t maskWidth

The width of the field to be checked against mask condition.

pxp_histogram_mask_condition_t condition

The mask condition.

uint16_t totalHeight

Total height for the buffer size, make sure it is aligned with the output of legacy flow or the WFE-A/B engine.

uint16_t totalWidth

Total width for the buffer size, make sure it is aligned with the output of legacy flow or the WFE-A/B engine.

struct __pxp_histogram_mask_result

#include <fsl_pxp.h> PXP Histogram mask result.

Public Members

uint32_t pixelCount

The total count of the pixels that pass the mask(collided pixels).

uint32_t minX

The x offset of the ULC of the minimal histogram that covers all passed pixels.

uint32_t minY

The y offset of the ULC of the minimal histogram that covers all passed pixels.

uint32_t maxX

The x offset of the LRC of the minimal histogram that covers all passed pixels.

uint32_t maxY

The y offset of the LRC of the minimal histogram that covers all passed pixels.

uint64_t lutlist

The 64-bit LUT list of collided pixels, if pixel of LUT17 is collided, bit17 in the list is set.

struct __pxp_wfea_engine_config

#include <fsl_pxp.h> PXP WFE-A engine configuration.

Public Members

uint32_t y4Addr

Address for Y4 buffer.

uint32_t y4cAddr

Address for Y4C buffer, {Y4[3:0],3'b000,collision}, 8bpp.

uint32_t wbAddr

Address for EPDC working buffer.

uint16_t updateWidth

Width of the update area.

uint16_t updateHeight

Height of the update area.

uint16_t updatePitch

Pitch of the update area.

uint16_t ulcX

X coordinate of upper left coordinate of the total input memory

uint16_t ulcY

Y coordinate of upper left coordinate of the total input memory

uint16_t resX

Horizontal resolution in pixels.

uint8_t lutNum

The EPDC LUT number for the update.

bool fullUpdateEnable

Enable full update.

bool alphaEnable

Enable alpha field, upd is {Y4[3:0],3'b000,alpha} format, otherwise its {Y4[3:0],4'b0000}.

bool detectionOnly

Detection only, do not write working buffer.

2.91 QTMR: Quad Timer Driver

```
void QTMR_Init(TMR_Type *base, qtmr_channel_selection_t channel, const qtmr_config_t *config)
```

Ungates the Quad Timer clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the Quad Timer driver.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- config – Pointer to user's Quad Timer config structure

```
void QTMR_Deinit(TMR_Type *base, qtmr_channel_selection_t channel)
```

Stops the counter and gates the Quad Timer clock.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

```
void QTMR_GetDefaultConfig(qtmr_config_t *config)
```

Fill in the Quad Timer config struct with the default settings.

The default values are:

```
config->debugMode = kQTMR_RunNormalInDebug;
config->enableExternalForce = false;
config->enableMasterMode = false;
config->faultFilterCount = 0;
config->faultFilterPeriod = 0;
config->primarySource = kQTMR_ClockDivide_2;
config->secondarySource = kQTMR_Counter0InputPin;
```

Parameters

- config – Pointer to user's Quad Timer config structure.

```
void QTMR_EnableInterrupts(TMR_Type *base, qtmr_channel_selection_t channel, uint32_t mask)
```

Enables the selected Quad Timer interrupts.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- mask – The interrupts to enable. This is a logical OR of members of the enumeration `qtmr_interrupt_enable_t`

```
void QTMR_DisableInterrupts(TMR_Type *base, qtmr_channel_selection_t channel, uint32_t mask)
```

Disables the selected Quad Timer interrupts.

Parameters

- base – Quad Timer peripheral base address

- channel – Quad Timer channel number
- mask – The interrupts to enable. This is a logical OR of members of the enumeration `qtmr_interrupt_enable_t`

`uint32_t QTMR_GetEnabledInterrupts(TMR_Type *base, qtmr_channel_selection_t channel)`

Gets the enabled Quad Timer interrupts.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

Returns

The enabled interrupts. This is the logical OR of members of the enumeration `qtmr_interrupt_enable_t`

`uint32_t QTMR_GetStatus(TMR_Type *base, qtmr_channel_selection_t channel)`

Gets the Quad Timer status flags.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

Returns

The status flags. This is the logical OR of members of the enumeration `qtmr_status_flags_t`

`void QTMR_ClearStatusFlags(TMR_Type *base, qtmr_channel_selection_t channel, uint32_t mask)`

Clears the Quad Timer status flags.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- mask – The status flags to clear. This is a logical OR of members of the enumeration `qtmr_status_flags_t`

`void QTMR_SetTimerPeriod(TMR_Type *base, qtmr_channel_selection_t channel, uint16_t ticks)`

Sets the timer period in ticks.

Timers counts from initial value till it equals the count value set here. The counter will then reinitialize to the value specified in the Load register.

Note:

- This function will write the time period in ticks to COMP1 or COMP2 register depending on the count direction
 - User can call the utility macros provided in `fsl_common.h` to convert to ticks
 - This function supports cases, providing only primary source clock without secondary source clock.
-

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- ticks – Timer period in units of ticks

```
void QTMR_SetCompareValue(TMR_Type *base, qtmr_channel_selection_t channel, uint16_t ticks)
```

Set compare value.

This function sets the value used for comparison with the counter value.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- ticks – Timer period in units of ticks.

```
static inline void QTMR_SetLoadValue(TMR_Type *base, qtmr_channel_selection_t channel, uint16_t value)
```

Set load value.

This function sets the value used to initialize the counter after a counter comparison.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- value – Load register initialization value.

```
static inline uint16_t QTMR_GetCurrentTimerCount(TMR_Type *base, qtmr_channel_selection_t channel)
```

Reads the current timer counting value.

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note: User can call the utility macros provided in `fsl_common.h` to convert ticks to usec or msec

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

Returns

Current counter value in ticks

```
static inline void QTMR_StartTimer(TMR_Type *base, qtmr_channel_selection_t channel, qtmr_counting_mode_t clockSource)
```

Starts the Quad Timer counter.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- clockSource – Quad Timer clock source

```
static inline void QTMR_StopTimer(TMR_Type *base, qtmr_channel_selection_t channel)
```

Stops the Quad Timer counter.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

```
void QTMR_EnableDma(TMR_Type *base, qtmr_channel_selection_t channel, uint32_t mask)
```

Enable the Quad Timer DMA.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- mask – The DMA to enable. This is a logical OR of members of the enumeration *qtmr_dma_enable_t*

```
void QTMR_DisableDma(TMR_Type *base, qtmr_channel_selection_t channel, uint32_t mask)
```

Disable the Quad Timer DMA.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- mask – The DMA to enable. This is a logical OR of members of the enumeration *qtmr_dma_enable_t*

```
void QTMR_SetPwmOutputToIdle(TMR_Type *base, qtmr_channel_selection_t channel, bool  
                             idleStatus)
```

Set PWM output in idle status (high or low).

Note: When the PWM is set again, the counting needs to be restarted.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- idleStatus – True: PWM output is high in idle status; false: PWM output is low in idle status.

```
static inline qtmr_pwm_out_state_t QTMR_GetPwmOutputStatus(TMR_Type *base,  
                                                           qtmr_channel_selection_t  
                                                           channel)
```

Get the channel output status.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

Returns

Current channel output status.

```
uint8_t QTMR_GetPwmChannelStatus(TMR_Type *base, qtmr_channel_selection_t channel)
```

Get the PWM channel duty cycle value.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

Returns

Current channel duty cycle value.

```
void QTMR_SetPwmClockMode(TMR_Type *base, qtmr_channel_selection_t channel,
                          qtmr_primary_count_source_t prescaler)
```

This function set the value of the prescaler on QTimer channels.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- prescaler – Set prescaler value

```
FSL_QTMR_DRIVER_VERSION
```

Version

```
enum _qtmr_primary_count_source
```

Quad Timer primary clock source selection.

Values:

```
enumerator kQTMR_ClockCounter0InputPin
```

Use counter 0 input pin

```
enumerator kQTMR_ClockCounter1InputPin
```

Use counter 1 input pin

```
enumerator kQTMR_ClockCounter2InputPin
```

Use counter 2 input pin

```
enumerator kQTMR_ClockCounter3InputPin
```

Use counter 3 input pin

```
enumerator kQTMR_ClockCounter0Output
```

Use counter 0 output

```
enumerator kQTMR_ClockCounter1Output
```

Use counter 1 output

```
enumerator kQTMR_ClockCounter2Output
```

Use counter 2 output

```
enumerator kQTMR_ClockCounter3Output
```

Use counter 3 output

```
enumerator kQTMR_ClockDivide_1
```

IP bus clock divide by 1 prescaler

```
enumerator kQTMR_ClockDivide_2
```

IP bus clock divide by 2 prescaler

```
enumerator kQTMR_ClockDivide_4
```

IP bus clock divide by 4 prescaler

```
enumerator kQTMR_ClockDivide_8
```

IP bus clock divide by 8 prescaler

```
enumerator kQTMR_ClockDivide_16
```

IP bus clock divide by 16 prescaler

```
enumerator kQTMR_ClockDivide_32
```

IP bus clock divide by 32 prescaler

```
enumerator kQTMR_ClockDivide_64
```

IP bus clock divide by 64 prescaler

enumerator kQTMR_ClockDivide_128
IP bus clock divide by 128 prescaler

enum __qtmr_input_source
Quad Timer input sources selection.

Values:

enumerator kQTMR_Counter0InputPin
Use counter 0 input pin

enumerator kQTMR_Counter1InputPin
Use counter 1 input pin

enumerator kQTMR_Counter2InputPin
Use counter 2 input pin

enumerator kQTMR_Counter3InputPin
Use counter 3 input pin

enum __qtmr_counting_mode
Quad Timer counting mode selection.

Values:

enumerator kQTMR_NoOperation
No operation

enumerator kQTMR_PriSrcRiseEdge
Count rising edges of primary source

enumerator kQTMR_PriSrcRiseAndFallEdge
Count rising and falling edges of primary source

enumerator kQTMR_PriSrcRiseEdgeSecInpHigh
Count rise edges of pri SRC while sec inp high active

enumerator kQTMR_QuadCountMode
Quadrature count mode, uses pri and sec sources

enumerator kQTMR_PriSrcRiseEdgeSecDir
Count rising edges of pri SRC; sec SRC specifies dir

enumerator kQTMR_SecSrcTrigPriCnt
Edge of sec SRC trigger primary count until compare

enumerator kQTMR_CascadeCount
Cascaded count mode (up/down)

enum __qtmr_pwm_out_state
Quad Timer PWM output state.

Values:

enumerator kQTMR_PwmLow
The output state of PWM channel is low

enumerator kQTMR_PwmHigh
The output state of PWM channel is low

enum __qtmr_output_mode
Quad Timer output mode selection.

Values:

enumerator kQTMR__AssertWhenCountActive
Assert OFLAG while counter is active

enumerator kQTMR__ClearOnCompare
Clear OFLAG on successful compare

enumerator kQTMR__SetOnCompare
Set OFLAG on successful compare

enumerator kQTMR__ToggleOnCompare
Toggle OFLAG on successful compare

enumerator kQTMR__ToggleOnAltCompareReg
Toggle OFLAG using alternating compare registers

enumerator kQTMR__SetOnCompareClearOnSecSrcInp
Set OFLAG on compare, clear on sec SRC input edge

enumerator kQTMR__SetOnCompareClearOnCountRoll
Set OFLAG on compare, clear on counter rollover

enumerator kQTMR__EnableGateClock
Enable gated clock output while count is active

enum __qtmr_input_capture_edge
Quad Timer input capture edge mode, rising edge, or falling edge.
Values:

enumerator kQTMR__NoCapture
Capture is disabled

enumerator kQTMR__RisingEdge
Capture on rising edge (IPS=0) or falling edge (IPS=1)

enumerator kQTMR__FallingEdge
Capture on falling edge (IPS=0) or rising edge (IPS=1)

enumerator kQTMR__RisingAndFallingEdge
Capture on both edges

enum __qtmr_preload_control
Quad Timer input capture edge mode, rising edge, or falling edge.
Values:

enumerator kQTMR__NoPreload
Never preload

enumerator kQTMR__LoadOnComp1
Load upon successful compare with value in COMP1

enumerator kQTMR__LoadOnComp2
Load upon successful compare with value in COMP2

enum __qtmr_debug_action
List of Quad Timer run options when in Debug mode.
Values:

enumerator kQTMR__RunNormalInDebug
Continue with normal operation

enumerator kQTMR__HaltCounter
Halt counter

enumerator kQTMR__ForceOutToZero
Force output to logic 0

enumerator kQTMR__HaltCountForceOutZero
Halt counter and force output to logic 0

enum __qtmr_interrupt_enable
List of Quad Timer interrupts.
Values:

enumerator kQTMR__CompareInterruptEnable
Compare interrupt.

enumerator kQTMR__Compare1InterruptEnable
Compare 1 interrupt.

enumerator kQTMR__Compare2InterruptEnable
Compare 2 interrupt.

enumerator kQTMR__OverflowInterruptEnable
Timer overflow interrupt.

enumerator kQTMR__EdgeInterruptEnable
Input edge interrupt.

enum __qtmr_status_flags
List of Quad Timer flags.
Values:

enumerator kQTMR__CompareFlag
Compare flag

enumerator kQTMR__Compare1Flag
Compare 1 flag

enumerator kQTMR__Compare2Flag
Compare 2 flag

enumerator kQTMR__OverflowFlag
Timer overflow flag

enumerator kQTMR__EdgeFlag
Input edge flag

enum __qtmr_channel_selection
List of channel selection.
Values:

enumerator kQTMR__Channel_0
TMR Channel 0

enumerator kQTMR__Channel_1
TMR Channel 1

enumerator kQTMR__Channel_2
TMR Channel 2

enumerator kQTMR_Channel_3
TMR Channel 3

enum _qtmr_dma_enable

List of Quad Timer DMA enable.

Values:

enumerator kQTMR_InputEdgeFlagDmaEnable
Input Edge Flag DMA Enable.

enumerator kQTMR_ComparatorPreload1DmaEnable
Comparator Preload Register 1 DMA Enable.

enumerator kQTMR_ComparatorPreload2DmaEnable
Comparator Preload Register 2 DMA Enable.

typedef uint16_t qtmrRegType

typedef enum _qtmr_primary_count_source qtmr_primary_count_source_t
Quad Timer primary clock source selection.

typedef enum _qtmr_input_source qtmr_input_source_t
Quad Timer input sources selection.

typedef enum _qtmr_counting_mode qtmr_counting_mode_t
Quad Timer counting mode selection.

typedef enum _qtmr_pwm_out_state qtmr_pwm_out_state_t
Quad Timer PWM output state.

typedef enum _qtmr_output_mode qtmr_output_mode_t
Quad Timer output mode selection.

typedef enum _qtmr_input_capture_edge qtmr_input_capture_edge_t
Quad Timer input capture edge mode, rising edge, or falling edge.

typedef enum _qtmr_preload_control qtmr_preload_control_t
Quad Timer input capture edge mode, rising edge, or falling edge.

typedef enum _qtmr_debug_action qtmr_debug_action_t
List of Quad Timer run options when in Debug mode.

typedef enum _qtmr_interrupt_enable qtmr_interrupt_enable_t
List of Quad Timer interrupts.

typedef enum _qtmr_status_flags qtmr_status_flags_t
List of Quad Timer flags.

typedef enum _qtmr_channel_selection qtmr_channel_selection_t
List of channel selection.

typedef enum _qtmr_dma_enable qtmr_dma_enable_t
List of Quad Timer DMA enable.

typedef struct _qtmr_config qtmr_config_t
Quad Timer config structure.

This structure holds the configuration settings for the Quad Timer peripheral. To initialize this structure to reasonable defaults, call the QTMR_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

```
status_t QTMR_SetupPwm(TMR_Type *base, qtmr_channel_selection_t channel, uint32_t
                        pwmFreqHz, uint8_t dutyCyclePercent, bool outputPolarity,
                        uint32_t srcClock_Hz)
```

Sets up Quad timer module for PWM signal output.

The function initializes the timer module according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- pwmFreqHz – PWM signal frequency in Hz
- dutyCyclePercent – PWM pulse width, value should be between 0 to 100
0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle)
- outputPolarity – true: invert polarity of the output signal, false: no inversion
- srcClock_Hz – Main counter clock in Hz.

Returns

Returns an error if there was error setting up the signal.

```
void QTMR_SetupInputCapture(TMR_Type *base, qtmr_channel_selection_t channel,
                            qtmr_input_source_t capturePin, bool inputPolarity, bool
                            reloadOnCapture, qtmr_input_capture_edge_t captureMode)
```

Allows the user to count the source clock cycles until a capture event arrives.

The count is stored in the capture register.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- capturePin – Pin through which we receive the input signal to trigger the capture
- inputPolarity – true: invert polarity of the input signal, false: no inversion
- reloadOnCapture – true: reload the counter when an input capture occurs, false: no reload
- captureMode – Specifies which edge of the input signal triggers a capture

```
TMR_CSCTRL_OFLAG_MASK
```

```
TMR_CSCTRL_OFLAG_SHIFT
```

```
struct __qtmr_config
```

#include <fsl_qtmr.h> Quad Timer config structure.

This structure holds the configuration settings for the Quad Timer peripheral. To initialize this structure to reasonable defaults, call the QTMR_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

qtmr_primary_count_source_t primarySource

Specify the primary count source

qtmr_input_source_t secondarySource

Specify the secondary count source

bool enableMasterMode

true: Broadcast compare function output to other counters; false no broadcast

bool enableExternalForce

true: Compare from another counter force state of OFLAG signal false: OFLAG controlled by local counter

uint8_t faultFilterCount

Fault filter count

uint8_t faultFilterPeriod

Fault filter period; value of 0 will bypass the filter

qtmr_debug_action_t debugMode

Operation in Debug mode

2.92 RDC: Resource Domain Controller

enum _rdc_interrupts

RDC interrupts.

Values:

enumerator kRDC_RestoreCompleteInterrupt

Interrupt generated when the RDC has completed restoring state to a recently re-powered memory regions.

enum _rdc_flags

RDC status.

Values:

enumerator kRDC_PowerDownDomainOn

Power down domain is ON.

enum _rdc_access_policy

Access permission policy.

Values:

enumerator kRDC_NoAccess

Could not read or write.

enumerator kRDC_WriteOnly

Write only.

enumerator kRDC_ReadOnly

Read only.

enumerator kRDC_ReadWrite

Read and write.

typedef struct *_rdc_hardware_config* rdc_hardware_config_t

RDC hardware configuration.

```
typedef struct _rdc_domain_assignment rdc_domain_assignment_t
```

Master domain assignment.

```
typedef struct _rdc_periph_access_config rdc_periph_access_config_t
```

Peripheral domain access permission configuration.

```
typedef struct _rdc_mem_access_config rdc_mem_access_config_t
```

Memory region domain access control configuration.

Note that when setting the `rdc_mem_access_config_t::baseAddress` and `rdc_mem_access_config_t::endAddress`, should be aligned to the region resolution, see `rdc_mem_t` definitions.

```
typedef struct _rdc_mem_status rdc_mem_status_t
```

Memory region access violation status.

```
void RDC_Init(RDC_Type *base)
```

Initializes the RDC module.

This function enables the RDC clock.

Parameters

- `base` – RDC peripheral base address.

```
void RDC_Deinit(RDC_Type *base)
```

De-initializes the RDC module.

This function disables the RDC clock.

Parameters

- `base` – RDC peripheral base address.

```
void RDC_GetHardwareConfig(RDC_Type *base, rdc_hardware_config_t *config)
```

Gets the RDC hardware configuration.

This function gets the RDC hardware configurations, including number of bus masters, number of domains, number of memory regions and number of peripherals.

Parameters

- `base` – RDC peripheral base address.
- `config` – Pointer to the structure to get the configuration.

```
static inline void RDC_EnableInterrupts(RDC_Type *base, uint32_t mask)
```

Enable interrupts.

Parameters

- `base` – RDC peripheral base address.
- `mask` – Interrupts to enable, it is OR'ed value of `enum _rdc_interrupts`.

```
static inline void RDC_DisableInterrupts(RDC_Type *base, uint32_t mask)
```

Disable interrupts.

Parameters

- `base` – RDC peripheral base address.
- `mask` – Interrupts to disable, it is OR'ed value of `enum _rdc_interrupts`.

```
static inline uint32_t RDC_GetInterruptStatus(RDC_Type *base)
```

Get the interrupt pending status.

Parameters

- `base` – RDC peripheral base address.

Returns

Interrupts pending status, it is OR'ed value of enum `_rdc_interrupts`.

```
static inline void RDC_ClearInterruptStatus(RDC_Type *base, uint32_t mask)
```

Clear interrupt pending status.

Parameters

- `base` – RDC peripheral base address.
- `mask` – Status to clear, it is OR'ed value of enum `_rdc_interrupts`.

```
static inline uint32_t RDC_GetStatus(RDC_Type *base)
```

Get RDC status.

Parameters

- `base` – RDC peripheral base address.

Returns

mask RDC status, it is OR'ed value of enum `_rdc_flags`.

```
static inline void RDC_ClearStatus(RDC_Type *base, uint32_t mask)
```

Clear RDC status.

Parameters

- `base` – RDC peripheral base address.
- `mask` – RDC status to clear, it is OR'ed value of enum `_rdc_flags`.

```
void RDC_SetMasterDomainAssignment(RDC_Type *base, rdc_master_t master, const  
                                   rdc_domain_assignment_t *domainAssignment)
```

Set master domain assignment.

Parameters

- `base` – RDC peripheral base address.
- `master` – Which master to set.
- `domainAssignment` – Pointer to the assignment.

```
void RDC_GetDefaultMasterDomainAssignment(rdc_domain_assignment_t *domainAssignment)
```

Get default master domain assignment.

The default configuration is:

```
assignment->domainId = 0U;  
assignment->lock = 0U;
```

Parameters

- `domainAssignment` – Pointer to the assignment.

```
static inline void RDC_LockMasterDomainAssignment(RDC_Type *base, rdc_master_t master)
```

Lock master domain assignment.

Once locked, it could not be unlocked until next reset.

Parameters

- `base` – RDC peripheral base address.
- `master` – Which master to lock.

`void RDC_SetPeriphAccessConfig(RDC_Type *base, const rdc_periph_access_config_t *config)`
Set peripheral access policy.

Parameters

- base – RDC peripheral base address.
- config – Pointer to the policy configuration.

`void RDC_GetDefaultPeriphAccessConfig(rdc_periph_access_config_t *config)`
Get default peripheral access policy.

The default configuration is:

```
config->lock = false;
config->enableSema = false;
config->policy = RDC_ACCESS_POLICY(0, kRDC_ReadWrite) |
                RDC_ACCESS_POLICY(1, kRDC_ReadWrite) |
                RDC_ACCESS_POLICY(2, kRDC_ReadWrite) |
                RDC_ACCESS_POLICY(3, kRDC_ReadWrite);
```

Parameters

- config – Pointer to the policy configuration.

`static inline void RDC_LockPeriphAccessConfig(RDC_Type *base, rdc_periph_t periph)`
Lock peripheral access policy configuration.

Once locked, it could not be unlocked until reset.

Parameters

- base – RDC peripheral base address.
- periph – Which peripheral to lock.

`static inline uint8_t RDC_GetPeriphAccessPolicy(RDC_Type *base, rdc_periph_t periph, uint8_t domainId)`

Get the peripheral access policy for specific domain.

Parameters

- base – RDC peripheral base address.
- periph – Which peripheral to get.
- domainId – Get policy for which domain.

Returns

Access policy, see `_rdc_access_policy`.

`void RDC_SetMemAccessConfig(RDC_Type *base, const rdc_mem_access_config_t *config)`
Set memory region access policy.

Note that when setting the baseAddress and endAddress in config, should be aligned to the region resolution, see `rdc_mem_t` definitions.

Parameters

- base – RDC peripheral base address.
- config – Pointer to the policy configuration.

`void RDC_GetDefaultMemAccessConfig(rdc_mem_access_config_t *config)`
Get default memory region access policy.

The default configuration is:

```

config->lock = false;
config->baseAddress = 0;
config->endAddress = 0;
config->policy = RDC_ACCESS_POLICY(0, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(1, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(2, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(3, kRDC_ReadWrite);

```

Parameters

- config – Pointer to the policy configuration.

static inline void RDC_LockMemAccessConfig(RDC_Type *base, rdc_mem_t mem)

Lock memory access policy configuration.

Once locked, it could not be unlocked until reset. After locked, you can only call RDC_SetMemAccessValid to enable the configuration, but can not disable it or change other settings.

Parameters

- base – RDC peripheral base address.
- mem – Which memory region to lock.

static inline void RDC_SetMemAccessValid(RDC_Type *base, rdc_mem_t mem, bool valid)

Enable or disable memory access policy configuration.

Parameters

- base – RDC peripheral base address.
- mem – Which memory region to operate.
- valid – Pass in true to valid, false to invalid.

void RDC_GetMemViolationStatus(RDC_Type *base, rdc_mem_t mem, rdc_mem_status_t *status)

Get the memory region violation status.

The first access violation is captured. Subsequent violations are ignored until the status register is cleared. Contents are cleared upon reading the register. Clearing of contents occurs only when the status is read by the memory region's associated domain ID(s).

Parameters

- base – RDC peripheral base address.
- mem – Which memory region to get.
- status – The returned status.

static inline void RDC_ClearMemViolationFlag(RDC_Type *base, rdc_mem_t mem)

Clear the memory region violation flag.

Parameters

- base – RDC peripheral base address.
- mem – Which memory region to clear.

static inline uint8_t RDC_GetMemAccessPolicy(RDC_Type *base, rdc_mem_t mem, uint8_t domainId)

Get the memory region access policy for specific domain.

Parameters

- base – RDC peripheral base address.
- mem – Which memory region to get.

- domainId – Get policy for which domain.

Returns

Access policy, see `_rdc_access_policy`.

static inline uint8_t RDC_GetCurrentMasterDomainId(RDC_Type *base)

Gets the domain ID of the current bus master.

This function returns the domain ID of the current bus master.

Parameters

- base – RDC peripheral base address.

Returns

Domain ID of current bus master.

FSL_RDC_DRIVER_VERSION

RDC_ACCESS_POLICY(domainID, policy)

struct _rdc_hardware_config

#include <fsl_rdc.h> RDC hardware configuration.

Public Members

uint32_t domainNumber

Number of domains.

uint32_t masterNumber

Number of bus masters.

uint32_t periphNumber

Number of peripherals.

uint32_t memNumber

Number of memory regions.

struct _rdc_domain_assignment

#include <fsl_rdc.h> Master domain assignment.

Public Members

uint32_t domainId

Domain ID.

uint32_t __pad0__

Reserved.

uint32_t lock

Lock the domain assignment.

struct _rdc_periph_access_config

#include <fsl_rdc.h> Peripheral domain access permission configuration.

Public Members

rdc_periph_t periph

Peripheral name.

bool lock

Lock the permission until reset.

bool enableSema

Enable semaphore or not, when enabled, master should call RDC_SEMA42_Lock to lock the semaphore gate accordingly before access the peripheral.

uint16_t policy

Access policy.

struct _rdc_mem_access_config

#include <fsl_rdc.h> Memory region domain access control configuration.

Note that when setting the rdc_mem_access_config_t::baseAddress and rdc_mem_access_config_t::endAddress, should be aligned to the region resolution, see rdc_mem_t definitions.

Public Members

rdc_mem_t mem

Memory region descriptor name.

bool lock

Lock the configuration.

uint64_t baseAddress

Start address of the memory region.

uint64_t endAddress

End address of the memory region.

uint16_t policy

Access policy.

struct _rdc_mem_status

#include <fsl_rdc.h> Memory region access violation status.

Public Members

bool hasViolation

Violating happens or not.

uint8_t domainID

Violating Domain ID.

uint64_t address

Violating Address.

2.93 RDC_SEMA42: Hardware Semaphores Driver

FSL_RDC_SEMA42_DRIVER_VERSION

RDC_SEMA42 driver version.

`void RDC_SEMA42_Init(RDC_SEMAPHORE_Type *base)`

Initializes the RDC_SEMA42 module.

This function initializes the RDC_SEMA42 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either RDC_SEMA42_ResetGate or RDC_SEMA42_ResetAllGates function.

Parameters

- `base` – RDC_SEMA42 peripheral base address.

`void RDC_SEMA42_Deinit(RDC_SEMAPHORE_Type *base)`

De-initializes the RDC_SEMA42 module.

This function de-initializes the RDC_SEMA42 module. It only disables the clock.

Parameters

- `base` – RDC_SEMA42 peripheral base address.

`status_t RDC_SEMA42_TryLock(RDC_SEMAPHORE_Type *base, uint8_t gateNum, uint8_t masterIndex, uint8_t domainId)`

Tries to lock the RDC_SEMA42 gate.

This function tries to lock the specific RDC_SEMA42 gate. If the gate has been locked by another processor, this function returns an error code.

Parameters

- `base` – RDC_SEMA42 peripheral base address.
- `gateNum` – Gate number to lock.
- `masterIndex` – Current processor master index.
- `domainId` – Current processor domain ID.

Return values

- `kStatus_Success` – Lock the sema42 gate successfully.
- `kStatus_Failed` – Sema42 gate has been locked by another processor.

`void RDC_SEMA42_Lock(RDC_SEMAPHORE_Type *base, uint8_t gateNum, uint8_t masterIndex, uint8_t domainId)`

Locks the RDC_SEMA42 gate.

This function locks the specific RDC_SEMA42 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

Parameters

- `base` – RDC_SEMA42 peripheral base address.
- `gateNum` – Gate number to lock.
- `masterIndex` – Current processor master index.
- `domainId` – Current processor domain ID.

`static inline void RDC_SEMA42_Unlock(RDC_SEMAPHORE_Type *base, uint8_t gateNum)`

Unlocks the RDC_SEMA42 gate.

This function unlocks the specific RDC_SEMA42 gate. It only writes unlock value to the RDC_SEMA42 gate register. However, it does not check whether the RDC_SEMA42 gate is locked by the current processor or not. As a result, if the RDC_SEMA42 gate is not locked by the current processor, this function has no effect.

Parameters

- `base` – RDC_SEMA42 peripheral base address.

- gateNum – Gate number to unlock.

static inline int32_t RDC_SEMA42_GetLockMasterIndex(RDC_SEMAPHORE_Type *base, uint8_t gateNum)

Gets which master has currently locked the gate.

Parameters

- base – RDC_SEMA42 peripheral base address.
- gateNum – Gate number.

Returns

Return -1 if the gate is not locked by any master, otherwise return the master index.

int32_t RDC_SEMA42_GetLockDomainID(RDC_SEMAPHORE_Type *base, uint8_t gateNum)

Gets which domain has currently locked the gate.

Parameters

- base – RDC_SEMA42 peripheral base address.
- gateNum – Gate number.

Returns

Return -1 if the gate is not locked by any domain, otherwise return the domain ID.

status_t RDC_SEMA42_ResetGate(RDC_SEMAPHORE_Type *base, uint8_t gateNum)

Resets the RDC_SEMA42 gate to an unlocked status.

This function resets a RDC_SEMA42 gate to an unlocked status.

Parameters

- base – RDC_SEMA42 peripheral base address.
- gateNum – Gate number.

Return values

- kStatus_Success – RDC_SEMA42 gate is reset successfully.
- kStatus_Failed – Some other reset process is ongoing.

static inline status_t RDC_SEMA42_ResetAllGates(RDC_SEMAPHORE_Type *base)

Resets all RDC_SEMA42 gates to an unlocked status.

This function resets all RDC_SEMA42 gate to an unlocked status.

Parameters

- base – RDC_SEMA42 peripheral base address.

Return values

- kStatus_Success – RDC_SEMA42 is reset successfully.
- kStatus_RDC_SEMA42_Reseting – Some other reset process is ongoing.

RDC_SEMA42_GATE_NUM_RESET_ALL

The number to reset all RDC_SEMA42 gates.

RDC_SEMA42_GATEn(base, n)

RDC_SEMA42 gate n register address.

RDC_SEMA42_GATE_COUNT

RDC_SEMA42 gate count.

RDC_SEMAPHORE_GATE_GTFSM_MASK

2.94 Romapi

Flash Pad Definitions.

Values:

enumerator kSerialFlash_1Pad

1-wire communication

enumerator kSerialFlash_2Pads

2-wire communication

enumerator kSerialFlash_4Pads

4-wire communication

enumerator kSerialFlash_8Pads

8-wire communication

FLEXSPI clock configuration type.

Values:

enumerator kFLEXSPIClk_SDR

Clock configure for SDR mode

enumerator kFLEXSPIClk_DDR

Clock configure for DDR mode

enum _flexspi_read_sample_clk

FLEXSPI Read Sample Clock Source definition.

Values:

enumerator kFLEXSPIReadSampleClk_LoopbackInternally

FLEXSPI Read Sample Clock Source from the Internal loopback

enumerator kFLEXSPIReadSampleClk_LoopbackFromDqsPad

FLEXSPI Read Sample Clock Source from the Dqs Pad loopback

enumerator kFLEXSPIReadSampleClk_LoopbackFromSckPad

FLEXSPI Read Sample Clock Source from the Sck Pad loopback

enumerator kFLEXSPIReadSampleClk_ExternalInputFromDqsPad

FLEXSPI Read Sample Clock Source from the External Input by the Dqs Pad

Flash Type Definition.

Values:

enumerator kFLEXSPIDeviceType_SerialNOR

Flash device is Serial NOR

Flash Configuration Command Type.

Values:

enumerator kDeviceConfigCmdType_Generic

Generic command, for example: configure dummy cycles, drive strength, etc.

enumerator kDeviceConfigCmdType_QuadEnable

Quad Enable command

enumerator kDeviceConfigCmdType_Spi2Xpi

Switch from SPI to DPI/QPI/OPI mode

enumerator kDeviceConfigCmdType_Xpi2Spi

Switch from DPI/QPI/OPI to SPI mode

enumerator kDeviceConfigCmdType_Spi2NoCmd

Switch to 0-4-4/0-8-8 mode

enumerator kDeviceConfigCmdType_Reset

Reset device command

enum _flexspi_serial_clk_freq

Definitions for FLEXSPI Serial Clock Frequency.

Values:

enumerator kFLEXSPISerialClk_NoChange

FlexSPI serial clock no changed

enumerator kFLEXSPISerialClk_30MHz

FlexSPI serial clock 30MHz

enumerator kFLEXSPISerialClk_50MHz

FlexSPI serial clock 50MHz

enumerator kFLEXSPISerialClk_60MHz

FlexSPI serial clock 60MHz

enumerator kFLEXSPISerialClk_75MHz

FlexSPI serial clock 75MHz

enumerator kFLEXSPISerialClk_80MHz

FlexSPI serial clock 80MHz

enumerator kFLEXSPISerialClk_100MHz

FlexSPI serial clock 100MHz

enumerator kFLEXSPISerialClk_133MHz

FlexSPI serial clock 133MHz

enumerator kFLEXSPISerialClk_166MHz

FlexSPI serial clock 166MHz

Misc feature bit definitions.

Values:

enumerator kFLEXSPIMiscOffset_DiffClkEnable

Bit for Differential clock enable

enumerator kFLEXSPIMiscOffset_Ck2Enable

Bit for CK2 enable

enumerator kFLEXSPIMiscOffset_ParallelEnable

Bit for Parallel mode enable

enumerator kFLEXSPIMiscOffset_WordAddressableEnable

Bit for Word Addressable enable

enumerator kFLEXSPIMiscOffset_SafeConfigFreqEnable

Bit for Safe Configuration Frequency enable

enumerator kFLEXSPIMiscOffset_PadSettingOverrideEnable

Bit for Pad setting override enable

enumerator kFLEXSPIMiscOffset_DdrModeEnable

Bit for DDR clock configuration indication.

enumerator kFLEXSPIMiscOffset_UseValidTimeForAllFreq

Bit for DLLCR settings under all modes

Manufacturer ID.

Values:

enumerator kSerialFlash_ISSI_ManufacturerID

Manufacturer ID of the ISSI serial flash

enumerator kSerialFlash_Adesto_ManufacturerID

Manufacturer ID of the Adesto Technologies serial flash

enumerator kSerialFlash_Winbond_ManufacturerID

Manufacturer ID of the Winbond serial flash

enumerator kSerialFlash_Cypress_ManufacturerID

Manufacturer ID for Cypress

enum _flexspi_nor_status

ROM FLEXSPI NOR flash status.

Values:

enumerator kStatus_ROM_FLEXSPI_SequenceExecutionTimeout

Status for Sequence Execution timeout

enumerator kStatus_ROM_FLEXSPI_InvalidSequence

Status for Invalid Sequence

enumerator kStatus_ROM_FLEXSPI_DeviceTimeout

Status for Device timeout

enumerator kStatus_ROM_FLEXSPINOR_SFDP_NotFound

Status for SFDP read failure

enumerator kStatus_ROM_FLEXSPINOR_Flash_NotFound

Status for Flash detection failure

enumerator kStatus_FLEXSPINOR_DTRRead_DummyProbeFailed

Status for DDR Read dummy probe failure

enum _flexspi_operation

FLEXSPI Operation Context.

Values:

enumerator kFLEXSPIOperation_Command

FLEXSPI operation: Only command, both TX and RX buffer are ignored.

enumerator kFLEXSPIOperation_Config

FLEXSPI operation: Configure device mode, the TX FIFO size is fixed in LUT.

enumerator kFLEXSPIOperation_Write

FLEXSPI operation: Write, only TX buffer is effective

enumerator kFLEXSPIOperation_Read

FLEXSPI operation: Read, only Rx Buffer is effective.

typedef struct *_flexspi_lut_seq* flexspi_lut_seq_t

FLEXSPI LUT Sequence structure.

typedef struct *_flexspi_mem_config* flexspi_mem_config_t

FLEXSPI Memory Configuration Block.

typedef struct *_flexspi_nor_config* flexspi_nor_config_t

Serial NOR configuration block.

typedef enum *_flexspi_operation* flexspi_operation_t

FLEXSPI Operation Context.

typedef struct *_flexspi_xfer* flexspi_xfer_t

FLEXSPI Transfer Context.

kFLEXSPIOperation_End

MISRA_CAST(to_type, to_var, from_type, from_var)

convert the type for MISRA

typedef struct *_serial_nor_config_option* serial_nor_config_option_t

Serial NOR Configuration Option.

void ROM_RunBootloader(void *arg)

Enter Bootloader.

Parameters

- arg – A pointer to the storage for the bootloader param. refer to System Boot Chapter in device reference manual for details.

status_t ROM_FLEXSPI_NorFlash_GetConfig(uint32_t instance, *flexspi_nor_config_t* *config, *serial_nor_config_option_t* *option)

Get FLEXSPI NOR Configuration Block based on specified option.

Parameters

- instance – storage the instance of FLEXSPI.
- config – A pointer to the storage for the driver runtime state.
- option – A pointer to the storage Serial NOR Configuration Option Context.

Return values

- kStatus_Success – Api was executed successfully.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_ROM_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_ROM_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.
- kStatus_ROM_FLEXSPI_DeviceTimeout – the device timeout

status_t ROM_FLEXSPI_NorFlash_Init(uint32_t instance, *flexspi_nor_config_t* *config)

Initialize Serial NOR devices via FLEXSPI.

This function checks and initializes the FLEXSPI module for the other FLEXSPI APIs.

Parameters

- `instance` – storage the instance of FLEXSPI.
- `config` – A pointer to the storage for the driver runtime state.

Return values

- `kStatus_Success` – Api was executed successfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_ROM_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_ROM_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.
- `kStatus_ROM_FLEXSPI_DeviceTimeout` – the device timeout

`status_t` ROM_FLEXSPI_NorFlash_ProgramPage(uint32_t instance, *flexspi_nor_config_t* *config, uint32_t dst_addr, const uint32_t *src)

Program data to Serial NOR via FLEXSPI.

This function programs the NOR flash memory with the dest address for a given flash area as determined by the dst address and the length.

Note: It is recommended that use page aligned access; If the `dst_addr` is not aligned to page, the driver automatically aligns address down with the page address.

Parameters

- `instance` – storage the instance of FLEXSPI.
- `config` – A pointer to the storage for the driver runtime state.
- `dst_addr` – A pointer to the desired flash memory to be programmed.
- `src` – A pointer to the source buffer of data that is to be programmed into the NOR flash.

Return values

- `kStatus_Success` – Api was executed successfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_ROM_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_ROM_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.
- `kStatus_ROM_FLEXSPI_DeviceTimeout` – the device timeout

`status_t` ROM_FLEXSPI_NorFlash_Read(uint32_t instance, *flexspi_nor_config_t* *config, uint32_t *dst, uint32_t start, uint32_t lengthInBytes)

Read data from Serial NOR via FLEXSPI.

This function read the NOR flash memory with the start address for a given flash area as determined by the dst address and the length.

Note: It is recommended that use page aligned access; If the `dstAddr` is not aligned to page, the driver automatically aligns address down with the page address.

Parameters

- `instance` – storage the instance of FLEXSPI.
- `config` – A pointer to the storage for the driver runtime state.

- `dst` – A pointer to the dest buffer of data that is to be read from the NOR flash.
- `start` – The start address of the desired NOR flash memory to be read.
- `lengthInBytes` – The length, given in bytes to be read.

Return values

- `kStatus_Success` – Api was executed successfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_ROM_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_ROM_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.
- `kStatus_ROM_FLEXSPI_DeviceTimeout` – the device timeout

`status_t` ROM_FLEXSPI_NorFlash_Erase(uint32_t instance, *flexspi_nor_config_t* *config, uint32_t start, uint32_t length)

Erase Flash Region specified by address and length.

This function erases the appropriate number of flash sectors based on the desired start address and length.

Note: It is recommended that use sector-aligned access nor device; If `dstAddr` is not aligned with the sector, the driver automatically aligns address down with the sector address.

Note: It is recommended that use sector-aligned access nor device; If `length` is not aligned with the sector, the driver automatically aligns up with the sector.

Parameters

- `instance` – storage the index of FLEXSPI.
- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired NOR flash memory to be erased.
- `length` – The length, given in bytes to be erased.

Return values

- `kStatus_Success` – Api was executed successfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_ROM_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_ROM_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.
- `kStatus_ROM_FLEXSPI_DeviceTimeout` – the device timeout

`status_t` ROM_FLEXSPI_NorFlash_EraseSector(uint32_t instance, *flexspi_nor_config_t* *config, uint32_t start)

Erase one sector specified by address.

This function erases one of NOR flash sectors based on the desired address.

Note: It is recommended that use sector-aligned access nor device; If `dstAddr` is not aligned with the sector, the driver automatically aligns address down with the sector address.

Parameters

- `instance` – storage the index of FLEXSPI.
- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired NOR flash memory to be erased.

Return values

- `kStatus_Success` – Api was executed successfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_ROM_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_ROM_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.
- `kStatus_ROM_FLEXSPI_DeviceTimeout` – the device timeout

`status_t` ROM_FLEXSPI_NorFlash_EraseBlock(uint32_t instance, *flexspi_nor_config_t* *config, uint32_t start)

Erase one block specified by address.

This function erases one block of NOR flash based on the desired address.

Note: It is recommended that use block-aligned access nor device; If `dstAddr` is not aligned with the block, the driver automatically aligns address down with the block address.

Parameters

- `instance` – storage the index of FLEXSPI.
- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired NOR flash memory to be erased.

Return values

- `kStatus_Success` – Api was executed successfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_ROM_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_ROM_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.
- `kStatus_ROM_FLEXSPI_DeviceTimeout` – the device timeout

`status_t` ROM_FLEXSPI_NorFlash_EraseAll(uint32_t instance, *flexspi_nor_config_t* *config)

Erase all the Serial NOR devices connected on FLEXSPI.

Parameters

- `instance` – storage the instance of FLEXSPI.
- `config` – A pointer to the storage for the driver runtime state.

Return values

- `kStatus_Success` – Api was executed successfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_ROM_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_ROM_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.

- `kStatus_ROM_FLEXSPI_DeviceTimeout` – the device timeout

`status_t ROM_FLEXSPI_NorFlash_CommandXfer(uint32_t instance, flexspi_xfer_t *xfer)`
FLEXSPI command.

This function is used to perform the command write sequence to the NOR device.

Parameters

- `instance` – storage the index of FLEXSPI.
- `xfer` – A pointer to the storage FLEXSPI Transfer Context.

Return values

- `kStatus_Success` – Api was executed successfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_ROM_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_ROM_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.

`status_t ROM_FLEXSPI_NorFlash_UpdateLut(uint32_t instance, uint32_t seqIndex, const uint32_t *lutBase, uint32_t seqNumber)`

Configure FLEXSPI Lookup table.

Parameters

- `instance` – storage the index of FLEXSPI.
- `seqIndex` – storage the sequence Id.
- `lutBase` – A pointer to the look-up-table for command sequences.
- `seqNumber` – storage sequence number.

Return values

- `kStatus_Success` – Api was executed successfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_ROM_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_ROM_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.

`status_t ROM_FLEXSPI_NorFlash_WaitBusy(uint32_t instance, flexspi_nor_config_t *config, bool isParallelMode, uint32_t address)`

Wait until device is idle.

Parameters

- `instance` – Indicates the index of FLEXSPI.
- `config` – A pointer to the storage for the driver runtime state
- `isParallelMode` – Indicates whether NOR flash is in parallel mode.
- `address` – Indicates the operation(erase/program/read) address for serial NOR flash.

Return values

- `kStatus_Success` – Api was executed successfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_ROM_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.

- kStatus_ROM_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_ROM_FLEXSPI_DeviceTimeout – Device timeout.

void ROM_FLEXSPI_NorFlash_ClearCache(uint32_t instance)

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

- instance – storage the index of FLEXSPI.

FSL_ROM_HAS_FLEXSPINOR_API

ROM has FLEXSPI NOR API.

FSL_ROM_HAS_RUNBOOTLOADER_API

ROM has run bootloader API.

FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_GET_CONFIG

ROM has FLEXSPI NOR get config API.

FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_FLASH_INIT

ROM has flash init API.

FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE

ROM has erase API.

FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE_SECTOR

ROM has erase sector API.

FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE_BLOCK

ROM has erase block API.

FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE_ALL

ROM has erase all API.

FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_READ

ROM has read API.

FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_UPDATE_LUT

ROM has update lut API.

FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_CMD_XFER

ROM has FLEXSPI command API.

kROM_StatusGroup_FLEXSPINOR

ROM FLEXSPI NOR status group number.

FSL_ROM_FLEXSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)

FSL_ROM_FLEXSPI_BITMASK(bit_offset)

Generate bit mask.

FLEXSPI_CFG_BLK_TAG

FLEXSPI memory config block related definitions.

ascii “FCFB” Big Endian

FLEXSPI_CFG_BLK_VERSION

V1.4.0

CMD_SDR

CMD_DDR
RADDR_SDR
RADDR_DDR
CADDR_SDR
CADDR_DDR
MODE1_SDR
MODE1_DDR
MODE2_SDR
MODE2_DDR
MODE4_SDR
MODE4_DDR
MODE8_SDR
MODE8_DDR
WRITE_SDR
WRITE_DDR
READ_SDR
READ_DDR
LEARN_SDR
LEARN_DDR
DATSZ_SDR
DATSZ_DDR
DUMMY_SDR
DUMMY_DDR
DUMMY_RWDS_SDR
DUMMY_RWDS_DDR
JMP_ON_CS
STOP
FLEXSPI_1PAD
FLEXSPI_2PAD
FLEXSPI_4PAD
FLEXSPI_8PAD

NOR_CMD_LUT_SEQ_IDX_READ

NOR LUT sequence index used for default LUT assignment.

Note: It will take effect if the lut sequences are not customized. READ LUT sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_READSTATUS

Read Status LUT sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_READSTATUS_XPI

Read status DPI/QPI/OPI sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_WRITEENABLE

Write Enable sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_WRITEENABLE_XPI

Write Enable DPI/QPI/OPI sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_ERASESECTOR

Erase Sector sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_READID

NOR_CMD_LUT_SEQ_IDX_ERASEBLOCK

Erase Block sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_PAGEPROGRAM

Program sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_CHIPERASE

Chip Erase sequence in lookupTable id stored in config block

NOR_CMD_LUT_SEQ_IDX_READ_SFDP

Read SFDP sequence in lookupTable id stored in config block

NOR_CMD_LUT_SEQ_IDX_RESTORE_NOCMD

Restore 0-4-4/0-8-8 mode sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_EXIT_NOCMD

Exit 0-4-4/0-8-8 mode sequence id in lookupTable stored in config block

uint32_t max_freq

Maximum supported Frequency

uint32_t misc_mode

miscellaneous mode

uint32_t quad_mode_setting

Quad mode setting

uint32_t cmd_pads

Command pads

uint32_t query_pads

SFDP read pads

uint32_t device_type

Device type

uint32_t option_size

Option size, in terms of uint32_t, size = (option_size + 1) * 4

uint32_t tag

Tag, must be 0x0E

struct *_serial_nor_config_option* B

uint32_t U

union *_serial_nor_config_option* option0

uint32_t dummy__cycles

Dummy cycles before read

uint32_t status__override

Override status register value during device mode configuration

uint32_t pinmux__group

The pinmux group selection

uint32_t dqs__pinmux__group

The DQS Pinmux Group Selection

uint32_t drive__strength

The Drive Strength of FlexSPI Pads

uint32_t flash__connection

Flash connection option: 0 - Single Flash connected to port A, 1 - Parallel mode, 2 - Single Flash connected to Port B

struct *_serial_nor_config_option* B

uint32_t U

union *_serial_nor_config_option* option1

uint8_t seqNum

Sequence Number, valid number: 1-16

uint8_t seqId

Sequence Index, valid number: 0-15

uint16_t reserved

uint8_t time__100ps

Data valid time, in terms of 100ps

uint8_t delay__cells

Data valid time, in terms of delay cells

uint32_t tag

[0x000-0x003] Tag, fixed value 0x42464346UL

uint32_t version

[0x004-0x007] Version,[31:24] -'V', [23:16] - Major, [15:8] - Minor, [7:0] - bugfix

uint32_t reserved0

[0x008-0x00b] Reserved for future use

uint8_t readSampleClkSrc

[0x00c-0x00c] Read Sample Clock Source, valid value: 0/1/3

uint8_t csHoldTime

[0x00d-0x00d] Data hold time, default value: 3

uint8_t csSetupTime

[0x00e-0x00e] Date setup time, default value: 3

uint8_t columnAddressWidth

[0x00f-0x00f] Column Address with, for HyperBus protocol, it is fixed to 3, For Serial NAND, need to refer to datasheet

uint8_t deviceModeCfgEnable

[0x010-0x010] Device Mode Configure enable flag, 1 - Enable, 0 - Disable

uint8_t deviceModeType

[0x011-0x011] Specify the configuration command type:Quad Enable, DPI/QPI/OPI switch, Generic configuration, etc.

uint16_t waitTimeCfgCommands

[0x012-0x013] Wait time for all configuration commands, unit: 100us, Used for DPI/QPI/OPI switch or reset command

flexspi_lut_seq_t deviceModeSeq

[0x014-0x017] Device mode sequence info, [7:0] - LUT sequence id, [15:8] - LUt sequence number, [31:16] Reserved

uint32_t deviceModeArg

[0x018-0x01b] Argument/Parameter for device configuration

uint8_t configCmdEnable

[0x01c-0x01c] Configure command Enable Flag, 1 - Enable, 0 - Disable

uint8_t configModeType[3]

[0x01d-0x01f] Configure Mode Type, similar as deviceModeType

flexspi_lut_seq_t configCmdSeqs[3]

[0x020-0x02b] Sequence info for Device Configuration command, similar as deviceModeSeq

uint32_t reserved1

[0x02c-0x02f] Reserved for future use

uint32_t configCmdArgs[3]

[0x030-0x03b] Arguments/Parameters for device Configuration commands

uint32_t reserved2

[0x03c-0x03f] Reserved for future use

uint32_t controllerMiscOption

[0x040-0x043] Controller Misc Options, see Misc feature bit definitions for more details

uint8_t deviceType

[0x044-0x044] Device Type: See Flash Type Definition for more details

uint8_t sflashPadType

[0x045-0x045] Serial Flash Pad Type: 1 - Single, 2 - Dual, 4 - Quad, 8 - Octal

uint8_t serialClkFreq

[0x046-0x046] Serial Flash Frequency, device specific definitions. See System Boot Chapter for more details

uint8_t lutCustomSeqEnable

[0x047-0x047] LUT customization Enable, it is required if the program/erase cannot be done using 1 LUT sequence, currently, only applicable to HyperFLASH

uint32_t reserved3[2]

[0x048-0x04f] Reserved for future use

`uint32_t sflashA1Size`
[0x050-0x053] Size of Flash connected to A1

`uint32_t sflashA2Size`
[0x054-0x057] Size of Flash connected to A2

`uint32_t sflashB1Size`
[0x058-0x05b] Size of Flash connected to B1

`uint32_t sflashB2Size`
[0x05c-0x05f] Size of Flash connected to B2

`uint32_t csPadSettingOverride`
[0x060-0x063] CS pad setting override value

`uint32_t sclkPadSettingOverride`
[0x064-0x067] SCK pad setting override value

`uint32_t dataPadSettingOverride`
[0x068-0x06b] data pad setting override value

`uint32_t dqsPadSettingOverride`
[0x06c-0x06f] DQS pad setting override value

`uint32_t timeoutInMs`
[0x070-0x073] Timeout threshold for read status command

`uint32_t commandInterval`
[0x074-0x077] CS deselect interval between two commands

flexspi_dll_time_t `dataValidTime[2]`
[0x078-0x07b] CLK edge to data valid time for PORT A and PORT B

`uint16_t busyOffset`
[0x07c-0x07d] Busy offset, valid value: 0-31

`uint16_t busyBitPolarity`
[0x07e-0x07f] Busy flag polarity, 0 - busy flag is 1 when flash device is busy, 1 - busy flag is 0 when flash device is busy

`uint32_t lookupTable[64]`
[0x080-0x17f] Lookup table holds Flash command sequences

flexspi_lut_seq_t `lutCustomSeq[12]`
[0x180-0x1af] Customizable LUT Sequences

`uint32_t reserved4[4]`
[0x1b0-0x1bf] Reserved for future use

flexspi_mem_config_t `memConfig`
Common memory configuration info via FLEXSPI

`uint32_t pageSize`
Page size of Serial NOR

`uint32_t sectorSize`
Sector size of Serial NOR

`uint8_t ipcmdSerialClkFreq`
Clock frequency for IP command

`uint8_t isUniformBlockSize`
Sector/Block size is the same

uint8_t isDataOrderSwapped

Data order (D0, D1, D2, D3) is swapped (D1,D0, D3, D2)

uint8_t reserved0[1]

Reserved for future use

uint8_t serialNorType

Serial NOR Flash type: 0/1/2/3

uint8_t needExitNoCmdMode

Need to exit NoCmd mode before other IP command

uint8_t halfClkForNonReadCmd

Half the Serial Clock for non-read command: true/false

uint8_t needRestoreNoCmdMode

Need to Restore NoCmd mode after IP command execution

uint32_t blockSize

Block size

uint32_t reserve2[11]

Reserved for future use

flexspi_operation_t operation

FLEXSPI operation

uint32_t baseAddress

FLEXSPI operation base address

uint32_t seqId

Sequence Id

uint32_t seqNum

Sequence Number

bool isParallelModeEnable

Is a parallel transfer

uint32_t *txBuffer

Tx buffer

uint32_t txSize

Tx size in bytes

uint32_t *rxBuffer

Rx buffer

uint32_t rxSize

Rx size in bytes

FSL_ROM_ROMAPI_VERSION

ROM API version 1.1.2.

FSL_ROM_FLEXSPINOR_DRIVER_VERSION

ROM FLEXSPI NOR driver version 1.7.0.

struct __serial_nor_config_option

#include <fsl_romapi.h> Serial NOR Configuration Option.

struct __flexspi_lut_seq

#include <fsl_romapi.h> FLEXSPI LUT Sequence structure.

```
struct flexspi_dll_time_t
    #include <fsl_romapi.h> FLEXSPI DLL time.
struct _flexspi_mem_config
    #include <fsl_romapi.h> FLEXSPI Memory Configuration Block.
struct _flexspi_nor_config
    #include <fsl_romapi.h> Serial NOR configuration block.
struct _flexspi_xfer
    #include <fsl_romapi.h> FLEXSPI Transfer Context.
union option0
```

Public Members

```
struct _serial_nor_config_option B
uint32_t U
struct B
```

Public Members

```
uint32_t max_freq
    Maximum supported Frequency
uint32_t misc_mode
    miscellaneous mode
uint32_t quad_mode_setting
    Quad mode setting
uint32_t cmd_pads
    Command pads
uint32_t query_pads
    SFDP read pads
uint32_t device_type
    Device type
uint32_t option_size
    Option size, in terms of uint32_t, size = (option_size + 1) * 4
uint32_t tag
    Tag, must be 0x0E
union option1
```

Public Members

```
struct _serial_nor_config_option B
uint32_t U
struct B
```

Public Members

uint32_t dummy__cycles

Dummy cycles before read

uint32_t status__override

Override status register value during device mode configuration

uint32_t pinmux__group

The pinmux group selection

uint32_t dqs__pinmux__group

The DQS Pinmux Group Selection

uint32_t drive__strength

The Drive Strength of FlexSPI Pads

uint32_t flash__connection

Flash connection option: 0 - Single Flash connected to port A, 1 - Parallel mode, 2 - Single Flash connected to Port B

2.95 RTWDOG: 32-bit Watchdog Timer

void RTWDOG_GetDefaultConfig(*rtwdog_config_t* *config)

Initializes the RTWDOG configuration structure.

This function initializes the RTWDOG configuration structure to default values. The default values are:

```
rtwdogConfig->enableRtwdog = true;
rtwdogConfig->clockSource = kRTWDOG_ClockSource1;
rtwdogConfig->prescaler = kRTWDOG_ClockPrescalerDivide1;
rtwdogConfig->workMode.enableWait = true;
rtwdogConfig->workMode.enableStop = false;
rtwdogConfig->workMode.enableDebug = false;
rtwdogConfig->testMode = kRTWDOG_TestModeDisabled;
rtwdogConfig->enableUpdate = true;
rtwdogConfig->enableInterrupt = false;
rtwdogConfig->enableWindowMode = false;
rtwdogConfig->windowValue = 0U;
rtwdogConfig->timeoutValue = 0xFFFFU;
```

See also:

rtwdog_config_t

Parameters

- *config* – Pointer to the RTWDOG configuration structure.

void RTWDOG_Init(RTWDOG_Type *base, const *rtwdog_config_t* *config)

Initializes the RTWDOG module.

This function initializes the RTWDOG. To reconfigure the RTWDOG without forcing a reset first, *enableUpdate* must be set to true in the configuration.

Example:

```
rtwdog_config_t config;
RTWDOG_GetDefaultConfig(&config);
config.timeoutValue = 0x7ffU;
config.enableUpdate = true;
RTWDOG_Init(wdog_base,&config);
```

Parameters

- base – RTWDOG peripheral base address.
- config – The configuration of the RTWDOG.

void RTWDOG_Deinit(RTWDOG_Type *base)

De-initializes the RTWDOG module.

This function shuts down the RTWDOG. Ensure that the WDOG_CS.UPDATE is 1, which means that the register update is enabled.

Parameters

- base – RTWDOG peripheral base address.

static inline void RTWDOG_Enable(RTWDOG_Type *base)

Enables the RTWDOG module.

This function writes a value into the WDOG_CS register to enable the RTWDOG. The WDOG_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – RTWDOG peripheral base address.

static inline void RTWDOG_Disable(RTWDOG_Type *base)

Disables the RTWDOG module.

This function writes a value into the WDOG_CS register to disable the RTWDOG. The WDOG_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – RTWDOG peripheral base address

static inline void RTWDOG_EnableInterrupts(RTWDOG_Type *base, uint32_t mask)

Enables the RTWDOG interrupt.

This function writes a value into the WDOG_CS register to enable the RTWDOG interrupt. The WDOG_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – RTWDOG peripheral base address.
- mask – The interrupts to enable. The parameter can be a combination of the following source if defined:
 - kRTWDOG_InterruptEnable

static inline void RTWDOG_DisableInterrupts(RTWDOG_Type *base, uint32_t mask)

Disables the RTWDOG interrupt.

This function writes a value into the WDOG_CS register to disable the RTWDOG interrupt. The WDOG_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – RTWDOG peripheral base address.
- mask – The interrupts to disabled. The parameter can be a combination of the following source if defined:
 - kRTWDOG_InterruptEnable

static inline uint32_t RTWDOG_GetStatusFlags(RTWDOG_Type *base)

Gets the RTWDOG all status flags.

This function gets all status flags.

Example to get the running flag:

```
uint32_t status;  
status = RTWDOG_GetStatusFlags(wdog_base) & kRTWDOG_RunningFlag;
```

See also:

_rtwdog_status_flags_t

- true: related status flag has been set.
- false: related status flag is not set.

Parameters

- base – RTWDOG peripheral base address

Returns

State of the status flag: asserted (true) or not-asserted (false).

static inline void RTWDOG_EnableWindowMode(RTWDOG_Type *base, bool enable)

Enables/disables the window mode.

Parameters

- base – RTWDOG peripheral base address.
- enable – Enables(true) or disables(false) the feature.

static inline uint32_t RTWDOG_CountToMesec(RTWDOG_Type *base, uint32_t count, uint32_t
clockFreqInHz)

Converts raw count value to millisecond.

Note that if the clock frequency is too high the timeout period can be less than 1 ms. In this case this api will return 0 value.

Parameters

- base – RTWDOG peripheral base address.
- count – Raw count value.
- clockFreqInHz – The frequency of the clock source RTWDOG uses.

Returns

Return converted time. Will return 0 if result is larger than 0xFFFFFFFF.

void RTWDOG_ClearStatusFlags(RTWDOG_Type *base, uint32_t mask)

Clears the RTWDOG flag.

This function clears the RTWDOG status flag.

Example to clear an interrupt flag:

```
RTWDOG_ClearStatusFlags(wdog_base,kRTWDOG_InterruptFlag);
```

Parameters

- base – RTWDOG peripheral base address.
- mask – The status flags to clear. The parameter can be any combination of the following values:
 - kRTWDOG_InterruptFlag

static inline void RTWDOG_SetTimeoutValue(RTWDOG_Type *base, uint16_t timeoutCount)

Sets the RTWDOG timeout value.

This function writes a timeout value into the WDOG_TOVAL register. The WDOG_TOVAL register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – RTWDOG peripheral base address
- timeoutCount – RTWDOG timeout value, count of RTWDOG clock ticks.

static inline void RTWDOG_SetWindowValue(RTWDOG_Type *base, uint16_t windowValue)

Sets the RTWDOG window value.

This function writes a window value into the WDOG_WIN register. The WDOG_WIN register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – RTWDOG peripheral base address.
- windowValue – RTWDOG window value.

__STATIC_FORCEINLINE void RTWDOG_Unlock (RTWDOG_Type *base)

Unlocks the RTWDOG register written.

This function unlocks the RTWDOG register written.

Before starting the unlock sequence and following the configuration, disable the global interrupts. Otherwise, an interrupt could effectively invalidate the unlock sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

Parameters

- base – RTWDOG peripheral base address

static inline void RTWDOG_Refresh(RTWDOG_Type *base)

Refreshes the RTWDOG timer.

This function feeds the RTWDOG. This function should be called before the Watchdog timer is in timeout. Otherwise, a reset is asserted.

Parameters

- base – RTWDOG peripheral base address

static inline uint32_t RTWDOG_GetCounterValue(RTWDOG_Type *base)

Gets the RTWDOG counter value.

This function gets the RTWDOG counter value.

Parameters

- base – RTWDOG peripheral base address.

Returns

Current RTWDOG counter value.

WD0G_FIRST_WORD_OF_UNLOCK

First word of unlock sequence

WDOG_SECOND_WORD_OF_UNLOCK

Second word of unlock sequence

WDOG_FIRST_WORD_OF_REFRESH

First word of refresh sequence

WDOG_SECOND_WORD_OF_REFRESH

Second word of refresh sequence

FSL_RTWDOG_DRIVER_VERSION

RTWDOG driver version.

enum _rtwdog_clock_source

Describes RTWDOG clock source.

Values:

enumerator kRTWDOG_ClockSource0

Clock source 0

enumerator kRTWDOG_ClockSource1

Clock source 1

enumerator kRTWDOG_ClockSource2

Clock source 2

enumerator kRTWDOG_ClockSource3

Clock source 3

enum _rtwdog_clock_prescaler

Describes the selection of the clock prescaler.

Values:

enumerator kRTWDOG_ClockPrescalerDivide1

Divided by 1

enumerator kRTWDOG_ClockPrescalerDivide256

Divided by 256

enum _rtwdog_test_mode

Describes RTWDOG test mode.

Values:

enumerator kRTWDOG_TestModeDisabled

Test Mode disabled

enumerator kRTWDOG_UserModeEnabled

User Mode enabled

enumerator kRTWDOG_LowByteTest

Test Mode enabled, only low byte is used

enumerator kRTWDOG_HighByteTest

Test Mode enabled, only high byte is used

enum _rtwdog_interrupt_enable_t

RTWDOG interrupt configuration structure.

This structure contains the settings for all of the RTWDOG interrupt configurations.

Values:

enumerator kRTWDOG_InterruptEnable

Interrupt is generated before forcing a reset

enum _rtwdog_status_flags_t

RTWDOG status flags.

This structure contains the RTWDOG status flags for use in the RTWDOG functions.

Values:

enumerator kRTWDOG_RunningFlag

Running flag, set when RTWDOG is enabled

enumerator kRTWDOG_InterruptFlag

Interrupt flag, set when interrupt occurs

typedef enum _rtwdog_clock_source rtwdog_clock_source_t

Describes RTWDOG clock source.

typedef enum _rtwdog_clock_prescaler rtwdog_clock_prescaler_t

Describes the selection of the clock prescaler.

typedef struct _rtwdog_work_mode rtwdog_work_mode_t

Defines RTWDOG work mode.

typedef enum _rtwdog_test_mode rtwdog_test_mode_t

Describes RTWDOG test mode.

typedef struct _rtwdog_config rtwdog_config_t

Describes RTWDOG configuration structure.

struct _rtwdog_work_mode

#include <fsl_rtwdog.h> Defines RTWDOG work mode.

Public Members

bool enableWait

Enables or disables RTWDOG in wait mode

bool enableStop

Enables or disables RTWDOG in stop mode

bool enableDebug

Enables or disables RTWDOG in debug mode

struct _rtwdog_config

#include <fsl_rtwdog.h> Describes RTWDOG configuration structure.

Public Members

bool enableRtwdog

Enables or disables RTWDOG

rtwdog_clock_source_t clockSource

Clock source select

rtwdog_clock_prescaler_t prescaler

Clock prescaler value

rtwdog_work_mode_t workMode

Configures RTWDOG work mode in debug stop and wait mode

rtwdog_test_mode_t testMode
Configures RTWDOG test mode

bool enableUpdate
Update write-once register enable

bool enableInterrupt
Enables or disables RTWDOG interrupt

bool enableWindowMode
Enables or disables RTWDOG window mode

uint16_t windowValue
Window value

uint16_t timeoutValue
Timeout value

2.96 SAI: Serial Audio Interface

2.97 SAI Driver

void SAI_Init(I2S_Type *base)

Initializes the SAI peripheral.

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

Parameters

- base – SAI base pointer.

void SAI_Deinit(I2S_Type *base)

De-initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless SAI_TxInit or SAI_RxInit is called to enable the clock.

Parameters

- base – SAI base pointer.

void SAI_TxReset(I2S_Type *base)

Resets the SAI Tx.

This function enables the software reset and FIFO reset of SAI Tx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

void SAI_RxReset(I2S_Type *base)

Resets the SAI Rx.

This function enables the software reset and FIFO reset of SAI Rx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

```
void SAI_TxEnable(I2S_Type *base, bool enable)
```

Enables/disables the SAI Tx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Tx, false means disable.

```
void SAI_RxEnable(I2S_Type *base, bool enable)
```

Enables/disables the SAI Rx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Rx, false means disable.

```
static inline void SAI_TxSetBitClockDirection(I2S_Type *base, sai_master_slave_t masterSlave)
```

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

```
static inline void SAI_RxSetBitClockDirection(I2S_Type *base, sai_master_slave_t masterSlave)
```

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

```
static inline void SAI_RxSetFrameSyncDirection(I2S_Type *base, sai_master_slave_t  
masterSlave)
```

Set Rx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

```
static inline void SAI_TxSetFrameSyncDirection(I2S_Type *base, sai_master_slave_t masterSlave)
```

Set Tx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

```
void SAI_TxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate,  
uint32_t bitWidth, uint32_t channelNumbers)
```

Transmitter bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.

- `sampleRate` – Audio data sample rate.
- `bitWidth` – Audio data bitWidth.
- `channelNumbers` – Audio channel numbers.

```
void SAI_RxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate,  
                           uint32_t bitWidth, uint32_t channelNumbers)
```

Receiver bit clock rate configurations.

Parameters

- `base` – SAI base pointer.
- `sourceClockHz` – Bit clock source frequency.
- `sampleRate` – Audio data sample rate.
- `bitWidth` – Audio data bitWidth.
- `channelNumbers` – Audio channel numbers.

```
void SAI_TxSetBitclockConfig(I2S_Type *base, sai_master_slave_t masterSlave, sai_bit_clock_t  
                             *config)
```

Transmitter Bit clock configurations.

Parameters

- `base` – SAI base pointer.
- `masterSlave` – master or slave.
- `config` – bit clock other configurations, can be NULL in slave mode.

```
void SAI_RxSetBitclockConfig(I2S_Type *base, sai_master_slave_t masterSlave, sai_bit_clock_t  
                             *config)
```

Receiver Bit clock configurations.

Parameters

- `base` – SAI base pointer.
- `masterSlave` – master or slave.
- `config` – bit clock other configurations, can be NULL in slave mode.

```
void SAI_SetMasterClockConfig(I2S_Type *base, sai_master_clock_t *config)
```

Master clock configurations.

Parameters

- `base` – SAI base pointer.
- `config` – master clock configurations.

```
void SAI_TxSetFifoConfig(I2S_Type *base, sai_fifo_t *config)
```

SAI transmitter fifo configurations.

Parameters

- `base` – SAI base pointer.
- `config` – fifo configurations.

```
void SAI_RxSetFifoConfig(I2S_Type *base, sai_fifo_t *config)
```

SAI receiver fifo configurations.

Parameters

- `base` – SAI base pointer.
- `config` – fifo configurations.

```
void SAI_TxSetFrameSyncConfig(I2S_Type *base, sai_master_slave_t masterSlave,  
                             sai_frame_sync_t *config)
```

SAI transmitter Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

```
void SAI_RxSetFrameSyncConfig(I2S_Type *base, sai_master_slave_t masterSlave,  
                             sai_frame_sync_t *config)
```

SAI receiver Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

```
void SAI_TxSetSerialDataConfig(I2S_Type *base, sai_serial_data_t *config)
```

SAI transmitter Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

```
void SAI_RxSetSerialDataConfig(I2S_Type *base, sai_serial_data_t *config)
```

SAI receiver Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

```
void SAI_TxSetConfig(I2S_Type *base, sai_transceiver_t *config)
```

SAI transmitter configurations.

Parameters

- base – SAI base pointer.
- config – transmitter configurations.

```
void SAI_RxSetConfig(I2S_Type *base, sai_transceiver_t *config)
```

SAI receiver configurations.

Parameters

- base – SAI base pointer.
- config – receiver configurations.

```
void SAI_GetClassicI2SConfig(sai_transceiver_t *config, sai_word_width_t bitWidth,  
                             sai_mono_stereo_t mode, uint32_t saiChannelMask)
```

Get classic I2S mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.

- `saiChannelMask` – mask value of the channel to be enable.

```
void SAI_GetLeftJustifiedConfig(sai_transceiver_t *config, sai_word_width_t bitWidth,  
                               sai_mono_stereo_t mode, uint32_t saiChannelMask)
```

Get left justified mode configurations.

Parameters

- `config` – transceiver configurations.
- `bitWidth` – audio data bitWidth.
- `mode` – audio data channel.
- `saiChannelMask` – mask value of the channel to be enable.

```
void SAI_GetRightJustifiedConfig(sai_transceiver_t *config, sai_word_width_t bitWidth,  
                                sai_mono_stereo_t mode, uint32_t saiChannelMask)
```

Get right justified mode configurations.

Parameters

- `config` – transceiver configurations.
- `bitWidth` – audio data bitWidth.
- `mode` – audio data channel.
- `saiChannelMask` – mask value of the channel to be enable.

```
void SAI_GetTDMConfig(sai_transceiver_t *config, sai_frame_sync_len_t frameSyncWidth,  
                     sai_word_width_t bitWidth, uint32_t dataWordNum, uint32_t  
                     saiChannelMask)
```

Get TDM mode configurations.

Parameters

- `config` – transceiver configurations.
- `frameSyncWidth` – length of frame sync.
- `bitWidth` – audio data word width.
- `dataWordNum` – word number in one frame.
- `saiChannelMask` – mask value of the channel to be enable.

```
void SAI_GetDSPConfig(sai_transceiver_t *config, sai_frame_sync_len_t frameSyncWidth,  
                     sai_word_width_t bitWidth, sai_mono_stereo_t mode, uint32_t  
                     saiChannelMask)
```

Get DSP mode configurations.

DSP/PCM MODE B configuration flow for TX. RX is similiar but uses `SAI_RxSetConfig` instead of `SAI_TxSetConfig`:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)  
SAI_TxSetConfig(base, config)
```

Note: DSP mode is also called PCM mode which support MODE A and MODE B, DSP/PCM MODE A configuration flow. RX is similiar but uses `SAI_RxSetConfig` instead of `SAI_TxSetConfig`:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)  
config->frameSync.frameSyncEarly = true;  
SAI_TxSetConfig(base, config)
```

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to enable.

```
static inline uint32_t SAI_TxGetStatusFlag(I2S_Type *base)
```

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns

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

```
static inline void SAI_TxClearStatusFlags(I2S_Type *base, uint32_t mask)
```

Clears the SAI Tx status flag state.

Parameters

- base – SAI base pointer
- mask – State mask. It can be a combination of the following source if defined:
 - kSAI_WordStartFlag
 - kSAI_SyncErrorFlag
 - kSAI_FIFOErrorFlag

```
static inline uint32_t SAI_RxGetStatusFlag(I2S_Type *base)
```

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns

SAI Rx status flag value. Use the Status Mask to get the status value needed.

```
static inline void SAI_RxClearStatusFlags(I2S_Type *base, uint32_t mask)
```

Clears the SAI Rx status flag state.

Parameters

- base – SAI base pointer
- mask – State mask. It can be a combination of the following sources if defined.
 - kSAI_WordStartFlag
 - kSAI_SyncErrorFlag
 - kSAI_FIFOErrorFlag

```
void SAI_TxSoftwareReset(I2S_Type *base, sai_reset_type_t resetType)
```

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But

software reset will not clear any configuration registers like TCR1~TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

void SAI_RxSoftwareReset(I2S_Type *base, *sai_reset_type_t* resetType)

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1~RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

void SAI_TxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)

Set the Tx channel FIFO enable mask.

Parameters

- base – SAI base pointer
- mask – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

void SAI_RxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)

Set the Rx channel FIFO enable mask.

Parameters

- base – SAI base pointer
- mask – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

void SAI_TxSetDataOrder(I2S_Type *base, *sai_data_order_t* order)

Set the Tx data order.

Parameters

- base – SAI base pointer
- order – Data order MSB or LSB

void SAI_RxSetDataOrder(I2S_Type *base, *sai_data_order_t* order)

Set the Rx data order.

Parameters

- base – SAI base pointer
- order – Data order MSB or LSB

void SAI_TxSetBitClockPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Tx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_RxSetBitClockPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Rx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_TxSetFrameSyncPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Tx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_RxSetFrameSyncPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Rx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_TxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Tx FIFO packing feature.

Parameters

- base – SAI base pointer.
- pack – FIFO pack type. It is element of *sai_fifo_packing_t*.

void SAI_RxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Rx FIFO packing feature.

Parameters

- base – SAI base pointer.
- pack – FIFO pack type. It is element of *sai_fifo_packing_t*.

static inline void SAI_TxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Tx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI_RxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Rx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

```
static inline void SAI_TxEnableInterrupts(I2S_Type *base, uint32_t mask)
```

Enables the SAI Tx interrupt requests.

Parameters

- `base` – SAI base pointer
- `mask` – interrupt source The parameter can be a combination of the following sources if defined.
 - `kSAI_WordStartInterruptEnable`
 - `kSAI_SyncErrorInterruptEnable`
 - `kSAI_FIFOWarningInterruptEnable`
 - `kSAI_FIFORequestInterruptEnable`
 - `kSAI_FIFOErrorInterruptEnable`

```
static inline void SAI_RxEnableInterrupts(I2S_Type *base, uint32_t mask)
```

Enables the SAI Rx interrupt requests.

Parameters

- `base` – SAI base pointer
- `mask` – interrupt source The parameter can be a combination of the following sources if defined.
 - `kSAI_WordStartInterruptEnable`
 - `kSAI_SyncErrorInterruptEnable`
 - `kSAI_FIFOWarningInterruptEnable`
 - `kSAI_FIFORequestInterruptEnable`
 - `kSAI_FIFOErrorInterruptEnable`

```
static inline void SAI_TxDisableInterrupts(I2S_Type *base, uint32_t mask)
```

Disables the SAI Tx interrupt requests.

Parameters

- `base` – SAI base pointer
- `mask` – interrupt source The parameter can be a combination of the following sources if defined.
 - `kSAI_WordStartInterruptEnable`
 - `kSAI_SyncErrorInterruptEnable`
 - `kSAI_FIFOWarningInterruptEnable`
 - `kSAI_FIFORequestInterruptEnable`
 - `kSAI_FIFOErrorInterruptEnable`

```
static inline void SAI_RxDisableInterrupts(I2S_Type *base, uint32_t mask)
```

Disables the SAI Rx interrupt requests.

Parameters

- `base` – SAI base pointer
- `mask` – interrupt source The parameter can be a combination of the following sources if defined.
 - `kSAI_WordStartInterruptEnable`
 - `kSAI_SyncErrorInterruptEnable`

- kSAI_FIFOWarningInterruptEnable
- kSAI_FIFORequestInterruptEnable
- kSAI_FIFOErrorInterruptEnable

static inline void SAI_TxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)

Enables/disables the SAI Tx DMA requests.

Parameters

- base – SAI base pointer
- mask – DMA source The parameter can be combination of the following sources if defined.
 - kSAI_FIFOWarningDMAEnable
 - kSAI_FIFORequestDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline void SAI_RxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)

Enables/disables the SAI Rx DMA requests.

Parameters

- base – SAI base pointer
- mask – DMA source The parameter can be a combination of the following sources if defined.
 - kSAI_FIFOWarningDMAEnable
 - kSAI_FIFORequestDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline uintptr_t SAI_TxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)

Gets the SAI Tx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- base – SAI base pointer.
- channel – Which data channel used.

Returns

data register address.

static inline uintptr_t SAI_RxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)

Gets the SAI Rx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- base – SAI base pointer.
- channel – Which data channel used.

Returns

data register address.

void SAI_WriteBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

```
void SAI_WriteMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask,  
                                  uint32_t bitWidth, uint8_t *buffer, uint32_t size)
```

Sends data to multi channel using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

```
static inline void SAI_WriteData(I2S_Type *base, uint32_t channel, uint32_t data)
```

Writes data into SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- data – Data needs to be written.

```
void SAI_ReadBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer,  
                     uint32_t size)
```

Receives data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

```
void SAI_ReadMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask,  
                                uint32_t bitWidth, uint8_t *buffer, uint32_t size)
```

Receives multi channel data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

```
static inline uint32_t SAI_ReadData(I2S_Type *base, uint32_t channel)
```

Reads data from the SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.

Returns

Data in SAI FIFO.

```
void SAI_TransferTxCreateHandle(I2S_Type *base, sai_handle_t *handle, sai_transfer_callback_t  
                              callback, void *userData)
```

Initializes the SAI Tx handle.

This function initializes the Tx handle for the SAI Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SAI base pointer
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function

```
void SAI_TransferRxCreateHandle(I2S_Type *base, sai_handle_t *handle, sai_transfer_callback_t  
                              callback, void *userData)
```

Initializes the SAI Rx handle.

This function initializes the Rx handle for the SAI Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function.

```
void SAI_TransferTxSetConfig(I2S_Type *base, sai_handle_t *handle, sai_transceiver_t *config)
```

SAI transmitter transfer configurations.

This function initializes the Tx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- config – transmitter configurations.

```
void SAI_TransferRxSetConfig(I2S_Type *base, sai_handle_t *handle, sai_transceiver_t *config)
```

SAI receiver transfer configurations.

This function initializes the Rx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- config – receiver configurations.

```
status_t SAI_TransferSendNonBlocking(I2S_Type *base, sai_handle_t *handle, sai_transfer_t *xfer)
```

Performs an interrupt non-blocking send transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the `SAI_TxGetTransferStatusIRQ` to poll the transfer status and check whether the transfer is finished. If the return status is not `kStatus_SAI_Busy`, the transfer is finished.

Parameters

- base – SAI base pointer.
- handle – Pointer to the `sai_handle_t` structure which stores the transfer state.
- xfer – Pointer to the `sai_transfer_t` structure.

Return values

- `kStatus_Success` – Successfully started the data receive.
- `kStatus_SAI_TxBusy` – Previous receive still not finished.
- `kStatus_InvalidArgument` – The input parameter is invalid.

```
status_t SAI_TransferReceiveNonBlocking(I2S_Type *base, sai_handle_t *handle, sai_transfer_t *xfer)
```

Performs an interrupt non-blocking receive transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the `SAI_RxGetTransferStatusIRQ` to poll the transfer status and check whether the transfer is finished. If the return status is not `kStatus_SAI_Busy`, the transfer is finished.

Parameters

- base – SAI base pointer

- `handle` – Pointer to the `sai_handle_t` structure which stores the transfer state.
- `xfer` – Pointer to the `sai_transfer_t` structure.

Return values

- `kStatus_Success` – Successfully started the data receive.
- `kStatus_SAI_RxBusy` – Previous receive still not finished.
- `kStatus_InvalidArgument` – The input parameter is invalid.

`status_t` SAI_TransferGetSendCount(I2S_Type *base, *sai_handle_t* *handle, size_t *count)

Gets a set byte count.

Parameters

- `base` – SAI base pointer.
- `handle` – Pointer to the `sai_handle_t` structure which stores the transfer state.
- `count` – Bytes count sent.

Return values

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

`status_t` SAI_TransferGetReceiveCount(I2S_Type *base, *sai_handle_t* *handle, size_t *count)

Gets a received byte count.

Parameters

- `base` – SAI base pointer.
- `handle` – Pointer to the `sai_handle_t` structure which stores the transfer state.
- `count` – Bytes count received.

Return values

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

`void` SAI_TransferAbortSend(I2S_Type *base, *sai_handle_t* *handle)

Aborts the current send.

Note: This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- `base` – SAI base pointer.
- `handle` – Pointer to the `sai_handle_t` structure which stores the transfer state.

void SAI_TransferAbortReceive(I2S_Type *base, sai_handle_t *handle)

Aborts the current IRQ receive.

Note: This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – SAI base pointer
- handle – Pointer to the sai_handle_t structure which stores the transfer state.

void SAI_TransferTerminateSend(I2S_Type *base, sai_handle_t *handle)

Terminate all SAI send.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSend.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTerminateReceive(I2S_Type *base, sai_handle_t *handle)

Terminate all SAI receive.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceive.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTxHandleIRQ(I2S_Type *base, sai_handle_t *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure.

void SAI_TransferRxHandleIRQ(I2S_Type *base, sai_handle_t *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure.

void SAI_DriverIRQHandler(uint32_t instance)

SAI driver IRQ handler common entry.

This function provides the common IRQ request entry for SAI.

Parameters

- instance – SAI instance.

FSL_SAI_DRIVER_VERSION

Version 2.4.10

`_sai_status_t`, SAI return status.

Values:

enumerator `kStatus_SAI_TxBusy`

SAI Tx is busy.

enumerator `kStatus_SAI_RxBusy`

SAI Rx is busy.

enumerator `kStatus_SAI_TxError`

SAI Tx FIFO error.

enumerator `kStatus_SAI_RxError`

SAI Rx FIFO error.

enumerator `kStatus_SAI_QueueFull`

SAI transfer queue is full.

enumerator `kStatus_SAI_TxIdle`

SAI Tx is idle

enumerator `kStatus_SAI_RxIdle`

SAI Rx is idle

`_sai_channel_mask`, sai channel mask value, actual channel numbers is depend soc specific

Values:

enumerator `kSAI_Channel0Mask`

channel 0 mask value

enumerator `kSAI_Channel1Mask`

channel 1 mask value

enumerator `kSAI_Channel2Mask`

channel 2 mask value

enumerator `kSAI_Channel3Mask`

channel 3 mask value

enumerator `kSAI_Channel4Mask`

channel 4 mask value

enumerator `kSAI_Channel5Mask`

channel 5 mask value

enumerator `kSAI_Channel6Mask`

channel 6 mask value

enumerator `kSAI_Channel7Mask`

channel 7 mask value

enum `_sai_protocol`

Define the SAI bus type.

Values:

enumerator `kSAI_BusLeftJustified`

Uses left justified format.

enumerator kSAI_BusRightJustified

Uses right justified format.

enumerator kSAI_BusI2S

Uses I2S format.

enumerator kSAI_BusPCMA

Uses I2S PCM A format.

enumerator kSAI_BusPCMB

Uses I2S PCM B format.

enum _sai_master_slave

Master or slave mode.

Values:

enumerator kSAI_Master

Master mode include bclk and frame sync

enumerator kSAI_Slave

Slave mode include bclk and frame sync

enumerator kSAI_Bclk_Master_FrameSync_Slave

bclk in master mode, frame sync in slave mode

enumerator kSAI_Bclk_Slave_FrameSync_Master

bclk in slave mode, frame sync in master mode

enum _sai_mono_stereo

Mono or stereo audio format.

Values:

enumerator kSAI_Stereo

Stereo sound.

enumerator kSAI_MonoRight

Only Right channel have sound.

enumerator kSAI_MonoLeft

Only left channel have sound.

enum _sai_data_order

SAI data order, MSB or LSB.

Values:

enumerator kSAI_DataLSB

LSB bit transferred first

enumerator kSAI_DataMSB

MSB bit transferred first

enum _sai_clock_polarity

SAI clock polarity, active high or low.

Values:

enumerator kSAI_PolarityActiveHigh

Drive outputs on rising edge

enumerator kSAI_PolarityActiveLow

Drive outputs on falling edge

enumerator kSAI_SampleOnFallingEdge

Sample inputs on falling edge

enumerator kSAI_SampleOnRisingEdge

Sample inputs on rising edge

enum __sai_sync_mode

Synchronous or asynchronous mode.

Values:

enumerator kSAI_ModeAsync

Asynchronous mode

enumerator kSAI_ModeSync

Synchronous mode (with receiver or transmit)

enumerator kSAI_ModeSyncWithOtherTx

Synchronous with another SAI transmit

enumerator kSAI_ModeSyncWithOtherRx

Synchronous with another SAI receiver

enum __sai_bclk_source

Bit clock source.

Values:

enumerator kSAI_BclkSourceBusclk

Bit clock using bus clock

enumerator kSAI_BclkSourceMclkOption1

Bit clock MCLK option 1

enumerator kSAI_BclkSourceMclkOption2

Bit clock MCLK option2

enumerator kSAI_BclkSourceMclkOption3

Bit clock MCLK option3

enumerator kSAI_BclkSourceMclkDiv

Bit clock using master clock divider

enumerator kSAI_BclkSourceOtherSai0

Bit clock from other SAI device

enumerator kSAI_BclkSourceOtherSai1

Bit clock from other SAI device

__sai_interrupt_enable_t, The SAI interrupt enable flag

Values:

enumerator kSAI_WordStartInterruptEnable

Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorInterruptEnable

Sync error flag, means the sync error is detected

enumerator kSAI_FIFOWarningInterruptEnable

FIFO warning flag, means the FIFO is empty

enumerator kSAI_FIFOErrorInterruptEnable
FIFO error flag

enumerator kSAI_FIFORequestInterruptEnable
FIFO request, means reached watermark

_sai_dma_enable_t, The DMA request sources

Values:

enumerator kSAI_FIFOWarningDMAEnable
FIFO warning caused by the DMA request

enumerator kSAI_FIFORequestDMAEnable
FIFO request caused by the DMA request

_sai_flags, The SAI status flag

Values:

enumerator kSAI_WordStartFlag
Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorFlag
Sync error flag, means the sync error is detected

enumerator kSAI_FIFOErrorFlag
FIFO error flag

enumerator kSAI_FIFORequestFlag
FIFO request flag.

enumerator kSAI_FIFOWarningFlag
FIFO warning flag

enum _sai_reset_type

The reset type.

Values:

enumerator kSAI_ResetTypeSoftware
Software reset, reset the logic state

enumerator kSAI_ResetTypeFIFO
FIFO reset, reset the FIFO read and write pointer

enumerator kSAI_ResetAll
All reset.

enum _sai_fifo_packing

The SAI packing mode The mode includes 8 bit and 16 bit packing.

Values:

enumerator kSAI_FifoPackingDisabled
Packing disabled

enumerator kSAI_FifoPacking8bit
8 bit packing enabled

enumerator kSAI_FifoPacking16bit
16bit packing enabled

enum _sai_sample_rate

Audio sample rate.

Values:

enumerator kSAI_SampleRate8KHz

Sample rate 8000 Hz

enumerator kSAI_SampleRate11025Hz

Sample rate 11025 Hz

enumerator kSAI_SampleRate12KHz

Sample rate 12000 Hz

enumerator kSAI_SampleRate16KHz

Sample rate 16000 Hz

enumerator kSAI_SampleRate22050Hz

Sample rate 22050 Hz

enumerator kSAI_SampleRate24KHz

Sample rate 24000 Hz

enumerator kSAI_SampleRate32KHz

Sample rate 32000 Hz

enumerator kSAI_SampleRate44100Hz

Sample rate 44100 Hz

enumerator kSAI_SampleRate48KHz

Sample rate 48000 Hz

enumerator kSAI_SampleRate96KHz

Sample rate 96000 Hz

enumerator kSAI_SampleRate192KHz

Sample rate 192000 Hz

enumerator kSAI_SampleRate384KHz

Sample rate 384000 Hz

enum _sai_word_width

Audio word width.

Values:

enumerator kSAI_WordWidth8bits

Audio data width 8 bits

enumerator kSAI_WordWidth16bits

Audio data width 16 bits

enumerator kSAI_WordWidth24bits

Audio data width 24 bits

enumerator kSAI_WordWidth32bits

Audio data width 32 bits

enum _sai_data_pin_state

sai data pin state definition

Values:

enumerator kSAI_DataPinStateTriState

transmit data pins are tri-stated when slots are masked or channels are disabled

enumerator kSAI_DataPinStateOutputZero

transmit data pins are never tri-stated and will output zero when slots are masked or channel disabled

enum _sai_fifo_combine

sai fifo combine mode definition

Values:

enumerator kSAI_FifoCombineDisabled

sai TX/RX fifo combine mode disabled

enumerator kSAI_FifoCombineModeEnabledOnRead

sai TX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnWrite

sai TX fifo combine mode enabled on FIFO write

enumerator kSAI_RxFifoCombineModeEnabledOnWrite

sai RX fifo combine mode enabled on FIFO write

enumerator kSAI_RXFifoCombineModeEnabledOnRead

sai RX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnReadWrite

sai TX/RX fifo combined mode enabled on FIFO read/writes

enum _sai_transceiver_type

sai transceiver type

Values:

enumerator kSAI_Transmitter

sai transmitter

enumerator kSAI_Receiver

sai receiver

enum _sai_frame_sync_len

sai frame sync len

Values:

enumerator kSAI_FrameSyncLenOneBitClk

1 bit clock frame sync len for DSP mode

enumerator kSAI_FrameSyncLenPerWordWidth

Frame sync length decided by word width

typedef enum _sai_protocol sai_protocol_t

Define the SAI bus type.

typedef enum _sai_master_slave sai_master_slave_t

Master or slave mode.

typedef enum _sai_mono_stereo sai_mono_stereo_t

Mono or stereo audio format.

typedef enum _sai_data_order sai_data_order_t

SAI data order, MSB or LSB.

`typedef enum _sai_clock_polarity sai_clock_polarity_t`
SAI clock polarity, active high or low.

`typedef enum _sai_sync_mode sai_sync_mode_t`
Synchronous or asynchronous mode.

`typedef enum _sai_bclk_source sai_bclk_source_t`
Bit clock source.

`typedef enum _sai_reset_type sai_reset_type_t`
The reset type.

`typedef enum _sai_fifo_packing sai_fifo_packing_t`
The SAI packing mode The mode includes 8 bit and 16 bit packing.

`typedef struct _sai_config sai_config_t`
SAI user configuration structure.

`typedef enum _sai_sample_rate sai_sample_rate_t`
Audio sample rate.

`typedef enum _sai_word_width sai_word_width_t`
Audio word width.

`typedef enum _sai_data_pin_state sai_data_pin_state_t`
sai data pin state definition

`typedef enum _sai_fifo_combine sai_fifo_combine_t`
sai fifo combine mode definition

`typedef enum _sai_transceiver_type sai_transceiver_type_t`
sai transceiver type

`typedef enum _sai_frame_sync_len sai_frame_sync_len_t`
sai frame sync len

`typedef struct _sai_transfer_format sai_transfer_format_t`
sai transfer format

`typedef struct _sai_master_clock sai_master_clock_t`
master clock configurations

`typedef struct _sai_fifo sai_fifo_t`
sai fifo configurations

`typedef struct _sai_bit_clock sai_bit_clock_t`
sai bit clock configurations

`typedef struct _sai_frame_sync sai_frame_sync_t`
sai frame sync configurations

`typedef struct _sai_serial_data sai_serial_data_t`
sai serial data configurations

`typedef struct _sai_transceiver sai_transceiver_t`
sai transceiver configurations

`typedef struct _sai_transfer sai_transfer_t`
SAI transfer structure.

`typedef struct _sai_handle sai_handle_t`


```
typedef void (*sai_transfer_callback_t)(I2S_Type *base, sai_handle_t *handle, status_t status, void *userData)
```

SAI transfer callback prototype.

MCUX_SDK_SAI_ALLOW_NULL_FIFO_WATERMARK

Used to control whether SAI_RxSetFifoConfig()/SAI_TxSetFifoConfig() allows a NULL FIFO watermark.

If this macro is set to 0 then SAI_RxSetFifoConfig()/SAI_TxSetFifoConfig() will set the watermark to half of the FIFO's depth if passed a NULL watermark.

MCUX_SDK_SAI_DISABLE_IMPLICIT_CHAN_CONFIG

Disable implicit channel data configuration within SAI_TxSetConfig()/SAI_RxSetConfig().

Use this macro to control whether SAI_RxSetConfig()/SAI_TxSetConfig() will attempt to implicitly configure the channel data. By channel data we mean the startChannel, channelMask, endChannel, and channelNums fields from the sai_transciever_t structure. By default, SAI_TxSetConfig()/SAI_RxSetConfig() will attempt to compute these fields, which may not be desired in cases where the user wants to set them before the call to said functions.

SAI_XFER_QUEUE_SIZE

SAI transfer queue size, user can refine it according to use case.

FSL_SAI_HAS_FIFO_EXTEND_FEATURE

sai fifo feature

```
struct __sai_config
```

#include <fsl_sai.h> SAI user configuration structure.

Public Members

sai_protocol_t protocol

Audio bus protocol in SAI

sai_sync_mode_t syncMode

SAI sync mode, control Tx/Rx clock sync

bool mclkOutputEnable

Master clock output enable, true means master clock divider enabled

sai_bclk_source_t bclkSource

Bit Clock source

sai_master_slave_t masterSlave

Master or slave

```
struct __sai_transfer_format
```

#include <fsl_sai.h> sai transfer format

Public Members

uint32_t sampleRate_Hz

Sample rate of audio data

uint32_t bitWidth

Data length of audio data, usually 8/16/24/32 bits

sai_mono_stereo_t stereo

Mono or stereo

uint32_t masterClockHz
Master clock frequency in Hz

uint8_t watermark
Watermark value

uint8_t channel
Transfer start channel

uint8_t channelMask
enabled channel mask value, reference `_sai_channel_mask`

uint8_t endChannel
end channel number

uint8_t channelNums
Total enabled channel numbers

sai_protocol_t protocol
Which audio protocol used

bool isFrameSyncCompact
True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.

struct _sai_master_clock
#include <fsl_sai.h> master clock configurations

Public Members

bool mclkOutputEnable
master clock output enable

uint32_t mclkHz
target mclk frequency

uint32_t mclkSourceClkHz
mclk source frequency

struct _sai_fifo
#include <fsl_sai.h> sai fifo configurations

Public Members

bool fifoContinueOnError
fifo continues when error occur

sai_fifo_combine_t fifoCombine
fifo combine mode

sai_fifo_packing_t fifoPacking
fifo packing mode

uint8_t fifoWatermark
fifo watermark

struct _sai_bit_clock
#include <fsl_sai.h> sai bit clock configurations

Public Members

bool bclkInputDelay
bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time
.

sai_clock_polarity_t bclkPolarity
bit clock polarity

sai_bclk_source_t bclkSource
bit Clock source

struct _sai_frame_sync
#include <fsl_sai.h> sai frame sync configurations

Public Members

uint8_t frameSyncWidth
frame sync width in number of bit clocks

bool frameSyncEarly
TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame

bool frameSyncGenerateOnDemand
internal frame sync is generated when FIFO waring flag is clear

sai_clock_polarity_t frameSyncPolarity
frame sync polarity

struct _sai_serial_data
#include <fsl_sai.h> sai serial data configurations

Public Members

sai_data_pin_state_t dataMode
sai data pin state when slots masked or channel disabled

sai_data_order_t dataOrder
configure whether the LSB or MSB is transmitted first

uint8_t dataWord0Length
configure the number of bits in the first word in each frame

uint8_t dataWordNLength
configure the number of bits in the each word in each frame, except the first word

uint8_t dataWordLength
used to record the data length for dma transfer

uint8_t dataFirstBitShifted
Configure the bit index for the first bit transmitted for each word in the frame

uint8_t dataWordNum
configure the number of words in each frame

uint32_t dataMaskedWord
configure whether the transmit word is masked

struct _sai_transceiver
#include <fsl_sai.h> sai transceiver configurations

Public Members

sai_serial_data_t serialData
serial data configurations

sai_frame_sync_t frameSync
ws configurations

sai_bit_clock_t bitClock
bit clock configurations

sai_fifo_t fifo
fifo configurations

sai_master_slave_t masterSlave
transceiver is master or slave

sai_sync_mode_t syncMode
transceiver sync mode

uint8_t startChannel
Transfer start channel

uint8_t channelMask
enabled channel mask value, reference `_sai_channel_mask`

uint8_t endChannel
end channel number

uint8_t channelNums
Total enabled channel numbers

struct `_sai_transfer`
#include <fsl_sai.h> SAI transfer structure.

Public Members

uint8_t *data
Data start address to transfer.

size_t dataSize
Transfer size.

struct `_sai_handle`
#include <fsl_sai.h> SAI handle structure.

Public Members

I2S_Type *base
base address

uint32_t state
Transfer status

sai_transfer_callback_t callback
Callback function called at transfer event

void *userData
Callback parameter passed to callback function

`uint8_t bitWidth`
Bit width for transfer, 8/16/24/32 bits

`uint8_t channel`
Transfer start channel

`uint8_t channelMask`
enabled channel mask value, refernece `_sai_channel_mask`

`uint8_t endChannel`
end channel number

`uint8_t channelNums`
Total enabled channel numbers

`sai_transfer_t saiQueue[(4U)]`
Transfer queue storing queued transfer

`size_t transferSize[(4U)]`
Data bytes need to transfer

`volatile uint8_t queueUser`
Index for user to queue transfer

`volatile uint8_t queueDriver`
Index for driver to get the transfer data and size

`uint8_t watermark`
Watermark value

2.98 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

2.99 SEMA4: Hardware Semaphores Driver

FSL_SEMA4_DRIVER_VERSION

SEMA4 driver version.

void SEMA4_Init(SEMA4_Type *base)

Initializes the SEMA4 module.

This function initializes the SEMA4 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either SEMA4_ResetGate or SEMA4_ResetAllGates function.

Parameters

- base – SEMA4 peripheral base address.

void SEMA4_Deinit(SEMA4_Type *base)

De-initializes the SEMA4 module.

This function de-initializes the SEMA4 module. It only disables the clock.

Parameters

- base – SEMA4 peripheral base address.

status_t SEMA4_TryLock(SEMA4_Type *base, uint8_t gateNum, uint8_t procNum)

Tries to lock the SEMA4 gate.

This function tries to lock the specific SEMA4 gate. If the gate has been locked by another processor, this function returns an error code.

Parameters

- base – SEMA4 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

Return values

- kStatus_Success – Lock the sema4 gate successfully.
- kStatus_Fail – Sema4 gate has been locked by another processor.

status_t SEMA4_Lock(SEMA4_Type *base, uint8_t gateNum, uint8_t procNum)

Locks the SEMA4 gate.

This function locks the specific SEMA4 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

If SEMA4_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

Parameters

- base – SEMA4 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

Return values

- `kStatus__Success` – The gate was successfully locked.
- `kStatus__Timeout` – Timeout occurred while waiting for the gate to be unlocked.

Returns

`status_t`

`static inline void SEMA4_Unlock(SEMA4_Type *base, uint8_t gateNum)`

Unlocks the SEMA4 gate.

This function unlocks the specific SEMA4 gate. It only writes unlock value to the SEMA4 gate register. However, it does not check whether the SEMA4 gate is locked by the current processor or not. As a result, if the SEMA4 gate is not locked by the current processor, this function has no effect.

Parameters

- `base` – SEMA4 peripheral base address.
- `gateNum` – Gate number to unlock.

`static inline int32_t SEMA4_GetLockProc(SEMA4_Type *base, uint8_t gateNum)`

Gets the status of the SEMA4 gate.

This function checks the lock status of a specific SEMA4 gate.

Parameters

- `base` – SEMA4 peripheral base address.
- `gateNum` – Gate number.

Returns

Return -1 if the gate is unlocked, otherwise return the processor number which has locked the gate.

`status_t SEMA4_ResetGate(SEMA4_Type *base, uint8_t gateNum)`

Resets the SEMA4 gate to an unlocked status.

This function resets a SEMA4 gate to an unlocked status.

Parameters

- `base` – SEMA4 peripheral base address.
- `gateNum` – Gate number.

Return values

- `kStatus__Success` – SEMA4 gate is reset successfully.
- `kStatus__Fail` – Some other reset process is ongoing.

`static inline status_t SEMA4_ResetAllGates(SEMA4_Type *base)`

Resets all SEMA4 gates to an unlocked status.

This function resets all SEMA4 gate to an unlocked status.

Parameters

- `base` – SEMA4 peripheral base address.

Return values

- `kStatus__Success` – SEMA4 is reset successfully.
- `kStatus__Fail` – Some other reset process is ongoing.

```
static inline void SEMA4_EnableGateNotifyInterrupt(SEMA4_Type *base, uint8_t procNum,  
                                                  uint16_t mask)
```

Enable the gate notification interrupt.

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle.

Parameters

- base – SEMA4 peripheral base address.
- procNum – Current processor number.
- mask – OR'ed value of the gate index, for example: (1«0) | (1«1) means gate 0 and gate 1.

```
static inline void SEMA4_DisableGateNotifyInterrupt(SEMA4_Type *base, uint8_t procNum,  
                                                  uint16_t mask)
```

Disable the gate notification interrupt.

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle.

Parameters

- base – SEMA4 peripheral base address.
- procNum – Current processor number.
- mask – OR'ed value of the gate index, for example: (1«0) | (1«1) means gate 0 and gate 1.

```
static inline uint32_t SEMA4_GetGateNotifyStatus(SEMA4_Type *base, uint8_t procNum)
```

Get the gate notification flags.

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle. The status flags are cleared automatically when the gate is locked by current core or locked again before the other core.

Parameters

- base – SEMA4 peripheral base address.
- procNum – Current processor number.

Returns

OR'ed value of the gate index, for example: (1«0) | (1«1) means gate 0 and gate 1 flags are pending.

```
status_t SEMA4_ResetGateNotify(SEMA4_Type *base, uint8_t gateNum)
```

Resets the SEMA4 gate IRQ notification.

This function resets a SEMA4 gate IRQ notification.

Parameters

- base – SEMA4 peripheral base address.
- gateNum – Gate number.

Return values

- kStatus_Success – Reset successfully.
- kStatus_Fail – Some other reset process is ongoing.

```
static inline status_t SEMA4_ResetAllGateNotify(SEMA4_Type *base)
```

Resets all SEMA4 gates IRQ notification.

This function resets all SEMA4 gate IRQ notifications.

Parameters

- base – SEMA4 peripheral base address.

Return values

- kStatus_Success – Reset successfully.
- kStatus_Fail – Some other reset process is ongoing.

SEMA4_GATE_NUM_RESET_ALL

The number to reset all SEMA4 gates.

SEMA4_GATE_n(base, n)

SEMA4 gate n register address.

SEMA4_BUSY_POLL_COUNT

Maximum polling iterations for SEMA4 waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the SEMA4 driver code before timing out and returning an error.

It applies to all waiting loops in SEMA4 driver, such as waiting for a gate to be unlocked, waiting for a reset to complete, or waiting for a resource to become available.

This is a count of loop iterations, not a time-based value.

If defined as 0, polling loops will continue indefinitely until their exit condition is met, which could potentially cause the system to hang if hardware doesn't respond or if a resource is never released.

2.100 SEMC: Smart External DRAM Controller Driver

void SEMC_GetDefaultConfig(*semc_config_t* *config)

Gets the SEMC default basic configuration structure.

The purpose of this API is to get the default SEMC configure structure for SEMC_Init(). User may use the initialized structure unchanged in SEMC_Init(), or modify some fields of the structure before calling SEMC_Init(). Example:

```
semc_config_t config;  
SEMC_GetDefaultConfig(&config);
```

Parameters

- config – The SEMC configuration structure pointer.

void SEMC_Init(SEMC_Type *base, *semc_config_t* *configure)

Initializes SEMC. This function ungates the SEMC clock and initializes SEMC. This function must be called before calling any other SEMC driver functions.

Parameters

- base – SEMC peripheral base address.
- configure – The SEMC configuration structure pointer.

void SEMC_Deinit(SEMC_Type *base)

Deinitializes the SEMC module and gates the clock.

This function gates the SEMC clock. As a result, the SEMC module doesn't work after calling this function, for some IDE, calling this API may cause the next downloading operation failed. so, please call this API cautiously. Additional, users can using "#define

FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL (1)” to disable the clock control operation in drivers.

Parameters

- base – SEMC peripheral base address.

status_t SEMC_ConfigureSDRAM(SEMC_Type *base, *semc_sdram_cs_t* cs, *semc_sdram_config_t* *config, uint32_t clkSrc_Hz)

Configures SDRAM controller in SEMC.

Parameters

- base – SEMC peripheral base address.
- cs – The chip selection.
- config – The sdram configuration.
- clkSrc_Hz – The SEMC clock frequency.

status_t SEMC_ConfigureNAND(SEMC_Type *base, *semc_nand_config_t* *config, uint32_t clkSrc_Hz)

Configures NAND controller in SEMC.

Parameters

- base – SEMC peripheral base address.
- config – The nand configuration.
- clkSrc_Hz – The SEMC clock frequency.

status_t SEMC_ConfigureNOR(SEMC_Type *base, *semc_nor_config_t* *config, uint32_t clkSrc_Hz)

Configures NOR controller in SEMC.

Parameters

- base – SEMC peripheral base address.
- config – The nor configuration.
- clkSrc_Hz – The SEMC clock frequency.

status_t SEMC_ConfigureSRAMWithChipSelection(SEMC_Type *base, *semc_sram_cs_t* cs, *semc_sram_config_t* *config, uint32_t clkSrc_Hz)

Configures SRAM controller in SEMC.

Parameters

- base – SEMC peripheral base address.
- cs – The chip selection.
- config – The sram configuration.
- clkSrc_Hz – The SEMC clock frequency.

status_t SEMC_ConfigureSRAM(SEMC_Type *base, *semc_sram_config_t* *config, uint32_t clkSrc_Hz)

Configures SRAM controller in SEMC.

Deprecated:

Do not use this function. It has been superceded by SEMC_ConfigureSRAMWithChipSelection.

Parameters

- `base` – SEMC peripheral base address.
- `config` – The sram configuration.
- `clkSrc_Hz` – The SEMC clock frequency.

`status_t SEMC_ConfigureDBI(SEMC_Type *base, semc_dbi_config_t *config, uint32_t clkSrc_Hz)`
Configures DBI controller in SEMC.

Parameters

- `base` – SEMC peripheral base address.
- `config` – The dbi configuration.
- `clkSrc_Hz` – The SEMC clock frequency.

`static inline void SEMC_EnableInterrupts(SEMC_Type *base, uint32_t mask)`

Enables the SEMC interrupt.

This function enables the SEMC interrupts according to the provided mask. The mask is a logical OR of enumeration members. See `semc_interrupt_enable_t`. For example, to enable the IP command done and error interrupt, do the following.

```
SEMC_EnableInterrupts(ENET, kSEMC_IPCmdDoneInterrupt | kSEMC_IPCmdErrInterrupt);
```

Parameters

- `base` – SEMC peripheral base address.
- `mask` – SEMC interrupts to enable. This is a logical OR of the enumeration
:: `semc_interrupt_enable_t`.

`static inline void SEMC_DisableInterrupts(SEMC_Type *base, uint32_t mask)`

Disables the SEMC interrupt.

This function disables the SEMC interrupts according to the provided mask. The mask is a logical OR of enumeration members. See `semc_interrupt_enable_t`. For example, to disable the IP command done and error interrupt, do the following.

```
SEMC_DisableInterrupts(ENET, kSEMC_IPCmdDoneInterrupt | kSEMC_IPCmdErrInterrupt);
```

Parameters

- `base` – SEMC peripheral base address.
- `mask` – SEMC interrupts to disable. This is a logical OR of the enumeration
:: `semc_interrupt_enable_t`.

`static inline bool SEMC_GetStatusFlag(SEMC_Type *base)`

Gets the SEMC status.

This function gets the SEMC interrupts event status. User can use the a logical OR of enumeration member as a mask. See `semc_interrupt_enable_t`.

Parameters

- `base` – SEMC peripheral base address.

Returns

status flag, use status flag in `semc_interrupt_enable_t` to get the related status.

`static inline void SEMC_ClearStatusFlags(SEMC_Type *base, uint32_t mask)`

Clears the SEMC status flag state.

The following status register flags can be cleared SEMC interrupt status.

Parameters

- base – SEMC base pointer
- mask – The status flag mask, a logical OR of enumeration member `semc_interrupt_enable_t`.

static inline bool SEMC_IsInIdle(SEMC_Type *base)

Check if SEMC is in idle.

Parameters

- base – SEMC peripheral base address.

Returns

True SEMC is in idle, false is not in idle.

status_t SEMC_SendIPCommand(SEMC_Type *base, *semc_mem_type_t* memType, uint32_t address, uint32_t command, uint32_t write, uint32_t *read)

SEMC IP command access.

Parameters

- base – SEMC peripheral base address.
- memType – SEMC memory type. refer to “`semc_mem_type_t`”
- address – SEMC device address.
- command – SEMC IP command. For NAND device, we should use the `SEMC_BuildNandIPCommand` to get the right nand command. For NOR/DBI device, take refer to “`semc_ipcmd_nor_dbi_t`”. For SRAM device, take refer to “`semc_ipcmd_sram_t`”. For SDRAM device, take refer to “`semc_ipcmd_sdram_t`”.
- write – Data for write access.
- read – Data pointer for read data out.

static inline uint16_t SEMC_BuildNandIPCommand(uint8_t userCommand, *semc_ipcmd_nand_addrmode_t* addrMode, *semc_ipcmd_nand_cmdmode_t* cmdMode)

Build SEMC IP command for NAND.

This function build SEMC NAND IP command. The command is build of user command code, SEMC address mode and SEMC command mode.

Parameters

- userCommand – NAND device normal command.
- addrMode – NAND address mode. Refer to “`semc_ipcmd_nand_addrmode_t`”.
- cmdMode – NAND command mode. Refer to “`semc_ipcmd_nand_cmdmode_t`”.

static inline bool SEMC_IsNandReady(SEMC_Type *base)

Check if the NAND device is ready.

Parameters

- base – SEMC peripheral base address.

Returns

True NAND is ready, false NAND is not ready.

status_t SEMC_IPCommandNandWrite(SEMC_Type *base, uint32_t address, uint8_t *data, uint32_t size_bytes)

SEMC NAND device memory write through IP command.

Parameters

- base – SEMC peripheral base address.
- address – SEMC NAND device address.
- data – Data for write access.
- size_bytes – Data length.

status_t SEMC_IPCommandNandRead(SEMC_Type *base, uint32_t address, uint8_t *data, uint32_t size_bytes)

SEMC NAND device memory read through IP command.

Parameters

- base – SEMC peripheral base address.
- address – SEMC NAND device address.
- data – Data pointer for data read out.
- size_bytes – Data length.

status_t SEMC_IPCommandNorWrite(SEMC_Type *base, uint32_t address, uint8_t *data, uint32_t size_bytes)

SEMC NOR device memory write through IP command.

Parameters

- base – SEMC peripheral base address.
- address – SEMC NOR device address.
- data – Data for write access.
- size_bytes – Data length.

status_t SEMC_IPCommandNorRead(SEMC_Type *base, uint32_t address, uint8_t *data, uint32_t size_bytes)

SEMC NOR device memory read through IP command.

Parameters

- base – SEMC peripheral base address.
- address – SEMC NOR device address.
- data – Data pointer for data read out.
- size_bytes – Data length.

FSL_SEMC_DRIVER_VERSION

SEMC driver version.

SEMC status, *_semc_status*.

Values:

enumerator *kStatus_SEMC_InvalidDeviceType*
Invalid device type.

enumerator *kStatus_SEMC_IpCommandExecutionError*
IP command execution error.

enumerator *kStatus_SEMC_AxiCommandExecutionError*
AXI command execution error.

enumerator kStatus_SEMC_InvalidMemorySize

Invalid memory size.

enumerator kStatus_SEMC_InvalidIpcmdDataSize

Invalid IP command data size.

enumerator kStatus_SEMC_InvalidAddressPortWidth

Invalid address port width.

enumerator kStatus_SEMC_InvalidDataPortWidth

Invalid data port width.

enumerator kStatus_SEMC_InvalidSwPinmuxSelection

Invalid SW pinmux selection.

enumerator kStatus_SEMC_InvalidBurstLength

Invalid burst length

enumerator kStatus_SEMC_InvalidColumnAddressBitWidth

Invalid column address bit width.

enumerator kStatus_SEMC_InvalidBaseAddress

Invalid base address.

enumerator kStatus_SEMC_InvalidTimerSetting

Invalid timer setting.

enum _semc_mem_type

SEMC memory device type.

Values:

enumerator kSEMC_MemType_SDRAM

SDRAM

enumerator kSEMC_MemType_SRAM

SRAM

enumerator kSEMC_MemType_NOR

NOR

enumerator kSEMC_MemType_NAND

NAND

enumerator kSEMC_MemType_8080

i.

enum _semc_waitready_polarity

SEMC WAIT/RDY polarity.

Values:

enumerator kSEMC_LowActive

Low active.

enumerator kSEMC_HighActive

High active.

enum _semc_sdram_cs

SEMC SDRAM Chip selection .

Values:

enumerator kSEMC_SDRAM_CS0
SEMC SDRAM CS0.

enumerator kSEMC_SDRAM_CS1
SEMC SDRAM CS1.

enumerator kSEMC_SDRAM_CS2
SEMC SDRAM CS2.

enumerator kSEMC_SDRAM_CS3
SEMC SDRAM CS3.

enum _semc_sram_cs
SEMC SRAM Chip selection .

Values:

enumerator kSEMC_SRAM_CS0
SEMC SRAM CS0.

enum _semc_nand_access_type
SEMC NAND device type.

Values:

enumerator kSEMC_NAND_ACCESS_BY_AXI
Access to NAND flash by AXI bus.

enumerator kSEMC_NAND_ACCESS_BY_IPCMD
Access to NAND flash by IP bus.

enum _semc_interrupt_enable
SEMC interrupts .

Values:

enumerator kSEMC_IPCmdDoneInterrupt
Ip command done interrupt.

enumerator kSEMC_IPCmdErrInterrupt
Ip command error interrupt.

enumerator kSEMC_AXICmdErrInterrupt
AXI command error interrupt.

enumerator kSEMC_AXIBusErrInterrupt
AXI bus error interrupt.

enum _semc_ipcmd_datasize
SEMC IP command data size in bytes.

Values:

enumerator kSEMC_IPcmdDataSize_1bytes
The IP command data size 1 byte.

enumerator kSEMC_IPcmdDataSize_2bytes
The IP command data size 2 byte.

enumerator kSEMC_IPcmdDataSize_3bytes
The IP command data size 3 byte.

enumerator kSEMC_IPcmdDataSize_4bytes
The IP command data size 4 byte.

enum _semc_refresh_time

SEMC auto-refresh timing.

Values:

enumerator kSEMC_RefreshThreeClocks

The refresh timing with three bus clocks.

enumerator kSEMC_RefreshSixClocks

The refresh timing with six bus clocks.

enumerator kSEMC_RefreshNineClocks

The refresh timing with nine bus clocks.

enum _semc_caslatency

CAS latency.

Values:

enumerator kSEMC_LatencyOne

Latency 1.

enumerator kSEMC_LatencyTwo

Latency 2.

enumerator kSEMC_LatencyThree

Latency 3.

enum _semc_sdram_column_bit_num

SEMC sdram column address bit number.

Values:

enumerator kSEMC_SdramColumn_12bit

12 bit.

enumerator kSEMC_SdramColumn_11bit

11 bit.

enumerator kSEMC_SdramColumn_10bit

10 bit.

enumerator kSEMC_SdramColumn_9bit

9 bit.

enumerator kSEMC_SdramColumn_8bit

8 bit.

enum _semc_sdram_burst_len

SEMC sdram burst length.

Values:

enumerator kSEMC_Sdram_BurstLen1

According to ERR050577, Auto-refresh command may possibly fail to be triggered during long time back-to-back write (or read) when SDRAM controller's burst length is greater than 1. Burst length 1

enum _semc_nand_column_bit_num

SEMC nand column address bit number.

Values:

enumerator kSEMC_NandColumn_16bit

16 bit.

enumerator kSEMC_NandColum_15bit
15 bit.

enumerator kSEMC_NandColum_14bit
14 bit.

enumerator kSEMC_NandColum_13bit
13 bit.

enumerator kSEMC_NandColum_12bit
12 bit.

enumerator kSEMC_NandColum_11bit
11 bit.

enumerator kSEMC_NandColum_10bit
10 bit.

enumerator kSEMC_NandColum_9bit
9 bit.

enum _semc_nand_burst_len
SEMC nand burst length.

Values:

enumerator kSEMC_Nand_BurstLen1
Burst length 1

enumerator kSEMC_Nand_BurstLen2
Burst length 2

enumerator kSEMC_Nand_BurstLen4
Burst length 4

enumerator kSEMC_Nand_BurstLen8
Burst length 8

enumerator kSEMC_Nand_BurstLen16
Burst length 16

enumerator kSEMC_Nand_BurstLen32
Burst length 32

enumerator kSEMC_Nand_BurstLen64
Burst length 64

enum _semc_norsram_column_bit_num
SEMC nor/sram column address bit number.

Values:

enumerator kSEMC_NorColum_12bit
12 bit.

enumerator kSEMC_NorColum_11bit
11 bit.

enumerator kSEMC_NorColum_10bit
10 bit.

enumerator kSEMC_NorColum_9bit
9 bit.

enumerator kSEMC_NorColum_8bit
8 bit.

enumerator kSEMC_NorColum_7bit
7 bit.

enumerator kSEMC_NorColum_6bit
6 bit.

enumerator kSEMC_NorColum_5bit
5 bit.

enumerator kSEMC_NorColum_4bit
4 bit.

enumerator kSEMC_NorColum_3bit
3 bit.

enumerator kSEMC_NorColum_2bit
2 bit.

enum _semc_norsram_burst_len
SEMC nor/sram burst length.

Values:

enumerator kSEMC_Nor_BurstLen1
Burst length 1

enumerator kSEMC_Nor_BurstLen2
Burst length 2

enumerator kSEMC_Nor_BurstLen4
Burst length 4

enumerator kSEMC_Nor_BurstLen8
Burst length 8

enumerator kSEMC_Nor_BurstLen16
Burst length 16

enumerator kSEMC_Nor_BurstLen32
Burst length 32

enumerator kSEMC_Nor_BurstLen64
Burst length 64

enum _semc_dbi_column_bit_num
SEMC dbi column address bit number.

Values:

enumerator kSEMC_Dbi_Colum_12bit
12 bit.

enumerator kSEMC_Dbi_Colum_11bit
11 bit.

enumerator kSEMC_Dbi_Colum_10bit
10 bit.

enumerator kSEMC_Dbi_Colum_9bit
9 bit.

enumerator kSEMC_Dbi_Colum_8bit
8 bit.

enumerator kSEMC_Dbi_Colum_7bit
7 bit.

enumerator kSEMC_Dbi_Colum_6bit
6 bit.

enumerator kSEMC_Dbi_Colum_5bit
5 bit.

enumerator kSEMC_Dbi_Colum_4bit
4 bit.

enumerator kSEMC_Dbi_Colum_3bit
3 bit.

enumerator kSEMC_Dbi_Colum_2bit
2 bit.

enum _semc_dbi_burst_len
SEMC dbi burst length.

Values:

enumerator kSEMC_Dbi_BurstLen1
Burst length 1

enumerator kSEMC_Dbi_BurstLen2
Burst length 2

enumerator kSEMC_Dbi_Dbi_BurstLen4
Burst length 4

enumerator kSEMC_Dbi_BurstLen8
Burst length 8

enumerator kSEMC_Dbi_BurstLen16
Burst length 16

enumerator kSEMC_Dbi_BurstLen32
Burst length 32

enumerator kSEMC_Dbi_BurstLen64
Burst length 64

enum _semc_iomux_pin
SEMC IOMUXC.

Values:

enumerator kSEMC_MUXA8
MUX A8 pin.

enumerator kSEMC_MUXCSX0
MUX CSX0 pin

enumerator kSEMC_MUXCSX1
MUX CSX1 Pin.

enumerator kSEMC_MUXCSX2
MUX CSX2 Pin.

enumerator kSEMC_MUXCSX3
MUX CSX3 Pin.

enumerator kSEMC_MUXRDY
MUX RDY pin.

enum _semc_iomux_nora27_pin
SEMC NOR/PSRAM Address bit 27 A27.

Values:

enumerator kSEMC_MORA27_NONE
No NOR/SRAM A27 pin.

enumerator kSEMC_NORA27_MUXCSX3
MUX CSX3 Pin.

enumerator kSEMC_NORA27_MUXRDY
MUX RDY pin.

enum _semc_port_size
SEMC port size.

Values:

enumerator kSEMC_PortSize8Bit
8-Bit port size.

enumerator kSEMC_PortSize16Bit
16-Bit port size.

enum _semc_addr_mode
SEMC address mode.

Values:

enumerator kSEMC_AddrDataMux
SEMC address/data mux mode.

enumerator kSEMC_AdvAddrdataMux
Advanced address/data mux mode.

enumerator kSEMC_AddrDataNonMux
Address/data non-mux mode.

enum _semc_dqs_mode
SEMC DQS read strobe mode.

Values:

enumerator kSEMC_Loopbackinternal
Dummy read strobe loopbacked internally.

enumerator kSEMC_Loopbackdqspad
Dummy read strobe loopbacked from DQS pad.

enum _semc_adv_polarity
SEMC ADV signal active polarity.

Values:

enumerator kSEMC_AdvActiveLow
Adv active low.

enumerator kSEMC_AdvActiveHigh
Adv active high.

enum _semc_sync_mode
SEMC sync mode.

Values:

enumerator kSEMC_AsyncMode
Async mode.

enumerator kSEMC_SyncMode
Sync mode.

enum _semc_adv_level_control
SEMC ADV signal level control.

Values:

enumerator kSEMC_AdvHigh
Adv is high during address hold state.

enumerator kSEMC_AdvLow
Adv is low during address hold state.

enum _semc_rdy_polarity
SEMC RDY signal active polarity.

Values:

enumerator kSEMC_RdyActiveLow
Adv active low.

enumerator kSEMC_RdyActivehigh
Adv active low.

enum _semc_ipcmd_nand_addrmode
SEMC IP command for NAND: address mode.

Values:

enumerator kSEMC_NANDAM_ColumnRow
Address mode: column and row address(5Byte-CA0/CA1/RA0/RA1/RA2).

enumerator kSEMC_NANDAM_ColumnCA0
Address mode: column address only(1 Byte-CA0).

enumerator kSEMC_NANDAM_ColumnCA0CA1
Address mode: column address only(2 Byte-CA0/CA1).

enumerator kSEMC_NANDAM_RawRA0
Address mode: row address only(1 Byte-RA0).

enumerator kSEMC_NANDAM_RawRA0RA1
Address mode: row address only(2 Byte-RA0/RA1).

enumerator kSEMC_NANDAM_RawRA0RA1RA2
Address mode: row address only(3 Byte-RA0).

enum _semc_ipcmd_nand_cmdmode
SEMC IP command for NAND□ command mode.

Values:

enumerator kSEMC_NANDCM_Command
command.

enumerator kSEMC_NANDCM_CommandHold
Command hold.

enumerator kSEMC_NANDCM_CommandAddress
Command address.

enumerator kSEMC_NANDCM_CommandAddressHold
Command address hold.

enumerator kSEMC_NANDCM_CommandAddressRead
Command address read.

enumerator kSEMC_NANDCM_CommandAddressWrite
Command address write.

enumerator kSEMC_NANDCM_CommandRead
Command read.

enumerator kSEMC_NANDCM_CommandWrite
Command write.

enumerator kSEMC_NANDCM_Read
Read.

enumerator kSEMC_NANDCM_Write
Write.

enum _semc_nand_address_option
SEMC NAND address option.

Values:

enumerator kSEMC_NandAddrOption_5byte_CA2RA3
CA0+CA1+RA0+RA1+RA2

enumerator kSEMC_NandAddrOption_4byte_CA2RA2
CA0+CA1+RA0+RA1

enumerator kSEMC_NandAddrOption_3byte_CA2RA1
CA0+CA1+RA0

enumerator kSEMC_NandAddrOption_4byte_CA1RA3
CA0+RA0+RA1+RA2

enumerator kSEMC_NandAddrOption_3byte_CA1RA2
CA0+RA0+RA1

enumerator kSEMC_NandAddrOption_2byte_CA1RA1
CA0+RA0

enum _semc_ipcmd_nor_dbi
SEMC IP command for NOR.

Values:

enumerator kSEMC_NORDBICM_Read
NOR read.

enumerator kSEMC_NORDBICM_Write
NOR write.

enum *_semc_ipcmd_sram*

SEMC IP command for SRAM.

Values:

enumerator *kSEMC_SRAMCM_ArrayRead*

SRAM memory array read.

enumerator *kSEMC_SRAMCM_ArrayWrite*

SRAM memory array write.

enumerator *kSEMC_SRAMCM_RegRead*

SRAM memory register read.

enumerator *kSEMC_SRAMCM_RegWrite*

SRAM memory register write.

enum *_semc_ipcmd_sdram*

SEMC IP command for SDARM.

Values:

enumerator *kSEMC_SDRAMCM_Read*

SDRAM memory read.

enumerator *kSEMC_SDRAMCM_Write*

SDRAM memory write.

enumerator *kSEMC_SDRAMCM_Modeset*

SDRAM MODE SET.

enumerator *kSEMC_SDRAMCM_Active*

SDRAM active.

enumerator *kSEMC_SDRAMCM_AutoRefresh*

SDRAM auto-refresh.

enumerator *kSEMC_SDRAMCM_SelfRefresh*

SDRAM self-refresh.

enumerator *kSEMC_SDRAMCM_Precharge*

SDRAM precharge.

enumerator *kSEMC_SDRAMCM_Prechargeall*

SDRAM precharge all.

typedef enum *_semc_mem_type* *semc_mem_type_t*

SEMC memory device type.

typedef enum *_semc_waitready_polarity* *semc_waitready_polarity_t*

SEMC WAIT/RDY polarity.

typedef enum *_semc_sdram_cs* *semc_sdram_cs_t*

SEMC SDRAM Chip selection .

typedef enum *_semc_sram_cs* *semc_sram_cs_t*

SEMC SRAM Chip selection .

typedef enum *_semc_nand_access_type* *semc_nand_access_type_t*

SEMC NAND device type.

typedef enum *_semc_interrupt_enable* *semc_interrupt_enable_t*

SEMC interrupts .

`typedef enum _semc_ipcmd_datasize` `semc_ipcmd_datasize_t`
SEMC IP command data size in bytes.

`typedef enum _semc_refresh_time` `semc_refresh_time_t`
SEMC auto-refresh timing.

`typedef enum _semc_caslatency` `semc_caslatency_t`
CAS latency.

`typedef enum _semc_sdram_column_bit_num` `semc_sdram_column_bit_num_t`
SEMC sdram column address bit number.

`typedef enum _semc_sdram_burst_len` `sem_sdram_burst_len_t`
SEMC sdram burst length.

`typedef enum _semc_nand_column_bit_num` `semc_nand_column_bit_num_t`
SEMC nand column address bit number.

`typedef enum _semc_nand_burst_len` `sem_nand_burst_len_t`
SEMC nand burst length.

`typedef enum _semc_norsram_column_bit_num` `semc_norsram_column_bit_num_t`
SEMC nor/sram column address bit number.

`typedef enum _semc_norsram_burst_len` `sem_norsram_burst_len_t`
SEMC nor/sram burst length.

`typedef enum _semc_dbi_column_bit_num` `semc_dbi_column_bit_num_t`
SEMC dbi column address bit number.

`typedef enum _semc_dbi_burst_len` `sem_dbi_burst_len_t`
SEMC dbi burst length.

`typedef enum _semc_iomux_pin` `semc_iomux_pin`
SEMC IOMUXC.

`typedef enum _semc_iomux_nora27_pin` `semc_iomux_nora27_pin`
SEMC NOR/PSRAM Address bit 27 A27.

`typedef enum _semc_port_size` `smec_port_size_t`
SEMC port size.

`typedef enum _semc_addr_mode` `semc_addr_mode_t`
SEMC address mode.

`typedef enum _semc_dqs_mode` `semc_dqs_mode_t`
SEMC DQS read strobe mode.

`typedef enum _semc_adv_polarity` `semc_adv_polarity_t`
SEMC ADV signal active polarity.

`typedef enum _semc_sync_mode` `semc_sync_mode_t`
SEMC sync mode.

`typedef enum _semc_adv_level_control` `semc_adv_level_control_t`
SEMC ADV signal level control.

`typedef enum _semc_rdy_polarity` `semc_rdy_polarity_t`
SEMC RDY signal active polarity.

`typedef enum _semc_ipcmd_nand_addrmode` `semc_ipcmd_nand_addrmode_t`
SEMC IP command for NAND: address mode.

typedef enum *_semc_ipcmd_nand_cmdmode* semc_ipcmd_nand_cmdmode_t
SEMC IP command for NAND command mode.

typedef enum *_semc_nand_address_option* semc_nand_address_option_t
SEMC NAND address option.

typedef enum *_semc_ipcmd_nor_dbi* semc_ipcmd_nor_dbi_t
SEMC IP command for NOR.

typedef enum *_semc_ipcmd_sram* semc_ipcmd_sram_t
SEMC IP command for SRAM.

typedef enum *_semc_ipcmd_sdram* semc_ipcmd_sdram_t
SEMC IP command for SDARM.

typedef struct *_semc_sdram_config* semc_sdram_config_t
SEMC SDRAM configuration structure.

- a. The memory size in the configuration is in the unit of KB. So memsize_kbytes should be set as 2^2 , 2^3 , 2^4 .etc which is base 2KB exponential function. Take refer to BR0~BR3 register in RM for details.
- b. The prescalePeriod_N16Cycle is in unit of 16 clock cycle. It is a exception for prescaleTimer_n16cycle = 0, it means the prescaler timer period is $256 * 16$ clock cycles. For precalerIf precalerTimer_n16cycle not equal to 0, The prescaler timer period is $\text{prescalePeriod\_N16Cycle} * 16$ clock cycles. idleTimeout_NprescalePeriod, refreshUrgThreshold_NprescalePeriod, refreshPeriod_NprescalePeriod are similar to prescalePeriod_N16Cycle.

typedef struct *_semc_nand_timing_config* semc_nand_timing_config_t
SEMC NAND device timing configuration structure.

typedef struct *_semc_nand_config* semc_nand_config_t
SEMC NAND configuration structure.

typedef struct *_semc_nor_config* semc_nor_config_t
SEMC NOR configuration structure.

typedef struct *_semc_sram_config* semc_sram_config_t
SEMC SRAM configuration structure.

typedef struct *_semc_dbi_config* semc_dbi_config_t
SEMC DBI configuration structure.

typedef struct *_semc_queuea_weight_struct* semc_queuea_weight_struct_t
SEMC AXI queue a weight setting structure.

typedef union *_semc_queuea_weight* semc_queuea_weight_t
SEMC AXI queue a weight setting union.

typedef struct *_semc_queueb_weight_struct* semc_queueb_weight_struct_t
SEMC AXI queue b weight setting structure.

typedef union *_semc_queueb_weight* semc_queueb_weight_t
SEMC AXI queue b weight setting union.

typedef struct *_semc_axi_queueweight* semc_axi_queueweight_t
SEMC AXI queue weight setting.

```
typedef struct _semc_config_t semc_config_t
```

SEMC configuration structure.

busTimeoutCycles: when busTimeoutCycles is zero, the bus timeout cycle is 255*1024. otherwise the bus timeout cycles is busTimeoutCycles*1024. cmdTimeoutCycles: is used for command execution timeout cycles. it's similar to the busTimeoutCycles.

```
struct _semc_sdram_config
```

#include <fsl_semc.h> SEMC SDRAM configuration structure.

- a. The memory size in the configuration is in the unit of KB. So memsize_kbytes should be set as 2^2 , 2^3 , 2^4 .etc which is base 2KB exponential function. Take refer to BR0~BR3 register in RM for details.
- b. The prescalePeriod_N16Cycle is in unit of 16 clock cycle. It is a exception for prescalerTimer_n16cycle = 0, it means the prescaler timer period is $256 * 16$ clock cycles. For precalerIf precalerTimer_n16cycle not equal to 0, The prescaler timer period is prescalePeriod_N16Cycle * 16 clock cycles. idleTimeout_NprescalePeriod, refreshUrgThreshold_NprescalePeriod, refreshPeriod_NprescalePeriod are similar to prescalePeriod_N16Cycle.

Public Members

semc_iomux_pin csxPinMux

CS pin mux. The kSEMC_MUXA8 is not valid in sdram pin mux setting.

uint32_t address

The base address.

uint32_t memsize_kbytes

The memory size in unit of kbytes.

smec_port_size_t portSize

Port size.

sem_sdram_burst_len_t burstLen

Burst length.

semc_sdram_column_bit_num_t columnAddrBitNum

Column address bit number.

semc_caslatency_t casLatency

CAS latency.

uint8_t tPrecharge2Act_Ns

Precharge to active wait time in unit of nanosecond.

uint8_t tAct2ReadWrite_Ns

Act to read/write wait time in unit of nanosecond.

uint8_t tRefreshRecovery_Ns

Refresh recovery time in unit of nanosecond.

uint8_t tWriteRecovery_Ns

write recovery time in unit of nanosecond.

uint8_t tCkeOff_Ns

CKE off minimum time in unit of nanosecond.

uint8_t tAct2Prechage_Ns

Active to precharge in unit of nanosecond.

uint8_t tSelfRefRecovery_Ns

Self refresh recovery time in unit of nanosecond.

uint8_t tRefresh2Refresh_Ns

Refresh to refresh wait time in unit of nanosecond.

uint8_t tAct2Act_Ns

Active to active wait time in unit of nanosecond.

uint32_t tPrescalePeriod_Ns

Prescaler timer period should not be larger than $256 * 16 * \text{clock cycle}$.

uint32_t tIdleTimeout_Ns

Idle timeout in unit of prescale time period.

uint32_t refreshPeriod_nsPerRow

Refresh timer period like $64\text{ms} * 1000000/8192$.

uint32_t refreshUrgThreshold

Refresh urgent threshold.

uint8_t refreshBurstLen

Refresh burst length.

uint8_t delayChain

Delay chain, which adds delays on DQS clock to compensate timings while DQS is faster than read data.

uint8_t autofreshTimes

Auto Refresh cycles times.

struct _semc_nand_timing_config

#include <fsl_semc.h> SEMC NAND device timing configuration structure.

Public Members

uint8_t tCeSetup_Ns

CE setup time: tCS.

uint8_t tCeHold_Ns

CE hold time: tCH.

uint8_t tCeInterval_Ns

CE interval time:tCEITV.

uint8_t tWeLow_Ns

WE low time: tWP.

uint8_t tWeHigh_Ns

WE high time: tWH.

uint8_t tReLow_Ns

RE low time: tRP.

uint8_t tReHigh_Ns

RE high time: tREH.

uint8_t tTurnAround_Ns

Turnaround time for async mode: tTA.

uint8_t tWehigh2Relow_Ns

WE# high to RE# wait time: tWHR.

uint8_t tRehigh2Welow_Ns

RE# high to WE# low wait time: tRHW.

uint8_t tAle2WriteStart_Ns

ALE to write start wait time: tADL.

uint8_t tReady2Relow_Ns

Ready to RE# low min wait time: tRR.

uint8_t tWehigh2Busy_Ns

WE# high to busy wait time: tWB.

struct _semc_nand_config

#include <fsl_semc.h> SEMC NAND configuration structure.

Public Members

semc_iomux_pin cePinMux

The CE pin mux setting. The kSEMC_MUXRDY is not valid for CE pin setting.

uint32_t axiAddress

The base address for AXI nand.

uint32_t axiMemsizes_kbytes

The memory size in unit of kbytes for AXI nand.

uint32_t ipgAddress

The base address for IPG nand .

uint32_t ipgMemsizes_kbytes

The memory size in unit of kbytes for IPG nand.

semc_rdy_polarity_t rdyactivePolarity

Wait ready polarity.

bool edoModeEnabled

EDO mode enabled.

semc_nand_column_bit_num_t columnAddrBitNum

Column address bit number.

semc_nand_address_option_t arrayAddrOption

Address option.

sem_nand_burst_len_t burstLen

Burst length.

smec_port_size_t portSize

Port size.

semc_nand_timing_config_t *timingConfig

SEMC nand timing configuration.

struct _semc_nor_config

#include <fsl_semc.h> SEMC NOR configuration structure.

Public Members

semc_iomux_pin cePinMux

The CE# pin mux setting.

semc_iomux_nora27_pin addr27

The Addr bit 27 pin mux setting.

uint32_t address

The base address.

uint32_t memsize_kbytes

The memory size in unit of kbytes.

uint8_t addrPortWidth

The address port width.

semc_rdy_polarity_t rdyactivePolarity

Wait ready polarity.

semc_adv_polarity_t advActivePolarity

ADV# polarity.

semc_norsram_column_bit_num_t columnAddrBitNum

Column address bit number.

semc_addr_mode_t addrMode

Address mode.

sem_norsram_burst_len_t burstLen

Burst length.

smec_port_size_t portSize

Port size.

uint8_t tCeSetup_Ns

The CE setup time.

uint8_t tCeHold_Ns

The CE hold time.

uint8_t tCeInterval_Ns

CE interval minimum time.

uint8_t tAddrSetup_Ns

The address setup time.

uint8_t tAddrHold_Ns

The address hold time.

uint8_t tWeLow_Ns

WE low time for async mode.

uint8_t tWeHigh_Ns

WE high time for async mode.

uint8_t tReLow_Ns

RE low time for async mode.

uint8_t tReHigh_Ns

RE high time for async mode.

uint8_t tTurnAround_Ns

Turnaround time for async mode.

uint8_t tAddr2WriteHold_Ns

Address to write data hold time for async mode.

uint8_t tWriteSetup_Ns

Write data setup time for sync mode.

uint8_t tWriteHold_Ns

Write hold time for sync mode.

uint8_t latencyCount

Latency count for sync mode.

uint8_t readCycle

Read cycle time for sync mode.

uint8_t delayChain

Delay chain, which adds delays on DQS clock to compensate timings while DQS is faster than read data.

struct __semc_sram_config

#include <fsl_semc.h> SEMC SRAM configuration structure.

Public Members

semc_iomux_pin cePinMux

The CE# pin mux setting.

semc_iomux_nora27_pin addr27

The Addr bit 27 pin mux setting.

uint32_t address

The base address.

uint32_t memsize_kbytes

The memory size in unit of kbytes.

uint8_t addrPortWidth

The address port width.

semc_adv_polarity_t advActivePolarity

ADV# polarity 1: active high, 0: active low.

semc_addr_mode_t addrMode

Address mode.

sem_norsram_burst_len_t burstLen

Burst length.

smec_port_size_t portSize

Port size.

semc_sync_mode_t syncMode

Sync mode.

bool waitEnable

Wait enable.

`uint8_t waitSample`
Wait sample.

`semc_adv_level_control_t advLevelCtrl`
ADV# level control during address hold state, 1: low, 0: high.

`uint8_t tCeSetup_Ns`
The CE setup time.

`uint8_t tCeHold_Ns`
The CE hold time.

`uint8_t tCeInterval_Ns`
CE interval minimum time.

`uint8_t readHoldTime_Ns`
read hold time.

`uint8_t tAddrSetup_Ns`
The address setup time.

`uint8_t tAddrHold_Ns`
The address hold time.

`uint8_t tWeLow_Ns`
WE low time for async mode.

`uint8_t tWeHigh_Ns`
WE high time for async mode.

`uint8_t tReLow_Ns`
RE low time for async mode.

`uint8_t tReHigh_Ns`
RE high time for async mode.

`uint8_t tTurnAround_Ns`
Turnaround time for async mode.

`uint8_t tAddr2WriteHold_Ns`
Address to write data hold time for async mode.

`uint8_t tWriteSetup_Ns`
Write data setup time for sync mode.

`uint8_t tWriteHold_Ns`
Write hold time for sync mode.

`uint8_t latencyCount`
Latency count for sync mode.

`uint8_t readCycle`
Read cycle time for sync mode.

`uint8_t delayChain`
Delay chain, which adds delays on DQS clock to compensate timings while DQS is faster than read data.

`struct __semc_dbi_config`
#include <fsl_semc.h> SEMC DBI configuration structure.

Public Members

semc_iomux_pin csxPinMux

The CE# pin mux.

uint32_t address

The base address.

uint32_t memsize_kbytes

The memory size in unit of 4kbytes.

semc_dbi_column_bit_num_t columnAddrBitNum

Column address bit number.

sem_dbi_burst_len_t burstLen

Burst length.

smec_port_size_t portSize

Port size.

uint8_t tCsxSetup_Ns

The CSX setup time.

uint8_t tCsxHold_Ns

The CSX hold time.

uint8_t tWexLow_Ns

WEX low time.

uint8_t tWexHigh_Ns

WEX high time.

uint8_t tRdxLow_Ns

RDX low time.

uint8_t tRdxHigh_Ns

RDX high time.

uint8_t tCsxInterval_Ns

Write data setup time.

struct __semc_queuea_weight_struct

#include <fsl_semc.h> SEMC AXI queue a weight setting structure.

Public Members

uint32_t qos

weight of qos for queue 0 .

uint32_t aging

weight of aging for queue 0.

uint32_t slaveHitNoswitch

weight of read/write no switch for queue 0 .

uint32_t slaveHitSwitch

weight of read/write switch for queue 0.

union __semc_queuea_weight

#include <fsl_semc.h> SEMC AXI queue a weight setting union.

Public Members

semc_queuea_weight_struct_t queueaConfig

Structure configuration for queueA.

uint32_t queueaValue

Configuration value for queueA which could directly write to the reg.

struct __semc_queueb_weight_struct

#include <fsl_semc.h> SEMC AXI queue b weight setting structure.

Public Members

uint32_t qos

weight of qos for queue 1.

uint32_t aging

weight of aging for queue 1.

uint32_t weightPagehit

weight of page hit for queue 1 only .

uint32_t slaveHitNoswitch

weight of read/write no switch for queue 1.

uint32_t bankRotation

weight of bank rotation for queue 1 only .

union __semc_queueb_weight

#include <fsl_semc.h> SEMC AXI queue b weight setting union.

Public Members

semc_queueb_weight_struct_t queuebConfig

Structure configuration for queueB.

uint32_t queuebValue

Configuration value for queueB which could directly write to the reg.

struct __semc_axi_queueweight

#include <fsl_semc.h> SEMC AXI queue weight setting.

Public Members

bool queueaEnable

Enable queue a.

semc_queuea_weight_t queueaWeight

Weight settings for queue a.

bool queuebEnable

Enable queue b.

semc_queueb_weight_t queuebWeight

Weight settings for queue b.

struct `_semc_config_t`

#include <fsl_semc.h> SEMC configuration structure.

`busTimeoutCycles`: when `busTimeoutCycles` is zero, the bus timeout cycle is 255*1024. otherwise the bus timeout cycles is `busTimeoutCycles*1024`. `cmdTimeoutCycles`: is used for command execution timeout cycles. it's similar to the `busTimeoutCycles`.

Public Members

`semc_dqs_mode_t` `dqsMode`

Dummy read strobe mode: use enum in “`semc_dqs_mode_t`”.

`uint8_t` `cmdTimeoutCycles`

Command execution timeout cycles.

`uint8_t` `busTimeoutCycles`

Bus timeout cycles.

`semc_axi_queueweight_t` `queueWeight`

AXI queue weight.

2.101 Smart Card

FSL_SMARTCARD_DRIVER_VERSION

Smart card driver version 2.3.0.

Smart card Error codes.

Values:

enumerator `kStatus_SMARTCARD_Success`

Transfer ends successfully

enumerator `kStatus_SMARTCARD_TxBusy`

Transmit in progress

enumerator `kStatus_SMARTCARD_RxBusy`

Receiving in progress

enumerator `kStatus_SMARTCARD_NoTransferInProgress`

No transfer in progress

enumerator `kStatus_SMARTCARD_Timeout`

Transfer ends with time-out

enumerator `kStatus_SMARTCARD_Initialized`

Smart card driver is already initialized

enumerator `kStatus_SMARTCARD_PhyInitialized`

Smart card PHY drive is already initialized

enumerator `kStatus_SMARTCARD_CardNotActivated`

Smart card is not activated

enumerator `kStatus_SMARTCARD_InvalidInput`

Function called with invalid input arguments

enumerator kStatus_SMARTCARD_OtherError

Some other error occur

enum _smartcard_control

Control codes for the Smart card protocol timers and misc.

Values:

enumerator kSMARTCARD_EnableADT

enumerator kSMARTCARD_DisableADT

enumerator kSMARTCARD_EnableGTV

enumerator kSMARTCARD_DisableGTV

enumerator kSMARTCARD_ResetWWT

enumerator kSMARTCARD_EnableWWT

enumerator kSMARTCARD_DisableWWT

enumerator kSMARTCARD_ResetCWT

enumerator kSMARTCARD_EnableCWT

enumerator kSMARTCARD_DisableCWT

enumerator kSMARTCARD_ResetBWT

enumerator kSMARTCARD_EnableBWT

enumerator kSMARTCARD_DisableBWT

enumerator kSMARTCARD_EnableInitDetect

enumerator kSMARTCARD_EnableAnack

enumerator kSMARTCARD_DisableAnack

enumerator kSMARTCARD_ConfigureBaudrate

enumerator kSMARTCARD_SetupATRMode

enumerator kSMARTCARD_SetupT0Mode

enumerator kSMARTCARD_SetupT1Mode

enumerator kSMARTCARD_EnableReceiverMode

enumerator kSMARTCARD_DisableReceiverMode

enumerator kSMARTCARD_EnableTransmitterMode

enumerator kSMARTCARD_DisableTransmitterMode

enumerator kSMARTCARD_ResetWaitTimeMultiplier

enum _smartcard_card_voltage_class

Defines Smart card interface voltage class values.

Values:

enumerator kSMARTCARD_VoltageClassUnknown

enumerator kSMARTCARD_VoltageClassA5_0V

enumerator kSMARTCARD_VoltageClassB3_3V

enumerator kSMARTCARD_VoltageClassC1_8V

enum _smartcard_transfer_state

Defines Smart card I/O transfer states.

Values:

enumerator kSMARTCARD_IdleState

enumerator kSMARTCARD_WaitingForTSSState

enumerator kSMARTCARD_InvalidTSDetectedState

enumerator kSMARTCARD_ReceivingState

enumerator kSMARTCARD_TransmittingState

enum _smartcard_reset_type

Defines Smart card reset types.

Values:

enumerator kSMARTCARD_ColdReset

enumerator kSMARTCARD_WarmReset

enumerator kSMARTCARD_NoColdReset

enumerator kSMARTCARD_NoWarmReset

enum _smartcard_transport_type

Defines Smart card transport protocol types.

Values:

enumerator kSMARTCARD_T0Transport

enumerator kSMARTCARD_T1Transport

enum _smartcard_parity_type

Defines Smart card data parity types.

Values:

enumerator kSMARTCARD_EvenParity

enumerator kSMARTCARD_OddParity

enum _smartcard_card_convention

Defines data Convention format.

Values:

enumerator kSMARTCARD_DirectConvention

enumerator kSMARTCARD_InverseConvention

enum _smartcard_interface_control

Defines Smart card interface IC control types.

Values:

enumerator kSMARTCARD_InterfaceSetVcc

enumerator kSMARTCARD_InterfaceSetClockToResetDelay

enumerator kSMARTCARD_InterfaceReadStatus

enum *_smartcard_direction*

Defines transfer direction.

Values:

enumerator kSMARTCARD_Receive

enumerator kSMARTCARD_Transmit

typedef enum *_smartcard_control* smartcard_control_t

Control codes for the Smart card protocol timers and misc.

typedef enum *_smartcard_card_voltage_class* smartcard_card_voltage_class_t

Defines Smart card interface voltage class values.

typedef enum *_smartcard_transfer_state* smartcard_transfer_state_t

Defines Smart card I/O transfer states.

typedef enum *_smartcard_reset_type* smartcard_reset_type_t

Defines Smart card reset types.

typedef enum *_smartcard_transport_type* smartcard_transport_type_t

Defines Smart card transport protocol types.

typedef enum *_smartcard_parity_type* smartcard_parity_type_t

Defines Smart card data parity types.

typedef enum *_smartcard_card_convention* smartcard_card_convention_t

Defines data Convention format.

typedef enum *_smartcard_interface_control* smartcard_interface_control_t

Defines Smart card interface IC control types.

typedef enum *_smartcard_direction* smartcard_direction_t

Defines transfer direction.

typedef void (*smartcard_interface_callback_t)(void *smartcardContext, void *param)

Smart card interface interrupt callback function type.

typedef void (*smartcard_transfer_callback_t)(void *smartcardContext, void *param)

Smart card transfer interrupt callback function type.

typedef void (*smartcard_time_delay_t)(uint32_t us)

Time Delay function used to passive waiting using RTOS [us].

typedef struct *_smartcard_card_params* smartcard_card_params_t

Defines card-specific parameters for Smart card driver.

typedef struct *_smartcard_timers_state* smartcard_timers_state_t

Smart card defines the state of the EMV timers in the Smart card driver.

typedef struct *_smartcard_interface_config* smartcard_interface_config_t

Defines user specified configuration of Smart card interface.

typedef struct *_smartcard_xfer* smartcard_xfer_t

Defines user transfer structure used to initialize transfer.

typedef struct *_smartcard_context* smartcard_context_t

Runtime state of the Smart card driver.

SMARTCARD_INIT_DELAY_CLOCK_CYCLES

Smart card global define which specify number of clock cycles until initial 'TS' character has to be received.

SMARTCARD_EMV_ATR_DURATION_ETU

Smart card global define which specify number of clock cycles during which ATR string has to be received.

SMARTCARD_TS_DIRECT_CONVENTION

Smart card specification initial TS character definition of direct convention.

SMARTCARD_TS_INVERSE_CONVENTION

Smart card specification initial TS character definition of inverse convention.

struct _smartcard_card_params

#include <fsl_smartcard.h> Defines card-specific parameters for Smart card driver.

Public Members

uint16_t Fi

4 bits Fi - clock rate conversion integer

uint8_t fMax

Maximum Smart card frequency in MHz

uint8_t WI

8 bits WI - work wait time integer

uint8_t Di

4 bits DI - baud rate divisor

uint8_t BWI

4 bits BWI - block wait time integer

uint8_t CWI

4 bits CWI - character wait time integer

uint8_t BGI

4 bits BGI - block guard time integer

uint8_t GTN

8 bits GTN - extended guard time integer

uint8_t IFSC

Indicates IFSC value of the card

uint8_t modeNegotiable

Indicates if the card acts in negotiable or a specific mode.

uint8_t currentD

4 bits DI - current baud rate divisor

uint8_t status

Indicates smart card status

bool t0Indicated

Indicates if T=0 indicated in TD1 byte

bool t1Indicated

Indicates if T=1 indicated in TD2 byte

bool atrComplete

Indicates whether the ATR received from the card was complete or not

bool atrValid

Indicates whether the ATR received from the card was valid or not

bool present

Indicates if a smart card is present

bool active

Indicates if the smart card is activated

bool faulty

Indicates whether smart card/interface is faulty

smartcard_card_convention_t convention

Card convention, kSMARTCARD_DirectConvention for direct convention, kSMARTCARD_InverseConvention for inverse convention

struct __smartcard_timers_state

#include <fsl_smartcard.h> Smart card defines the state of the EMV timers in the Smart card driver.

Public Members

volatile bool adtExpired

Indicates whether ADT timer expired

volatile bool wwtExpired

Indicates whether WWT timer expired

volatile bool cwtExpired

Indicates whether CWT timer expired

volatile bool bwtExpired

Indicates whether BWT timer expired

volatile bool initCharTimerExpired

Indicates whether reception timer for initialization character (TS) after the RST has expired

struct __smartcard_interface_config

#include <fsl_smartcard.h> Defines user specified configuration of Smart card interface.

Public Members

uint32_t smartCardClock

Smart card interface clock [Hz]

uint32_t clockToResetDelay

Indicates clock to RST apply delay [smart card clock cycles]

uint8_t clockModule

Smart card clock module number

uint8_t clockModuleChannel

Smart card clock module channel number

uint8_t clockModuleSourceClock

Smart card clock module source clock [e.g., BusClk]

smartcard_card_voltage_class_t vcc
Smart card voltage class

uint8_t controlPort
Smart card PHY control port instance

uint8_t controlPin
Smart card PHY control pin instance

uint8_t irqPort
Smart card PHY Interrupt port instance

uint8_t irqPin
Smart card PHY Interrupt pin instance

uint8_t resetPort
Smart card reset port instance

uint8_t resetPin
Smart card reset pin instance

uint8_t vsel0Port
Smart card PHY Vsel0 control port instance

uint8_t vsel0Pin
Smart card PHY Vsel0 control pin instance

uint8_t vsel1Port
Smart card PHY Vsel1 control port instance

uint8_t vsel1Pin
Smart card PHY Vsel1 control pin instance

uint8_t dataPort
Smart card PHY data port instance

uint8_t dataPin
Smart card PHY data pin instance

uint8_t dataPinMux
Smart card PHY data pin mux option

uint8_t tsTimerId
Numerical identifier of the External HW timer for Initial character detection

struct *_smartcard_xfer*
#include <fsl_smartcard.h> Defines user transfer structure used to initialize transfer.

Public Members

smartcard_direction_t direction
Direction of communication. (RX/TX)

uint8_t *buff
The buffer of data.

size_t size
The number of transferred units.

struct *_smartcard_context*
#include <fsl_smartcard.h> Runtime state of the Smart card driver.

Public Members

`void *base`

Smart card module base address

`smartcard_direction_t direction`

Direction of communication. (RX/TX)

`uint8_t *xBuff`

The buffer of data being transferred.

`volatile size_t xSize`

The number of bytes to be transferred.

`volatile bool xIsBusy`

True if there is an active transfer.

`uint8_t txFifoEntryCount`

Number of data word entries in transmit FIFO.

`uint8_t rxFifoThreshold`

The max value of the receiver FIFO threshold.

`smartcard_interface_callback_t interfaceCallback`

Callback to invoke after interface IC raised interrupt.

`smartcard_transfer_callback_t transferCallback`

Callback to invoke after transfer event occur.

`void *interfaceCallbackParam`

Interface callback parameter pointer.

`void *transferCallbackParam`

Transfer callback parameter pointer.

`smartcard_time_delay_t timeDelay`

Function which handles time delay defined by user or RTOS.

`smartcard_reset_type_t resetType`

Indicates whether a Cold reset or Warm reset was requested.

`smartcard_transport_type_t tType`

Indicates current transfer protocol (T0 or T1)

`volatile smartcard_transfer_state_t transferState`

Indicates the current transfer state

`smartcard_timers_state_t timersState`

Indicates the state of different protocol timers used in driver

`smartcard_card_params_t cardParams`

Smart card parameters(ATR and current) and interface slots states(ATR and current)

`uint8_t IFSD`

Indicates the terminal IFSD

`smartcard_parity_type_t parity`

Indicates current parity even/odd

`volatile bool rxtCrossed`

Indicates whether RXT thresholds has been crossed

volatile bool txtCrossed

Indicates whether TXT thresholds has been crossed

volatile bool wtxRequested

Indicates whether WTX has been requested or not

volatile bool parityError

Indicates whether a parity error has been detected

uint8_t statusBytes[2]

Used to store Status bytes SW1, SW2 of the last executed card command response

smartcard_interface_config_t interfaceConfig

Smart card interface configuration structure

bool abortTransfer

Used to abort transfer.

2.102 Smart Card EMVSIM Driver

void SMARTCARD_EMVSIM_GetDefaultConfig(*smartcard_card_params_t* *cardParams)

Fills in the *smartcard_card_params* structure with default values according to the EMV 4.3 specification.

Parameters

- *cardParams* – The configuration structure of type *smartcard_interface_config_t*. Function fill in members: Fi = 372; Di = 1; currentD = 1; WI = 0x0A; GTN = 0x00; with default values.

status_t SMARTCARD_EMVSIM_Init(*EMVSIM_Type* *base, *smartcard_context_t* *context, uint32_t srcClock_Hz)

Initializes an EMVSIM peripheral for the Smart card/ISO-7816 operation.

This function un-gates the EMVSIM clock, initializes the module to EMV default settings, configures the IRQ, enables the module-level interrupt to the core and, initializes the driver context.

Parameters

- *base* – The EMVSIM peripheral base address.
- *context* – A pointer to the smart card driver context structure.
- *srcClock_Hz* – Smart card clock generation module source clock.

Returns

An error code or *kStatus_SMARTCARD_Success*.

void SMARTCARD_EMVSIM_Deinit(*EMVSIM_Type* *base)

This function disables the EMVSIM interrupts, disables the transmitter and receiver, flushes the FIFOs, and gates EMVSIM clock in SIM.

Parameters

- *base* – The EMVSIM module base address.

int32_t SMARTCARD_EMVSIM_GetTransferRemainingBytes(*EMVSIM_Type* *base, *smartcard_context_t* *context)

Returns whether the previous EMVSIM transfer has finished.

When performing an async transfer, call this function to ascertain the context of the current transfer: in progress (or busy) or complete (success). If the transfer is still in progress, the user can obtain the number of words that have not been transferred.

Parameters

- `base` – The EMVSIM module base address.
- `context` – A pointer to a smart card driver context structure.

Returns

The number of bytes not transferred.

`status_t SMARTCARD_EMVSIM_AbortTransfer(EMVSIM_Type *base, smartcard_context_t *context)`

Terminates an asynchronous EMVSIM transfer early.

During an async EMVSIM transfer, the user can terminate the transfer early if the transfer is still in progress.

Parameters

- `base` – The EMVSIM peripheral address.
- `context` – A pointer to a smart card driver context structure.

Return values

- `kStatus_SMARTCARD_Success` – The transmit abort was successful.
- `kStatus_SMARTCARD_NoTransmitInProgress` – No transmission is currently in progress.

`status_t SMARTCARD_EMVSIM_TransferNonBlocking(EMVSIM_Type *base, smartcard_context_t *context, smartcard_xfer_t *xfer)`

Transfer data using interrupts.

A non-blocking (also known as asynchronous) function means that the function returns immediately after initiating the transfer function. The application has to get the transfer status to see when the transfer is complete. In other words, after calling the non-blocking (asynchronous) transfer function, the application must get the transfer status to check if the transmit is completed or not.

Parameters

- `base` – The EMVSIM peripheral base address.
- `context` – A pointer to a smart card driver context structure.
- `xfer` – A pointer to the smart card transfer structure where the linked buffers and sizes are stored.

Returns

An error code or `kStatus_SMARTCARD_Success`.

`status_t SMARTCARD_EMVSIM_Control(EMVSIM_Type *base, smartcard_context_t *context, smartcard_control_t control, uint32_t param)`

Controls the EMVSIM module per different user request.

return `kStatus_SMARTCARD_Success` in success. return `kStatus_SMARTCARD_OtherError` in case of error.

Parameters

- `base` – The EMVSIM peripheral base address.
- `context` – A pointer to a smart card driver context structure.
- `control` – Control type.
- `param` – Integer value of specific to control command.

`void SMARTCARD_EMVSIM_IRQHandler(EMVSIM_Type *base, smartcard_context_t *context)`
Handles EMVSIM module interrupts.

Parameters

- `base` – The EMVSIM peripheral base address.
- `context` – A pointer to a smart card driver context structure.

`enum __emvsim_gpc_clock_select`

General Purpose Counter clock selections.

Values:

enumerator `kEMVSIM_GPCClockDisable`
Disabled

enumerator `kEMVSIM_GPCCardClock`
Card clock

enumerator `kEMVSIM_GPCRxClock`
Receive clock

enumerator `kEMVSIM_GPCTxClock`
Transmit ETU clock

`enum __presence_detect_edge`

EMVSIM card presence detection edge control.

Values:

enumerator `kEMVSIM_DetectOnFallingEdge`
Presence detected on the falling edge

enumerator `kEMVSIM_DetectOnRisingEdge`
Presence detected on the rising edge

`enum __presence_detect_status`

EMVSIM card presence detection status.

Values:

enumerator `kEMVSIM_DetectPinIsLow`
Presence detected pin is logic low

enumerator `kEMVSIM_DetectPinIsHigh`
Presence detected pin is logic high

`typedef enum __emvsim_gpc_clock_select emvsim_gpc_clock_select_t`

General Purpose Counter clock selections.

`typedef enum __presence_detect_edge emvsim_presence_detect_edge_t`

EMVSIM card presence detection edge control.

`typedef enum __presence_detect_status emvsim_presence_detect_status_t`

EMVSIM card presence detection status.

`SMARTCARD_EMV_RX_NACK_THRESHOLD`

EMV RX NACK interrupt generation threshold.

`SMARTCARD_EMV_TX_NACK_THRESHOLD`

EMV TX NACK interrupt generation threshold.

`SMARTCARD_WWT_ADJUSTMENT`

Smart card Word Wait Timer adjustment value.

SMARTCARD_CWT_ADJUSTMENT

Smart card Character Wait Timer adjustment value.

2.103 Smart Card PHY Driver

void SMARTCARD_PHY_GetDefaultConfig(*smartcard_interface_config_t* *config)

Fills in the configuration structure with default values.

Parameters

- config – The Smart card user configuration structure which contains configuration structure of type *smartcard_interface_config_t*. Function fill in members: clockToResetDelay = 42000, vcc = kSmartcardVoltageClassB3_3V, with default values.

status_t SMARTCARD_PHY_Init(void *base, *smartcard_interface_config_t* const *config, uint32_t srcClock_Hz)

Initializes a Smart card interface instance.

Parameters

- base – The Smart card peripheral base address.
- config – The user configuration structure of type *smartcard_interface_config_t*. Call the function SMARTCARD_PHY_GetDefaultConfig() to fill the configuration structure.
- srcClock_Hz – Smart card clock generation module source clock.

Return values

kStatus_SMARTCARD_Success – or kStatus_SMARTCARD_OtherError in case of error.

void SMARTCARD_PHY_Deinit(void *base, *smartcard_interface_config_t* const *config)

De-initializes a Smart card interface, stops the Smart card clock, and disables the VCC.

Parameters

- base – The Smart card peripheral module base address.
- config – The user configuration structure of type *smartcard_interface_config_t*.

status_t SMARTCARD_PHY_Activate(void *base, *smartcard_context_t* *context, *smartcard_reset_type_t* resetType)

Activates the Smart card IC.

Parameters

- base – The Smart card peripheral module base address.
- context – A pointer to a Smart card driver context structure.
- resetType – type of reset to be performed, possible values = kSmartcardColdReset, kSmartcardWarmReset

Return values

kStatus_SMARTCARD_Success – or kStatus_SMARTCARD_OtherError in case of error.

status_t SMARTCARD_PHY_Deactivate(void *base, *smartcard_context_t* *context)

De-activates the Smart card IC.

Parameters

- `base` – The Smart card peripheral module base address.
- `context` – A pointer to a Smart card driver context structure.

Return values

`kStatus_SMARTCARD_Success` – or `kStatus_SMARTCARD_OtherError` in case of error.

`status_t SMARTCARD_PHY_Control(void *base, smartcard_context_t *context, smartcard_interface_control_t control, uint32_t param)`

Controls the Smart card interface IC.

Parameters

- `base` – The Smart card peripheral module base address.
- `context` – A pointer to a Smart card driver context structure.
- `control` – A interface command type.
- `param` – Integer value specific to control type

Return values

`kStatus_SMARTCARD_Success` – or `kStatus_SMARTCARD_OtherError` in case of error.

`SMARTCARD_ATR_DURATION_ADJUSTMENT`

Smart card definition which specifies the adjustment number of clock cycles during which an ATR string has to be received.

`SMARTCARD_INIT_DELAY_CLOCK_CYCLES_ADJUSTMENT`

Smart card definition which specifies the adjustment number of clock cycles until an initial 'TS' character has to be received.

2.104 Smart Card PHY EMVSIM Driver

2.105 Smart Card PHY TDA8035 Driver

2.106 SNVS: Secure Non-Volatile Storage

2.107 Secure Non-Volatile Storage High-Power

`void SNVS_HP_Init(SNVS_Type *base)`

Initialize the SNVS.

Note: This API should be called at the beginning of the application using the SNVS driver.

Parameters

- `base` – SNVS peripheral base address

`void SNVS_HP_Deinit(SNVS_Type *base)`

Deinitialize the SNVS.

Parameters

- `base` – SNVS peripheral base address

`void SNVS_HP_RTC_Init(SNVS_Type *base, const snvs_hp_rtc_config_t *config)`
Ungates the SNVS clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the SNVS driver.

Parameters

- `base` – SNVS peripheral base address
- `config` – Pointer to the user's SNVS configuration structure.

`void SNVS_HP_RTC_Deinit(SNVS_Type *base)`
Stops the RTC and SRTC timers.

Parameters

- `base` – SNVS peripheral base address

`void SNVS_HP_RTC_GetDefaultConfig(snvs_hp_rtc_config_t *config)`
Fills in the SNVS config struct with the default settings.

The default values are as follows.

```
config->rtccalenable = false;  
config->rtccalvalue = 0U;  
config->PIFreq = 0U;
```

Parameters

- `config` – Pointer to the user's SNVS configuration structure.

`status_t SNVS_HP_RTC_SetDatetime(SNVS_Type *base, const snvs_hp_rtc_datetime_t *datetime)`

Sets the SNVS RTC date and time according to the given time structure.

Parameters

- `base` – SNVS peripheral base address
- `datetime` – Pointer to the structure where the date and time details are stored.

Returns

`kStatus_Success`: Success in setting the time and starting the SNVS RTC
`kStatus_InvalidArgument`: Error because the datetime format is incorrect

`void SNVS_HP_RTC_GetDatetime(SNVS_Type *base, snvs_hp_rtc_datetime_t *datetime)`
Gets the SNVS RTC time and stores it in the given time structure.

Parameters

- `base` – SNVS peripheral base address
- `datetime` – Pointer to the structure where the date and time details are stored.

`status_t SNVS_HP_RTC_SetAlarm(SNVS_Type *base, const snvs_hp_rtc_datetime_t *alarmTime)`

Sets the SNVS RTC alarm time.

The function sets the RTC alarm. It also checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

Parameters

- `base` – SNVS peripheral base address

- `alarmTime` – Pointer to the structure where the alarm time is stored.

Returns

`kStatus_Success`: success in setting the SNVS RTC alarm
`kStatus_InvalidArgument`: Error because the alarm datetime format is incorrect
`kStatus_Fail`: Error because the alarm time has already passed

`void SNVS_HP_RTC_GetAlarm(SNVS_Type *base, snvs_hp_rtc_datetime_t *datetime)`

Returns the SNVS RTC alarm time.

Parameters

- `base` – SNVS peripheral base address
- `datetime` – Pointer to the structure where the alarm date and time details are stored.

`void SNVS_HP_RTC_TimeSynchronize(SNVS_Type *base)`

The function synchronizes RTC counter value with SRTC.

Parameters

- `base` – SNVS peripheral base address

`static inline void SNVS_HP_RTC_EnableInterrupts(SNVS_Type *base, uint32_t mask)`

Enables the selected SNVS interrupts.

Parameters

- `base` – SNVS peripheral base address
- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `::_snvs_hp_interrupts_t`

`static inline void SNVS_HP_RTC_DisableInterrupts(SNVS_Type *base, uint32_t mask)`

Disables the selected SNVS interrupts.

Parameters

- `base` – SNVS peripheral base address
- `mask` – The interrupts to disable. This is a logical OR of members of the enumeration `::_snvs_hp_interrupts_t`

`uint32_t SNVS_HP_RTC_GetEnabledInterrupts(SNVS_Type *base)`

Gets the enabled SNVS interrupts.

Parameters

- `base` – SNVS peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration `::_snvs_hp_interrupts_t`

`uint32_t SNVS_HP_RTC_GetStatusFlags(SNVS_Type *base)`

Gets the SNVS status flags.

Parameters

- `base` – SNVS peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration `::_snvs_hp_status_flags_t`

```
static inline void SNVS_HP_RTC_ClearStatusFlags(SNVS_Type *base, uint32_t mask)
```

Clears the SNVS status flags.

Parameters

- base – SNVS peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration :: `_snvs_hp_status_flags_t`

```
static inline void SNVS_HP_RTC_StartTimer(SNVS_Type *base)
```

Starts the SNVS RTC time counter.

Parameters

- base – SNVS peripheral base address

```
static inline void SNVS_HP_RTC_StopTimer(SNVS_Type *base)
```

Stops the SNVS RTC time counter.

Parameters

- base – SNVS peripheral base address

```
static inline void SNVS_HP_EnableHighAssuranceCounter(SNVS_Type *base, bool enable)
```

Enable or disable the High Assurance Counter (HAC)

Parameters

- base – SNVS peripheral base address
- enable – Pass true to enable, false to disable.

```
static inline void SNVS_HP_StartHighAssuranceCounter(SNVS_Type *base, bool start)
```

Start or stop the High Assurance Counter (HAC)

Parameters

- base – SNVS peripheral base address
- start – Pass true to start, false to stop.

```
static inline void SNVS_HP_SetHighAssuranceCounterInitialValue(SNVS_Type *base, uint32_t  
                                                                value)
```

Set the High Assurance Counter (HAC) initialize value.

Parameters

- base – SNVS peripheral base address
- value – The initial value to set.

```
static inline void SNVS_HP_LoadHighAssuranceCounter(SNVS_Type *base)
```

Load the High Assurance Counter (HAC)

This function loads the HAC initialize value to counter register.

Parameters

- base – SNVS peripheral base address

```
static inline uint32_t SNVS_HP_GetHighAssuranceCounter(SNVS_Type *base)
```

Get the current High Assurance Counter (HAC) value.

Parameters

- base – SNVS peripheral base address

Returns

HAC current value.

static inline void SNVS_HP_ClearHighAssuranceCounter(SNVS_Type *base)

Clear the High Assurance Counter (HAC)

This function can be called in a functional or soft fail state. When the HAC is enabled:

- If the HAC is cleared in the soft fail state, the SSM transitions to the hard fail state immediately;
- If the HAC is cleared in functional state, the SSM will transition to hard fail immediately after transitioning to soft fail.

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_LockHighAssuranceCounter(SNVS_Type *base)

Lock the High Assurance Counter (HAC)

Once locked, the HAC initialize value could not be changed, the HAC enable status could not be changed. This could only be unlocked by system reset.

Parameters

- base – SNVS peripheral base address

FSL_SNVS_HP_DRIVER_VERSION

Version 2.3.2

enum _snvs_hp_interrupts

List of SNVS interrupts.

Values:

enumerator kSNVS_RTC_AlarmInterrupt

RTC time alarm

enumerator kSNVS_RTC_PeriodicInterrupt

RTC periodic interrupt

enum _snvs_hp_status_flags

List of SNVS flags.

Values:

enumerator kSNVS_RTC_AlarmInterruptFlag

RTC time alarm flag

enumerator kSNVS_RTC_PeriodicInterruptFlag

RTC periodic interrupt flag

enumerator kSNVS_ZMK_ZeroFlag

The ZMK is zero

enumerator kSNVS_OTPMK_ZeroFlag

The OTPMK is zero

enum _snvs_hp_sv_status_flags

List of SNVS security violation flags.

Values:

enumerator kSNVS_LP_ViolationFlag

Low Power section Security Violation

enumerator kSNVS_ZMK_EccFailFlag

Zeroizable Master Key Error Correcting Code Check Failure

enumerator kSNVS_LP_SoftwareViolationFlag
LP Software Security Violation

enumerator kSNVS_FatalSoftwareViolationFlag
Software Fatal Security Violation

enumerator kSNVS_SoftwareViolationFlag
Software Security Violation

enumerator kSNVS_Violation0Flag
Security Violation 0

enumerator kSNVS_Violation1Flag
Security Violation 1

enumerator kSNVS_Violation2Flag
Security Violation 2

enumerator kSNVS_Violation4Flag
Security Violation 4

enumerator kSNVS_Violation5Flag
Security Violation 5

enum _snvs_hp_ssm_state
List of SNVS Security State Machine State.

Values:

enumerator kSNVS_SSMInit
Init

enumerator kSNVS_SSMHardFail
Hard Fail

enumerator kSNVS_SSMSoftFail
Soft Fail

enumerator kSNVS_SSMInitInter
Init Intermediate (transition state between Init and Check)

enumerator kSNVS_SSMCheck
Check

enumerator kSNVS_SSMNonSecure
Non-Secure

enumerator kSNVS_SSMTrusted
Trusted

enumerator kSNVS_SSMSecure
Secure

typedef enum _snvs_hp_interrupts snvs_hp_interrupts_t
List of SNVS interrupts.

typedef enum _snvs_hp_status_flags snvs_hp_status_flags_t
List of SNVS flags.

typedef enum _snvs_hp_sv_status_flags snvs_hp_sv_status_flags_t
List of SNVS security violation flags.

```
typedef struct _snvs_hp_rtc_datetime snvs_hp_rtc_datetime_t
```

Structure is used to hold the date and time.

```
typedef struct _snvs_hp_rtc_config snvs_hp_rtc_config_t
```

SNVS config structure.

This structure holds the configuration settings for the SNVS peripheral. To initialize this structure to reasonable defaults, call the SNVS_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

```
typedef enum _snvs_hp_ssm_state snvs_hp_ssm_state_t
```

List of SNVS Security State Machine State.

```
static inline void SNVS_HP_EnableMasterKeySelection(SNVS_Type *base, bool enable)
```

Enable or disable master key selection.

Parameters

- base – SNVS peripheral base address
- enable – Pass true to enable, false to disable.

```
static inline void SNVS_HP_ProgramZeroizableMasterKey(SNVS_Type *base)
```

Trigger to program Zeroizable Master Key.

Parameters

- base – SNVS peripheral base address

```
static inline void SNVS_HP_ChangeSSMState(SNVS_Type *base)
```

Trigger SSM State Transition.

Trigger state transition of the system security monitor (SSM). It results only the following transitions of the SSM:

- Check State -> Non-Secure (when Non-Secure Boot and not in Fab Configuration)
- Check State -> Trusted (when Secure Boot or in Fab Configuration)
- Trusted State -> Secure
- Secure State -> Trusted
- Soft Fail -> Non-Secure

Parameters

- base – SNVS peripheral base address

```
static inline void SNVS_HP_SetSoftwareFatalSecurityViolation(SNVS_Type *base)
```

Trigger Software Fatal Security Violation.

The result SSM state transition is:

- Check State -> Soft Fail
- Non-Secure State -> Soft Fail
- Trusted State -> Soft Fail
- Secure State -> Soft Fail

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_SetSoftwareSecurityViolation(SNVS_Type *base)

Trigger Software Security Violation.

The result SSM state transition is:

- Check -> Non-Secure
- Trusted -> Soft Fail
- Secure -> Soft Fail

Parameters

- base – SNVS peripheral base address

static inline snvs_hp_ssm_state_t SNVS_HP_GetSSMState(SNVS_Type *base)

Get current SSM State.

Parameters

- base – SNVS peripheral base address

Returns

Current SSM state

static inline void SNVS_HP_ResetLP(SNVS_Type *base)

Reset the SNVS LP section.

Reset the LP section except SRTC and Time alarm.

Parameters

- base – SNVS peripheral base address

static inline uint32_t SNVS_HP_GetStatusFlags(SNVS_Type *base)

Get the SNVS HP status flags.

The flags are returned as the OR'ed value of the enumeration :: `_snvs_hp_status_flags_t`.

Parameters

- base – SNVS peripheral base address

Returns

The OR'ed value of status flags.

static inline void SNVS_HP_ClearStatusFlags(SNVS_Type *base, uint32_t mask)

Clear the SNVS HP status flags.

The flags to clear are passed in as the OR'ed value of the enumeration :: `_snvs_hp_status_flags_t`. Only these flags could be cleared using this API.

- `kSNVS_RTC_PeriodicInterruptFlag`
- `kSNVS_RTC_AlarmInterruptFlag`

Parameters

- base – SNVS peripheral base address
- mask – OR'ed value of the flags to clear.

static inline uint32_t SNVS_HP_GetSecurityViolationStatusFlags(SNVS_Type *base)

Get the SNVS HP security violation status flags.

The flags are returned as the OR'ed value of the enumeration :: `_snvs_hp_sv_status_flags_t`.

Parameters

- base – SNVS peripheral base address

Returns

The OR'ed value of security violation status flags.

static inline void SNVS_HP_ClearSecurityViolationStatusFlags(SNVS_Type *base, uint32_t mask)

Clear the SNVS HP security violation status flags.

The flags to clear are passed in as the OR'ed value of the enumeration :: `_snvs_hp_sv_status_flags_t`. Only these flags could be cleared using this API.

- `kSNVS_ZMK_EccFailFlag`
- `kSNVS_Violation0Flag`
- `kSNVS_Violation1Flag`
- `kSNVS_Violation2Flag`
- `kSNVS_Violation3Flag`
- `kSNVS_Violation4Flag`
- `kSNVS_Violation5Flag`

Parameters

- `base` – SNVS peripheral base address
- `mask` – OR'ed value of the flags to clear.

`SNVS_HPSVSR_SV0_MASK`

`SNVS_HPSVSR_SV1_MASK`

`SNVS_HPSVSR_SV2_MASK`

`SNVS_HPSVSR_SV4_MASK`

`SNVS_HPSVSR_SV5_MASK`

`SNVS_MAKE_HP_SV_FLAG(x)`

Macro to make security violation flag.

Macro help to make security violation flag `kSNVS_Violation0Flag` to `kSNVS_Violation5Flag`, For example, `SNVS_MAKE_HP_SV_FLAG(0)` is `kSNVS_Violation0Flag`.

struct `_snvs_hp_rtc_datetime`

#include <fsl_snvs_hp.h> Structure is used to hold the date and time.

Public Members

`uint16_t` year

Range from 1970 to 2099.

`uint8_t` month

Range from 1 to 12.

`uint8_t` day

Range from 1 to 31 (depending on month).

`uint8_t` hour

Range from 0 to 23.

`uint8_t` minute

Range from 0 to 59.

uint8_t second

Range from 0 to 59.

struct __snvs__hp__rtc__config

#include <fsl_snvs_hp.h> SNVS config structure.

This structure holds the configuration settings for the SNVS peripheral. To initialize this structure to reasonable defaults, call the SNVS_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

Public Members

bool rtcCalEnable

true: RTC calibration mechanism is enabled; false: No calibration is used

uint32_t rtcCalValue

Defines signed calibration value for nonsecure RTC; This is a 5-bit 2's complement value, range from -16 to +15

uint32_t periodicInterruptFreq

Defines frequency of the periodic interrupt; Range from 0 to 15

2.108 Secure Non-Volatile Storage Low-Power

void SNVS_LP_Init(SNVS_Type *base)

Un gates the SNVS clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the SNVS driver.

Parameters

- base – SNVS peripheral base address

void SNVS_LP_Deinit(SNVS_Type *base)

Deinit the SNVS LP section.

Parameters

- base – SNVS peripheral base address

status_t SNVS_LP_SRTC_SetDatetime(SNVS_Type *base, const snvs_lp_srtc_datetime_t *datetime)

Sets the SNVS SRTC date and time according to the given time structure.

Parameters

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

Returns

kStatus_Success: Success in setting the time and starting the SNVS SRTC
kStatus_InvalidArgument: Error because the datetime format is incorrect

```
void SNVS_LP_SRTC_GetDatetime(SNVS_Type *base, snvs_lp_srtc_datetime_t *datetime)
```

Gets the SNVS SRTC time and stores it in the given time structure.

Parameters

- *base* – SNVS peripheral base address
- *datetime* – Pointer to the structure where the date and time details are stored.

```
status_t SNVS_LP_SRTC_SetAlarm(SNVS_Type *base, const snvs_lp_srtc_datetime_t *alarmTime)
```

Sets the SNVS SRTC alarm time.

The function sets the SRTC alarm. It also checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error. Please note, that SRTC alarm has limited resolution because only 32 most significant bits of SRTC counter are compared to SRTC Alarm register. If the alarm time is beyond SRTC resolution, the function does not set the alarm and returns an error.

Parameters

- *base* – SNVS peripheral base address
- *alarmTime* – Pointer to the structure where the alarm time is stored.

Returns

kStatus_Success: success in setting the SNVS SRTC alarm
kStatus_InvalidArgument: Error because the alarm datetime format is incorrect
kStatus_Fail: Error because the alarm time has already passed or is beyond resolution

```
void SNVS_LP_SRTC_GetAlarm(SNVS_Type *base, snvs_lp_srtc_datetime_t *datetime)
```

Returns the SNVS SRTC alarm time.

Parameters

- *base* – SNVS peripheral base address
- *datetime* – Pointer to the structure where the alarm date and time details are stored.

```
static inline void SNVS_LP_SRTC_EnableInterrupts(SNVS_Type *base, uint32_t mask)
```

Enables the selected SNVS interrupts.

Parameters

- *base* – SNVS peripheral base address
- *mask* – The interrupts to enable. This is a logical OR of members of the enumeration :: *_snvs_lp_srtc_interrupts*

```
static inline void SNVS_LP_SRTC_DisableInterrupts(SNVS_Type *base, uint32_t mask)
```

Disables the selected SNVS interrupts.

Parameters

- *base* – SNVS peripheral base address
- *mask* – The interrupts to enable. This is a logical OR of members of the enumeration :: *_snvs_lp_srtc_interrupts*

```
uint32_t SNVS_LP_SRTC_GetEnabledInterrupts(SNVS_Type *base)
```

Gets the enabled SNVS interrupts.

Parameters

- *base* – SNVS peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration
:: `_snvs_lp_srtc_interrupts`

`uint32_t SNVS_LP_SRTC_GetStatusFlags(SNVS_Type *base)`

Gets the SNVS status flags.

Parameters

- `base` – SNVS peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration ::
`_snvs_lp_srtc_status_flags`

`static inline void SNVS_LP_SRTC_ClearStatusFlags(SNVS_Type *base, uint32_t mask)`

Clears the SNVS status flags.

Parameters

- `base` – SNVS peripheral base address
- `mask` – The status flags to clear. This is a logical OR of members of the enumeration :: `_snvs_lp_srtc_status_flags`

`static inline void SNVS_LP_SRTC_StartTimer(SNVS_Type *base)`

Starts the SNVS SRTC time counter.

Parameters

- `base` – SNVS peripheral base address

`static inline void SNVS_LP_SRTC_StopTimer(SNVS_Type *base)`

Stops the SNVS SRTC time counter.

Parameters

- `base` – SNVS peripheral base address

`void SNVS_LP_EnablePassiveTamper(SNVS_Type *base, snvs_lp_external_tamper_t pin,
snvs_lp_passive_tamper_t config)`

Enables the specified SNVS external tamper.

Parameters

- `base` – SNVS peripheral base address
- `pin` – SNVS external tamper pin
- `config` – Configuration structure of external passive tamper

`status_t SNVS_LP_EnableTxActiveTamper(SNVS_Type *base, snvs_lp_active_tx_tamper_t pin,
tamper_active_tx_config_t config)`

Enable active tamper tx external pad.

Parameters

- `base` – SNVS peripheral base address
- `pin` – SNVS active tamper pin
- `config` – Configuration structure of external active tamper

`status_t SNVS_LP_EnableRxActiveTamper(SNVS_Type *base, snvs_lp_external_tamper_t rx,
tamper_active_rx_config_t config)`

Enable active tamper rx external pad.

Parameters

- `base` – SNVS peripheral base address

- rx – SNVS external RX tamper pin
- config – SNVS RX tamper config structure

status_t SNVS_LP_SetVoltageTamper(SNVS_Type *base, bool enable)

Sets voltage tamper detect.

Parameters

- base – SNVS peripheral base address
- enable – True if enable false if disable

status_t SNVS_LP_SetTemperatureTamper(SNVS_Type *base, bool enable)

Sets temperature tamper detect.

Parameters

- base – SNVS peripheral base address
- enable – True if enable false if disable

status_t SNVS_LP_SetClockTamper(SNVS_Type *base, bool enable)

Sets clock tamper detect.

Parameters

- base – SNVS peripheral base address
- enable – True if enable false if disable

snvs_lp_external_tamper_status_t SNVS_LP_CheckVoltageTamper(SNVS_Type *base)

brief Check voltage tamper

param base SNVS peripheral base address

snvs_lp_external_tamper_status_t SNVS_LP_CheckTemperatureTamper(SNVS_Type *base)

Check temperature tamper.

Parameters

- base – SNVS peripheral base address

snvs_lp_external_tamper_status_t SNVS_LP_CheckClockTamper(SNVS_Type *base)

brief Check clock tamper

param base SNVS peripheral base address

void SNVS_LP_TamperPinTx_GetDefaultConfig(*tamper_active_tx_config_t* *config)

Fills in the SNVS tamper pin config struct with the default settings.

The default values are as follows. code config->clock = kSNVS_ActiveTamper16HZ; config->seed = 0U; config->polynomial = 0U; endcode

Parameters

- config – Pointer to the user's SNVS configuration structure.

void SNVS_LP_TamperPinRx_GetDefaultConfig(*tamper_active_rx_config_t* *config)

brief Fills in the SNVS tamper pin config struct with the default settings.

The default values are as follows. code config->filterenable = 0U; config->filter = 0U; config->tx = kSNVS_ActiveTamper1; endcode param config Pointer to the user's SNVS configuration structure.

void SNVS_LP_PassiveTamperPin_GetDefaultConfig(*snvs_lp_passive_tamper_t* *config)

Fills in the SNVS tamper pin config struct with the default settings.

The default values are as follows. code config->polarity = 0U; config->filterenable = 0U; if available on SoC config->filter = 0U; if available on SoC endcode

Parameters

- config – Pointer to the user's SNVS configuration structure.

void SNVS_LP_DisableExternalTamper(SNVS_Type *base, *snvs_lp_external_tamper_t* pin)

Disables the specified SNVS external tamper.

Parameters

- base – SNVS peripheral base address
- pin – SNVS external tamper pin

void SNVS_LP_DisableAllExternalTamper(SNVS_Type *base)

Disable all external tamper.

Parameters

- base – SNVS peripheral base address

snvs_lp_external_tamper_status_t SNVS_LP_GetExternalTamperStatus(SNVS_Type *base,
snvs_lp_external_tamper_t
pin)

Returns status of the specified external tamper.

Parameters

- base – SNVS peripheral base address
- pin – SNVS external tamper pin

Returns

The status flag. This is the enumeration :: *_snvs_lp_external_tamper_status*

void SNVS_LP_ClearExternalTamperStatus(SNVS_Type *base, *snvs_lp_external_tamper_t* pin)

Clears status of the specified external tamper.

Parameters

- base – SNVS peripheral base address
- pin – SNVS external tamper pin

void SNVS_LP_ClearAllExternalTamperStatus(SNVS_Type *base)

Clears status of the all external tamper.

Parameters

- base – SNVS peripheral base address

static inline void SNVS_LP_EnableMonotonicCounter(SNVS_Type *base, bool enable)

Enable or disable the Monotonic Counter.

Parameters

- base – SNVS peripheral base address
- enable – Pass true to enable, false to disable.

uint64_t SNVS_LP_GetMonotonicCounter(SNVS_Type *base)

Get the current Monotonic Counter.

Parameters

- base – SNVS peripheral base address

Returns

Current Monotonic Counter value.

```
static inline void SNVS_LP_IncreaseMonotonicCounter(SNVS_Type *base)
```

Increase the Monotonic Counter.

Increase the Monotonic Counter by 1.

Parameters

- base – SNVS peripheral base address

```
void SNVS_LP_WriteZeroizableMasterKey(SNVS_Type *base, uint32_t ZMKey[8U])
```

Write Zeroizable Master Key (ZMK) to the SNVS registers.

Parameters

- base – SNVS peripheral base address
- ZMKey – The ZMK write to the SNVS register.

```
static inline void SNVS_LP_SetZeroizableMasterKeyValid(SNVS_Type *base, bool valid)
```

Set Zeroizable Master Key valid.

This API could only be called when using software programming mode. After writing ZMK using SNVS_LP_WriteZeroizableMasterKey, call this API to make the ZMK valid.

Parameters

- base – SNVS peripheral base address
- valid – Pass true to set valid, false to set invalid.

```
static inline bool SNVS_LP_GetZeroizableMasterKeyValid(SNVS_Type *base)
```

Get Zeroizable Master Key valid status.

In hardware programming mode, call this API to check whether the ZMK is valid.

Parameters

- base – SNVS peripheral base address

Returns

true if valid, false if invalid.

```
static inline void SNVS_LP_SetZeroizableMasterKeyProgramMode(SNVS_Type *base,  
                                                             snvs_lp_zmk_program_mode_t  
                                                             mode)
```

Set Zeroizable Master Key programming mode.

Parameters

- base – SNVS peripheral base address
- mode – ZMK programming mode.

```
static inline void SNVS_LP_EnableZeroizableMasterKeyECC(SNVS_Type *base, bool enable)
```

Enable or disable Zeroizable Master Key ECC.

Parameters

- base – SNVS peripheral base address
- enable – Pass true to enable, false to disable.

```
static inline void SNVS_LP_SetMasterKeyMode(SNVS_Type *base, snvs_lp_master_key_mode_t  
                                             mode)
```

Set SNVS Master Key mode.

Note: When kSNVS_ZMK or kSNVS_CMK used, the SNVS_HP must be configured to enable the master key selection.

Parameters

- base – SNVS peripheral base address
- mode – Master Key mode.

FSL_SNVS_LP_DRIVER_VERSION

Version 2.4.6

enum _snvs_lp_srtc_interrupts

List of SNVS_LP interrupts.

Values:

enumerator kSNVS_SRTC_AlarmInterrupt

SRTC time alarm.

enum _snvs_lp_srtc_status_flags

List of SNVS_LP flags.

Values:

enumerator kSNVS_SRTC_AlarmInterruptFlag

SRTC time alarm flag

enum _snvs_lp_external_tamper

List of SNVS_LP external tampers.

Values:

enumerator kSNVS_ExternalTamper1

enum _snvs_lp_active_tamper

List of SNVS_LP active tampers.

Values:

enumerator kSNVS_ActiveTamper1

enumerator kSNVS_ActiveTamper2

enumerator kSNVS_ActiveTamper3

enumerator kSNVS_ActiveTamper4

enumerator kSNVS_ActiveTamper5

enum _snvs_lp_active_clock

List of SNVS_LP external tampers.

Values:

enumerator kSNVS_ActiveTamper16HZ

enumerator kSNVS_ActiveTamper8HZ

enumerator kSNVS_ActiveTamper4HZ

enumerator kSNVS_ActiveTamper2HZ

enum _snvs_lp_external_tamper_status

List of SNVS_LP external tampers status.

Values:

enumerator kSNVS_TamperNotDetected

enumerator kSNVS_TamperDetected

enum _snvs_lp_external_tamper_polarity
SNVS_LP external tamper polarity.

Values:

enumerator kSNVS_ExternalTamperActiveLow

enumerator kSNVS_ExternalTamperActiveHigh

enum _snvs_lp_zmk_program_mode
SNVS_LP Zeroizable Master Key programming mode.

Values:

enumerator kSNVS_ZMKSoftwareProgram
Software programming mode.

enumerator kSNVS_ZMKHardwareProgram
Hardware programming mode.

enum _snvs_lp_master_key_mode
SNVS_LP Master Key mode.

Values:

enumerator kSNVS_OTPMK
One Time Programmable Master Key.

enumerator kSNVS_ZMK
Zeroizable Master Key.

enumerator kSNVS_CMK
Combined Master Key, it is XOR of OPTMK and ZMK.

typedef enum _snvs_lp_srtc_interrupts snvs_lp_srtc_interrupts_t
List of SNVS_LP interrupts.

typedef enum _snvs_lp_srtc_status_flags snvs_lp_srtc_status_flags_t
List of SNVS_LP flags.

typedef enum _snvs_lp_external_tamper snvs_lp_external_tamper_t
List of SNVS_LP external tampers.

typedef enum _snvs_lp_active_tamper snvs_lp_active_tx_tamper_t
List of SNVS_LP active tampers.

typedef enum _snvs_lp_active_clock snvs_lp_active_clock_t
List of SNVS_LP external tampers.

typedef enum _snvs_lp_external_tamper_status snvs_lp_external_tamper_status_t
List of SNVS_LP external tampers status.

typedef enum _snvs_lp_external_tamper_polarity snvs_lp_external_tamper_polarity_t
SNVS_LP external tamper polarity.

typedef struct _snvs_lp_srtc_datetime snvs_lp_srtc_datetime_t
Structure is used to hold the date and time.

```
typedef struct _snvs_lp_srtc_config snvs_lp_srtc_config_t
```

SNVS_LP config structure.

This structure holds the configuration settings for the SNVS_LP peripheral. To initialize this structure to reasonable defaults, call the SNVS_LP_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

```
typedef enum _snvs_lp_zmk_program_mode snvs_lp_zmk_program_mode_t
```

SNVS_LP Zeroizable Master Key programming mode.

```
typedef enum _snvs_lp_master_key_mode snvs_lp_master_key_mode_t
```

SNVS_LP Master Key mode.

```
void SNVS_LP_SRTC_Init(SNVS_Type *base, const snvs_lp_srtc_config_t *config)
```

Ungates the SNVS clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the SNVS driver.

Parameters

- base – SNVS peripheral base address
- config – Pointer to the user's SNVS configuration structure.

```
void SNVS_LP_SRTC_Deinit(SNVS_Type *base)
```

Stops the SRTC timer.

Parameters

- base – SNVS peripheral base address

```
void SNVS_LP_SRTC_GetDefaultConfig(snvs_lp_srtc_config_t *config)
```

Fills in the SNVS_LP config struct with the default settings.

The default values are as follows.

```
config->srtccalenable = false;  
config->srtccalvalue = 0U;
```

Parameters

- config – Pointer to the user's SNVS configuration structure.

```
SNVS_ZMK_REG_COUNT
```

Define of SNVS_LP Zeroizable Master Key registers.

```
SNVS_LP_MAX_TAMPER
```

Define of SNVS_LP Max possible tamper.

```
struct tamper_active_tx_config_t
```

#include <fsl_snvs_lp.h> Structure is used to configure SNVS LP active TX tamper pins.

```
struct tamper_active_rx_config_t
```

#include <fsl_snvs_lp.h> Structure is used to configure SNVS LP active RX tamper pins.

```
struct snvs_lp_passive_tamper_t
```

#include <fsl_snvs_lp.h> Structure is used to configure SNVS LP passive tamper pins.

```
struct _snvs_lp_srtc_datetime
```

#include <fsl_snvs_lp.h> Structure is used to hold the date and time.

Public Members

uint16_t year

Range from 1970 to 2099.

uint8_t month

Range from 1 to 12.

uint8_t day

Range from 1 to 31 (depending on month).

uint8_t hour

Range from 0 to 23.

uint8_t minute

Range from 0 to 59.

uint8_t second

Range from 0 to 59.

struct _snvs_lp_srtc_config

#include <fsl_snvs_lp.h> SNVS_LP config structure.

This structure holds the configuration settings for the SNVS_LP peripheral. To initialize this structure to reasonable defaults, call the SNVS_LP_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

Public Members

bool srtcCalEnable

true: SRTC calibration mechanism is enabled; false: No calibration is used

uint32_t srtcCalValue

Defines signed calibration value for SRTC; This is a 5-bit 2's complement value, range from -16 to +15

2.109 Soc_mipi_csi2rx

void MIPI_CSI2RX_SoftwareReset(MIPI_CSI2RX_Type *base, bool reset)

Assert or deassert CSI2RX reset in system level.

Note: Don't call this function directly.

Parameters

- base – The CSI2RX peripheral base address.
- reset – Pass in true to set to reset state, false to release reset.

void MIPI_CSI2RX_InitInterface(MIPI_CSI2RX_Type *base, uint8_t tHsSettle_EscClk)

Initialize the CSI2RX interface.

Note: Don't call this function directly.

Parameters

- base – The CSI2RX peripheral base address.
- tHsSettle_EscClk – t-HS_SETTLE in esc clock period.

void MIPI_CSI2RX_DeinitInterface(MIPI_CSI2RX_Type *base)

Deinitialize the CSI2RX interface.

Note: Don't call this function directly.

Parameters

- base – The CSI2RX peripheral base address.

FSL_SOC_MIPI_CSI2RX_DRIVER_VERSION

Driver version.

2.110 Soc_mipi_dsi

FSL_SOC_MIPI_DSI_DRIVER_VERSION

Driver version 2.0.0.

DSI_DPHY_PLL_VCO_MAX

DSI_DPHY_PLL_VCO_MIN

FSL_COMPONENT_ID

2.111 SPDIF: Sony/Philips Digital Interface

void SPDIF_Init(SPDIF_Type *base, const *spdif_config_t* *config)

Initializes the SPDIF peripheral.

Un gates the SPDIF clock, resets the module, and configures SPDIF with a configuration structure. The configuration structure can be custom filled or set with default values by SPDIF_GetDefaultConfig().

Note: This API should be called at the beginning of the application to use the SPDIF driver. Otherwise, accessing the SPDIF module can cause a hard fault because the clock is not enabled.

Parameters

- base – SPDIF base pointer
- config – SPDIF configuration structure.

void SPDIF_GetDefaultConfig(*spdif_config_t* *config)

Sets the SPDIF configuration structure to default values.

This API initializes the configuration structure for use in SPDIF_Init. The initialized structure can remain unchanged in SPDIF_Init, or it can be modified before calling SPDIF_Init. This is an example.

```
spdif_config_t config;
SPDIF_GetDefaultConfig(&config);
```

Parameters

- config – pointer to master configuration structure

void SPDIF_Deinit(SPDIF_Type *base)

De-initializes the SPDIF peripheral.

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

Parameters

- base – SPDIF base pointer

uint32_t SPDIF_GetInstance(SPDIF_Type *base)

Get the instance number for SPDIF.

Parameters

- base – SPDIF base pointer.

static inline void SPDIF_TxFIFOReset(SPDIF_Type *base)

Resets the SPDIF Tx.

This function makes Tx FIFO in reset mode.

Parameters

- base – SPDIF base pointer

static inline void SPDIF_RxFIFOReset(SPDIF_Type *base)

Resets the SPDIF Rx.

This function enables the software reset and FIFO reset of SPDIF Rx. After reset, clear the reset bit.

Parameters

- base – SPDIF base pointer

void SPDIF_TxEnable(SPDIF_Type *base, bool enable)

Enables/disables the SPDIF Tx.

Parameters

- base – SPDIF base pointer
- enable – True means enable SPDIF Tx, false means disable.

static inline void SPDIF_RxEnable(SPDIF_Type *base, bool enable)

Enables/disables the SPDIF Rx.

Parameters

- base – SPDIF base pointer
- enable – True means enable SPDIF Rx, false means disable.

static inline uint32_t SPDIF_GetStatusFlag(SPDIF_Type *base)

Gets the SPDIF status flag state.

Parameters

- base – SPDIF base pointer

Returns

SPDIF status flag value. Use the `_spdif_interrupt_enable_t` to get the status value needed.

```
static inline void SPDIF__ClearStatusFlags(SPDIF_Type *base, uint32_t mask)
```

Clears the SPDIF status flag state.

Parameters

- `base` – SPDIF base pointer
- `mask` – State mask. It can be a combination of the `_spdif_interrupt_enable_t` member. Notice these members cannot be included, as these flags cannot be cleared by writing 1 to these bits:
 - `kSPDIF_UChannelReceiveRegisterFull`
 - `kSPDIF_QChannelReceiveRegisterFull`
 - `kSPDIF_TxFIFOEmpty`
 - `kSPDIF_RxFIFOFull`

```
static inline void SPDIF__EnableInterrupts(SPDIF_Type *base, uint32_t mask)
```

Enables the SPDIF Tx interrupt requests.

Parameters

- `base` – SPDIF base pointer
- `mask` – interrupt source The parameter can be a combination of the following sources if defined.
 - `kSPDIF_WordStartInterruptEnable`
 - `kSPDIF_SyncErrorInterruptEnable`
 - `kSPDIF_FIFOWarningInterruptEnable`
 - `kSPDIF_FIFORequestInterruptEnable`
 - `kSPDIF_FIFOErrorInterruptEnable`

```
static inline void SPDIF__DisableInterrupts(SPDIF_Type *base, uint32_t mask)
```

Disables the SPDIF Tx interrupt requests.

Parameters

- `base` – SPDIF base pointer
- `mask` – interrupt source The parameter can be a combination of the following sources if defined.
 - `kSPDIF_WordStartInterruptEnable`
 - `kSPDIF_SyncErrorInterruptEnable`
 - `kSPDIF_FIFOWarningInterruptEnable`
 - `kSPDIF_FIFORequestInterruptEnable`
 - `kSPDIF_FIFOErrorInterruptEnable`

```
static inline void SPDIF__EnableDMA(SPDIF_Type *base, uint32_t mask, bool enable)
```

Enables/disables the SPDIF DMA requests.

Parameters

- `base` – SPDIF base pointer
- `mask` – SPDIF DMA enable mask, The parameter can be a combination of the following sources if defined

- kSPDIF_RxDMAEnable
- kSPDIF_TxDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t SPDIF__TxGetLeftDataRegisterAddress(SPDIF_Type *base)

Gets the SPDIF Tx left data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

- base – SPDIF base pointer.

Returns

data register address.

static inline uint32_t SPDIF__TxGetRightDataRegisterAddress(SPDIF_Type *base)

Gets the SPDIF Tx right data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

- base – SPDIF base pointer.

Returns

data register address.

static inline uint32_t SPDIF__RxGetLeftDataRegisterAddress(SPDIF_Type *base)

Gets the SPDIF Rx left data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

- base – SPDIF base pointer.

Returns

data register address.

static inline uint32_t SPDIF__RxGetRightDataRegisterAddress(SPDIF_Type *base)

Gets the SPDIF Rx right data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

- base – SPDIF base pointer.

Returns

data register address.

void SPDIF__TxSetSampleRate(SPDIF_Type *base, uint32_t sampleRate_Hz, uint32_t sourceClockFreq_Hz)

Configures the SPDIF Tx sample rate.

The audio format can be changed at run-time. This function configures the sample rate.

Parameters

- base – SPDIF base pointer.
- sampleRate_Hz – SPDIF sample rate frequency in Hz.
- sourceClockFreq_Hz – SPDIF tx clock source frequency in Hz.

uint32_t SPDIF_GetRxSampleRate(SPDIF_Type *base, uint32_t clockSourceFreq_Hz)

Configures the SPDIF Rx audio format.

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – SPDIF base pointer.
- clockSourceFreq_Hz – SPDIF system clock frequency in hz.

void SPDIF_WriteBlocking(SPDIF_Type *base, uint8_t *buffer, uint32_t size)

Sends data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SPDIF base pointer.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

static inline void SPDIF_WriteLeftData(SPDIF_Type *base, uint32_t data)

Writes data into SPDIF FIFO.

Parameters

- base – SPDIF base pointer.
- data – Data needs to be written.

static inline void SPDIF_WriteRightData(SPDIF_Type *base, uint32_t data)

Writes data into SPDIF FIFO.

Parameters

- base – SPDIF base pointer.
- data – Data needs to be written.

static inline void SPDIF_WriteChannelStatusHigh(SPDIF_Type *base, uint32_t data)

Writes data into SPDIF FIFO.

Parameters

- base – SPDIF base pointer.
- data – Data needs to be written.

static inline void SPDIF_WriteChannelStatusLow(SPDIF_Type *base, uint32_t data)

Writes data into SPDIF FIFO.

Parameters

- base – SPDIF base pointer.
- data – Data needs to be written.

void SPDIF_ReadBlocking(SPDIF_Type *base, uint8_t *buffer, uint32_t size)

Receives data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SPDIF base pointer.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

```
static inline uint32_t SPDIF_ReadLeftData(SPDIF_Type *base)
```

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns

Data in SPDIF FIFO.

```
static inline uint32_t SPDIF_ReadRightData(SPDIF_Type *base)
```

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns

Data in SPDIF FIFO.

```
static inline uint32_t SPDIF_ReadChannelStatusHigh(SPDIF_Type *base)
```

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns

Data in SPDIF FIFO.

```
static inline uint32_t SPDIF_ReadChannelStatusLow(SPDIF_Type *base)
```

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns

Data in SPDIF FIFO.

```
static inline uint32_t SPDIF_ReadQChannel(SPDIF_Type *base)
```

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns

Data in SPDIF FIFO.

```
static inline uint32_t SPDIF_ReadUChannel(SPDIF_Type *base)
```

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns

Data in SPDIF FIFO.

```
void SPDIF__TransferTxCreateHandle(SPDIF_Type *base, spdif_handle_t *handle,  
                                  spdif_transfer_callback_t callback, void *userData)
```

Initializes the SPDIF Tx handle.

This function initializes the Tx handle for the SPDIF Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SPDIF base pointer
- handle – SPDIF handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function

```
void SPDIF__TransferRxCreateHandle(SPDIF_Type *base, spdif_handle_t *handle,  
                                  spdif_transfer_callback_t callback, void *userData)
```

Initializes the SPDIF Rx handle.

This function initializes the Rx handle for the SPDIF Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SPDIF base pointer.
- handle – SPDIF handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function.

```
status_t SPDIF__TransferSendNonBlocking(SPDIF_Type *base, spdif_handle_t *handle,  
                                         spdif_transfer_t *xfer)
```

Performs an interrupt non-blocking send transfer on SPDIF.

Note: This API returns immediately after the transfer initiates. Call the SPDIF_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SPDIF_Busy, the transfer is finished.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the *spdif_handle_t* structure which stores the transfer state.
- xfer – Pointer to the *spdif_transfer_t* structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SPDIF_TxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

```
status_t SPDIF__TransferReceiveNonBlocking(SPDIF_Type *base, spdif_handle_t *handle,  
                                           spdif_transfer_t *xfer)
```

Performs an interrupt non-blocking receive transfer on SPDIF.

Note: This API returns immediately after the transfer initiates. Call the SPDIF_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SPDIF_Busy, the transfer is finished.

Parameters

- base – SPDIF base pointer
- handle – Pointer to the spdif_handle_t structure which stores the transfer state.
- xfer – Pointer to the spdif_transfer_t structure.

Return values

- kStatus__Success – Successfully started the data receive.
- kStatus_SPDIF_RxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SPDIF_TransferGetSendCount(SPDIF_Type *base, *spdif_handle_t* *handle, *size_t* *count)

Gets a set byte count.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the spdif_handle_t structure which stores the transfer state.
- count – Bytes count sent.

Return values

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

status_t SPDIF_TransferGetReceiveCount(SPDIF_Type *base, *spdif_handle_t* *handle, *size_t* *count)

Gets a received byte count.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the spdif_handle_t structure which stores the transfer state.
- count – Bytes count received.

Return values

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

void SPDIF_TransferAbortSend(SPDIF_Type *base, *spdif_handle_t* *handle)

Aborts the current send.

Note: This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the `spdif_handle_t` structure which stores the transfer state.

`void SPDIF__TransferAbortReceive(SPDIF_Type *base, spdif_handle_t *handle)`

Aborts the current IRQ receive.

Note: This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – SPDIF base pointer
- handle – Pointer to the `spdif_handle_t` structure which stores the transfer state.

`void SPDIF__TransferTxHandleIRQ(SPDIF_Type *base, spdif_handle_t *handle)`

Tx interrupt handler.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the `spdif_handle_t` structure.

`void SPDIF__TransferRxHandleIRQ(SPDIF_Type *base, spdif_handle_t *handle)`

Tx interrupt handler.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the `spdif_handle_t` structure.

`FSL_SPDIF_DRIVER_VERSION`

Version 2.0.7

SPDIF return status.

Values:

enumerator `kStatus_SPDIF_RxDPLLLocked`

SPDIF Rx PLL locked.

enumerator `kStatus_SPDIF_TxFIFOError`

SPDIF Tx FIFO error.

enumerator `kStatus_SPDIF_TxFIFOResync`

SPDIF Tx left and right FIFO resync.

enumerator `kStatus_SPDIF_RxCnew`

SPDIF Rx status channel value updated.

enumerator `kStatus_SPDIF_ValidatyNoGood`

SPDIF validaty flag not good.

enumerator `kStatus_SPDIF_RxIllegalSymbol`

SPDIF Rx receive illegal symbol.

enumerator kStatus_SPDIF_RxParityBitError

SPDIF Rx parity bit error.

enumerator kStatus_SPDIF_UChannelOverrun

SPDIF receive U channel overrun.

enumerator kStatus_SPDIF_QChannelOverrun

SPDIF receive Q channel overrun.

enumerator kStatus_SPDIF_UQChannelSync

SPDIF U/Q channel sync found.

enumerator kStatus_SPDIF_UQChannelFrameError

SPDIF U/Q channel frame error.

enumerator kStatus_SPDIF_RxFIFOError

SPDIF Rx FIFO error.

enumerator kStatus_SPDIF_RxFIFOResync

SPDIF Rx left and right FIFO resync.

enumerator kStatus_SPDIF_LockLoss

SPDIF Rx PLL clock lock loss.

enumerator kStatus_SPDIF_TxIdle

SPDIF Tx is idle

enumerator kStatus_SPDIF_RxIdle

SPDIF Rx is idle

enumerator kStatus_SPDIF_QueueFull

SPDIF queue full

enum _spdif_rxfull_select

SPDIF Rx FIFO full falg select, it decides when assert the rx full flag.

Values:

enumerator kSPDIF_RxFull1Sample

Rx full at least 1 sample in left and right FIFO

enumerator kSPDIF_RxFull4Samples

Rx full at least 4 sample in left and right FIFO

enumerator kSPDIF_RxFull8Samples

Rx full at least 8 sample in left and right FIFO

enumerator kSPDIF_RxFull16Samples

Rx full at least 16 sample in left and right FIFO

enum _spdif_txempty_select

SPDIF tx FIFO EMPTY falg select, it decides when assert the tx empty flag.

Values:

enumerator kSPDIF_TxEmpty0Sample

Tx empty at most 0 sample in left and right FIFO

enumerator kSPDIF_TxEmpty4Samples

Tx empty at most 4 sample in left and right FIFO

enumerator kSPDIF_TxEmpty8Samples

Tx empty at most 8 sample in left and right FIFO

enumerator kSPDIF_TxEmpty12Samples

Tx empty at most 12 sample in left and right FIFO

enum __spdif_uchannel_source

SPDIF U channel source.

Values:

enumerator kSPDIF_NoUChannel

No embedded U channel

enumerator kSPDIF_UChannelFromRx

U channel from receiver, it is CD mode

enumerator kSPDIF_UChannelFromTx

U channel from on chip tx

enum __spdif_gain_select

SPDIF clock gain.

Values:

enumerator kSPDIF_GAIN_24

Gain select is 24

enumerator kSPDIF_GAIN_16

Gain select is 16

enumerator kSPDIF_GAIN_12

Gain select is 12

enumerator kSPDIF_GAIN_8

Gain select is 8

enumerator kSPDIF_GAIN_6

Gain select is 6

enumerator kSPDIF_GAIN_4

Gain select is 4

enumerator kSPDIF_GAIN_3

Gain select is 3

enum __spdif_tx_source

SPDIF tx data source.

Values:

enumerator kSPDIF_txFromReceiver

Tx data directly through SPDIF receiver

enumerator kSPDIF_txNormal

Normal operation, data from processor

enum __spdif_validity_config

SPDIF tx data source.

Values:

enumerator kSPDIF_validityFlagAlwaysSet

Outgoing validity flags always set

enumerator kSPDIF_validityFlagAlwaysClear

Outgoing validity flags always clear

The SPDIF interrupt enable flag.

Values:

enumerator kSPDIF_RxDPLLLocked
SPDIF DPLL locked

enumerator kSPDIF_TxFIFOError
Tx FIFO underrun or overrun

enumerator kSPDIF_TxFIFOResync
Tx FIFO left and right channel resync

enumerator kSPDIF_RxControlChannelChange
SPDIF Rx control channel value changed

enumerator kSPDIF_VValidityFlagNoGood
SPDIF validity flag no good

enumerator kSPDIF_RxIllegalSymbol
SPDIF receiver found illegal symbol

enumerator kSPDIF_RxParityBitError
SPDIF receiver found parity bit error

enumerator kSPDIF_UChannelReceiveRegisterFull
SPDIF U channel receive register full

enumerator kSPDIF_UChannelReceiveRegisterOverrun
SPDIF U channel receive register overrun

enumerator kSPDIF_QChannelReceiveRegisterFull
SPDIF Q channel receive register full

enumerator kSPDIF_QChannelReceiveRegisterOverrun
SPDIF Q channel receive register overrun

enumerator kSPDIF_UQChannelSync
SPDIF U/Q channel sync found

enumerator kSPDIF_UQChannelFrameError
SPDIF U/Q channel frame error

enumerator kSPDIF_RxFIFOError
SPDIF Rx FIFO underrun/overrun

enumerator kSPDIF_RxFIFOResync
SPDIF Rx left and right FIFO resync

enumerator kSPDIF_LockLoss
SPDIF receiver loss of lock

enumerator kSPDIF_TxFIFOEmpty
SPDIF Tx FIFO empty

enumerator kSPDIF_RxFIFOFull
SPDIF Rx FIFO full

enumerator kSPDIF_AllInterrupt
all interrupt

The DMA request sources.

Values:

enumerator kSPDIF_RxDMAEnable

Rx FIFO full

enumerator kSPDIF_TxDMAEnable

Tx FIFO empty

typedef enum *_spdif_rxfull_select* spdif_rxfull_select_t

SPDIF Rx FIFO full falg select, it decides when assert the rx full flag.

typedef enum *_spdif_txempty_select* spdif_txempty_select_t

SPDIF tx FIFO EMPTY falg select, it decides when assert the tx empty flag.

typedef enum *_spdif_uchannel_source* spdif_uchannel_source_t

SPDIF U channel source.

typedef enum *_spdif_gain_select* spdif_gain_select_t

SPDIF clock gain.

typedef enum *_spdif_tx_source* spdif_tx_source_t

SPDIF tx data source.

typedef enum *_spdif_validity_config* spdif_validity_config_t

SPDIF tx data source.

typedef struct *_spdif_config* spdif_config_t

SPDIF user configuration structure.

typedef struct *_spdif_transfer* spdif_transfer_t

SPDIF transfer structure.

typedef struct *_spdif_handle* spdif_handle_t

typedef void (*spdif_transfer_callback_t)(SPDIF_Type *base, *spdif_handle_t* *handle, *status_t* status, void *userData)

SPDIF transfer callback prototype.

SPDIF_XFER_QUEUE_SIZE

SPDIF transfer queue size, user can refine it according to use case.

struct *_spdif_config*

#include <fsl_spdif.h> SPDIF user configuration structure.

Public Members

bool isTxAutoSync

If auto sync mechanism open

bool isRxAutoSync

If auto sync mechanism open

uint8_t DPLLClkSource

SPDIF DPLL clock source, range from 0~15, meaning is chip-specific

uint8_t txClkSource

SPDIF tx clock source, range from 0~7, meaning is chip-specific

spdif_rxfull_select_t rxFullSelect
SPDIF rx buffer full select

spdif_txempty_select_t txFullSelect
SPDIF tx buffer empty select

spdif_uchannel_source_t uChannelSrc
U channel source

spdif_tx_source_t txSource
SPDIF tx data source

spdif_validity_config_t validityConfig
Validity flag config

spdif_gain_select_t gain
Rx receive clock measure gain parameter.

struct *_spdif_transfer*
#include <fsl_spdif.h> SPDIF transfer structure.

Public Members

uint8_t *data
Data start address to transfer.

uint8_t *qdata
Data buffer for Q channel

uint8_t *udata
Data buffer for C channel

size_t dataSize
Transfer size.

struct *_spdif_handle*
#include <fsl_spdif.h> SPDIF handle structure.

Public Members

uint32_t state
Transfer status

spdif_transfer_callback_t callback
Callback function called at transfer event

void *userData
Callback parameter passed to callback function

spdif_transfer_t spdifQueue[(4U)]
Transfer queue storing queued transfer

size_t transferSize[(4U)]
Data bytes need to transfer

volatile uint8_t queueUser
Index for user to queue transfer

volatile uint8_t queueDriver
Index for driver to get the transfer data and size

uint8_t watermark
Watermark value

2.112 SPDIF eDMA Driver

```
void SPDIF__TransferTxCreateHandleEDMA(SPDIF_Type *base, spdif_edma_handle_t *handle,  
                                       spdif_edma_callback_t callback, void *userData,  
                                       edma_handle_t *dmaLeftHandle, edma_handle_t  
                                       *dmaRightHandle)
```

Initializes the SPDIF eDMA handle.

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

Parameters

- base – SPDIF base pointer.
- handle – SPDIF eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- dmaLeftHandle – eDMA handle pointer for left channel, this handle shall be static allocated by users.
- dmaRightHandle – eDMA handle pointer for right channel, this handle shall be static allocated by users.

```
void SPDIF__TransferRxCreateHandleEDMA(SPDIF_Type *base, spdif_edma_handle_t *handle,  
                                       spdif_edma_callback_t callback, void *userData,  
                                       edma_handle_t *dmaLeftHandle, edma_handle_t  
                                       *dmaRightHandle)
```

Initializes the SPDIF Rx eDMA handle.

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

Parameters

- base – SPDIF base pointer.
- handle – SPDIF eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- dmaLeftHandle – eDMA handle pointer for left channel, this handle shall be static allocated by users.
- dmaRightHandle – eDMA handle pointer for right channel, this handle shall be static allocated by users.

```
status_t SPDIF__TransferSendEDMA(SPDIF_Type *base, spdif_edma_handle_t *handle,  
                                 spdif_edma_transfer_t *xfer)
```

Performs a non-blocking SPDIF transfer using DMA.

Note: This interface returns immediately after the transfer initiates. Call SPDIF_GetTransferStatus to poll the transfer status and check whether the SPDIF transfer is finished.

Parameters

- base – SPDIF base pointer.
- handle – SPDIF eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure.

Return values

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

status_t SPDIF_TransferReceiveEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle, *spdif_edma_transfer_t* *xfer)

Performs a non-blocking SPDIF receive using eDMA.

Note: This interface returns immediately after the transfer initiates. Call the SPDIF_GetReceiveRemainingBytes to poll the transfer status and check whether the SPDIF transfer is finished.

Parameters

- base – SPDIF base pointer
- handle – SPDIF eDMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a SPDIF eDMA receive successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_RxBusy – SPDIF is busy receiving data.

void SPDIF_TransferAbortSendEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle)

Aborts a SPDIF transfer using eDMA.

Parameters

- base – SPDIF base pointer.
- handle – SPDIF eDMA handle pointer.

void SPDIF_TransferAbortReceiveEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle)

Aborts a SPDIF receive using eDMA.

Parameters

- base – SPDIF base pointer
- handle – SPDIF eDMA handle pointer.

status_t SPDIF_TransferGetSendCountEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle, *size_t* *count)

Gets byte count sent by SPDIF.

Parameters

- `base` – SPDIF base pointer.
- `handle` – SPDIF eDMA handle pointer.
- `count` – Bytes count sent by SPDIF.

Return values

- `kStatus_Success` – Succeed get the transfer count.
- `kStatus_NoTransferInProgress` – There is no non-blocking transaction in progress.

`status_t SPDIF_TransferGetReceiveCountEDMA(SPDIF_Type *base, spdif_edma_handle_t *handle, size_t *count)`

Gets byte count received by SPDIF.

Parameters

- `base` – SPDIF base pointer
- `handle` – SPDIF eDMA handle pointer.
- `count` – Bytes count received by SPDIF.

Return values

- `kStatus_Success` – Succeed get the transfer count.
- `kStatus_NoTransferInProgress` – There is no non-blocking transaction in progress.

`FSL_SPDIF_EDMA_DRIVER_VERSION`

Version 2.0.8

`typedef struct spdif_edma_handle spdif_edma_handle_t`

`typedef void (*spdif_edma_callback_t)(SPDIF_Type *base, spdif_edma_handle_t *handle, status_t status, void *userData)`

SPDIF eDMA transfer callback function for finish and error.

`typedef struct spdif_edma_transfer spdif_edma_transfer_t`

SPDIF transfer structure.

`struct _spdif_edma_transfer`

`#include <fsl_spdif_edma.h>` SPDIF transfer structure.

Public Members

`uint8_t *leftData`

Data start address to transfer.

`uint8_t *rightData`

Data start address to transfer.

`size_t dataSize`

Transfer size.

`struct _spdif_edma_handle`

`#include <fsl_spdif_edma.h>` SPDIF DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *dmaLeftHandle
DMA handler for SPDIF left channel

edma_handle_t *dmaRightHandle
DMA handler for SPDIF right channel

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 SPDIF eDMA transfer

spdif_edma_callback_t callback
Callback for users while transfer finish or error occurs

void *userData
User callback parameter

edma_tcd_t leftTcd[(4U) + 1U]
TCD pool for eDMA transfer.

edma_tcd_t rightTcd[(4U) + 1U]
TCD pool for eDMA transfer.

spdif_edma_transfer_t spdifQueue[(4U)]
Transfer queue storing queued transfer.

size_t transferSize[(4U)]
Data bytes need to transfer, left and right are the same, so use one

volatile uint8_t queueUser
Index for user to queue transfer.

volatile uint8_t queueDriver
Index for driver to get the transfer data and size

2.113 SSARC: State Save and Restore Controller

```
static inline uint32_t SSARC_GetDescriptorRegisterAddress(SSARC_HP_Type *base, uint32_t
                                                         index)
```

Gets the address of the register to be saved/restored.

Parameters

- base – SSARC_HP peripheral base address.
- index – The index of descriptor. Range from 0 to 1023.

Returns

The address of the register.

```
static inline uint32_t SSARC_GetDescriptorRegisterData(SSARC_HP_Type *base, uint32_t index)
```

Gets the value of the register to be saved/restored.

Parameters

- base – SSARC_HP peripheral base address.

- index – The index of descriptor. Range from 0 to 1023.

Returns

The value of the register.

```
void SSARC_SetDescriptorConfig(SSARC_HP_Type *base, uint32_t index, const
                               ssarc_descriptor_config_t *config)
```

Sets the configuration of the descriptor.

Parameters

- base – SSARC_HP peripheral base address.
- index – The index of descriptor. Range from 0 to 1023.
- config – Pointer to the structure `ssarc_descriptor_config_t`. Please refer to `ssarc_descriptor_config_t` for details.

```
void SSARC_GroupInit(SSARC_LP_Type *base, uint8_t groupID, const ssarc_group_config_t
                    *config)
```

Initializes the selected group.

Note: For the groups with the same save priority or restore priority, the save/restore operation runs in the group order.

Parameters

- base – SSARC_LP peripheral base address.
- groupID – The index of the group. Range from 0 to 15.
- config – Pointer to the structure `ssarc_group_config_t`. Please refer to `ssarc_group_config_t` for details.

```
static inline void SSARC_GroupDeinit(SSARC_LP_Type *base, uint8_t groupID)
```

De-initializes the selected group.

Parameters

- base – SSARC_LP peripheral base address.
- groupID – The index of the group. Range from 0 to 15.

```
static inline void SSARC_LockGroupDomain(SSARC_LP_Type *base, uint8_t groupID)
```

Locks the configuration of the domain.

This function locks the configuration of the domain. Once locked, only the access from the same domain is allowed, access from other domains will be blocked. Once locked, it can only be unlocked by a hardware reset.

Parameters

- base – SSARC_LP peripheral base address.
- groupID – The index of the group. Range from 0 to 15.

```
static inline void SSARC_LockGroupWrite(SSARC_LP_Type *base, uint8_t groupID)
```

Locks the write access to the control registers and descriptors for the selected group.

This function locks the write access to the control registers and descriptors for the selected group. All writes are blocked. Once locked, it can only be unlocked by a hardware reset.

Parameters

- base – SSARC_LP peripheral base address.
- groupID – The index of the group. Range from 0 to 15.

```
static inline void SSARC_LockGroupRead(SSARC_LP_Type *base, uint8_t groupID)
```

Locks the read access to the control registers and descriptors for the selected group.

This function Locks the read access to the control registers and descriptors for the selected group. All reads are blocked. Once locked, it can only be unlocked by a hardware reset.

Parameters

- base – SSARC_LP peripheral base address.
- groupID – The index of the group. Range from 0 to 15.

```
void SSARC_TriggerSoftwareRequest(SSARC_LP_Type *base, uint8_t groupID,  
                                ssarc_software_trigger_mode_t mode)
```

Triggers software request.

Note: Each group allows software to trigger the save/restore operation without getting the request from basic power controller.

Parameters

- base – SSARC_LP peripheral base address.
- groupID – The index of the group. Range from 0 to 15.
- mode – Software trigger mode. Please refer to `ssarc_software_trigger_mode_t` for details.

```
static inline void SSARC_ResetWholeBlock(SSARC_LP_Type *base)
```

Resets the whole SSARC block by software.

Note: Only reset the SSARC registers, not include the DESC in SRAM.

Parameters

- base – SSARC_LP peripheral base address.

```
static inline void SSARC_EnableHardwareRequest(SSARC_LP_Type *base, bool enable)
```

Enables/Disables save/restore request from the PGMC module.

Parameters

- base – SSARC_LP peripheral base address.
- enable – Used to enable/disable save/restore hardware request.
 - **true** Enable GPC save/restore requests.
 - **false** Disable GPC save/restore requests.

```
static inline uint32_t SSARC_GetStatusFlags(SSARC_LP_Type *base)
```

Gets status flags.

Parameters

- base – SSARC_LP peripheral base address.

Returns

The value of status flags. See `_ssarc_interrupt_status_flags` for details.

```
static inline void SSARC_ClearStatusFlags(SSARC_LP_Type *base, uint32_t mask)
```

Clears status flags.

Note: Only kSSARC_AddressErrorFlag, kSSARC_AHBErrorFlag, kSSARC_TimeoutFlag and kSSARC_GroupConflictFlag can be cleared.

Parameters

- base – SSARC_LP peripheral base address.
- mask – The mask value for flags to be cleared. See `_ssarc_interrupt_status_flags` for details.

```
static inline uint32_t SSARC_GetErrorIndex(SSARC_LP_Type *base)
```

Gets the error index that indicates which descriptor will trigger the AHB_ERR or ADDR_ERR interrupt.

Parameters

- base – SSARC_LP peripheral base address.

Returns

The error index.

```
static inline void SSARC_SetTimeoutValue(SSARC_LP_Type *base, uint32_t value)
```

Sets timeout value for the entire group to complete.

This function sets timeout value for the entire group to complete. Setting timeout value to 0 will disable this feature.

Parameters

- base – SSARC_LP peripheral base address.
- value – The timeout value, 0 means disable time out feature.

```
static inline uint32_t SSARC_GetTimeoutValue(SSARC_LP_Type *base)
```

Gets timeout value for AHB clock.

Parameters

- base – SSARC_LP peripheral base address.

Returns

The timeout value.

```
static inline uint16_t SSARC_GetHardwareRequestRestorePendingGroup(SSARC_LP_Type *base)
```

Gets the value that indicates which groups are pending for restore from hardware request.

Parameters

- base – SSARC_LP peripheral base address.

Returns

The value of the pending groups.

```
static inline uint16_t SSARC_GetHardwareRequestSavePendingGroup(SSARC_LP_Type *base)
```

Gets the value that indicates which groups are pending for save from hardware request.

Parameters

- base – SSARC_LP peripheral base address.

Returns

The value of the pending groups.

```
static inline uint16_t SSARC_GetSoftwareRequestRestorePendingGroup(SSARC_LP_Type *base)
```

Gets the value that indicates which groups are pending for restore from software request.

Parameters

- base – SSARC_LP peripheral base address.

Returns

The value of the pending groups.

```
static inline uint16_t SSARC_GetSoftwareRequestSavePendingGroup(SSARC_LP_Type *base)
```

Gets the value that indicates which groups are pending for save from software request.

Parameters

- base – SSARC_LP peripheral base address.

Returns

The value of the pending groups.

```
FSL_SSARC_DRIVER_VERSION
```

SSARC driver version 2.1.0.

```
enum __ssarc_interrupt_status_flags
```

The enumeration of ssarc status flags.

Values:

```
enumerator kSSARC_AddressErrorFlag
```

If the descriptor is not in the range, assert address error.

```
enumerator kSSARC_AHBErrorFlag
```

If any AHB master access receives none-OKAY, assert AHB error.

```
enumerator kSSARC_SoftwareRequestDoneFlag
```

If a software triggered save or restore process is completed, assert software request done .

```
enumerator kSSARC_TimeoutFlag
```

If processing of a group has exceeded the timeout value, assert timeout.

```
enumerator kSSARC_GroupConflictFlag
```

Group conflict.

```
enum __ssarc_descriptor_register_size
```

The size of the register to be saved/restored.

Values:

```
enumerator kSSARC_DescriptorRegister8bitWidth
```

The register to be saved/restored is 8 bit width.

```
enumerator kSSARC_DescriptorRegister16bitWidth
```

The register to be saved/restored is 16 bit width.

```
enumerator kSSARC_DescriptorRegister32bitWidth
```

The register to be saved/restored is 32 bit width.

```
enum __ssarc_descriptor_operation
```

The operation of the descriptor.

Values:

```
enumerator kSSARC_SaveDisableRestoreDisable
```

Disable Save operation, disable restore operation.

enumerator kSSARC_SaveEnableRestoreDisable

Enable Save operation, disable restore operation.

enumerator kSSARC_SaveDisableRestoreEnable

Disable Save operation, enable restore operation.

enumerator kSSARC_SaveEnableRestoreEnable

Enable Save operation, enable restore operation.

enum _ssarc_descriptor_type

The type of operation.

Values:

enumerator kSSARC_ReadValueWriteBack

Read the register value on save operation and write it back on restore operation

enumerator kSSARC_WriteFixedValue

Always write a fixed value from DATA[31:0]

enumerator kSSARC_RMWOOr

Read register, OR with the DATA[31:0], and write it back

enumerator kSSARC_RMWAnd

Read register, AND with the DATA[31:0], and write it back

enumerator kSSARC_DelayCycles

Delay for number of cycles based on the DATA[31:0]

enumerator kSSARC_Polling0

Read the register until read_data[31:0] & DATA[31:0] == 0

enumerator kSSARC_Polling1

Read the register until read_data[31:0] & DATA[31:0] != 0

enum _ssarc_save_restore_order

The order of the restore/save operation.

Values:

enumerator kSSARC_ProcessFromStartToEnd

Descriptors within the group are processed from start to end.

enumerator kSSARC_ProcessFromEndToStart

Descriptors within the group are processed from end to start.

enum _ssarc_software_trigger_mode

Software trigger mode.

Values:

enumerator kSSARC_TriggerSaveRequest

Don't trigger restore operation, trigger the save operation by software.

enumerator kSSARC_TriggerRestoreRequest

Trigger the restore operation, don't trigger the save operation.

typedef enum _ssarc_descriptor_register_size ssarc_descriptor_register_size_t

The size of the register to be saved/restored.

typedef enum _ssarc_descriptor_operation ssarc_descriptor_operation_t

The operation of the descriptor.

`typedef enum _ssarc_descriptor_type ssarc_descriptor_type_t`

The type of operation.

`typedef enum _ssarc_save_restore_order ssarc_save_restore_order_t`

The order of the restore/save operation.

`typedef enum _ssarc_software_trigger_mode ssarc_software_trigger_mode_t`

Software trigger mode.

`typedef struct _ssarc_descriptor_config ssarc_descriptor_config_t`

The configuration of descriptor.

`typedef struct _ssarc_group_config ssarc_group_config_t`

The configuration of the group.

`SSARC_INT_STATUS_ALL`

`struct _ssarc_descriptor_config`

`#include <fsl_ssarc.h>` The configuration of descriptor.

Public Members

`uint32_t address`

The address of the register/memory to be saved/restored.

`uint32_t data`

The value of the register/memory to be saved/restored, please note that if the type is selected as `kSSARC_ReadValueWriteBack`, this data field is useless.

`ssarc_descriptor_register_size_t size`

The size of register to be saved/restored.

`ssarc_descriptor_operation_t operation`

The operation mode of descriptor.

`ssarc_descriptor_type_t type`

The type of operation.

`struct _ssarc_group_config`

`#include <fsl_ssarc.h>` The configuration of the group.

Public Members

`ssarc_cpu_domain_name_t cpuDomain`

CPU domain, define the ownership of this group.

`uint32_t startIndex`

The index of the first descriptor of the group.

`uint32_t endIndex`

The index of the last descriptor of the group.

`ssarc_save_restore_order_t restoreOrder`

The restore order.

`ssarc_save_restore_order_t saveOrder`

The save order.

`uint8_t restorePriority`

Restore priority of current group. 0 is the highest priority, 15 is the lowest priority

uint8_t savePriority

Save priority of current group. 0 is the highest priority, 15 is the lowest priority.

ssarc_power_domain_name_t powerDomain

Power domain.

uint32_t highestAddress

Highest address that can be accessed for the descriptors in the group.

uint32_t lowestAddress

Lowest address that can be accessed for the descriptors in the group.

2.114 TEMPSensor: Temperature Sensor Module

FSL_TMPSNS_DRIVER_VERSION

TMPSNS interrupt status enable type, tmpsns_interrupt_status_enable_t.

Values:

enumerator kTEMPSENSOR_HighTempInterruptStatusEnable

High temperature interrupt status enable.

enumerator kTEMPSENSOR_LowTempInterruptStatusEnable

Low temperature interrupt status enable.

enumerator kTEMPSENSOR_PanicTempInterruptStatusEnable

Panic temperature interrupt status enable.

enumerator kTEMPSENSOR_FinishInterruptStatusEnable

Finish interrupt enable.

TMPSNS interrupt status type, tmpsns_interrupt_status_t.

Values:

enumerator kTEMPSENSOR_HighTempInterruptStatus

High temperature interrupt status.

enumerator kTEMPSENSOR_LowTempInterruptStatus

Low temperature interrupt status.

enumerator kTEMPSENSOR_PanicTempInterruptStatus

Panic temperature interrupt status.

enum tmpsns_measure_mode_t

TMPSNS measure mode, tempsensor_measure_mode.

Values:

enumerator kTEMPSENSOR_SingleMode

Single measurement mode.

enumerator kTEMPSENSOR_ContinuousMode

Continuous measurement mode.

enum _tmpsns_alarm_mode

TMPSNS alarm mode.

Values:

enumerator kTEMPMON_HighAlarmMode

The high alarm temperature interrupt mode.

enumerator kTEMPMON_PanicAlarmMode

The panic alarm temperature interrupt mode.

enumerator kTEMPMON_LowAlarmMode

The low alarm temperature interrupt mode.

typedef struct *tmplsns_config* tmplsns_config_t

TMPSNS temperature structure.

typedef enum *tmplsns_alarm_mode* tmplsns_alarm_mode_t

TMPSNS alarm mode.

void TMPSNS_Init(TMPSNS_Type *base, const *tmplsns_config_t* *config)

Initializes the TMPSNS module.

Parameters

- base – TMPSNS base pointer
- config – Pointer to configuration structure.

void TMPSNS_Deinit(TMPSNS_Type *base)

Deinitializes the TMPSNS module.

Parameters

- base – TMPSNS base pointer

void TMPSNS_GetDefaultConfig(*tmplsns_config_t* *config)

Gets the default configuration structure.

This function initializes the TMPSNS configuration structure to a default value. The default values are: tempmonConfig->frequency = 0x02U; tempmonConfig->highAlarmTemp = 44U; tempmonConfig->panicAlarmTemp = 90U; tempmonConfig->lowAlarmTemp = 39U;

Parameters

- config – Pointer to a configuration structure.

void TMPSNS_StartMeasure(TMPSNS_Type *base)

start the temperature measurement process.

Parameters

- base – TMPSNS base pointer.

void TMPSNS_StopMeasure(TMPSNS_Type *base)

stop the measurement process.

Parameters

- base – TMPSNS base pointer

float TMPSNS_GetCurrentTemperature(TMPSNS_Type *base)

Get current temperature with the fused temperature calibration data.

Parameters

- base – TMPSNS base pointer

Returns

current temperature with degrees Celsius.


```
void TMPSNS_SetTempAlarm(TMPSNS_Type *base, int32_t tempVal, tmplsns_alarm_mode_t
                        alarmMode)
```

Set the temperature count (raw sensor output) that will generate an alarm interrupt.

Parameters

- base – TMPSNS base pointer
- tempVal – The alarm temperature with degrees Celsius
- alarmMode – The alarm mode.

```
void TMPSNS_EnableInterrupt(TMPSNS_Type *base, uint32_t mask)
```

Enable interrupt status.

Parameters

- base – TMPSNS base pointer
- mask – The interrupts to enable from *tmplsns_interrupt_status_enable_t*.

```
void TMPSNS_DisableInterrupt(TMPSNS_Type *base, uint32_t mask)
```

Disable interrupt status.

Parameters

- base – TMPSNS base pointer
- mask – The interrupts to disable from *tmplsns_interrupt_status_enable_t*.

```
static inline uint32_t TMPSNS_GetInterruptFlags(TMPSNS_Type *base)
```

Get interrupt status flag.

Parameters

- base – TMPSNS base pointer

```
static inline void TMPSNS_ClearInterruptFlags(TMPSNS_Type *base, uint32_t mask)
```

Clear interrupt status flag.

Parameters

- base – TMPSNS base pointer
- mask – The interrupts to disable from *tmplsns_interrupt_status_t*.

```
struct _tmplsns_config
```

#include <fsl_tmpsns.h> TMPSNS temperature structure.

Public Members

tmplsns_measure_mode_t measureMode

The temperature measure mode.

uint16_t frequency

The temperature measure frequency.

int32_t highAlarmTemp

The high alarm temperature.

int32_t panicAlarmTemp

The panic alarm temperature.

int32_t lowAlarmTemp

The low alarm temperature.

2.115 USDHC: Ultra Secured Digital Host Controller Driver

`void USDHC_Init(USDHC_Type *base, const usdhc_config_t *config)`

USDHC module initialization function.

Configures the USDHC according to the user configuration.

Example:

```
usdhc_config_t config;
config.cardDetectDat3 = false;
config.endianMode = kUSDHC_EndianModeLittle;
config.dmaMode = kUSDHC_DmaModeAdma2;
config.readWatermarkLevel = 128U;
config.writeWatermarkLevel = 128U;
USDHC_Init(USDHC, &config);
```

Parameters

- `base` – USDHC peripheral base address.
- `config` – USDHC configuration information.

Return values

`kStatus_Success` – Operate successfully.

`void USDHC_Deinit(USDHC_Type *base)`

Deinitializes the USDHC.

Parameters

- `base` – USDHC peripheral base address.

`bool USDHC_Reset(USDHC_Type *base, uint32_t mask, uint32_t timeout)`

Resets the USDHC.

Parameters

- `base` – USDHC peripheral base address.
- `mask` – The reset type mask(`_usdhc_reset`).
- `timeout` – Timeout for reset.

Return values

- `true` – Reset successfully.
- `false` – Reset failed.

`status_t USDHC_SetAdmaTableConfig(USDHC_Type *base, usdhc_adma_config_t *dmaConfig, usdhc_data_t *dataConfig, uint32_t flags)`

Sets the DMA descriptor table configuration. A high level DMA descriptor configuration function.

Parameters

- `base` – USDHC peripheral base address.
- `dmaConfig` – ADMA configuration
- `dataConfig` – Data descriptor
- `flags` – ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum `_usdhc_adma_flag`.

Return values

- `kStatus_OutOfRange` – ADMA descriptor table length isn't enough to describe data.
- `kStatus_Success` – Operate successfully.

status_t USDHC_SetInternalDmaConfig(USDHC_Type *base, *usdhc_adma_config_t* *dmaConfig, const uint32_t *dataAddr, bool enAutoCmd23)

Internal DMA configuration. This function is used to config the USDHC DMA related registers.

Parameters

- base – USDHC peripheral base address.
- dmaConfig – ADMA configuration.
- dataAddr – Transfer data address, a simple DMA parameter, if ADMA is used, leave it to NULL.
- enAutoCmd23 – Flag to indicate Auto CMD23 is enable or not, a simple DMA parameter, if ADMA is used, leave it to false.

Return values

- `kStatus_OutOfRange` – ADMA descriptor table length isn't enough to describe data.
- `kStatus_Success` – Operate successfully.

status_t USDHC_SetADMA2Descriptor(uint32_t *admaTable, uint32_t admaTableWords, const uint32_t *dataBufferAddr, uint32_t dataBytes, uint32_t flags)

Sets the ADMA2 descriptor table configuration.

Parameters

- admaTable – ADMA table address.
- admaTableWords – ADMA table length.
- dataBufferAddr – Data buffer address.
- dataBytes – Data Data length.
- flags – ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum `_usdhc_adma_flag`.

Return values

- `kStatus_OutOfRange` – ADMA descriptor table length isn't enough to describe data.
- `kStatus_Success` – Operate successfully.

status_t USDHC_SetADMA1Descriptor(uint32_t *admaTable, uint32_t admaTableWords, const uint32_t *dataBufferAddr, uint32_t dataBytes, uint32_t flags)

Sets the ADMA1 descriptor table configuration.

Parameters

- admaTable – ADMA table address.
- admaTableWords – ADMA table length.
- dataBufferAddr – Data buffer address.
- dataBytes – Data length.
- flags – ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum `_usdhc_adma_flag`.

Return values

- `kStatus_OutOfRange` – ADMA descriptor table length isn't enough to describe data.
- `kStatus_Success` – Operate successfully.

`static inline void USDHC_EnableInternalDMA(USDHC_Type *base, bool enable)`

Enables internal DMA.

Parameters

- `base` – USDHC peripheral base address.
- `enable` – enable or disable flag

`static inline void USDHC_EnableInterruptStatus(USDHC_Type *base, uint32_t mask)`

Enables the interrupt status.

Parameters

- `base` – USDHC peripheral base address.
- `mask` – Interrupt status flags mask(`_usdhc_interrupt_status_flag`).

`static inline void USDHC_DisableInterruptStatus(USDHC_Type *base, uint32_t mask)`

Disables the interrupt status.

Parameters

- `base` – USDHC peripheral base address.
- `mask` – The interrupt status flags mask(`_usdhc_interrupt_status_flag`).

`static inline void USDHC_EnableInterruptSignal(USDHC_Type *base, uint32_t mask)`

Enables the interrupt signal corresponding to the interrupt status flag.

Parameters

- `base` – USDHC peripheral base address.
- `mask` – The interrupt status flags mask(`_usdhc_interrupt_status_flag`).

`static inline void USDHC_DisableInterruptSignal(USDHC_Type *base, uint32_t mask)`

Disables the interrupt signal corresponding to the interrupt status flag.

Parameters

- `base` – USDHC peripheral base address.
- `mask` – The interrupt status flags mask(`_usdhc_interrupt_status_flag`).

`static inline uint32_t USDHC_GetEnabledInterruptStatusFlags(USDHC_Type *base)`

Gets the enabled interrupt status.

Parameters

- `base` – USDHC peripheral base address.

Returns

Current interrupt status flags mask(`_usdhc_interrupt_status_flag`).

`static inline uint32_t USDHC_GetInterruptStatusFlags(USDHC_Type *base)`

Gets the current interrupt status.

Parameters

- `base` – USDHC peripheral base address.

Returns

Current interrupt status flags mask(`_usdhc_interrupt_status_flag`).

static inline void USDHC_ClearInterruptStatusFlags(USDHC_Type *base, uint32_t mask)

Clears a specified interrupt status. write 1 clears.

Parameters

- base – USDHC peripheral base address.
- mask – The interrupt status flags mask(_usdhc_interrupt_status_flag).

static inline uint32_t USDHC_GetAutoCommand12ErrorStatusFlags(USDHC_Type *base)

Gets the status of auto command 12 error.

Parameters

- base – USDHC peripheral base address.

Returns

Auto command 12 error status flags mask(_usdhc_auto_command12_error_status_flag).

static inline uint32_t USDHC_GetAdmaErrorStatusFlags(USDHC_Type *base)

Gets the status of the ADMA error.

Parameters

- base – USDHC peripheral base address.

Returns

ADMA error status flags mask(_usdhc_adma_error_status_flag).

static inline uint32_t USDHC_GetPresentStatusFlags(USDHC_Type *base)

Gets a present status.

This function gets the present USDHC's status except for an interrupt status and an error status.

Parameters

- base – USDHC peripheral base address.

Returns

Present USDHC's status flags mask(_usdhc_present_status_flag).

void USDHC_GetCapability(USDHC_Type *base, *usdhc_capability_t* *capability)

Gets the capability information.

Parameters

- base – USDHC peripheral base address.
- capability – Structure to save capability information.

static inline void USDHC_ForceClockOn(USDHC_Type *base, bool enable)

Forces the card clock on.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

uint32_t USDHC_SetSdClock(USDHC_Type *base, uint32_t srcClock_Hz, uint32_t busClock_Hz)

Sets the SD bus clock frequency.

Parameters

- base – USDHC peripheral base address.
- srcClock_Hz – USDHC source clock frequency united in Hz.
- busClock_Hz – SD bus clock frequency united in Hz.

Returns

The nearest frequency of busClock_Hz configured for SD bus.

bool USDHC_SetCardActive(USDHC_Type *base, uint32_t timeout)

Sends 80 clocks to the card to set it to the active state.

This function must be called each time the card is inserted to ensure that the card can receive the command correctly.

Parameters

- base – USDHC peripheral base address.
- timeout – Timeout to initialize card.

Return values

- true – Set card active successfully.
- false – Set card active failed.

static inline void USDHC_AssertHardwareReset(USDHC_Type *base, bool high)

Triggers a hardware reset.

Parameters

- base – USDHC peripheral base address.
- high – 1 or 0 level

static inline void USDHC_SetDataBusWidth(USDHC_Type *base, *usdhc_data_bus_width_t* width)

Sets the data transfer width.

Parameters

- base – USDHC peripheral base address.
- width – Data transfer width.

static inline void USDHC_WriteData(USDHC_Type *base, uint32_t data)

Fills the data port.

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

- base – USDHC peripheral base address.
- data – The data about to be sent.

static inline uint32_t USDHC_ReadData(USDHC_Type *base)

Retrieves the data from the data port.

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

- base – USDHC peripheral base address.

Returns

The data has been read.

void USDHC_SendCommand(USDHC_Type *base, *usdhc_command_t* *command)

Sends command function.

Parameters

- base – USDHC peripheral base address.
- command – configuration

static inline void USDHC_EnableWakeupEvent(USDHC_Type *base, uint32_t mask, bool enable)

Enables or disables a wakeup event in low-power mode.

Parameters

- base – USDHC peripheral base address.
- mask – Wakeup events mask(_usdhc_wakeup_event).
- enable – True to enable, false to disable.

static inline void USDHC_CardDetectByData3(USDHC_Type *base, bool enable)

Detects card insert status.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

static inline bool USDHC_DetectCardInsert(USDHC_Type *base)

Detects card insert status.

Parameters

- base – USDHC peripheral base address.

static inline void USDHC_EnableSdioControl(USDHC_Type *base, uint32_t mask, bool enable)

Enables or disables the SDIO card control.

Parameters

- base – USDHC peripheral base address.
- mask – SDIO card control flags mask(_usdhc_sdio_control_flag).
- enable – True to enable, false to disable.

static inline void USDHC_SetContinueRequest(USDHC_Type *base)

Restarts a transaction which has stopped at the block GAP for the SDIO card.

Parameters

- base – USDHC peripheral base address.

static inline void USDHC_RequestStopAtBlockGap(USDHC_Type *base, bool enable)

Request stop at block gap function.

Parameters

- base – USDHC peripheral base address.
- enable – True to stop at block gap, false to normal transfer.

void USDHC_SetMmcBootConfig(USDHC_Type *base, const *usdhc_boot_config_t* *config)

Configures the MMC boot feature.

Example:

```
usdhc_boot_config_t config;
config.ackTimeoutCount = 4;
config.bootMode = kUSDHC_BootModeNormal;
config.blockCount = 5;
config.enableBootAck = true;
config.enableBoot = true;
config.enableAutoStopAtBlockGap = true;
USDHC_SetMmcBootConfig(USDHC, &config);
```

Parameters

- base – USDHC peripheral base address.
- config – The MMC boot configuration information.

static inline void USDHC_EnableMmcBoot(USDHC_Type *base, bool enable)

Enables or disables the mmc boot mode.

Parameters

- base – USDHC peripheral base address.
- enable – True to enable, false to disable.

static inline void USDHC_SetForceEvent(USDHC_Type *base, uint32_t mask)

Forces generating events according to the given mask.

Parameters

- base – USDHC peripheral base address.
- mask – The force events bit position (_usdhc_force_event).

static inline bool USDHC_RequestTuningForSDR50(USDHC_Type *base)

Checks the SDR50 mode request tuning bit. When this bit set, application shall perform tuning for SDR50 mode.

Parameters

- base – USDHC peripheral base address.

static inline bool USDHC_RequestReTuning(USDHC_Type *base)

Checks the request re-tuning bit. When this bit is set, user should do manual tuning or standard tuning function.

Parameters

- base – USDHC peripheral base address.

static inline void USDHC_EnableAutoTuning(USDHC_Type *base, bool enable)

The SDR104 mode auto tuning enable and disable. This function should be called after tuning function execute pass, auto tuning will handle by hardware.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

void USDHC_EnableAutoTuningForCmdAndData(USDHC_Type *base)

The auto tuning enable for CMD/DATA line.

Parameters

- base – USDHC peripheral base address.

void USDHC_EnableManualTuning(USDHC_Type *base, bool enable)

Manual tuning trigger or abort. User should handle the tuning cmd and find the boundary of the delay then calculate a average value which will be configured to the **CLK_TUNE_CTRL_STATUS**. This function should be called before function **USDHC_AdjustDelayForManualTuning**.

Parameters

- base – USDHC peripheral base address.
- enable – tuning enable flag

static inline uint32_t USDHC_GetTuningDelayStatus(USDHC_Type *base)

Get the tuning delay cell setting.

Parameters

- base – USDHC peripheral base address.

Return values

CLK – Tuning Control and Status register value.

status_t USDHC_SetTuningDelay(USDHC_Type *base, uint32_t preDelay, uint32_t outDelay,
uint32_t postDelay)

The tuning delay cell setting.

Parameters

- base – USDHC peripheral base address.
- preDelay – Set the number of delay cells on the feedback clock between the feedback clock and CLK_PRE.
- outDelay – Set the number of delay cells on the feedback clock between CLK_PRE and CLK_OUT.
- postDelay – Set the number of delay cells on the feedback clock between CLK_OUT and CLK_POST.

Return values

- kStatus_Fail – config the delay setting fail
- kStatus_Success – config the delay setting success

status_t USDHC_AdjustDelayForManualTuning(USDHC_Type *base, uint32_t delay)

Adjusts delay for manual tuning.

Deprecated:

Do not use this function. It has been superseded by USDHC_SetTuningDelay

Parameters

- base – USDHC peripheral base address.
- delay – setting configuration

Return values

- kStatus_Fail – config the delay setting fail
- kStatus_Success – config the delay setting success

static inline void USDHC_SetStandardTuningCounter(USDHC_Type *base, uint8_t counter)

set tuning counter tuning.

Parameters

- base – USDHC peripheral base address.
- counter – tuning counter

Return values

- kStatus_Fail – config the delay setting fail
- kStatus_Success – config the delay setting success

```
void USDHC_EnableStandardTuning(USDHC_Type *base, uint32_t tuningStartTap, uint32_t step,
                                bool enable)
```

The enable standard tuning function. The standard tuning window and tuning counter using the default config tuning cmd is sent by the software, user need to check whether the tuning result can be used for SDR50, SDR104, and HS200 mode tuning.

Parameters

- base – USDHC peripheral base address.
- tuningStartTap – start tap
- step – tuning step
- enable – enable/disable flag

```
static inline uint32_t USDHC_GetExecuteStdTuningStatus(USDHC_Type *base)
```

Gets execute STD tuning status.

Parameters

- base – USDHC peripheral base address.

```
static inline uint32_t USDHC_CheckStdTuningResult(USDHC_Type *base)
```

Checks STD tuning result.

Parameters

- base – USDHC peripheral base address.

```
static inline uint32_t USDHC_CheckTuningError(USDHC_Type *base)
```

Checks tuning error.

Parameters

- base – USDHC peripheral base address.

```
void USDHC_EnableDDRMMode(USDHC_Type *base, bool enable, uint32_t nibblePos)
```

The enable/disable DDR mode.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag
- nibblePos – nibble position

```
static inline void USDHC_EnableHS400Mode(USDHC_Type *base, bool enable)
```

The enable/disable HS400 mode.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

```
static inline void USDHC_ResetStrobeDLL(USDHC_Type *base)
```

Resets the strobe DLL.

Parameters

- base – USDHC peripheral base address.

```
static inline void USDHC_EnableStrobeDLL(USDHC_Type *base, bool enable)
```

Enables/disables the strobe DLL.

Parameters

- base – USDHC peripheral base address.

- enable – enable/disable flag

void USDHC_ConfigStrobeDLL(USDHC_Type *base, uint32_t delayTarget, uint32_t updateInterval)

Configs the strobe DLL delay target and update interval.

Parameters

- base – USDHC peripheral base address.
- delayTarget – delay target
- updateInterval – update interval

static inline void USDHC_SetStrobeDllOverride(USDHC_Type *base, uint32_t delayTaps)
Enables manual override for slave delay chain using **STROBE_SLV_OVERRIDE_VAL**.

Parameters

- base – USDHC peripheral base address.
- delayTaps – Valid delay taps range from 1 - 128 taps. A value of 0 selects tap 1, and a value of 0x7F selects tap 128.

static inline uint32_t USDHC_GetStrobeDLLStatus(USDHC_Type *base)
Gets the strobe DLL status.

Parameters

- base – USDHC peripheral base address.

void USDHC_SetDataConfig(USDHC_Type *base, *usdhc_transfer_direction_t* dataDirection, uint32_t blockCount, uint32_t blockSize)

USDHC data configuration.

Parameters

- base – USDHC peripheral base address.
- dataDirection – Data direction, tx or rx.
- blockCount – Data block count.
- blockSize – Data block size.

void USDHC_TransferCreateHandle(USDHC_Type *base, *usdhc_handle_t* *handle, const *usdhc_transfer_callback_t* *callback, void *userData)

Creates the USDHC handle.

Parameters

- base – USDHC peripheral base address.
- handle – USDHC handle pointer.
- callback – Structure pointer to contain all callback functions.
- userData – Callback function parameter.

status_t USDHC_TransferNonBlocking(USDHC_Type *base, *usdhc_handle_t* *handle, *usdhc_adma_config_t* *dmaConfig, *usdhc_transfer_t* *transfer)

Transfers the command/data using an interrupt and an asynchronous method.

This function sends a command and data and returns immediately. It doesn't wait for the transfer to complete or to encounter an error. The application must not call this API in multiple threads at the same time. Because of that this API doesn't support the re-entry mechanism.

Note: Call API `USDHC_TransferCreateHandle` when calling this API.

Parameters

- `base` – USDHC peripheral base address.
- `handle` – USDHC handle.
- `dmaConfig` – ADMA configuration.
- `transfer` – Transfer content.

Return values

- `kStatus_InvalidArgument` – Argument is invalid.
- `kStatus_USDHC_BusyTransferring` – Busy transferring.
- `kStatus_USDHC_PrepareAdmaDescriptorFailed` – Prepare ADMA descriptor failed.
- `kStatus_Success` – Operate successfully.

status_t USDHC_TransferBlocking(USDHC_Type *base, *usdhc_adma_config_t* *dmaConfig, *usdhc_transfer_t* *transfer)

Transfers the command/data using a blocking method.

This function waits until the command response/data is received or the USDHC encounters an error by polling the status flag.

The application must not call this API in multiple threads at the same time. Because this API doesn't support the re-entry mechanism.

Note: There is no need to call API `USDHC_TransferCreateHandle` when calling this API.

Parameters

- `base` – USDHC peripheral base address.
- `dmaConfig` – adma configuration
- `transfer` – Transfer content.

Return values

- `kStatus_InvalidArgument` – Argument is invalid.
- `kStatus_USDHC_PrepareAdmaDescriptorFailed` – Prepare ADMA descriptor failed.
- `kStatus_USDHC_SendCommandFailed` – Send command failed.
- `kStatus_USDHC_TransferDataFailed` – Transfer data failed.
- `kStatus_Success` – Operate successfully.

void USDHC_TransferHandleIRQ(USDHC_Type *base, *usdhc_handle_t* *handle)

IRQ handler for the USDHC.

This function deals with the IRQs on the given host controller.

Parameters

- `base` – USDHC peripheral base address.
- `handle` – USDHC handle.

FSL_USDHC_DRIVER_VERSION

Driver version 2.8.5.

Enum _usdhc_status. USDHC status.

Values:

enumerator kStatus_USDHC_BusyTransferring
Transfer is on-going.

enumerator kStatus_USDHC_PrepareAdmaDescriptorFailed
Set DMA descriptor failed.

enumerator kStatus_USDHC_SendCommandFailed
Send command failed.

enumerator kStatus_USDHC_TransferDataFailed
Transfer data failed.

enumerator kStatus_USDHC_DMADDataAddrNotAlign
Data address not aligned.

enumerator kStatus_USDHC_ReTuningRequest
Re-tuning request.

enumerator kStatus_USDHC_TuningError
Tuning error.

enumerator kStatus_USDHC_NotSupport
Not support.

enumerator kStatus_USDHC_TransferDataComplete
Transfer data complete.

enumerator kStatus_USDHC_SendCommandSuccess
Transfer command complete.

enumerator kStatus_USDHC_TransferDMAComplete
Transfer DMA complete.

Enum _usdhc_capability_flag. Host controller capabilities flag mask. .

Values:

enumerator kUSDHC_SupportAdmaFlag
Support ADMA.

enumerator kUSDHC_SupportHighSpeedFlag
Support high-speed.

enumerator kUSDHC_SupportDmaFlag
Support DMA.

enumerator kUSDHC_SupportSuspendResumeFlag
Support suspend/resume.

enumerator kUSDHC_SupportV330Flag
Support voltage 3.3V.

enumerator kUSDHC_SupportV300Flag
Support voltage 3.0V.

enumerator kUSDHC_Support4BitFlag

Flag in HTCAPBLT_MBL's position, supporting 4-bit mode.

enumerator kUSDHC_Support8BitFlag

Flag in HTCAPBLT_MBL's position, supporting 8-bit mode.

enumerator kUSDHC_SupportDDR50Flag

SD version 3.0 new feature, supporting DDR50 mode.

enumerator kUSDHC_SupportSDR104Flag

Support SDR104 mode.

enumerator kUSDHC_SupportSDR50Flag

Support SDR50 mode.

Enum _usdhc_wakeup_event. Wakeup event mask. .

Values:

enumerator kUSDHC_WakeupEventOnCardInt

Wakeup on card interrupt.

enumerator kUSDHC_WakeupEventOnCardInsert

Wakeup on card insertion.

enumerator kUSDHC_WakeupEventOnCardRemove

Wakeup on card removal.

enumerator kUSDHC_WakeupEventsAll

All wakeup events

Enum _usdhc_reset. Reset type mask. .

Values:

enumerator kUSDHC_ResetAll

Reset all except card detection.

enumerator kUSDHC_ResetCommand

Reset command line.

enumerator kUSDHC_ResetData

Reset data line.

enumerator kUSDHC_ResetTuning

Reset tuning circuit.

enumerator kUSDHC_ResetsAll

All reset types

Enum _usdhc_transfer_flag. Transfer flag mask.

Values:

enumerator kUSDHC_EnableDmaFlag

Enable DMA.

enumerator kUSDHC_CommandTypeSuspendFlag

Suspend command.

enumerator kUSDHC_CommandTypeResumeFlag
Resume command.

enumerator kUSDHC_CommandTypeAbortFlag
Abort command.

enumerator kUSDHC_EnableBlockCountFlag
Enable block count.

enumerator kUSDHC_EnableAutoCommand12Flag
Enable auto CMD12.

enumerator kUSDHC_DataReadFlag
Enable data read.

enumerator kUSDHC_MultipleBlockFlag
Multiple block data read/write.

enumerator kUSDHC_EnableAutoCommand23Flag
Enable auto CMD23.

enumerator kUSDHC_ResponseLength136Flag
136-bit response length.

enumerator kUSDHC_ResponseLength48Flag
48-bit response length.

enumerator kUSDHC_ResponseLength48BusyFlag
48-bit response length with busy status.

enumerator kUSDHC_EnableCrcCheckFlag
Enable CRC check.

enumerator kUSDHC_EnableIndexCheckFlag
Enable index check.

enumerator kUSDHC_DataPresentFlag
Data present flag.

Enum _usdhc_present_status_flag. Present status flag mask. .

Values:

enumerator kUSDHC_CommandInhibitFlag
Command inhibit.

enumerator kUSDHC_DataInhibitFlag
Data inhibit.

enumerator kUSDHC_DataLineActiveFlag
Data line active.

enumerator kUSDHC_SdClockStableFlag
SD bus clock stable.

enumerator kUSDHC_WriteTransferActiveFlag
Write transfer active.

enumerator kUSDHC_ReadTransferActiveFlag
Read transfer active.

enumerator kUSDHC_BufferWriteEnableFlag
Buffer write enable.

enumerator kUSDHC_BufferReadEnableFlag
Buffer read enable.

enumerator kUSDHC_ReTuningRequestFlag
Re-tuning request flag, only used for SDR104 mode.

enumerator kUSDHC_DelaySettingFinishedFlag
Delay setting finished flag.

enumerator kUSDHC_CardInsertedFlag
Card inserted.

enumerator kUSDHC_CommandLineLevelFlag
Command line signal level.

enumerator kUSDHC_Data0LineLevelFlag
Data0 line signal level.

enumerator kUSDHC_Data1LineLevelFlag
Data1 line signal level.

enumerator kUSDHC_Data2LineLevelFlag
Data2 line signal level.

enumerator kUSDHC_Data3LineLevelFlag
Data3 line signal level.

enumerator kUSDHC_Data4LineLevelFlag
Data4 line signal level.

enumerator kUSDHC_Data5LineLevelFlag
Data5 line signal level.

enumerator kUSDHC_Data6LineLevelFlag
Data6 line signal level.

enumerator kUSDHC_Data7LineLevelFlag
Data7 line signal level.

Enum _usdhc_interrupt_status_flag. Interrupt status flag mask. .

Values:

enumerator kUSDHC_CommandCompleteFlag
Command complete.

enumerator kUSDHC_DataCompleteFlag
Data complete.

enumerator kUSDHC_BlockGapEventFlag
Block gap event.

enumerator kUSDHC_DmaCompleteFlag
DMA interrupt.

enumerator kUSDHC_BufferWriteReadyFlag
Buffer write ready.

enumerator kUSDHC_BufferReadReadyFlag
Buffer read ready.

enumerator kUSDHC_CardInsertionFlag
Card inserted.

enumerator kUSDHC_CardRemovalFlag
Card removed.

enumerator kUSDHC_CardInterruptFlag
Card interrupt.

enumerator kUSDHC_ReTuningEventFlag
Re-Tuning event, only for SD3.0 SDR104 mode.

enumerator kUSDHC_TuningPassFlag
SDR104 mode tuning pass flag.

enumerator kUSDHC_TuningErrorFlag
SDR104 tuning error flag.

enumerator kUSDHC_CommandTimeoutFlag
Command timeout error.

enumerator kUSDHC_CommandCrcErrorFlag
Command CRC error.

enumerator kUSDHC_CommandEndBitErrorFlag
Command end bit error.

enumerator kUSDHC_CommandIndexErrorFlag
Command index error.

enumerator kUSDHC_DataTimeoutFlag
Data timeout error.

enumerator kUSDHC_DataCrcErrorFlag
Data CRC error.

enumerator kUSDHC_DataEndBitErrorFlag
Data end bit error.

enumerator kUSDHC_AutoCommand12ErrorFlag
Auto CMD12 error.

enumerator kUSDHC_DmaErrorFlag
DMA error.

enumerator kUSDHC_CommandErrorFlag
Command error

enumerator kUSDHC_DataErrorFlag
Data error

enumerator kUSDHC_ErrorFlag
All error

enumerator kUSDHC_DataFlag
Data interrupts

enumerator kUSDHC_DataDMAFlag
Data interrupts

enumerator kUSDHC_CommandFlag

Command interrupts

enumerator kUSDHC_CardDetectFlag

Card detection interrupts

enumerator kUSDHC_SDR104TuningFlag

SDR104 tuning flag.

enumerator kUSDHC_AllInterruptFlags

All flags mask

Enum _usdhc_auto_command12_error_status_flag. Auto CMD12 error status flag mask. .

Values:

enumerator kUSDHC_AutoCommand12NotExecutedFlag

Not executed error.

enumerator kUSDHC_AutoCommand12TimeoutFlag

Timeout error.

enumerator kUSDHC_AutoCommand12EndBitErrorFlag

End bit error.

enumerator kUSDHC_AutoCommand12CrcErrorFlag

CRC error.

enumerator kUSDHC_AutoCommand12IndexErrorFlag

Index error.

enumerator kUSDHC_AutoCommand12NotIssuedFlag

Not issued error.

Enum _usdhc_standard_tuning. Standard tuning flag.

Values:

enumerator kUSDHC_ExecuteTuning

Used to start tuning procedure.

enumerator kUSDHC_TuningSampleClockSel

When **std_tuning_en** bit is set, this bit is used to select sampling clock.

Enum _usdhc_adma_error_status_flag. ADMA error status flag mask. .

Values:

enumerator kUSDHC_AdmaLenghMismatchFlag

Length mismatch error.

enumerator kUSDHC_AdmaDescriptorErrorFlag

Descriptor error.

Enum _usdhc_adma_error_state. ADMA error state.

This state is the detail state when ADMA error has occurred.

Values:

enumerator kUSDHC_AdmaErrorStateStopDma

Stop DMA, previous location set in the ADMA system address is errored address.

enumerator kUSDHC_AdmaErrorStateFetchDescriptor

Fetch descriptor, current location set in the ADMA system address is errored address.

enumerator kUSDHC_AdmaErrorStateChangeAddress

Change address, no DMA error has occurred.

enumerator kUSDHC_AdmaErrorStateTransferData

Transfer data, previous location set in the ADMA system address is errored address.

enumerator kUSDHC_AdmaErrorStateInvalidLength

Invalid length in ADMA descriptor.

enumerator kUSDHC_AdmaErrorStateInvalidDescriptor

Invalid descriptor fetched by ADMA.

enumerator kUSDHC_AdmaErrorState

ADMA error state

Enum _usdhc_force_event. Force event bit position. .

Values:

enumerator kUSDHC_ForceEventAutoCommand12NotExecuted

Auto CMD12 not executed error.

enumerator kUSDHC_ForceEventAutoCommand12Timeout

Auto CMD12 timeout error.

enumerator kUSDHC_ForceEventAutoCommand12CrcError

Auto CMD12 CRC error.

enumerator kUSDHC_ForceEventEndBitError

Auto CMD12 end bit error.

enumerator kUSDHC_ForceEventAutoCommand12IndexError

Auto CMD12 index error.

enumerator kUSDHC_ForceEventAutoCommand12NotIssued

Auto CMD12 not issued error.

enumerator kUSDHC_ForceEventCommandTimeout

Command timeout error.

enumerator kUSDHC_ForceEventCommandCrcError

Command CRC error.

enumerator kUSDHC_ForceEventCommandEndBitError

Command end bit error.

enumerator kUSDHC_ForceEventCommandIndexError

Command index error.

enumerator kUSDHC_ForceEventDataTimeout

Data timeout error.

enumerator kUSDHC_ForceEventDataCrcError

Data CRC error.

enumerator kUSDHC_ForceEventDataEndBitError

Data end bit error.

enumerator kUSDHC_ForceEventAutoCommand12Error

Auto CMD12 error.

enumerator kUSDHC_ForceEventCardInt

Card interrupt.

enumerator kUSDHC_ForceEventDmaError

Dma error.

enumerator kUSDHC_ForceEventTuningError

Tuning error.

enumerator kUSDHC_ForceEventsAll

All force event flags mask.

enum __usdhc_transfer_direction

Data transfer direction.

Values:

enumerator kUSDHC_TransferDirectionReceive

USDHC transfer direction receive.

enumerator kUSDHC_TransferDirectionSend

USDHC transfer direction send.

enum __usdhc_data_bus_width

Data transfer width.

Values:

enumerator kUSDHC_DataBusWidth1Bit

1-bit mode

enumerator kUSDHC_DataBusWidth4Bit

4-bit mode

enumerator kUSDHC_DataBusWidth8Bit

8-bit mode

enum __usdhc_endian_mode

Endian mode.

Values:

enumerator kUSDHC_EndianModeBig

Big endian mode.

enumerator kUSDHC_EndianModeHalfWordBig

Half word big endian mode.

enumerator kUSDHC_EndianModeLittle

Little endian mode.

enum __usdhc_dma_mode

DMA mode.

Values:

enumerator kUSDHC_DmaModeSimple

External DMA.

enumerator kUSDHC_DmaModeAdma1
ADMA1 is selected.

enumerator kUSDHC_DmaModeAdma2
ADMA2 is selected.

enumerator kUSDHC_ExternalDMA
External DMA mode selected.

Enum _usdhc_sdio_control_flag. SDIO control flag mask. .

Values:

enumerator kUSDHC_StopAtBlockGapFlag
Stop at block gap.

enumerator kUSDHC_ReadWaitControlFlag
Read wait control.

enumerator kUSDHC_InterruptAtBlockGapFlag
Interrupt at block gap.

enumerator kUSDHC_ReadDoneNo8CLK
Read done without 8 clk for block gap.

enumerator kUSDHC_ExactBlockNumberReadFlag
Exact block number read.

enum _usdhc_boot_mode
MMC card boot mode.

Values:

enumerator kUSDHC_BootModeNormal
Normal boot

enumerator kUSDHC_BootModeAlternative
Alternative boot

enum _usdhc_card_command_type
The command type.

Values:

enumerator kCARD_CommandTypeNormal
Normal command

enumerator kCARD_CommandTypeSuspend
Suspend command

enumerator kCARD_CommandTypeResume
Resume command

enumerator kCARD_CommandTypeAbort
Abort command

enumerator kCARD_CommandTypeEmpty
Empty command

enum _usdhc_card_response_type
The command response type.

Defines the command response type from card to host controller.

Values:

enumerator kCARD_ResponseTypeNone

Response type: none

enumerator kCARD_ResponseTypeR1

Response type: R1

enumerator kCARD_ResponseTypeR1b

Response type: R1b

enumerator kCARD_ResponseTypeR2

Response type: R2

enumerator kCARD_ResponseTypeR3

Response type: R3

enumerator kCARD_ResponseTypeR4

Response type: R4

enumerator kCARD_ResponseTypeR5

Response type: R5

enumerator kCARD_ResponseTypeR5b

Response type: R5b

enumerator kCARD_ResponseTypeR6

Response type: R6

enumerator kCARD_ResponseTypeR7

Response type: R7

Enum _usdhc_adma1_descriptor_flag. The mask for the control/status field in ADMA1 descriptor.

Values:

enumerator kUSDHC_Adma1DescriptorValidFlag

Valid flag.

enumerator kUSDHC_Adma1DescriptorEndFlag

End flag.

enumerator kUSDHC_Adma1DescriptorInterruptFlag

Interrupt flag.

enumerator kUSDHC_Adma1DescriptorActivity1Flag

Activity 1 flag.

enumerator kUSDHC_Adma1DescriptorActivity2Flag

Activity 2 flag.

enumerator kUSDHC_Adma1DescriptorTypeNop

No operation.

enumerator kUSDHC_Adma1DescriptorTypeTransfer

Transfer data.

enumerator kUSDHC_Adma1DescriptorTypeLink

Link descriptor.

enumerator kUSDHC_Adma1DescriptorTypeSetLength

Set data length.

Enum _usdhc_adma2_descriptor_flag. ADMA1 descriptor control and status mask.

Values:

enumerator kUSDHC__Adma2DescriptorValidFlag

Valid flag.

enumerator kUSDHC__Adma2DescriptorEndFlag

End flag.

enumerator kUSDHC__Adma2DescriptorInterruptFlag

Interrupt flag.

enumerator kUSDHC__Adma2DescriptorActivity1Flag

Activity 1 mask.

enumerator kUSDHC__Adma2DescriptorActivity2Flag

Activity 2 mask.

enumerator kUSDHC__Adma2DescriptorTypeNop

No operation.

enumerator kUSDHC__Adma2DescriptorTypeReserved

Reserved.

enumerator kUSDHC__Adma2DescriptorTypeTransfer

Transfer type.

enumerator kUSDHC__Adma2DescriptorTypeLink

Link type.

Enum _usdhc_adma_flag. ADMA descriptor configuration flag. .

Values:

enumerator kUSDHC__AdmaDescriptorSingleFlag

Try to finish the transfer in a single ADMA descriptor. If transfer size is bigger than one ADMA descriptor's ability, new another descriptor for data transfer.

enumerator kUSDHC__AdmaDescriptorMultipleFlag

Create multiple ADMA descriptors within the ADMA table, this is used for mmc boot mode specifically, which need to modify the ADMA descriptor on the fly, so the flag should be used combining with stop at block gap feature.

enum _usdhc_burst_len

DMA transfer burst len config.

Values:

enumerator kUSDHC__EnBurstLenForINCR

Enable burst len for INCR.

enumerator kUSDHC__EnBurstLenForINCR4816

Enable burst len for INCR4/INCR8/INCR16.

enumerator kUSDHC__EnBurstLenForINCR4816WRAP

Enable burst len for INCR4/8/16 WRAP.

Enum _usdhc_transfer_data_type. Transfer data type definition.

Values:

enumerator `kUSDHC_TransferDataNormal`
Transfer normal read/write data.

enumerator `kUSDHC_TransferDataTuning`
Transfer tuning data.

enumerator `kUSDHC_TransferDataBoot`
Transfer boot data.

enumerator `kUSDHC_TransferDataBootcontinuous`
Transfer boot data continuously.

typedef enum `_usdhc_transfer_direction` `usdhc_transfer_direction_t`
Data transfer direction.

typedef enum `_usdhc_data_bus_width` `usdhc_data_bus_width_t`
Data transfer width.

typedef enum `_usdhc_endian_mode` `usdhc_endian_mode_t`
Endian mode.

typedef enum `_usdhc_dma_mode` `usdhc_dma_mode_t`
DMA mode.

typedef enum `_usdhc_boot_mode` `usdhc_boot_mode_t`
MMC card boot mode.

typedef enum `_usdhc_card_command_type` `usdhc_card_command_type_t`
The command type.

typedef enum `_usdhc_card_response_type` `usdhc_card_response_type_t`
The command response type.
Defines the command response type from card to host controller.

typedef enum `_usdhc_burst_len` `usdhc_burst_len_t`
DMA transfer burst len config.

typedef uint32_t `usdhc_adma1_descriptor_t`
Defines the ADMA1 descriptor structure.

typedef struct `_usdhc_adma2_descriptor` `usdhc_adma2_descriptor_t`
Defines the ADMA2 descriptor structure.

typedef struct `_usdhc_capability` `usdhc_capability_t`
USDHC capability information.
Defines a structure to save the capability information of USDHC.

typedef struct `_usdhc_boot_config` `usdhc_boot_config_t`
Data structure to configure the MMC boot feature.

typedef struct `_usdhc_config` `usdhc_config_t`
Data structure to initialize the USDHC.

typedef struct `_usdhc_command` `usdhc_command_t`
Card command descriptor.
Defines card command-related attribute.

typedef struct `_usdhc_adma_config` `usdhc_adma_config_t`
ADMA configuration.

`typedef struct _usdhc_scatter_gather_data_list usdhc_scatter_gather_data_list_t`

Card scatter gather data list.

Allow application register uncontinuous data buffer for data transfer.

`typedef struct _usdhc_scatter_gather_data usdhc_scatter_gather_data_t`

Card scatter gather data descriptor.

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

`typedef struct _usdhc_scatter_gather_transfer usdhc_scatter_gather_transfer_t`

usdhc scatter gather transfer.

`typedef struct _usdhc_data usdhc_data_t`

Card data descriptor.

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

`typedef struct _usdhc_transfer usdhc_transfer_t`

Transfer state.

`typedef struct _usdhc_handle usdhc_handle_t`

USDHC handle typedef.

`typedef struct _usdhc_transfer_callback usdhc_transfer_callback_t`

USDHC callback functions.

`typedef status_t (*usdhc_transfer_function_t)(USDHC_Type *base, usdhc_transfer_t *content)`

USDHC transfer function.

`typedef struct _usdhc_host usdhc_host_t`

USDHC host descriptor.

`USDHC_MAX_BLOCK_COUNT`

Maximum block count can be set one time.

`FSL_USDHC_ENABLE_SCATTER_GATHER_TRANSFER`

USDHC scatter gather feature control macro.

`USDHC_ADMA1_ADDRESS_ALIGN`

The alignment size for ADDRESS filed in ADMA1's descriptor.

`USDHC_ADMA1_LENGTH_ALIGN`

The alignment size for LENGTH field in ADMA1's descriptor.

`USDHC_ADMA2_ADDRESS_ALIGN`

The alignment size for ADDRESS field in ADMA2's descriptor.

`USDHC_ADMA2_LENGTH_ALIGN`

The alignment size for LENGTH filed in ADMA2's descriptor.

`USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT`

The bit shift for ADDRESS filed in ADMA1's descriptor.

Address/page field	Reserved	Attribute						
31 12	11 6	05	04	03	02	01	00	
address or data length	000000	Act2	Act1	0	Int	End	Valid	

Act2	Act1	Comment	31-28	27-12
0	0	No op	Don't care	
0	1	Set data length	0000	Data Length
1	0	Transfer data	Data address	
1	1	Link descriptor	Descriptor address	

USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK

The bit mask for ADDRESS field in ADMA1's descriptor.

USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT

The bit shift for LENGTH field in ADMA1's descriptor.

USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK

The mask for LENGTH field in ADMA1's descriptor.

USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY

The maximum value of LENGTH field in ADMA1's descriptor. Since the max transfer size ADMA1 support is 65535 which is indivisible by 4096, so to make sure a large data load transfer (>64KB) continuously (require the data address be always align with 4096), software will set the maximum data length for ADMA1 to (64 - 4)KB.

USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT

The bit shift for LENGTH field in ADMA2's descriptor.

Address field	Length	Reserved	Attribute						
63 32	31 16	15 06	05	04	03	02	01	00	
32-bit address	16-bit length	0000000000	Act2	Act1	0	Int	End	Valid	

Act2	Act1	Comment	Operation
0	0	No op	Don't care
0	1	Reserved	Read this line and go to next one
1	0	Transfer data	Transfer data with address and length set in this descriptor line
1	1	Link descriptor	Link to another descriptor

USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK

The bit mask for LENGTH field in ADMA2's descriptor.

USDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_ENTRY

The maximum value of LENGTH field in ADMA2's descriptor.

struct _usdhc_adma2_descriptor

#include <fsl_usdhc.h> Defines the ADMA2 descriptor structure.

Public Members

uint32_t attribute
The control and status field.

uint32_t address
The address field.

struct __usdhc__capability
#include <fsl_usdhc.h> USDHC capability information.
Defines a structure to save the capability information of USDHC.

Public Members

uint32_t sdVersion
Support SD card/sdio version.

uint32_t mmcVersion
Support EMMC card version.

uint32_t maxBlockLength
Maximum block length united as byte.

uint32_t maxBlockCount
Maximum block count can be set one time.

uint32_t flags
Capability flags to indicate the support information(*__usdhc__capability_flag*).

struct __usdhc__boot__config
#include <fsl_usdhc.h> Data structure to configure the MMC boot feature.

Public Members

uint32_t ackTimeoutCount
Timeout value for the boot ACK. The available range is 0 ~ 15.

usdhc_boot_mode_t bootMode
Boot mode selection.

uint32_t blockCount
Stop at block gap value of automatic mode. Available range is 0 ~ 65535.

size_t blockSize
Block size.

bool enableBootAck
Enable or disable boot ACK.

bool enableAutoStopAtBlockGap
Enable or disable auto stop at block gap function in boot period.

struct __usdhc__config
#include <fsl_usdhc.h> Data structure to initialize the USDHC.

Public Members

uint32_t dataTimeout

Data timeout value.

usdhc_endian_mode_t endianMode

Endian mode.

uint8_t readWatermarkLevel

Watermark level for DMA read operation. Available range is 1 ~ 128.

uint8_t writeWatermarkLevel

Watermark level for DMA write operation. Available range is 1 ~ 128.

struct __usdhc_command

#include <fsl_usdhc.h> Card command descriptor.

Defines card command-related attribute.

Public Members

uint32_t index

Command index.

uint32_t argument

Command argument.

usdhc_card_command_type_t type

Command type.

usdhc_card_response_type_t responseType

Command response type.

uint32_t response[4U]

Response for this command.

uint32_t responseErrorFlags

Response error flag, which need to check the command reponse.

uint32_t flags

Cmd flags.

struct __usdhc_adma_config

#include <fsl_usdhc.h> ADMA configuration.

Public Members

usdhc_dma_mode_t dmaMode

DMA mode.

uint32_t *admaTable

ADMA table address, can't be null if transfer way is ADMA1/ADMA2.

uint32_t admaTableWords

ADMA table length united as words, can't be 0 if transfer way is ADMA1/ADMA2.

struct __usdhc_scatter_gather_data_list

#include <fsl_usdhc.h> Card scatter gather data list.

Allow application register uncontinuous data buffer for data transfer.

struct __usdhc_scatter_gather_data

#include <fsl_usdhc.h> Card scatter gather data descriptor.

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

Public Members

bool enableAutoCommand12

Enable auto CMD12.

bool enableAutoCommand23

Enable auto CMD23.

bool enableIgnoreError

Enable to ignore error event to read/write all the data.

usdhc_transfer_direction_t dataDirection

data direction

uint8_t dataType

this is used to distinguish the normal/tuning/boot data.

size_t blockSize

Block size.

usdhc_scatter_gather_data_list_t sgData

scatter gather data

struct __usdhc_scatter_gather_transfer

#include <fsl_usdhc.h> usdhc scatter gather transfer.

Public Members

usdhc_scatter_gather_data_t *data

Data to transfer.

usdhc_command_t *command

Command to send.

struct __usdhc_data

#include <fsl_usdhc.h> Card data descriptor.

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

Public Members

bool enableAutoCommand12

Enable auto CMD12.

bool enableAutoCommand23

Enable auto CMD23.

`bool enableIgnoreError`
Enable to ignore error event to read/write all the data.

`uint8_t dataType`
this is used to distinguish the normal/tuning/boot data.

`size_t blockSize`
Block size.

`uint32_t blockCount`
Block count.

`uint32_t *rxData`
Buffer to save data read.

`const uint32_t *txData`
Data buffer to write.

`struct __usdhc_transfer`
#include <fsl_usdhc.h> Transfer state.

Public Members

`usdhc_data_t *data`
Data to transfer.

`usdhc_command_t *command`
Command to send.

`struct __usdhc_transfer_callback`
#include <fsl_usdhc.h> USDHC callback functions.

Public Members

`void (*CardInserted)(USDHC_Type *base, void *userData)`
Card inserted occurs when DAT3/CD pin is for card detect

`void (*CardRemoved)(USDHC_Type *base, void *userData)`
Card removed occurs

`void (*SdioInterrupt)(USDHC_Type *base, void *userData)`
SDIO card interrupt occurs

`void (*BlockGap)(USDHC_Type *base, void *userData)`
stopped at block gap event

`void (*TransferComplete)(USDHC_Type *base, usdhc_handle_t *handle, status_t status, void *userData)`
Transfer complete callback.

`void (*ReTuning)(USDHC_Type *base, void *userData)`
Handle the re-tuning.

`struct __usdhc_handle`
#include <fsl_usdhc.h> USDHC handle.
Defines the structure to save the USDHC state information and callback function.

Note: All the fields except interruptFlags and transferredWords must be allocated by the user.

Public Members

usdhc_data_t *volatile data

Transfer parameter. Data to transfer.

usdhc_command_t *volatile command

Transfer parameter. Command to send.

volatile uint32_t transferredWords

Transfer status. Words transferred by DATAPORT way.

usdhc_transfer_callback_t callback

Callback function.

void *userData

Parameter for transfer complete callback.

struct __usdhc_host

#include <fsl_usdhc.h> USDHC host descriptor.

Public Members

USDHC_Type *base

USDHC peripheral base address.

uint32_t sourceClock_Hz

USDHC source clock frequency united in Hz.

usdhc_config_t config

USDHC configuration.

usdhc_capability_t capability

USDHC capability information.

usdhc_transfer_function_t transfer

USDHC transfer function.

2.116 WDOG: Watchdog Timer Driver

void WDOG_GetDefaultConfig(*wdog_config_t* *config)

Initializes the WDOG configuration structure.

This function initializes the WDOG configuration structure to default values. The default values are as follows.

```
wdogConfig->enableWdog = true;
wdogConfig->workMode.enableWait = true;
wdogConfig->workMode.enableStop = true;
wdogConfig->workMode.enableDebug = true;
wdogConfig->enableInterrupt = false;
wdogConfig->enablePowerdown = false;
wdogConfig->resetExtension = false;
wdogConfig->timeoutValue = 0xFFU;
wdogConfig->interruptTimeValue = 0x04u;
```

See also:

wdog_config_t

Parameters

- config – Pointer to the WDOG configuration structure.

void WDOG_Init(WDOG_Type *base, const *wdog_config_t* *config)

Initializes the WDOG.

This function initializes the WDOG. When called, the WDOG runs according to the configuration.

This is an example.

```
wdog_config_t config;
WDOG_GetDefaultConfig(&config);
config.timeoutValue = 0xffU;
config->interruptTimeValue = 0x04u;
WDOG_Init(wdog_base,&config);
```

Parameters

- base – WDOG peripheral base address
- config – The configuration of WDOG

void WDOG_Deinit(WDOG_Type *base)

Shuts down the WDOG.

This function shuts down the WDOG. Watchdog Enable bit is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. This bit(WDE) can be set/reset only in debug mode(exception).

static inline void WDOG_Enable(WDOG_Type *base)

Enables the WDOG module.

This function writes a value into the WDOG_WCR register to enable the WDOG. This is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. only debug mode exception.

Parameters

- base – WDOG peripheral base address

static inline void WDOG_Disable(WDOG_Type *base)

Disables the WDOG module.

This function writes a value into the WDOG_WCR register to disable the WDOG. This is a write one once only bit. It is not possible to clear this bit by a software write,once the bit is set. only debug mode exception

Parameters

- base – WDOG peripheral base address

static inline void WDOG_TriggerSystemSoftwareReset(WDOG_Type *base)

Trigger the system software reset.

This function will write to the WCR[SRS] bit to trigger a software system reset. This bit will automatically resets to “1” after it has been asserted to “0”. Note: Calling this API will reset the system right now, please using it with more attention.

Parameters

- base – WDOG peripheral base address


```
static inline void WDOG_TriggerSoftwareSignal(WDOG_Type *base)
```

Trigger an output assertion.

This function will write to the WCR[WDA] bit to trigger WDOG_B signal assertion. The WDOG_B signal can be routed to external pin of the chip, the output pin will turn to assertion along with WDOG_B signal. Note: The WDOG_B signal will remain assert until a power on reset occurred, so, please take more attention while calling it.

Parameters

- base – WDOG peripheral base address

```
static inline void WDOG_EnableInterrupts(WDOG_Type *base, uint16_t mask)
```

Enables the WDOG interrupt.

This bit is a write once only bit. Once the software does a write access to this bit, it will get locked and cannot be reprogrammed until the next system reset assertion

Parameters

- base – WDOG peripheral base address
- mask – The interrupts to enable The parameter can be combination of the following source if defined.
 - kWDOG_InterruptEnable

```
uint16_t WDOG_GetStatusFlags(WDOG_Type *base)
```

Gets the WDOG all reset status flags.

This function gets all reset status flags.

```
uint16_t status;  
status = WDOG_GetStatusFlags (wdog_base);
```

See also:

`_wdog_status_flags`

- true: a related status flag has been set.
- false: a related status flag is not set.

Parameters

- base – WDOG peripheral base address

Returns

State of the status flag: asserted (true) or not-asserted (false).

```
void WDOG_ClearInterruptStatus(WDOG_Type *base, uint16_t mask)
```

Clears the WDOG flag.

This function clears the WDOG status flag.

This is an example for clearing the interrupt flag.

```
WDOG_ClearStatusFlags(wdog_base,KWDOG_InterruptFlag);
```

Parameters

- base – WDOG peripheral base address
- mask – The status flags to clear. The parameter could be any combination of the following values. kWDOG_TimeoutFlag

```
static inline void WDOG_SetTimeoutValue(WDOG_Type *base, uint16_t timeoutCount)
```

Sets the WDOG timeout value.

This function sets the timeout value. This function writes a value into WCR registers. The time-out value can be written at any point of time but it is loaded to the counter at the time when WDOG is enabled or after the service routine has been performed.

Parameters

- base – WDOG peripheral base address
- timeoutCount – WDOG timeout value; count of WDOG clock tick.

```
static inline void WDOG_SetInterruptTimeoutValue(WDOG_Type *base, uint16_t timeoutCount)
```

Sets the WDOG interrupt count timeout value.

This function sets the interrupt count timeout value. This function writes a value into WIC registers which are write-once. This field is write once only. Once the software does a write access to this field, it will get locked and cannot be reprogrammed until the next system reset assertion.

Parameters

- base – WDOG peripheral base address
- timeoutCount – WDOG timeout value; count of WDOG clock tick.

```
static inline void WDOG_DisablePowerDownEnable(WDOG_Type *base)
```

Disable the WDOG power down enable bit.

This function disable the WDOG power down enable(PDE). This function writes a value into WMCR registers which are write-once. This field is write once only. Once software sets this bit it cannot be reset until the next system reset.

Parameters

- base – WDOG peripheral base address

```
void WDOG_Refresh(WDOG_Type *base)
```

Refreshes the WDOG timer.

This function feeds the WDOG. This function should be called before the WDOG timer is in timeout. Otherwise, a reset is asserted.

Parameters

- base – WDOG peripheral base address

```
FSL_WDOG_DRIVER_VERSION
```

Defines WDOG driver version.

```
WDOG_REFRESH_KEY
```

```
enum _wdog_interrupt_enable
```

WDOG interrupt configuration structure, default settings all disabled.

This structure contains the settings for all of the WDOG interrupt configurations.

Values:

enumerator kWDOG_InterruptEnable

WDOG timeout generates an interrupt before reset

```
enum _wdog_status_flags
```

WDOG status flags.

This structure contains the WDOG status flags for use in the WDOG functions.

Values:

enumerator kWDOG_RunningFlag

Running flag, set when WDOG is enabled

enumerator kWDOG_PowerOnResetFlag

Power On flag, set when reset is the result of a powerOnReset

enumerator kWDOG_TimeoutResetFlag

Timeout flag, set when reset is the result of a timeout

enumerator kWDOG_SoftwareResetFlag

Software flag, set when reset is the result of a software

enumerator kWDOG_InterruptFlag

interrupt flag, whether interrupt has occurred or not

typedef struct *_wdog_work_mode* wdog_work_mode_t

Defines WDOG work mode.

typedef struct *_wdog_config* wdog_config_t

Describes WDOG configuration structure.

struct *_wdog_work_mode*

#include <fsl_wdog.h> Defines WDOG work mode.

Public Members

bool enableWait

If set to true, WDOG continues in wait mode

bool enableStop

If set to true, WDOG continues in stop mode

bool enableDebug

If set to true, WDOG continues in debug mode

struct *_wdog_config*

#include <fsl_wdog.h> Describes WDOG configuration structure.

Public Members

bool enableWdog

Enables or disables WDOG

wdog_work_mode_t workMode

Configures WDOG work mode in debug stop and wait mode

bool enableInterrupt

Enables or disables WDOG interrupt

uint16_t timeoutValue

Timeout value

uint16_t interruptTimeValue

Interrupt count timeout value

bool softwareResetExtension

software reset extension

bool enablePowerDown

power down enable bit

`bool enableTimeOutAssert`
 Enable WDOG_B timeout assertion.

2.117 XBARA: Inter-Peripheral Crossbar Switch

`void XBARA_Init(XBARA_Type *base)`

Initializes the XBARA module.

This function un-gates the XBARA clock.

Parameters

- `base` – XBARA peripheral address.

`void XBARA_Deinit(XBARA_Type *base)`

Shuts down the XBARA module.

This function disables XBARA clock.

Parameters

- `base` – XBARA peripheral address.

`void XBARA_SetSignalsConnection(XBARA_Type *base, xbar_input_signal_t input, xbar_output_signal_t output)`

Sets a connection between the selected XBARA_IN[*] input and the XBARA_OUT[*] output signal.

This function connects the XBARA input to the selected XBARA output. If more than one XBARA module is available, only the inputs and outputs from the same module can be connected.

Example:

```
XBARA_SetSignalsConnection(XBARA, kXBARA_InputPIT_TRG0, kXBARA_
↪OutputDMAMUX18);
```

Parameters

- `base` – XBARA peripheral address.
- `input` – XBARA input signal.
- `output` – XBARA output signal.

`uint32_t XBARA_GetStatusFlags(XBARA_Type *base)`

Gets the active edge detection status.

This function gets the active edge detect status of all XBARA_OUTs. If the active edge occurs, the return value is asserted. When the interrupt or the DMA functionality is enabled for the XBARA_OUTx, this field is 1 when the interrupt or DMA request is asserted and 0 when the interrupt or DMA request has been cleared.

Parameters

- `base` – XBARA peripheral address.

Returns

the mask of these status flag bits.

`void XBARA_ClearStatusFlags(XBARA_Type *base, uint32_t mask)`

Clears the edge detection status flags of relative mask.

Parameters

- base – XBARA peripheral address.
- mask – the status flags to clear.

```
void XBARA_SetOutputSignalConfig(XBARA_Type *base, xbar_output_signal_t output, const  
                                xbara_control_config_t *controlConfig)
```

Configures the XBARA control register.

This function configures an XBARA control register. The active edge detection and the DMA/IRQ function on the corresponding XBARA output can be set.

Example:

```
xbara_control_config_t userConfig;  
userConfig.activeEdge = kXBARA_EdgeRising;  
userConfig.requestType = kXBARA_RequestInterruptEnable;  
XBARA_SetOutputSignalConfig(XBARA, kXBARA_OutputDMAMUX18, &userConfig);
```

Parameters

- base – XBARA peripheral address.
- output – XBARA output number.
- controlConfig – Pointer to structure that keeps configuration of control register.

enum __xbara_active_edge

XBARA active edge for detection.

Values:

enumerator kXBARA_EdgeNone

Edge detection status bit never asserts.

enumerator kXBARA_EdgeRising

Edge detection status bit asserts on rising edges.

enumerator kXBARA_EdgeFalling

Edge detection status bit asserts on falling edges.

enumerator kXBARA_EdgeRisingAndFalling

Edge detection status bit asserts on rising and falling edges.

enum __xbar_request

Defines the XBARA DMA and interrupt configurations.

Values:

enumerator kXBARA_RequestDisable

Interrupt and DMA are disabled.

enumerator kXBARA_RequestDMAEnable

DMA enabled, interrupt disabled.

enumerator kXBARA_RequestInterruptEnable

Interrupt enabled, DMA disabled.

enum __xbara_status_flag_t

XBARA status flags.

This provides constants for the XBARA status flags for use in the XBARA functions.

Values:

enumerator kXBARA_EdgeDetectionOut0

XBAR_OUT0 active edge interrupt flag, sets when active edge detected.

enumerator kXBARA_EdgeDetectionOut1

XBAR_OUT1 active edge interrupt flag, sets when active edge detected.

enumerator kXBARA_EdgeDetectionOut2

XBAR_OUT2 active edge interrupt flag, sets when active edge detected.

enumerator kXBARA_EdgeDetectionOut3

XBAR_OUT3 active edge interrupt flag, sets when active edge detected.

typedef enum *xbara_active_edge* xbara_active_edge_t

XBARA active edge for detection.

typedef enum *xbar_request* xbara_request_t

Defines the XBARA DMA and interrupt configurations.

typedef enum *xbara_status_flag_t* xbara_status_flag_t

XBARA status flags.

This provides constants for the XBARA status flags for use in the XBARA functions.

typedef struct *XBARAControlConfig* xbara_control_config_t

Defines the configuration structure of the XBARA control register.

This structure keeps the configuration of XBARA control register for one output. Control registers are available only for a few outputs. Not every XBARA module has control registers.

FSL_XBARA_DRIVER_VERSION

XBARA_SELx(base, output)

XBARA_WR_SELx_SELx(base, input, output)

kXBARA_RequestInterruptEnalbe

struct XBARAControlConfig

#include <fsl_xbara.h> Defines the configuration structure of the XBARA control register.

This structure keeps the configuration of XBARA control register for one output. Control registers are available only for a few outputs. Not every XBARA module has control registers.

Public Members

xbara_active_edge_t activeEdge

Active edge to be detected.

xbara_request_t requestType

Selects DMA/Interrupt request.

2.118 XBARB: Inter-Peripheral Crossbar Switch

void XBARB_Init(XBARB_Type *base)

Initializes the XBARB module.

This function un-gates the XBARB clock.

Parameters

- base – XBARB peripheral address.

void XBARB_Deinit(XBARB_Type *base)

Shuts down the XBARB module.

This function disables XBARB clock.

Parameters

- base – XBARB peripheral address.

void XBARB_SetSignalsConnection(XBARB_Type *base, xbar_input_signal_t input, xbar_output_signal_t output)

Configures a connection between the selected XBARB_IN[*] input and the XBARB_OUT[*] output signal.

This function configures which XBARB input is connected to the selected XBARB output. If more than one XBARB module is available, only the inputs and outputs from the same module can be connected.

Parameters

- base – XBARB peripheral address.
- input – XBARB input signal.
- output – XBARB output signal.

FSL_XBARB_DRIVER_VERSION

XBARB_SELx(base, output)

XBARB_WR_SELx_SELx(base, input, output)

2.119 XECC: external error correction code controller

void XECC_Init(XECC_Type *base, const *xecc_config_t* *config)

XECC module initialization function.

Parameters

- base – XECC base address.
- config – pointer to the XECC configuration structure.

void XECC_Deinit(XECC_Type *base)

Deinitializes the XECC.

Parameters

- base – XECC base address.

void XECC_GetDefaultConfig(*xecc_config_t* *config)

Sets the XECC configuration structure to default values.

Parameters

- config – pointer to the XECC configuration structure.

static inline uint32_t XECC_GetStatusFlags(XECC_Type *base)

Gets XECC status flags.

Parameters

- base – XECC peripheral base address.

Returns

XECC status flags.

```
static inline void XECC_ClearStatusFlags(XECC_Type *base, uint32_t mask)
```

XECC module clear interrupt status.

Parameters

- base – XECC base address.
- mask – status to clear from `xecc_interrupt_status_t`.

```
static inline void XECC_EnableInterruptStatus(XECC_Type *base, uint32_t mask)
```

XECC module enable interrupt status.

Parameters

- base – XECC base address.
- mask – status to enable from `xecc_interrupt_status_enable_t`.

```
static inline void XECC_DisableInterruptStatus(XECC_Type *base, uint32_t mask)
```

XECC module disable interrupt status.

Parameters

- base – XECC base address.
- mask – status to disable from `xecc_interrupt_status_enable_t`.

```
static inline void XECC_EnableInterrupts(XECC_Type *base, uint32_t mask)
```

XECC module enable interrupt.

Parameters

- base – XECC base address.
- mask – The interrupts to enable from `xecc_interrupt_enable_t`.

```
static inline void XECC_DisableInterrupts(XECC_Type *base, uint32_t mask)
```

XECC module disable interrupt.

Parameters

- base – XECC base address.
- mask – The interrupts to disable from `xecc_interrupt_enable_t`.

```
static inline void XECC_WriteECCEnable(XECC_Type *base, bool enable)
```

XECC module write ECC function enable.

Parameters

- base – XECC base address.
- enable – enable or disable.

```
static inline void XECC_ReadECCEnable(XECC_Type *base, bool enable)
```

XECC module read ECC function enable.

Parameters

- base – XECC base address.
- enable – enable or disable.

```
static inline void XECC_SwapECCEnable(XECC_Type *base, bool enable)
```

XECC module swap data enable.

Parameters

- base – XECC base address.

- enable – enable or disable.

status_t XECC_ErrorInjection(XECC_Type *base, uint32_t errordata, uint8_t erroreccdata)
XECC module error injection.

Parameters

- base – XECC base address.
- errordata – error data.
- erroreccdata – ecc code.

Return values

kStatus_Success. –

void XECC_GetSingleErrorInfo(XECC_Type *base, *xecc_single_error_info_t* *info)
XECC module get single error information.

Parameters

- base – XECC base address.
- info – single error information.

void XECC_GetMultiErrorInfo(XECC_Type *base, *xecc_multi_error_info_t* *info)
XECC module get multiple error information.

Parameters

- base – XECC base address.
- info – multiple error information.

FSL_XECC_DRIVER_VERSION

Driver version 2.0.0.

XECC interrupt configuration structure, , *xecc_interrupt_enable_t*.

This structure contains the settings for all of the XECC interrupt configurations.

Values:

enumerator kXECC_SingleErrorInterruptEnable

Single bit error interrupt enable

enumerator kXECC_MultiErrorInterruptEnable

Multiple bit error interrupt enable

enumerator kXECC_AllInterruptsEnable

all interrupts enable

XECC interrupt status configuration structure, *xecc_interrupt_status_enable_t*.

This structure contains the settings for all of the XECC interrupt status configurations.

Values:

enumerator kXECC_SingleErrorInterruptStatusEnable

Single bit error interrupt status enable

enumerator kXECC_MultiErrorInterruptStatusEnable

Multiple bits error interrupt status enable

enumerator kXECC_AllInterruptsStatusEnable

all interrupts enable

XECC status flags, `xecc_interrupt_status_t`.

This provides constants for the XECC status flags for use in the XECC functions.

Values:

enumerator `kXECC_SingleErrorInterruptFlag`

Single bit error interrupt happens on read data

enumerator `kXECC_MultiErrorInterruptFlag`

Multiple bits error interrupt happens on read data

enumerator `kXECC_AllInterruptsFlag`

all interrupts happens on read data

typedef struct `_xecc_config` `xecc_config_t`

XECC user configuration.

typedef struct `_xecc_single_error_info` `xecc_single_error_info_t`

XECC single error information, including single error address, ECC code, error data, error bit position and error bit field.

typedef struct `_xecc_multi_error_info` `xecc_multi_error_info_t`

XECC multiple error information, including multiple error address, ECC code, error data and error bit field.

struct `_xecc_config`

`#include <fsl_xecc.h>` XECC user configuration.

Public Members

bool `enableXECC`

Enable the XECC function.

bool `enableWriteECC`

Enable write ECC function.

bool `enableReadECC`

Enable read ECC function.

bool `enableSwap`

Enable swap function. The minimum ECC region range is 4k, so the lower 12 bits of this register must be 0.

uint32_t `Region0BaseAddress`

ECC region 0 base address.

uint32_t `Region0EndAddress`

ECC region 0 end address.

uint32_t `Region1BaseAddress`

ECC region 1 base address.

uint32_t `Region1EndAddress`

ECC region 1 end address.

uint32_t `Region2BaseAddress`

ECC region 2 base address.

uint32_t `Region2EndAddress`

ECC region 2 end address.

uint32_t Region3BaseAddress
ECC region 3 base address.

uint32_t Region3EndAddress
ECC region 3 end address.

struct __xecc_single_error_info
#include <fsl_xecc.h> XECC single error information, including single error address, ECC code, error data, error bit position and error bit field.

Public Members

uint32_t singleErrorAddress
Single error address

uint32_t singleErrorData
Single error read data

uint32_t singleErrorEccCode
Single error ECC code

uint32_t singleErrorBitPos
Single error bit position

uint32_t singleErrorBitField
Single error bit field

struct __xecc_multi_error_info
#include <fsl_xecc.h> XECC multiple error information, including multiple error address, ECC code, error data and error bit field.

Public Members

uint32_t multiErrorAddress
Multiple error address

uint32_t multiErrorData
Multiple error read data

uint32_t multiErrorEccCode
Multiple error ECC code

uint32_t multiErrorBitField
Single error bit field

2.120 XRDC2: Extended Resource Domain Controller 2

void XRDC2_SetGlobalValid(XRDC2_Type *base, bool valid)
Sets the XRDC2 global valid.

This function sets the XRDC2 global valid or invalid. When the XRDC2 is global invalid, all accesses from all bus masters to all slaves are allowed.

Parameters

- base – XRDC2 peripheral base address.
- valid – True to valid XRDC2.

```
static inline uint8_t XRDC2_GetCurrentMasterDomainId(XRDC2_Type *base)
```

Gets the domain ID of the current bus master.

This function returns the domain ID of the current bus master.

Parameters

- base – XRDC2 peripheral base address.

Returns

Domain ID of current bus master.

```
static inline void XRDC2_SetGlobalConfigLock(XRDC2_Type *base, xrdc2_global_config_lock_t mode)
```

Set the global configuration lock mode.

Once change the lock mode, it could not be changed until next reset.

Parameters

- base – XRDC2 peripheral base address.
- mode – The lock mode.

```
static inline uint8_t XRDC2_GetCurrentGlobalConfigLockOwnerDomainId(XRDC2_Type *base)
```

Gets the domain ID of global configuration lock owner.

Parameters

- base – XRDC2 peripheral base address.

Returns

Domain ID of the global configuration lock owner.

```
void XRDC2_GetDefaultMasterDomainAssignment(xrdc2_master_domain_assignment_t *assignment)
```

Gets the default master domain assignment.

This function sets the assignment as follows:

```
config->lock = false;
config->privilegeAttr = kXRDC2_MasterPrivilege;
config->secureAttr = kXRDC2_MasterSecure;
config->domainId = 0U;
config->mask = 0U;
config->match = 0U;
```

Parameters

- assignment – Pointer to the assignment structure.

```
void XRDC2_SetMasterDomainAssignment(XRDC2_Type *base, xrdc2_master_t master, uint8_t assignIndex, const xrdc2_master_domain_assignment_t *assignment)
```

Sets the processor bus master domain assignment.

Parameters

- base – XRDC2 peripheral base address.
- master – Which master to configure.
- assignIndex – Which assignment register to set.
- assignment – Pointer to the assignment structure.

```
static inline void XRDC2_LockMasterDomainAssignment(XRDC2_Type *base, xrdc2_master_t
                                                    master, uint8_t assignIndex)
```

Locks the bus master domain assignment register.

This function locks the master domain assignment. One bus master might have multiple domain assignment registers. The parameter `assignIndex` specifies which assignment register to lock. After it is locked, the register can't be changed until next reset.

Parameters

- `base` – XRDC2 peripheral base address.
- `master` – Which master to configure.
- `assignIndex` – Which assignment register to lock.

```
static inline void XRDC2_SetMasterDomainAssignmentValid(XRDC2_Type *base, xrdc2_master_t
                                                         master, uint8_t assignIndex, bool
                                                         valid)
```

Sets the master domain assignment as valid or invalid.

This function sets the master domain assignment as valid or invalid. One bus master might have multiple domain assignment registers. The parameter `assignIndex` specifies which assignment register to configure.

Parameters

- `base` – XRDC2 peripheral base address.
- `master` – Which master to configure.
- `assignIndex` – Index for the domain assignment register.
- `valid` – True to set valid, false to set invalid.

```
void XRDC2_GetMemSlotAccessDefaultConfig(xrdc2_mem_slot_access_config_t *config)
```

Gets the default memory slot access configuration.

This function sets the assignment as follows:

```
config->lockMode = kXRDC2_AccessConfigLockDisabled;
config->policy[0] = kXRDC2_AccessPolicyNone;
config->policy[1] = kXRDC2_AccessPolicyNone;
...
```

Parameters

- `config` – Pointer to the configuration.

```
void XRDC2_SetMemSlotAccessConfig(XRDC2_Type *base, xrdc2_mem_slot_t memSlot, const
                                   xrdc2_mem_slot_access_config_t *config)
```

Sets the memory slot access policy.

Parameters

- `base` – XRDC2 peripheral base address.
- `memSlot` – Which memory slot descriptor to set.
- `config` – Pointer to the access policy configuration structure.

```
void XRDC2_SetMemSlotAccessValid(XRDC2_Type *base, xrdc2_mem_slot_t memSlot, bool
                                   valid)
```

Sets the memory slot descriptor as valid or invalid.

Parameters

- `base` – XRDC2 peripheral base address.

- memSlot – Which memory slot descriptor to set.
- valid – True to set valid, false to set invalid.

```
void XRDC2_SetMemSlotAccessLockMode(XRDC2_Type *base, xrdc2_mem_slot_t memSlot,  
                                     xrdc2_access_config_lock_t lockMode)
```

Sets the memory slot descriptor lock mode.

Parameters

- base – XRDC2 peripheral base address.
- memSlot – Which memory slot descriptor to set.
- lockMode – The lock mode to set.

```
void XRDC2_SetMemSlotDomainAccessPolicy(XRDC2_Type *base, xrdc2_mem_slot_t memSlot,  
                                         uint8_t domainId, xrdc2_access_policy_t policy)
```

Sets the memory slot access policy for specific domain.

Parameters

- base – XRDC2 peripheral base address.
- memSlot – The memory slot to operate.
- domainId – The ID of the domain whose policy will be changed.
- policy – The access policy to set.

```
void XRDC2_EnableMemSlotExclAccessLock(XRDC2_Type *base, xrdc2_mem_slot_t memSlot,  
                                        bool enable)
```

Enable or disable the memory slot exclusive access lock.

The lock must be enabled first before use. Once disabled, it could not be enabled until reset.

Parameters

- base – XRDC2 peripheral base address.
- memSlot – The memory slot to operate.
- enable – True to enable, false to disable.

```
uint8_t XRDC2_GetMemSlotExclAccessLockDomainOwner(XRDC2_Type *base, xrdc2_mem_slot_t  
                                                    memSlot)
```

Get current memory slot exclusive access lock owner.

Parameters

- base – XRDC2 peripheral base address.
- memSlot – The memory slot to operate.

Returns

The domain ID of the lock owner.

```
status_t XRDC2_TryLockMemSlotExclAccess(XRDC2_Type *base, xrdc2_mem_slot_t memSlot)
```

Try to lock the memory slot exclusive access.

Parameters

- base – XRDC2 peripheral base address.
- memSlot – The memory slot to operate.

Return values

- kStatus_Fail – Failed to lock.
- kStatus_Success – Locked successfully.

void XRDC2_LockMemSlotExclAccess(XRDC2_Type *base, xrdc2_mem_slot_t memSlot)

Lock the memory slot exclusive access using blocking method.

Note: This function must be called when the lock is not disabled.

Parameters

- base – XRDC2 peripheral base address.
- memSlot – The memory slot to operate.

static inline void XRDC2_UnlockMemSlotExclAccess(XRDC2_Type *base, xrdc2_mem_slot_t memSlot)

Unlock the memory slot exclusive access.

Note: This function must be called by the lock owner.

Parameters

- base – XRDC2 peripheral base address.
- memSlot – The memory slot to operate.

static inline void XRDC2_ForceMemSlotExclAccessLockRelease(XRDC2_Type *base, xrdc2_mem_slot_t memSlot)

Force the memory slot exclusive access lock release.

The master does not own the lock could call this function to force release the lock.

Parameters

- base – XRDC2 peripheral base address.
- memSlot – The memory slot to operate.

void XRDC2_GetMemAccessDefaultConfig(xrdc2_mem_access_config_t *config)

Gets the default memory access configuration.

This function sets the assignment as follows:

```
config->startAddr = 0U;
config->endAddr = 0xFFFFFFFFU;
config->lockMode = kXRDC2_AccessConfigLockDisabled;
config->policy[0] = kXRDC2_AccessPolicyNone;
config->policy[1] = kXRDC2_AccessPolicyNone;
...
```

Parameters

- config – Pointer to the configuration.

void XRDC2_SetMemAccessConfig(XRDC2_Type *base, xrdc2_mem_t mem, const xrdc2_mem_access_config_t *config)

Sets the memory region access policy.

Parameters

- base – XRDC2 peripheral base address.
- mem – Which memory region descriptor to set.
- config – Pointer to the access policy configuration structure.

`void XRDC2_SetMemAccessValid(XRDC2_Type *base, xrdc2_mem_t mem, bool valid)`

Sets the memory region descriptor as valid or invalid.

Parameters

- `base` – XRDC2 peripheral base address.
- `mem` – Which memory region descriptor to set.
- `valid` – True to set valid, false to set invalid.

`void XRDC2_SetMemAccessLockMode(XRDC2_Type *base, xrdc2_mem_t mem, xrdc2_access_config_lock_t lockMode)`

Sets the memory descriptor lock mode.

Parameters

- `base` – XRDC2 peripheral base address.
- `mem` – Which memory descriptor to set.
- `lockMode` – The lock mode to set.

`void XRDC2_SetMemDomainAccessPolicy(XRDC2_Type *base, xrdc2_mem_t mem, uint8_t domainId, xrdc2_access_policy_t policy)`

Sets the memory region access policy for specific domain.

Parameters

- `base` – XRDC2 peripheral base address.
- `mem` – The memory region to operate.
- `domainId` – The ID of the domain whose policy will be changed.
- `policy` – The access policy to set.

`void XRDC2_EnableMemExclAccessLock(XRDC2_Type *base, xrdc2_mem_t mem, bool enable)`

Enable or disable the memory region exclusive access lock.

Once disabled, it could not be enabled until reset.

Parameters

- `base` – XRDC2 peripheral base address.
- `mem` – The memory region to operate.
- `enable` – True to enable, false to disable.

`uint8_t XRDC2_GetMemExclAccessLockDomainOwner(XRDC2_Type *base, xrdc2_mem_t mem)`

Get current memory region exclusive access lock owner.

Parameters

- `base` – XRDC2 peripheral base address.
- `mem` – The memory region to operate.

Returns

The domain ID of the lock owner.

`status_t XRDC2_TryLockMemExclAccess(XRDC2_Type *base, xrdc2_mem_t mem)`

Try to lock the memory region exclusive access.

Parameters

- `base` – XRDC2 peripheral base address.
- `mem` – The memory region to operate.

Return values

- kStatus_Fail – Failed to lock.
- kStatus_Success – Locked successfully.

void XRDC2_LockMemExclAccess(XRDC2_Type *base, xrdc2_mem_t mem)
Lock the memory region exclusive access using blocking method.

Note: This function must be called when the lock is not disabled.

Parameters

- base – XRDC2 peripheral base address.
- mem – The memory region to operate.

void XRDC2_UnlockMemExclAccess(XRDC2_Type *base, xrdc2_mem_t mem)
Unlock the memory region exclusive access.

Note: This function must be called by the lock owner.

Parameters

- base – XRDC2 peripheral base address.
- mem – The memory region to operate.

void XRDC2_ForceMemExclAccessLockRelease(XRDC2_Type *base, xrdc2_mem_t mem)
Force the memory region exclusive access lock release.
The master does not own the lock could call this function to force release the lock.

Parameters

- base – XRDC2 peripheral base address.
- mem – The memory region to operate.

void XRDC2_GetPeriphAccessDefaultConfig(*xrdc2_periph_access_config_t* *config)
Gets the default peripheral access configuration.
The default configuration is set as follows:

```
config->lockMode      = kXRDC2_AccessConfigLockWritable;
config->policy[0]      = kXRDC2_AccessPolicyNone;
config->policy[1]      = kXRDC2_AccessPolicyNone;
...
config->policy[15]     = kXRDC2_AccessPolicyNone;
```

Parameters

- config – Pointer to the configuration structure.

void XRDC2_SetPeriphAccessConfig(XRDC2_Type *base, xrdc2_periph_t periph, const *xrdc2_periph_access_config_t* *config)
Sets the peripheral access policy.

Parameters

- base – XRDC2 peripheral base address.
- periph – Which peripheral descriptor to set.
- config – Pointer to the access policy configuration structure.

```
void XRDC2_SetPeriphAccessValid(XRDC2_Type *base, xrdc2_periph_t periph, bool valid)
```

Sets the peripheral descriptor as valid or invalid.

Parameters

- base – XRDC2 peripheral base address.
- periph – Which peripheral descriptor to set.
- valid – True to set valid, false to set invalid.

```
void XRDC2_SetPeriphAccessLockMode(XRDC2_Type *base, xrdc2_periph_t periph,  
                                   xrdc2_access_config_lock_t lockMode)
```

Sets the peripheral descriptor lock mode.

Parameters

- base – XRDC2 peripheral base address.
- periph – Which peripheral descriptor to set.
- lockMode – The lock mode to set.

```
void XRDC2_SetPeriphDomainAccessPolicy(XRDC2_Type *base, xrdc2_periph_t periph, uint8_t  
                                       domainId, xrdc2_access_policy_t policy)
```

Sets the peripheral access policy for specific domain.

Parameters

- base – XRDC2 peripheral base address.
- periph – The peripheral to operate.
- domainId – The ID of the domain whose policy will be changed.
- policy – The access policy to set.

```
void XRDC2_EnablePeriphExclAccessLock(XRDC2_Type *base, xrdc2_periph_t periph, bool  
                                       enable)
```

Disable the peripheral exclusive access lock.

Once disabled, it could not be enabled until reset.

Parameters

- base – XRDC2 peripheral base address.
- periph – The peripheral to operate.
- enable – True to enable, false to disable.

```
uint8_t XRDC2_GetPeriphExclAccessLockDomainOwner(XRDC2_Type *base, xrdc2_periph_t  
                                                  periph)
```

Get current peripheral exclusive access lock owner.

Parameters

- base – XRDC2 peripheral base address.
- periph – The peripheral to operate.

Returns

The domain ID of the lock owner.

```
status_t XRDC2_TryLockPeriphExclAccess(XRDC2_Type *base, xrdc2_periph_t periph)
```

Try to lock the peripheral exclusive access.

Parameters

- base – XRDC2 peripheral base address.

- `periph` – The peripheral to operate.

Return values

- `kStatus_Fail` – Failed to lock.
- `kStatus_Success` – Locked successfully.

`void XRDC2_LockPeriphExclAccess(XRDC2_Type *base, xrdc2_periph_t periph)`

Lock the peripheral exclusive access using blocking method.

Note: This function must be called when the lock is not disabled.

Parameters

- `base` – XRDC2 peripheral base address.
- `periph` – The peripheral to operate.

`void XRDC2_UnlockPeriphExclAccess(XRDC2_Type *base, xrdc2_periph_t periph)`

Unlock the peripheral exclusive access.

Note: This function must be called by the lock owner.

Parameters

- `base` – XRDC2 peripheral base address.
- `periph` – The peripheral to operate.

`void XRDC2_ForcePeriphExclAccessLockRelease(XRDC2_Type *base, xrdc2_periph_t periph)`

Force the peripheral exclusive access lock release.

The master does not own the lock could call this function to force release the lock.

Parameters

- `base` – XRDC2 peripheral base address.
- `periph` – The peripheral to operate.

`enum _xrdc2_global_config_lock`

Global configuration lock.

Values:

enumerator `kXRDC2_GlobalConfigLockDisabled`

Lock disabled, registers can be written by any domain.

enumerator `kXRDC2_GlobalConfigLockDisabledUntilReset`

Lock disabled until the next reset.

enumerator `kXRDC2_GlobalConfigLockOwnerOnly`

Lock enabled, only the lock owner can write.

enumerator `kXRDC2_GlobalConfigLockEnabledUntilReset`

Lock enabled, all registers are read only until the next reset.

`enum _xrdc2_secure_attr`

XRDC2 secure attribute, the register bit `MDACi_MDAj_W0[SA]`, secure/nonsecure attribute output on a hit.

Values:

enumerator kXRDC2_MasterSecure

Use the bus master's secure/nonsecure attribute directly.

enumerator kXRDC2_ForceSecure

Force the bus attribute for this master to secure.

enumerator kXRDC2_ForceNonSecure

Force the bus attribute for this master to non-secure.

enum _xrdc2_privilege_attr

XRDC2 privileged attribute, the register bit MDACi_MDAj_W0[PA], defines the privileged/user attribute on a hit.

Values:

enumerator kXRDC2_MasterPrivilege

Use the bus master's attribute directly.

enumerator kXRDC2_ForceUser

Force the bus attribute for this master to user.

enumerator kXRDC2_ForcePrivilege

Force the bus attribute for this master to privileged.

enum _xrdc2_access_policy

XRDC2 domain access control policy.

Values:

enumerator kXRDC2_AccessPolicyNone

enumerator kXRDC2_AccessPolicyAlt1

enumerator kXRDC2_AccessPolicyAlt2

enumerator kXRDC2_AccessPolicyAlt3

enumerator kXRDC2_AccessPolicyAlt4

enumerator kXRDC2_AccessPolicyAlt5

enumerator kXRDC2_AccessPolicyAlt6

enumerator kXRDC2_AccessPolicyAll

enum _xrdc2_access_config_lock

Access configuration lock mode, the register field PDAC and MRGD LK2.

Values:

enumerator kXRDC2_AccessConfigLockDisabled

Entire PDACn/MRGDn/MSD can be written.

enumerator kXRDC2_AccessConfigLockDisabledUntilReset

Entire PDACn/MRGDn/MSD can be written until next reset.

enumerator kXRDC2_AccessConfigLockDomainXOnly

Domain x only write the DxACP field.

enumerator kXRDC2_AccessConfigLockEnabledUntilReset

PDACn/MRGDn/MSD is read-only until the next reset.

typedef enum _xrdc2_global_config_lock xrdc2_global_config_lock_t

Global configuration lock.

typedef enum *_xrdc2_secure_attr* xrdc2_secure_attr_t

XRDC2 secure attribute, the register bit MDACi_MDAj_W0[SA], secure/nonsecure attribute output on a hit.

typedef enum *_xrdc2_privilege_attr* xrdc2_privilege_attr_t

XRDC2 privileged attribute, the register bit MDACi_MDAj_W0[PA], defines the privileged/user attribute on a hit.

typedef struct *_xrdc2_master_domain_assignment* xrdc2_master_domain_assignment_t

Domain assignment for the bus master.

XRDC2 compares the bus master *match input* with the parameter *xrdc2_master_domain_assignment_t::mask* and *xrdc2_master_domain_assignment_t::match* in this structure. If hit, the domain ID, privilege attribute, and secure attribute are used for the access.

typedef enum *_xrdc2_access_policy* xrdc2_access_policy_t

XRDC2 domain access control policy.

typedef enum *_xrdc2_access_config_lock* xrdc2_access_config_lock_t

Access configuration lock mode, the register field PDAC and MRGD LK2.

typedef struct *_xrdc2_periph_access_config* xrdc2_periph_access_config_t

XRDC2 peripheral domain access control configuration.

typedef struct *_xrdc2_mem_access_config* xrdc2_mem_access_config_t

XRDC2 memory region domain access control configuration.

typedef struct *_xrdc2_mem_slot_access_config* xrdc2_mem_slot_access_config_t

XRDC2 memory slot domain access control configuration.

void XRDC2_Init(XRDC2_Type *base)

Initializes the XRDC2 module.

Parameters

- base – XRDC2 peripheral base address.

void XRDC2_Deinit(XRDC2_Type *base)

De-initializes the XRDC2 module.

Parameters

- base – XRDC2 peripheral base address.

FSL_XRDC2_DRIVER_VERSION

Driver version.

XRDC2_EAL_FORCE_RELEASE_MAGIC_0

XRDC2_EAL_FORCE_RELEASE_MAGIC_1

XRDC2_EAL_DISABLE

XRDC2_EAL_DISABLE_UNTIL_RESET

XRDC2_EAL_UNLOCKED

XRDC2_EAL_LOCKED

XRDC2_EAL_MASK

struct `_xrdc2_master_domain_assignment`

#include <fsl_xrdc2.h> Domain assignment for the bus master.

XRDC2 compares the bus master *match input* with the parameter `xrdc2_master_domain_assignment_t::mask` and `xrdc2_master_domain_assignment_t::match` in this structure. If hit, the domain ID, privilege attribute, and secure attribute are used for the access.

Public Members

`bool lock`

Set true to lock the descriptor.

`xrdc2_privilege_attr_t privilegeAttr`

Privilege attribute.

`xrdc2_secure_attr_t secureAttr`

Secure attribute.

`uint8_t domainId`

Domain ID used when this descriptor hit.

`uint16_t mask`

Mask used for descriptor hit.

`uint16_t match`

Match used for descriptor hit.

struct `_xrdc2_periph_access_config`

#include <fsl_xrdc2.h> XRDC2 peripheral domain access control configuration.

Public Members

`xrdc2_access_config_lock_t lockMode`

PDACn lock configuration.

`xrdc2_access_policy_t policy[1]`

Access policy for each domain.

struct `_xrdc2_mem_access_config`

#include <fsl_xrdc2.h> XRDC2 memory region domain access control configuration.

Public Members

`uint32_t startAddr`

Memory region start address, should be 4k aligned.

`uint32_t endAddr`

Memory region end address, (endAddr + 1) should be 4k aligned.

`xrdc2_access_config_lock_t lockMode`

MRGDn lock configuration.

`xrdc2_access_policy_t policy[1]`

Access policy for each domain.

struct `_xrdc2_mem_slot_access_config`

#include <fsl_xrdc2.h> XRDC2 memory slot domain access control configuration.

Public Members

xrdc2_access_config_lock_t lockMode
Descriptor lock configuration.

xrdc2_access_policy_t policy[1]
Access policy for each domain.

Chapter 3

Middleware

3.1 Boot

3.1.1 MCUXpresso SDK : mcuxsdk-middleware-mcuboot_opensource

Overview

This repository is a fork of MCUboot (<https://github.com/mcu-tools/mcuboot>) for MCUXpresso SDK delivery and it contains the components officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository (mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

Documentation

Overall details can be reviewed here: [MCUXpresso SDK Online Documentation](#)

Visit [MCUboot - Documentation](#) to review details on the contents in this sub-repo.

Setup

Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest [Getting Started with SDK - Detailed Installation Instructions](#)

Contribution

Contributions are not currently accepted. If the intended contribution is not related to NXP specific code, consider contributing directly to the upstream MCUboot project. Once this MCUboot fork is synchronized with the upstream project, such contributions will end up here as well. If the intended contribution is a bugfix or improvement for NXP porting layer or for code added or modified by NXP, please open an issue or contact NXP support.

NXP Fork

This fork of MCUboot contains specific modifications and enhancements for NXP MCUXpresso SDK integration.

See *changelog* for details.

3.1.2 MCUboot



This is MCUboot version 2.2.0

MCUboot is a secure bootloader for 32-bits microcontrollers. It defines a common infrastructure for the bootloader and the system flash layout on microcontroller systems, and provides a secure bootloader that enables easy software upgrade.

MCUboot is not dependent on any specific operating system and hardware and relies on hardware porting layers from the operating system it works with. Currently, MCUboot works with the following operating systems and SoCs:

- [Zephyr](#)
- [Apache Mynewt](#)
- [Apache NuttX](#)
- [RIOT](#)
- [Mbed OS](#)
- [Espressif](#)
- [Cypress/Infineon](#)

RIOT is supported only as a boot target. We will accept any new port contributed by the community once it is good enough.

MCUboot How-tos

See the following pages for instructions on using MCUboot with different operating systems and SoCs:

- [Zephyr](#)
- [Apache Mynewt](#)
- [Apache NuttX](#)
- [RIOT](#)
- [Mbed OS](#)
- [Espressif](#)
- [Cypress/Infineon](#)

There are also instructions for the *Simulator*.

Roadmap

The issues being planned and worked on are tracked using GitHub issues. To give your input, visit [MCUboot GitHub Issues](#).

Source files

You can find additional documentation on the bootloader in the source files. For more information, use the following links:

- [boot/bootutil](#) - The core of the bootloader itself.
- [boot/boot_serial](#) - Support for serial upgrade within the bootloader itself.
- [boot/zephyr](#) - Port of the bootloader to Zephyr.
- [boot/mynewt](#) - Bootloader application for Apache Mynewt.
- [boot/nuttX](#) - Bootloader application and port of MCUboot interfaces for Apache NuttX.
- [boot/mbed](#) - Port of the bootloader to Mbed OS.
- [boot/espressif](#) - Bootloader application and MCUboot port for Espressif SoCs.
- [boot/cypress](#) - Bootloader application and MCUboot port for Cypress/Infineon SoCs.
- [imgtool](#) - A tool to securely sign firmware images for booting by MCUboot.
- [sim](#) - A bootloader simulator for testing and regression.

Joining the project

Developers are welcome!

Use the following links to join or see more about the project:

- [Our developer mailing list](#)
- [Our Discord channel](#) [Get your invite](#)

3.2 eIQ

3.2.1 eIQ

eIQ TensorFlow Lite for Micro Library User Guide

- [Overview](#)
- [TensorFlow Lite for Microcontrollers](#)
- [Build Status](#)
 - [Official Builds](#)
 - [Community Supported TFLM Examples](#)
 - [Community Supported Kernels and Unit Tests](#)
- [Contributing](#)
- [Getting Help](#)
- [Additional Documentation](#)

- [RFCs](#)

Overview TensorFlow Lite is an open source software library for running machine learning models on mobile and embedded devices. For more information, see www.tensorflow.org/lite.

For memory constrained devices, the library contains TensorFlow Lite for Microcontrollers. For more information, see www.tensorflow.org/lite/microcontrollers.

The MCUXpresso Software Development Kit (MCUXpresso SDK) provides a comprehensive software package with a pre-integrated TensorFlow Lite for Microcontrollers based on version 25-04-08 (from the 8th of April 2025 with [commit](#)). This document describes the steps required to download and start using the library. Additionally, the document describes the steps required to create an application for running pre-trained models.

Note: The document also assumes knowledge of machine learning frameworks for model training.

TensorFlow Lite for Microcontrollers TensorFlow Lite for Microcontrollers is a port of TensorFlow Lite designed to run machine learning models on DSPs, microcontrollers and other devices with limited memory.

Additional Links:

- [Tensorflow github repository](#)
- [TFLM at tensorflow.org](#)

Build Status

- [GitHub Status](#)

Official Builds

Build Type	Status
CI (Linux)	 Run-CI passing
Code Sync	 Sync from Upstream TF passing

Community Supported TFLM Examples This table captures platforms that TFLM has been ported to. Please see *New Platform Support* for additional documentation.

Platform	Status
Arduino	 CI no status  Arduino examples tests no status
Coral Dev Board Micro	TFLM + EdgeTPU Examples for Coral Dev Board Micro
Espressif Systems Dev Boards	 CI failing
Renesas Boards	TFLM Examples for Renesas Boards
Silicon Labs Dev Kits	TFLM Examples for Silicon Labs Dev Kits
Sparkfun Edge	 CI no status
Texas Instruments Dev Boards	

Community Supported Kernels and Unit Tests This is a list of targets that have optimized kernel implementations and/or run the TFLM unit tests using software emulation or instruction set simulators.

Build Type	Status
Cortex-M	 Cortex-M passing
Hexagon	 Run-Hexagon passing
RISC-V	 RISC-V passing
Xtensa	 Run-Xtensa passing
Generate Integration Test	 Generate Integration Tests passing

Contributing See our *contribution documentation*.

Getting Help A [Github issue](#) should be the primary method of getting in touch with the TensorFlow Lite Micro (TFLM) team.

The following resources may also be useful:

1. SIG Micro [email group](#) and [monthly meetings](#).
2. SIG Micro [gitter chat room](#).
3. For questions that are not specific to TFLM, please consult the broader TensorFlow project, e.g.:
 - Create a topic on the [TensorFlow Discourse forum](#)
 - Send an email to the [TensorFlow Lite mailing list](#)
 - Create a [TensorFlow issue](#)
 - Create a [Model Optimization Toolkit issue](#)

Additional Documentation

- *Continuous Integration*
- *Benchmarks*
- *Profiling*
- *Memory Management*
- *Logging*
- *Porting Reference Kernels from TfLite to TFLM*
- *Optimized Kernel Implementations*
- *New Platform Support*
- Platform/IP support
 - *Arm IP support*
- *Software Emulation with Renode*
- *Software Emulation with QEMU*

- *Python Dev Guide*
- *Automatically Generated Files*
- *Python Interpreter Guide*

RFCs

1. *Pre-allocated tensors*
2. *TensorFlow Lite for Microcontrollers Port of 16x8 Quantized Operators*

Deployment The eIQ TensorFlow Lite for Microcontrollers library is part of the eIQ machine learning software package, which is an optional middleware component of MCUXpresso SDK. The eIQ component is integrated into the MCUXpresso SDK Builder delivery system available on mcuxpresso.nxp.com. To include eIQ machine learning into the MCUXpresso SDK package, the eIQ middleware component is selected in the software component selector on the SDK Builder page when building a new package. See *Figure 1*.

MCUXpresso SDK Builder

SDK Builder

Generate a downloadable SDK archive for use with desktop MCUXpresso Tools.

Developer Environment Settings

Selections here will impact files and examples projects included in the SDK and Generated Projects

Toolchain / IDE: All toolchains
Host OS: Windows
Embedded real-time operating system: Bare-Metal

Filter by Name, Category, or Descriptor Select All Unselect All

	Name	Category	Description	Dependencies
☒	CMSIS DSP Library	CMSIS DSP Lib	CMSIS DSP Software Library	
☐	canopen	Middleware	canopen library	
☒	eIQ	Middleware	eIQ machine learning SDK containing: - ARM CMSIS-NN library (neural network kern... (more)	
☐	Embedded Wizard GUI	Middleware	Embedded Wizard GUI	
☐	emWin	Middleware	emWin graphics library	
☐	FatFS	Middleware	FAT File System	

Once the MCUXpresso SDK package is downloaded, it can be extracted on a local machine or imported into the MCUXpresso IDE. For more information on the MCUXpresso SDK folder structure, see the Getting Started with MCUXpresso SDK User's Guide (document: MCUXSDKGSUG). The package directory structure is similar to *Figure 2*. The eIQ TensorFlow Lite library directories are highlighted in red.

- SDK_2_15_000_EVKB-IMXRT1050
 - boards
 - evkbimxrt1050
 - cmsis_driver_examples
 - component_examples
 - demo_apps
 - driver_examples
 - eiq_examples
 - deepviewrt_camera_label_image
 - deepviewrt_image_detection
 - deepviewrt_labelimage
 - glow_cifar10
 - glow_cifar10_camera
 - glow_lenet_mnist
 - glow_lenet_mnist_camera
 - tflm_cifar10
 - tflm_kws
 - tflm_label_image
 - tflm_lib
 - littlefs_examples
 - lwip_examples
 - project_template
 - sdmmc_examples
 - xip
 - CMSIS
 - components
 - devices
 - docs
 - middleware
 - bm
 - cjson
 - eiq
 - deepviewrt
 - doc
 - glow
 - mpp
 - tensorflow-lite
 - lib
 - signal
 - tensorflow
 - third_party

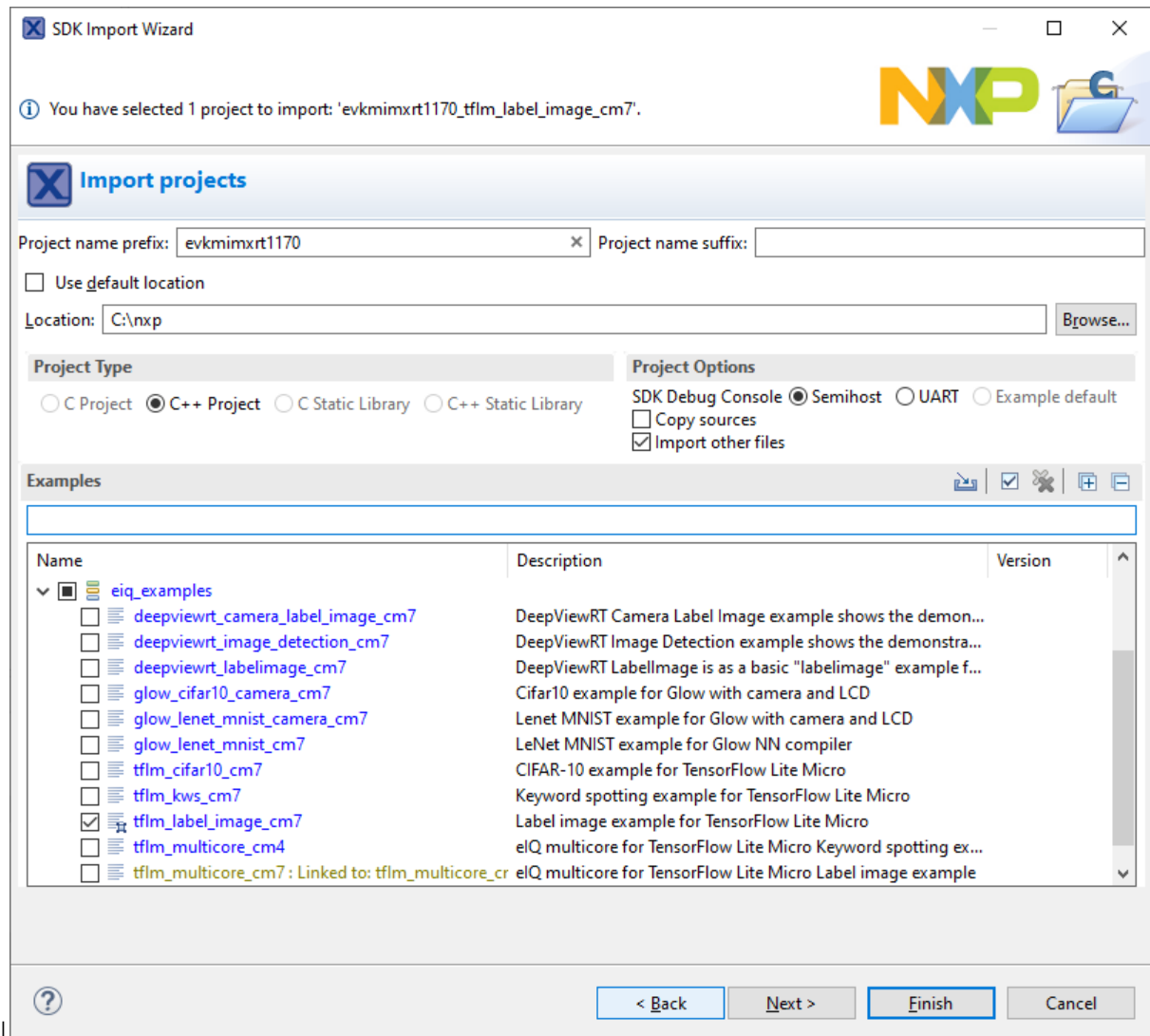
- MIMXRT700-EVK
 - arch
 - boards
 - mimxrt700evk
 - **eiq_examples**
 - mpp_static_image_mobilenet_view
 - mpp_static_image_ultraface_view_tflm
 - tflm_cifar10
 - tflm_cifar10_hifi4
 - tflm_kws
 - tflm_label_image
 - tflm_label_image_ext_mem
 - tflm_label_image_hifi4
 - tflm_lib
 - tflm_modelrunner
 - flash_config
 - project_template
 - CMSIS
 - components
 - devices
 - docs
 - merged_data
 - middleware
 - aws_iot
 - bm
 - dsp
 - eiq
 - doc
 - mpp
 - tensorflow-lite
 - lib
 - signal
 - tensorflow
 - third_party
 - cmsis_nn
 - fft2d
 - flatbuffers
 - gemmlowp
 - kissfft
 - neutron
 - common
 - driver
 - rt700
 - ruy
 - xa nnlib hifi4

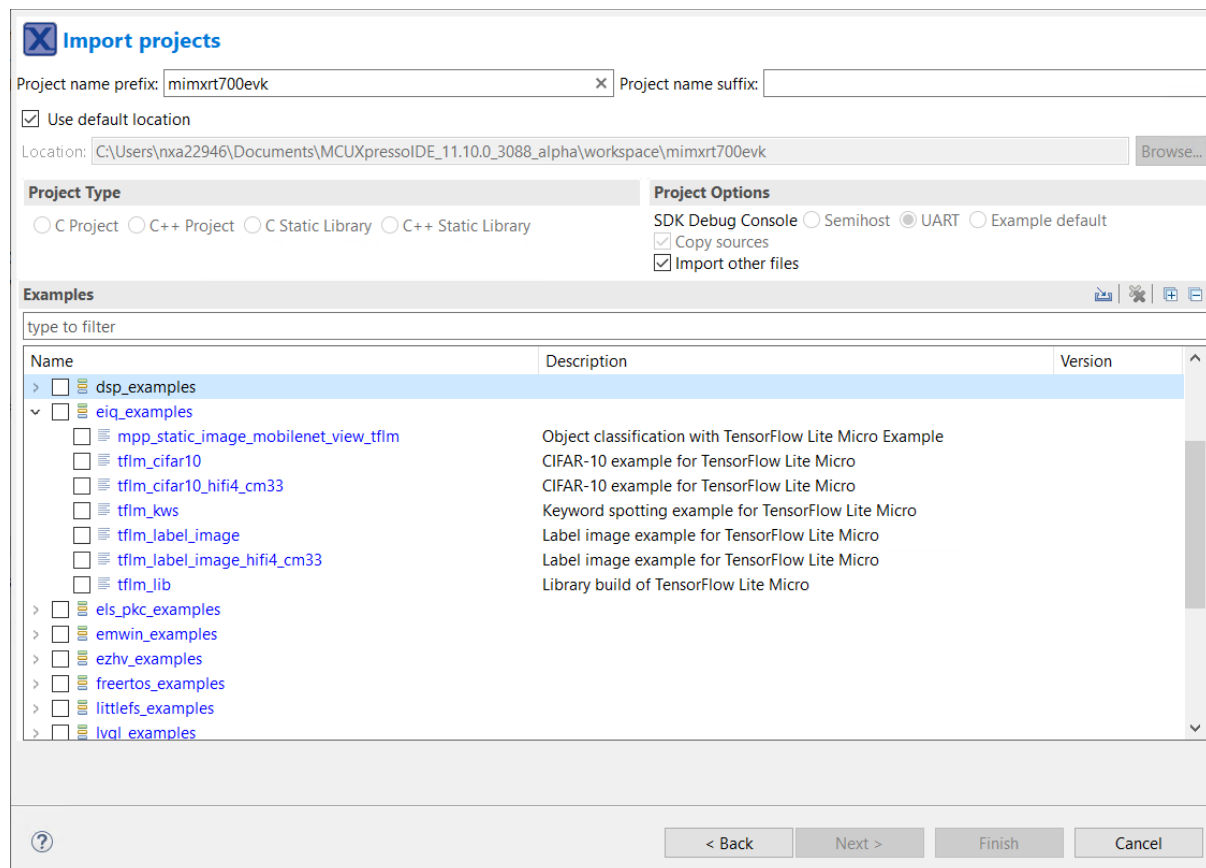
The *boards* directory contains example application projects for supported toolchains. For the list of supported toolchains, see the *MCUXpresso SDK Release Notes*. The *middleware* directory contains the eIQ library source code and example application source code and data.

Example applications The eIQ TensorFlow Lite library is provided with a set of example applications. For details, see *Table 1*. The applications demonstrate the usage of the library in several use cases.

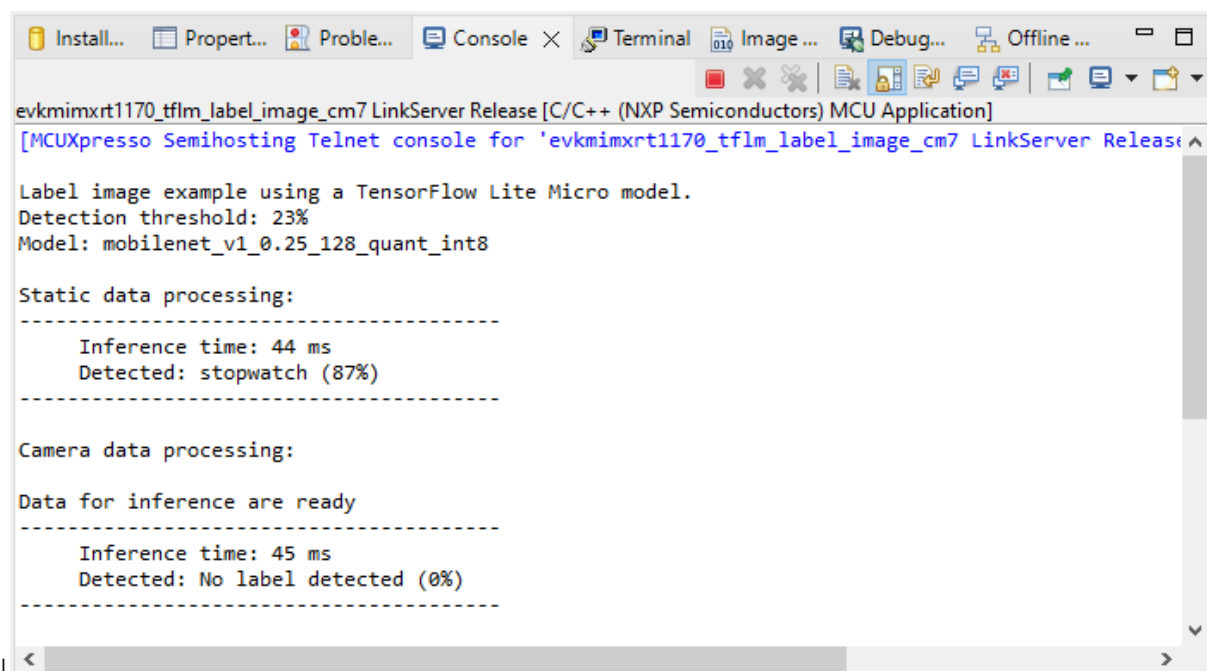
Name	Description	Availability
tflm_c	CIFAR-10 classification of 32×32 RGB pixel images into 10 categories using a small Convolutional Neural Network (CNN).	MCX-N947-EVK (no camera and display support) MCX-N947-FRDM (no camera and display support) MCX-N547-EVK (no camera and display support) MIMXRT700-EVK (no camera and display support)
tflm_l	Keyword spotting application using a neural network for word detection in pre-processed audio input.	MCX-N947-EVK (no audio support) MCX-N947-FRDM (no audio support) MCX-N547-EVK (no audio support) MIMXRT700-EVK (no audio support)
tflm_l	Image recognition application using a MobileNet model architecture to classify 128×128 RGB pixel images into 1000 categories with eIQ Neutron NPU.	MCX-N947-EVK (no camera and display support) MCX-N947-FRDM (no camera and display support) MCX-N547-EVK (no camera and display support) MIMXRT700-EVK (no camera and display support)
tflm_l	Image recognition application using a MobileNet model architecture to classify 224×224 RGB pixel images into 1000 categories with eIQ Neutron NPU. In this example, it demonstrates how to fetch model's weight from external memory (xSPI flash) to internal SRAM for Neutron NPU execution.	MIMXRT700-EVK (no camera and display support)
tflm_c	CIFAR-10 classification of 32×32 RGB pixel images into 10 categories using a small Convolutional Neural Network. In this example, M33 core0 starts HiFi4 DSP core with HiFi4 DSP image. HiFi4 DSP does the inference for CIFAR-10 classification.	MIMXRT700-EVK (no camera and display support)
tflm_l	Image recognition application using a MobileNet model architecture to classify 128×128 RGB pixel images into 1000 categories. In this example, M33 core0 starts HiFi4 DSP core with HiFi4 DSP image. HiFi4 DSP does the inference for image recognition application.	MIMXRT700-EVK (no camera and display support)

For details on how to build and run the example applications with supported toolchains, see *Getting Started with MCUXpresso SDK User's Guide* (document: MCUXSDKGSUG). When using MCUXpresso IDE, the example applications can be imported through the SDK Import Wizard as shown in *Figure 1*.





After building the example application and downloading it to the target, the execution stops in the *main* function. When the execution resumes, an output message displays on the connected terminal. For example, *Figure 2* shows the output of the `tflm_label_image_cm7` `tflm_label_image` example application printed to the MCUXpresso IDE Console window when semihosting debug console is selected in the SDK Import Wizard.



```

Label image example using a TensorFlow Lite Micro model.
Detection threshold: 23%
Model: mobilenet_v1_0.25_128_quant_int8_npu

Static data processing:
-----
      Inference time: 3987 us
      Detected: stopwatch (87%)
-----

```

Model Conversion to TensorFlow Lite Format The eIQ® Toolkit provides a comprehensive end-to-end environment for machine learning (ML) model development and deployment. Designed for NXP EdgeVerse processors, the toolkit includes both an intuitive GUI-based tool (eIQ Portal) and command-line utilities for advanced workflows.

One key component, the eIQ ModelTool, enables seamless conversion of ML models from popular formats such as TensorFlow, PyTorch, and ONNX into the TensorFlow Lite (TFLite) format. These converted models can be further optimized and deployed on NXP platforms for inference acceleration.

Model Conversion for NXP eIQ Neutron NPU To leverage the NXP eIQ Neutron NPU for hardware acceleration, models must undergo additional processing using the Neutron Converter Tool. This tool transforms standard quantized TensorFlow Lite models into a format optimized for execution on the Neutron NPU.

The key steps involved in this process are as follows:

1. **Convert to Quantized TensorFlow Lite Model:** Ensure the model is in a quantized TFLite format before running the Neutron Converter.
2. **Run the Neutron Converter Tool:** The Neutron Converter analyzes the TFLite model, identifies supported operators, and replaces them with specialized NPU-compatible nodes. Unsupported operations are executed using fallback mechanisms, such as:
 - CMSIS-NN for optimized CPU execution
 - Reference Operators for unsupported cases
3. **Execute on Target Platform:** The converted model runs efficiently on the Neutron NPU using a custom TFLite Micro-operator implementation.

Example: Converting a Quantized TensorFlow Lite Model for Neutron NPU The following is a sample command-line invocation for the Neutron Converter tool:

```

neutron-converter --input mobilenet_v1_0.25_128_quant.tflite \
                  --output mobilenet_v1_0.25_128_quant_npu.tflite \
                  --target imxrt700 \
                  --dump-header-file-output

```

Note: This will convert the source tflite model to neutron compatible model, meanwhile, it will dump the model as one headfile name as “mobilenet_v1_0.25_128_quant_npu.h”.

Run and debug eIQ HiFi4 and HiFi1 DSP examples using Xplorer IDE This section lists the steps to Prepare CM33 Core for the examples and Prepare DSP core for the examples.

Prepare CM33 Core for the examples

1. The `tflm_cifar10_hifi4` and `tflm_label_image_hifi4` examples consist of two separate applications that run on the CM33 core0 and DSP core. The CM33 core0 application initializes the DSP core and starts it.

To debug the application:

1. Set up and execute the CM33 application using an environment of your choice.
2. Build and execute the examples located in:

`<SDK_ROOT>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/cm33/`

`<SDK_ROOT>/boards/mimxrt700evk/eiq_examples/tflm_label_image_hifi4/cm33/`

2. The `tflm_cifar10_hifi1` example consists of three separate applications that run on the CM33 core0, CM33 core1, and DSP core. The CM33 core0 application initializes the CM33 core1 core and starts it. The CM33 core1 application initializes the DSP core and starts it.

To debug the application:

1. Set up and build the CM33 core1 application using an environment of your choice.
2. Set up and execute the CM33 core0 application using an environment of your choice.
3. Build and execute the example located in:

`<SDK_ROOT>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi1/cm33_core1/`

`<SDK_ROOT>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi1/cm33_core0/`

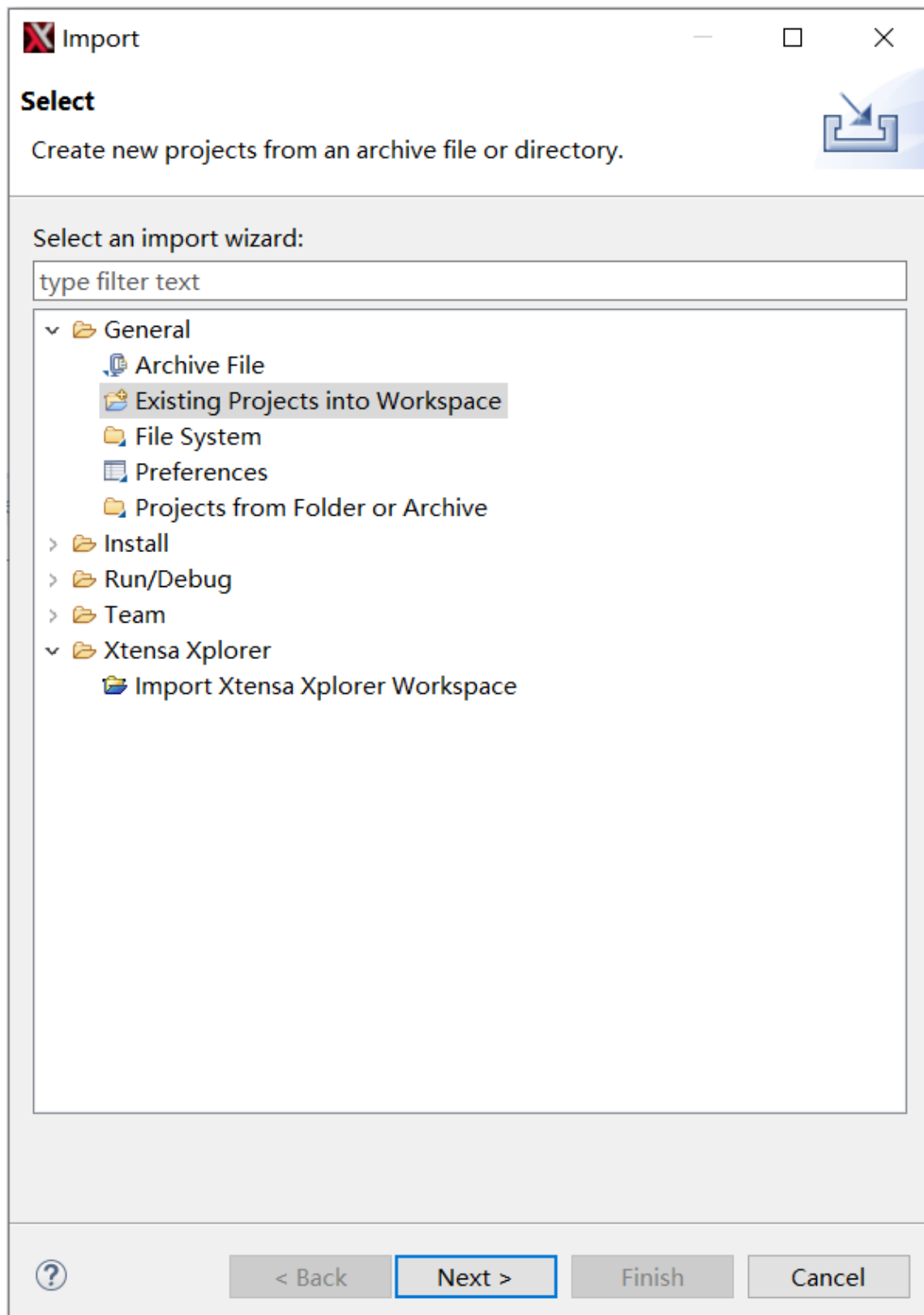
Note: ARMGCC toolchain and IAR Embedded Workbench are both supported. To enable compatibility with RT700, IAR Embedded Workbench may require a patch. There are default DSP core images in the SDK. For details on how to build the examples, refer to Prepare DSP core for the examples.

Parent topic: [Run and debug eIQ HiFi4 and HiFi1 DSP examples using Xplorer IDE](#)

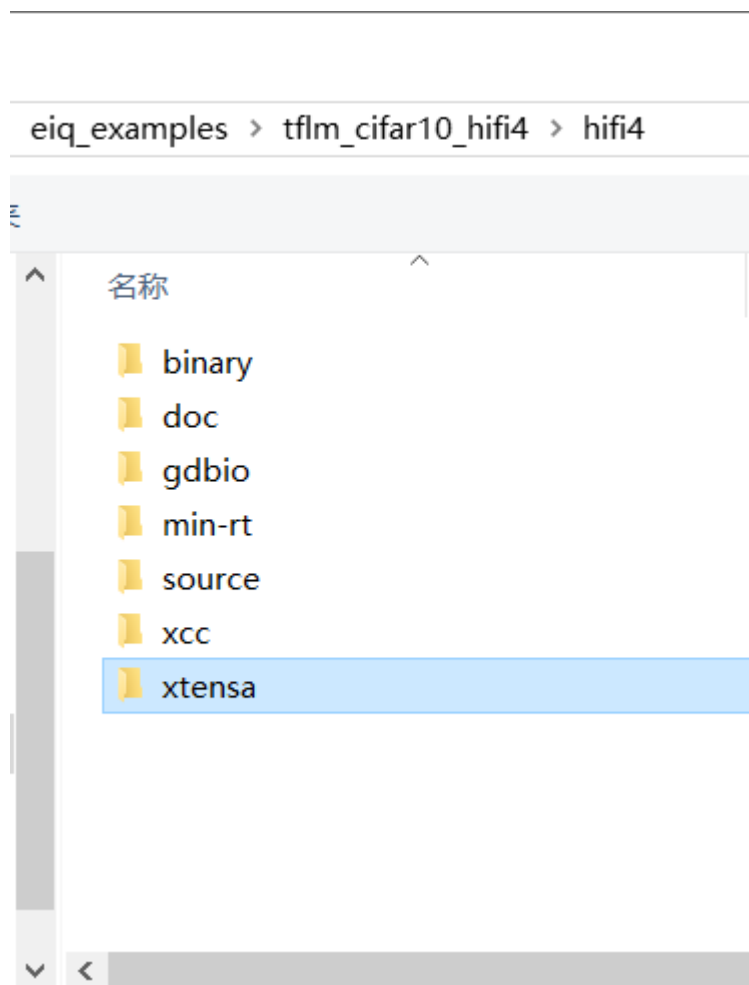
Prepare DSP core for the examples The projects for different supported toolchains are built. The “xcc” project builds on the command line and the “xtensa” directory is an Xplorer IDE project.

To run the `tflm_cifar10_hifi4` example, import the SDK sources into the Xplorer IDE.

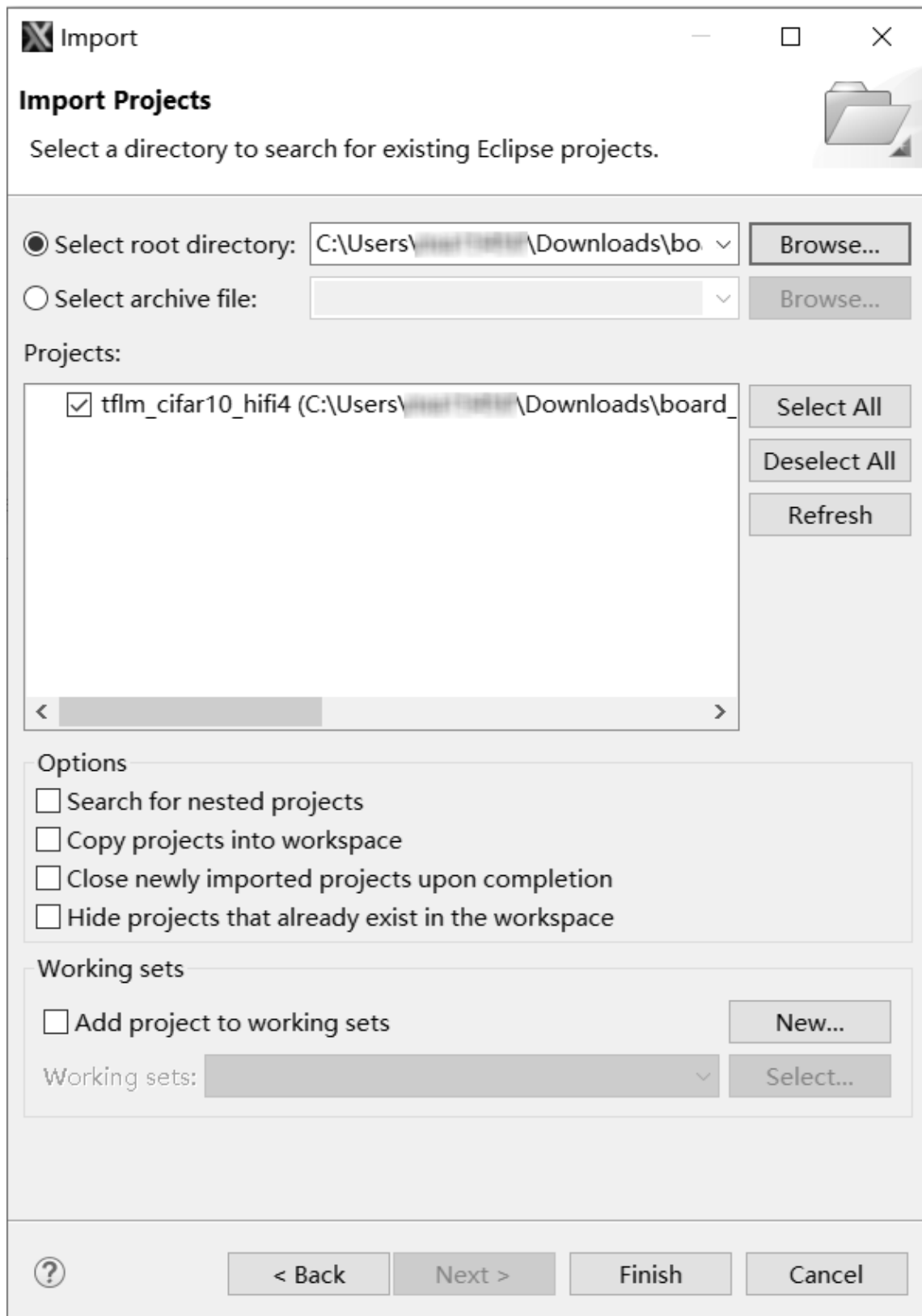
1. Select **File > Import > General > Existing Projects into Workspace**.



2. Click **Next**.
3. Select the SDK directory/boards/mimxrt700evk/eiq_examples/tfm_cifar10_hifi4/hifi4/xtensa as the root directory.

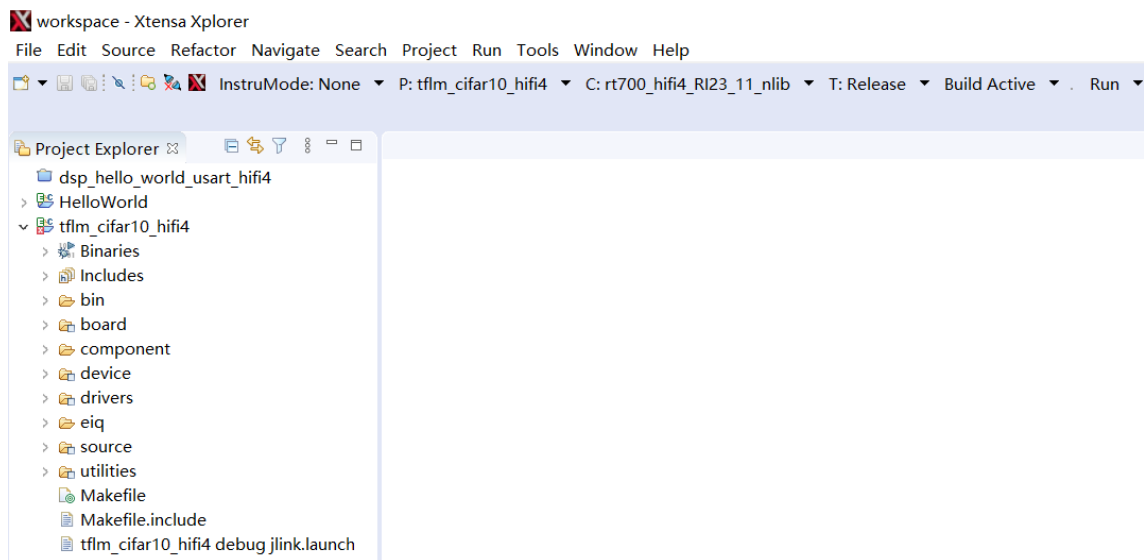


4. Click **Select Folder**.
5. Leave all the other options check boxes blank.

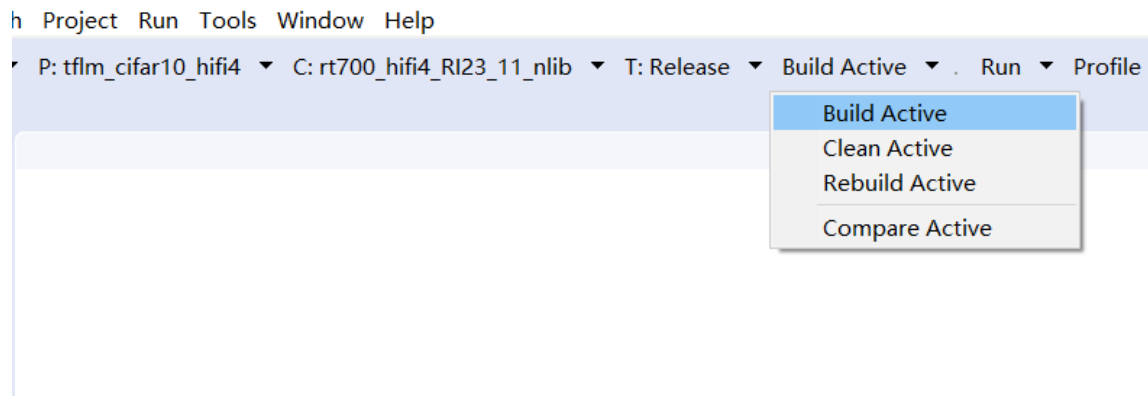


Once imported, the `tflm_cifar10_hifi4` example appears in the **Project Explorer**.

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



7. To build the DSP application image for the CM33 application, select the **Release target** option in the Xplorer IDE as below.



8. Three DSP binaries are generated and are loaded into different TCM or SRAM address segments:
- <SDK_ROOT>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/hifi4/binary/dsp_data_release.bin
 - <SDK_ROOT>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/hifi4/binary/dsp_literal_release.bin
 - <SDK_ROOT>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/hifi4/binary/dsp_text_release.bin

Parent topic: [Run and debug eIQ HiFi4 and HiFi1 DSP examples using Xplorer IDE](#)

Running an inference After converting the model to the TensorFlow Lite format, it is converted into a C language array to include it in the application source code. The `xxd` utility can be used for this purpose (distributed with the *Vim* editor for many platforms on <https://www.vim.org/>) as shown in *Converting a model to a C language header file*. The utility converts a TensorFlow Lite model into a C header file with an array definition containing the binary image of the model and a variable containing the data size.

Converting a model to a C language header file {#EXAMPLE_4.section}

```
xxd -i mobilenet_v1_0.25_128_quant.tflite > mobilenet_v1_0.25_128_quant_model.h
```

After the header file is generated, the type of the array is changed from `unsigned char` to `const char` to match the library API input parameters and the default array name can be changed to a more convenient one. The user must align the buffer to at least 64-bit boundary (the size of a double-precision floating-point number) to avoid misaligned memory access. The alignment can be achieved by using the `__ALIGNED(16)` macro from the `cmsis_compiler.h` header file (available in the MCUXpresso SDK) in the array declaration before the data assignment.

The easiest way to create an application with the proper configuration is to copy and modify an existing example application. To learn where to find the example applications and how to build them, see the [Example applications](#).

Running an inference using TensorFlow Lite for Microcontrollers involves several steps (shown for quantized model with signed 8-bit values as input and 32-floating point values as output):

1. Include the necessary eIQ TensorFlow Lite Micro library header files and the converted model.

Including header files

```
#include "tensorflow/lite/micro/micro_error_reporter.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/micro/all_ops_resolver.h"
#include "mobilenet_v1_0.25_128_quant_model.h"
```

2. Allocate a static memory buffer for input and output tensors and intermediate arrays. Load the FlatBuffer model image (assuming the `mobilenet_v1_0.25_128_quant_model.h` file generated in *Converting a model to a C language header file* defines an array named `mobilenet_model` and a size variable named `mobilenet_model_len`), build the interpreter object and allocate memory for tensors.

Loading the FlatBuffer model

```
constexpr int kTensorArenaSize = 1024 * 1024;
static uint8_t tensorArena[kTensorArenaSize];
const tflite::Model* model = tflite::GetModel(mobilenet_model);
// TODO: Report an error if model->version() != TFLITE_SCHEMA_VERSION
static tflite::AllOpsResolver microOpResolver;
static tflite::MicroErrorReporter microErrorReporter;
static tflite::MicroInterpreter interpreter(model,
    microOpResolver, tensorArena, kTensorArenaSize,
    microErrorReporter);
interpreter->AllocateTensors();
// TODO: Check return value for kTfLiteOk
```

3. Fill the input data into the input tensor. For example, if a speech recognition model, image data from a camera or audio data from a microphone. The dimensions of the input data must be the same as the dimensions of the input tensor. These dimensions were specified when the model was created.

Fill-in input data

```
// Get access to the input tensor data
TfLiteTensor* inputTensor = interpreter->input(0);
// Copy the input tensor data from an application buffer
for (int i = 0; i < inputTensor->bytes; i++)
    inputTensor->data.int8[i] = input_data[i];
```

4. Run the inference and read the output data from the output tensor. The dimensions of the output data must be the same as the dimensions of the output tensor. These dimensions were specified when the model was created.

Running inference and reading output data

```
// Run the inference
interpreter->Invoke();
// TODO: Check the return value for TfLiteOk
// Get access to the output tensor data
TfLiteTensor* outputTensor = interpreter->output(0);
// Copy the output tensor data to an application buffer
for (int i = 0; i < outputTensor->bytes / sizeof(float32); i++)
    output_data[i] = outputTensor->data.f[i];
```

NPU inference {#npu_infer.section} Running an inference using a model converted for the NPU requires registration of a custom operator implementation. First the header file with the custom operator implementation interface must be included.

```
#include "tensorflow/lite/micro/kernels/micro_ops.h"
#include "tensorflow/lite/micro/all_ops_resolver.h"
#include "tensorflow/lite/micro/kernels/neutron/neutron.h"
```

Next, the specialized implementation has to be registered in the operator resolver object.

```
static tf::AllOpsResolver microOpResolver;
microOpResolver.AddCustom(tf::GetString_NEUTRON_GRAPH(),
    tf::Register_NEUTRON_GRAPH());
```

The specialized NPU nodes from the converted model are the executed using this newly registered implementation.

Adjusting the tensor arena size {#adjust_arena.section} The tensor arena is a static memory buffer used for intermediate tensor and scratch buffer allocation. The size of the tensor arena buffer is set by the `kTensorArenaSize` constant in the example above. The value depends on the tensor sizes used in the model and on the hardware-specific implementations of kernels, which may require various sizes of scratch buffers for intermediate computations. The value can be determined experimentally by running an inference with a small value, so the library fails with an insufficient tensor memory error and prints the missing amount. Continue adjusting the size until the error stops being reported. If the target hardware changes, readjust the value.

Code size optimization Typically, models do not use all the operators that are available in TensorFlow Lite. However, because of the default operator registration mechanism used in the library, the toolchain linker is not able to remove the code of unused operators. In order to reduce code size, it is possible to only register the specific operators used by a model. To determine which operators are used by a particular model, a model visualizer tool like Netron can be used. Then a mutable operator resolver object can be created that only registers the operators that are used by the model being inferred.

Use the `tf::MicroMutableOpResolver` object template, which is later passed to the `tf::MicroInterpreter` object. Depending on the list of used operators, the result should be similar to the following code snippet. Make sure to update the `MicroMutableOpResolver` template parameter to reflect the number of operators that need to be registered.

Register only used operators in TensorFlow Lite Micro {#SECTION_SS1_DJQ_QPB.section}

```
#include "tensorflow/lite/micro/kernels/micro_ops.h"
#include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
tf::MicroMutableOpResolver<6> microOpResolver;
microOpResolver.AddAveragePool2D();
microOpResolver.AddConv2D();
microOpResolver.AddDepthwiseConv2D();
```

(continues on next page)

(continued from previous page)

```
microOpResolver.AddDequantize();
microOpResolver.AddReshape();
microOpResolver.AddSoftmax();
static tfLite::MicroInterpreter interpreter(
    model, microOpResolver, tensorArena, kTensorArenaSize, microErrorReporter);
```

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

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

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

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

eIQ ExecuTorch Library User Guide

Overview ExecuTorch is an end-to-end solution for enabling on-device inference capabilities across mobile and edge devices including wearables, embedded devices and microcontrollers. It is part of the PyTorch Edge ecosystem and enables efficient deployment of PyTorch models to edge devices. For more information, see <https://pytorch.org/executorch-overview>.

The MCUXpresso Software Development Kit (MCUXpresso SDK) provides a comprehensive software package with a pre-integrated ExecuTorch based on version v0.5.0 with initial support for Neutron Backend. Neutron Backend enables acceleration of ML models on the **eIQ® Neutron Neural Processing Unit (NPU)**.

This document describes the steps required to download and start using the ExecuTorch. Additionally, the document describes the steps required to create an application for running pre-trained models.

Note: The document also assumes knowledge of machine learning frameworks for model training.

Supported platforms:

- **i.MX RT700**

Installation The ExecuTorch, with the Neutron Backend consists of:

- ExecuTorch with Neutron Backend for Ahead of Time ML Model Compilation
- Neutron Converter
- MCUXpresso SDK

Here we briefly describe each components purpose and steps to install them.

The **ExecuTorch AoT** and **Neutron Converter** are needed to convert a PyTorch model to ExecuTorch and Delegate it to eIQ Neutron NPU using the Neutron Backend. The **MCUXpresso SDK** provides project to build the ExecuTorch Runtime Library, the example application with simple CNN, toolchains and other middleware libraries to build and deploy the application on the target platform.

If you want run to prepared example application on the i.MX RT700 platform, and skip the model preparation phase continue with the *MCUXpresso SDK Part*.

ExecuTorch for Ahead of Time model preparation The ExecuTorch enables to deploy PyTorch models on edge devices. For this purpose the PyTorch model must be processed and converter by the ExecuTorch Ahead of Time (AoT) part. You can obtain the full ExecuTorch including the AoT part aligned with this version of MCUX SDK from the [mcuxsdk-middleware-executorch](#) release/mcux-full branch.

Installation Prerequisites:

- x86 Linux Machine with GLIBC-2.29 or higher (e.g. Ubuntu 20.04 or higher)
- Python 3.10, 3.11 or 3.12

To build and install the ExecuTorch follow these steps:

1. (Optional) Setup python virtual environment on desired location and activate it.

```
$ python3 -m venv venv
$ source venv/bin/activate
```

2. Clone the ExecuTorch from [mcuxsdk-middleware-executorch](#)

```
$ git clone --branch release/mcux-full https://github.com/nxp-mcuxpresso/mcuxsdk-middleware-executorch.git
$ cd mcuxsdk-middleware-executorch
$ git submodule update --init --recursive
```

3. Build and install the ExecuTorch and its dependencies:

```
$ ./install_requirements.sh
```

[!WARNING] The install_requirements.sh installs the CPU version of torch from <https://download.pytorch.org/whl/cpu>. If you are behind corporate proxy, it might have issues accessing it and you will see warnings like:

```
WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None,
↪status=None)) after connection broken by 'SSLError(SSLCertVerificationError(1, '[SSL:
↪CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer
↪certificate (_ssl.c:1006')))': /whl/test/cpu/torch/
```

In this case the CUDA version of torch is installed and the install_requirements.sh script fails with:

```
PyTorch: CUDA cannot be found. Depending on whether you are building
```

Make sure the pip can access the <https://download.pytorch.org/whl/cpu> PyPI.

Next continue with installation of the [Neutron Converter](#)

Neutron Converter The eIQ Neutron Backend uses the Neutron Converter to convert the ExecuTorch program to the eIQ Neutron NPU microcode.

Installation The Neutron Converter is available as a Python package and can be installed by the pip command from eiq.nxp.com/repository:

```
pip install --index-url https://eiq.nxp.com/repository neutron_converter_SDK_25_09==1.0.0
```

The Neutron Converter is used internally by the ExecuTorch, and it is tied to the particular BSP you are using - the suffix of the python package name. In the code snippet above the flavor is the SDK_25_09. In the `aot_neutron_convert.py` example script by the `--neutron_converter_flavor` parameter.

MCUXpresso SDK The MCUXpresso SDK is used to build, debug and deploy the application using the ExecuTorch on the target platform.

You can obtain the MCUXpresso SDK from [MCUXpresso SDK Builder](#) including the IDE. See the [getting_mcuxpress](#) for details.

In the MCUXpresso SDK, there are 2 projects available related to ExecuTorch:

- `executorch_lib`
- `executorch_cifarnet`

For more details see [example_applications](#). Here you will find the details to run build and run the demo applications.

Getting the MCUXpresso SDK with eIQ ExecuTorch The eIQ ExecuTorch library is part of the eIQ machine learning software package, which is an optional middleware component of MCUXpresso SDK. The eIQ component is integrated into the MCUXpresso SDK Builder delivery system available on mcuxpresso.nxp.com. To include eIQ machine learning into the MCUXpresso SDK package, the eIQ middleware component is selected in the software component selector on the SDK Builder page when building a new package. See *Figure 1*.

mcuxpresso.nxp.com/en/builder

MCUXpresso SDK Builder

SDK Builder

Generate a downloadable SDK archive for use with desktop MCUXpresso Tools.

Developer Environment Settings

Selections here will impact files and examples projects included in the SDK and Generated Projects

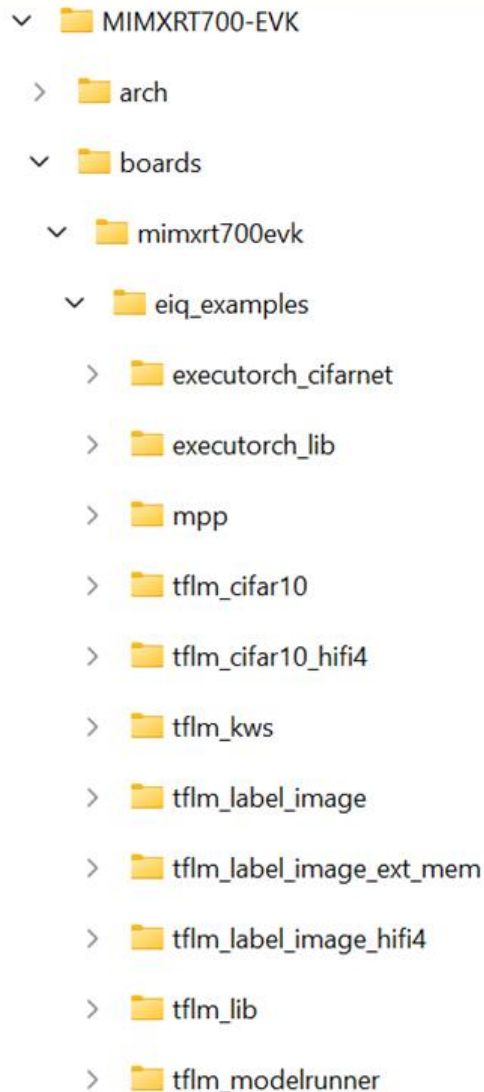
Toolchain / IDE: All toolchains ▼ Host OS: Windows ▼

Embedded real-time operating system: Bare-Metal ▼

Filter by Name, Category, or Descriptor Select All Unselect All

	Name	Category	Description	Dependencies
☒	CMSIS DSP Library	CMSIS DSP Lib	CMSIS DSP Software Library	
☐	canopen	Middleware	canopen library	
☒	eIQ	Middleware	eIQ machine learning SDK containing: - ARM CMSIS-NN library (neural network kern... (more)	
☐	Embedded Wizard GUI	Middleware	Embedded Wizard GUI	
☐	emWin	Middleware	emWin graphics library	
☐	FatFS	Middleware	FAT File System	

Once the MCUXpresso SDK package is downloaded, it can be extracted on a local machine or imported into the MCUXpresso IDE. For more information on the MCUXpresso SDK folder structure, see the Getting Started with MCUXpresso SDK User's Guide (document: MCUXSDKGSUG). The package directory structure is similar to *Figure 2*.



The *boards* directory contains example application projects for supported toolchains. For the list of supported toolchains, see the *MCUXpresso SDK Release Notes*. The *middleware* directory contains the eIQ library source code and example application source code and data.

PyTorch Model Conversion to ExecuTorch Format In this guideline we will show how to use the ExecuTorch AoT part to convert a PyTorch model to ExecuTorch format and delegate the model computation to eIQ Neutron NPU using the eIQ Neutron Backend.

First we will start with an example script converting the model. This example show the CifarNet model preparation. It is the same model which is part of the `example_cifar10`

The steps are expected to be executed from the `executorch` root folder, in our case the `mcuxsdk-middleware-executorch`

1. After building the ExecuTorch you shall have the `libquantized_ops_aot_lib.so` located in the `pip-out` folder. We will need this library when generating the quantized cifar10 ExecuTorch model. So as first step we will find it:

```
$ find ./pip-out -name 'libquantized_ops_aot_lib.so'
./pip-out/temp.linux-x86_64-cpython-310/cmake-out/kernels/quantized/libquantized_ops_aot_lib.so
./pip-out/lib.linux-x86_64-cpython-310/executorch/kernels/quantized/libquantized_ops_aot_lib.so
```

2. Now run the `aot_neutron_compile.py` example with the `cifar10` model


```
$ python examples/nxp/aot_neutron_compile.py \
--quantize --so_library ./pip-out/lib.linux-x86_64-cpython-310/executorch/kernels/quantized/libquantized_
ops_aot_lib.so \
--delegate --neutron_converter_flavor SDK_25_09 -m cifar10
```

3. It will generate you `cifar10_nxp_delegate.pte` file which can be used with the MCUXpresso SDK `cifar10_example` project.

The generated PTE file is used in the `executorch_cifar10` example application, see [example application](#).

MCUXpresso SDK Example applications The MCUXpresso SDK provides a set of projects and example application with the eIQ ExecuTorch. For details, see *Table 1*.

The eIQ ExecuTorch library is provided with a set of example applications. For details, see *Table 1*. The applications demonstrate the usage of the library in several use cases.

Name	Description	Availability
ex-ecu-torch_	This project contains the ExecuTorch Runtime Library source code and is used to build the ExecuTorch Runtime Library. The library is further used to build a full application using the leveraging ExecuTorch.	MIMXRT700-EVK (no camera and display support)
ex-ecu-torch_	Example application demonstrating the use of the ExecuTorch running a CifarNet classification model accelerated on the eIQ Neutron NPU. The CifarNet is a small Convolutional Neural Network (CNN), trained on CIFAR-10 [1] dataset. The model clasifies the input images into 10 caterories.	MIMXRT700-EVK (no camera and display support)

For details on how to build and run the example applications with supported toolchains, see *Getting Started with MCUXpresso SDK User's Guide* (document: MCUXSDKGSUG).

How to build and run `executorch_cifar10` example The example needs ExecuTorch Runtime Library and Neutron Libraries.

ExecuTorch Runtime Library:

- `middleware/eiq/executorch/lib/cm33/armgcc/libexecutorch.a`

Neutron Libraries:

- `middleware/eiq/executorch/third-party/neutron/rt700/libNeutronDriver.a` and
- `middleware/eiq/executorch/third-party/neutron/rt700/libNeutronFirmware.a`

In the example the model and the input image is already embedded into the program and ready to build and deploy to i.MX RT700, so you can continue right to the [building and deployment](#) section.

Convert the model and example input to C array In this section we describe where the model and example input is located in the example application sources, and how it was generated.

The **cifar10 model** ExecuTorch model is stored in `boards/mimxrt700evk/eiq_examples/executorch_cifar10/cm33_core0/model_pte.h`. and was generated from the `cifar10_nxp_delegate.pte` (see [convert model](#)).

We use the `xxd` command to get the C array containing the model data and array size:

```
$ xxd -i cifar10_nxp_delegate.pte > model_pte_data.h
```

then use the array data and size in the model_pte.h.

As **input image** we use the image from **CIFAR-10** dataset [1]. After preprocessing and normalization it is converted to bytes and located here boards/mimxrt700evk/eiq_examples/executorch_cifar10/cm33_core0/image_data.h. The preprocessing is performed as follows:

```
import torch
import torchvision
import numpy as np

batch_size = 1

transform = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

test_set = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=batch_size, shuffle=False, num_workers=0)

index = 0
num_images = 10
for data in test_loader:
    images, labels = data
    for image, label in zip(images, labels):
        arr = image.numpy().astype(np.float32)
        arr.tofile("img" + str(index) + "_" + str(int(label)) + ".bin")
        index = index + 1
    if index >= num_images:
        break
if index >= num_images:
    break
```

This generates the num_images count of images from Cifar10 dataset, as input tensors for the cifar10 model and store them in corresponding .bin files. Then we use the xxd command to get the C array data and size:

```
$ xxd -i img0_3.bin > image_data_base.h
```

and again copy the array data and size in the image_data.h

Note, the img0 is the image picturing a cat, what is a class number 3.

Build, Deploy and Run

1. When using ARMGCC toolchain, the example application can be built as below. After building the example application, download it to the target with JLink as shown in *Figure 3*, an output message displays on the connected terminal as *Figure 4*.

```
$ boards/mimxrt700evk/eiq_examples/executorch_cifar10/cm33_core0/armgcc$ ./build_flash_release.sh
```

```

J-Link>loadfile C:\rt700\executorch_cifarnet_cm33_core0.elf
'loadfile': Performing implicit reset & halt of MCU.
ResetTarget() start
-- JLINK ResetTarget --
Set CM33_SYSRESETREQ_EN
CPU0 CSW: 0x03000002
ResetTarget() end - Took 88.9ms
Downloading file [C:\rt700\executorch_cifarnet_cm33_core0.elf]...
J-Link: Flash download: Bank 0 @ 0x28000000: 1 range affected (315392 bytes)
J-Link: Flash download: Total: 20.220s (Prepare: 0.145s, Compare: 2.874s, Erase: 2.558s, Program: 13.212s, Verify: 1.392s, Restore: 0.036s)
J-Link: Flash download: Program speed: 22 KB/s
O.K.
J-Link>reset
Reset delay: 0 ms
Reset type NORMAL: Resets core & peripherals via SYSRESETREQ & VECTRESET bit.
ResetTarget() start
-- JLINK ResetTarget --
Set CM33_SYSRESETREQ_EN
CPU0 CSW: 0x03000002
ResetTarget() end - Took 71.5ms
J-Link>go
Memory map 'after startup completion point' is active

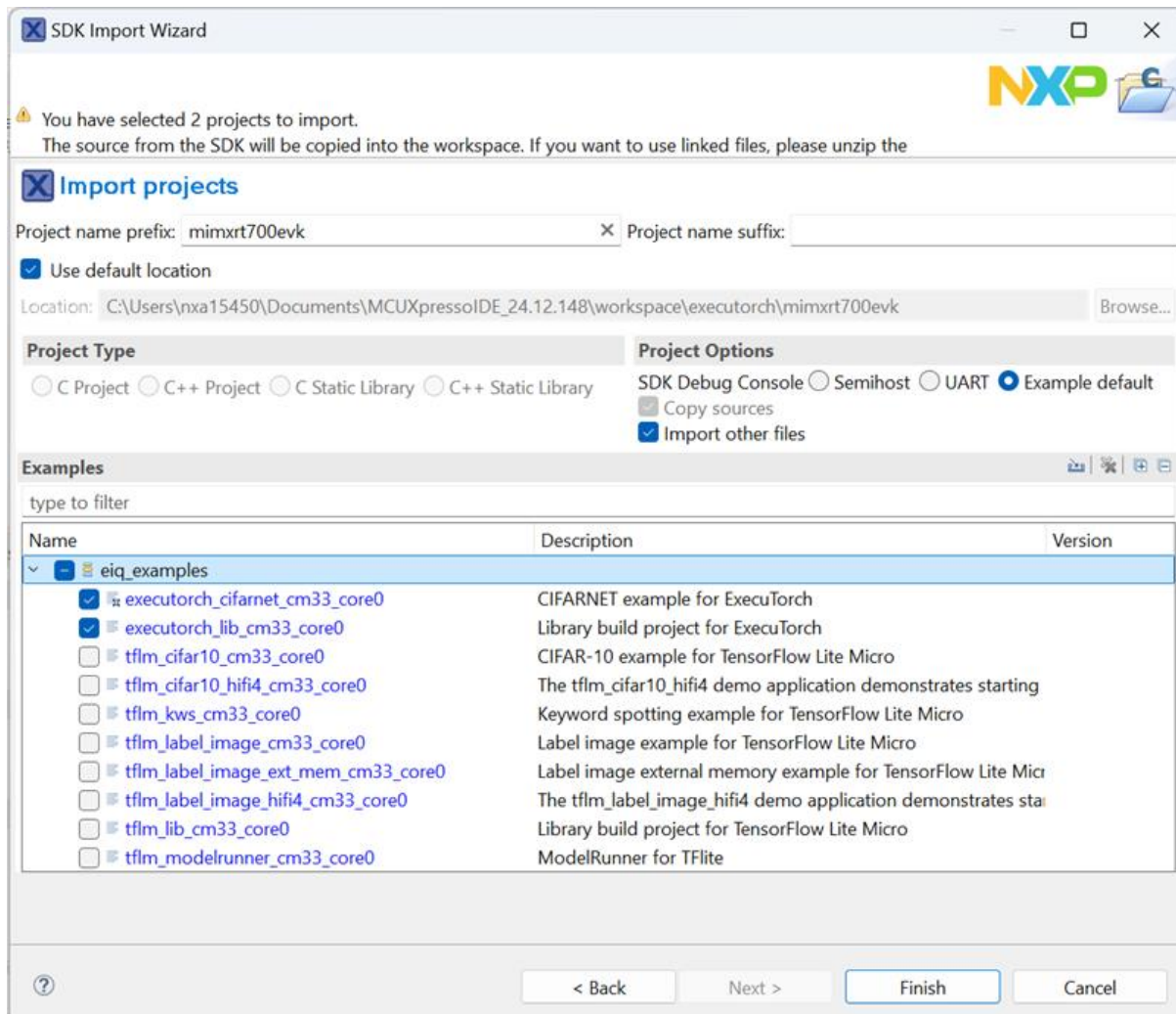
```

```

Model PTE file loaded. Size: 99376 bytes.
Model buffer loaded, has 1 methods
Running method forward
Setting up planned buffer 0, size 53760.
Method loaded.
Preparing inputs...
Input prepared.
Starting the model execution...
Model executed successfully.
-----
Inference time: 11950 us
-----
1 outputs:
Output[0][0]: 0
Output[0][1]: 0
Output[0][2]: 0
Output[0][3]: 0.996094
Output[0][4]: 0
Output[0][5]: 0
Output[0][6]: 0
Output[0][7]: 0
Output[0][8]: 0
Output[0][9]: 0
Program complete, exiting.

```

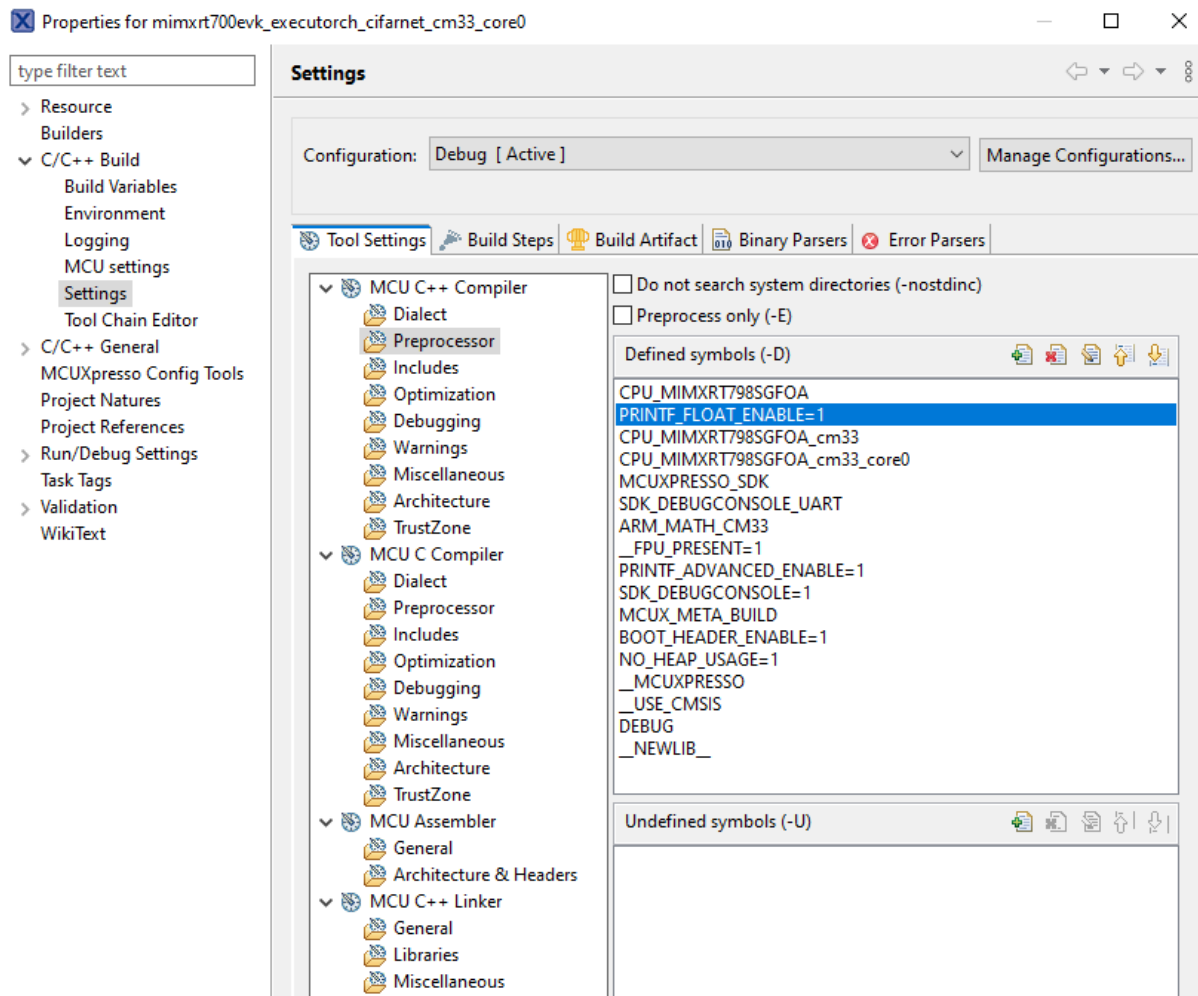
2. When using MCUXpresso IDE, the example applications can be imported through the SDK Import Wizard as shown in *Figure 5*.



After building the example application and downloading it to the target, the execution stops in the *main* function. When the execution resumes, an output message displays on the connected terminal. For example, *Figure 6* shows the output of the `executortch_cifar10` example application.

```
Model PTE file loaded. Size: 99376 bytes.
Model buffer loaded, has 1 methods
Running method forward
Setting up planned buffer 0, size 53760.
Method loaded.
Preparing inputs...
Input prepared.
Starting the model execution...
Model executed successfully.
-----
      Inference time: 14855 us
-----
1 outputs:
Output[0][0]: 0
Output[0][1]: 0
Output[0][2]: 0
Output[0][3]: 0.996094
Output[0][4]: 0
Output[0][5]: 0
Output[0][6]: 0
Output[0][7]: 0
Output[0][8]: 0
Output[0][9]: 0
Program complete, exiting.
```

In case of missing probabilities in the printed output, add `PRINTF_FLOAT_ENABLE=1` to the Pre-processor settings for C++ and C compiler:



How to build executorch_lib example If you want to build a new ExecuTorch Runtime Library, follow the commands as below and use the new library to replace the default Runtime library `middleware/eq/executorch/lib/cm33/armgcc/libexecutorch.a`.

1. When using ARMGCC toolchain, the example application can be built as below.

```
$ boards/mimxrt700evk/eq_examples/executorch_lib/cm33_core0/armgcc$ ./build_release.sh
$ boards/mimxrt700evk/eq_examples/executorch_lib/cm33_core0/armgcc$ cp release/libexecutorch_lib_cm33_core0.a.a ../../../../../../middleware/eq/executorch/lib/cm33/armgcc/libexecutorch.a
```

2. When using MCUXpresso IDE, the example applications can be imported through the SDK Import Wizard as shown in the above *Figure 5*.

After building the example application, copy the new library `mimxrt700evk_executorch_lib_cm33_core0\Debug\libmimxrt700evk_executorch_lib_cm33_core0.a` to replace the default Runtime library `mimxrt700evk_executorch_cifarnet_cm33_core0\eq\executorch\lib\cm33\armgcc\libexecutorch.a`.

[1] [Learning Multiple Layers of Features from Tiny Images](#), Alex Krizhevsky, 2009

3.3 Motor Control

3.3.1 FreeMASTER

Communication Driver User Guide

Introduction

What is FreeMASTER? FreeMASTER is a PC-based application developed by NXP for NXP customers. It is a versatile tool usable as a real-time monitor, visualization tool, and a graphical control panel of embedded applications based on the NXP processing units.

This document describes the embedded-side software driver which implements an interface between the application and the host PC. The interface covers the following communication:

- **Serial** UART communication either over plain RS232 interface or more typically over a USB-to-Serial either external or built in a debugger probe.
- **USB** direct connection to target microcontroller
- **CAN bus**
- **TCP/IP network** wired or WiFi
- **Segger J-Link RTT**
- **JTAG** debug port communication
- ...and all of the above also using a **Zephyr** generic drivers.

The driver also supports so-called “packet-driven BDM” interface which enables a protocol-based communication over a debugging port. The BDM stands for Background Debugging Module and its physical implementation is different on each platform. Some platforms leverage a semi-standard JTAG interface, other platforms provide a custom implementation called BDM. Regardless of the name, this debugging interface enables non-intrusive access to the memory space while the target CPU is running. For basic memory read and write operations, there is no communication driver required on the target when communicating with the host PC. Use this driver to get more advanced FreeMASTER protocol features over the BDM interface. The driver must be configured for the packet-driven BDM mode, in which the host PC uses the debugging interface to write serial command frames directly to the target memory buffer. The same method is then used to read response frames from that memory buffer.

Similar to “packet-driven BDM”, the FreeMASTER also supports a communication over [J-Link RTT](<https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/>) interface defined by SEGGER Microcontroller GmbH for ARM CortexM-based microcontrollers. This method also uses JTAG physical interface and enables high-speed real time communication to run over the same channel as used for application debugging.

Driver version 3 This document describes version 3 of the FreeMASTER Communication Driver. This version features the implementation of the new Serial Protocol, which significantly extends the features and security of its predecessor. The new protocol internal number is v4 and its specification is available in the documentation accompanying the driver code.

Driver V3 is deployed to modern 32-bit MCU platforms first, so the portfolio of supported platforms is smaller than for the previous V2 versions. It is recommended to keep using the V2 driver for legacy platforms, such as S08, S12, ColdFire, or Power Architecture. Reach out to [FreeMASTER community](#) or to the local NXP representative with requests for more information or to port the V3 driver to legacy MCU devices.

Thanks to a layered approach, the new driver simplifies the porting of the driver to new UART, CAN or networking communication interfaces significantly. Users are encouraged to port the driver to more NXP MCU platforms and contribute the code back to NXP for integration into future releases. Existing code and low-level driver layers may be used as an example when porting to new targets.

Note: Using the FreeMASTER tool and FreeMASTER Communication Driver is only allowed in systems based on NXP microcontroller or microprocessor unit. Use with non-NXP MCU platforms is **not permitted** by the license terms.

Target platforms The driver implementation uses the following abstraction mechanisms which simplify driver porting and supporting new communication modules:

- **General CPU Platform** (see source code in the `src/platforms` directory). The code in this layer is only specific to native data type sizes and CPU architectures (for example; alignment-aware memory copy routines). This driver version brings two generic implementations of 32-bit platforms supporting both little-endian and big-endian architectures. There are also implementations customized for the 56F800E family of digital signal controllers and S12Z MCUs. **Zephyr** is treated as a specific CPU platform as it brings unified user configuration (Kconfig) and generic hardware device drivers. With Zephyr, the transport layer and low-level communication layers described below are configured automatically using Kconfig and Device Tree technologies.
- **Transport Communication Layer** - The Serial, CAN, Networking, PD-BDM, and other methods of transport logic are implemented as a driver layer called `FMSTR_TRANSPORT` with a uniform API. A support of the Network transport also extends single-client modes of operation which are native for Serial, USB and CAN by a concept of multiple client sessions.
- **Low-level Communication Driver** - Each type of transport further defines a low-level API used to access the physical communication module. For example, the Serial transport defines a character-oriented API implemented by different serial communication modules like UART, LPUART, USART, and also USB-CDC. Similarly, the CAN transport defines a message-oriented API implemented by the FlexCAN or MCAN modules. Moreover, there are multiple different implementations for the same kind of communication peripherals. The difference between the implementation is in the way the low-level hardware registers are accessed. The `mcuxsdk` folder contains implementations which use MCUXpresso SDK drivers. These drivers should be used in applications based on the NXP MCUXpresso SDK. The “`ampsdk`” drivers target automotive-specific MCUs and their respective SDKs. The “`dreg`” implementations use a plain C-language access to hardware register addresses which makes it a universal and the most portable solution. In this case, users are encouraged to add more drivers for other communication modules or other respective SDKs and contribute the code back to NXP for integration.

The low-level drivers defined for the Networking transport enable datagram-oriented UDP and stream TCP communication. This implementation is demonstrated using the lwIP software stack but shall be portable to other TCP/IP stacks. It may sound surprisingly, but also the Segger J-Link RTT communication driver is linked to the Networking transport (RTT is stream oriented communication handled similarly to TCP).

Replacing existing drivers For all supported platforms, the driver described in this document replaces the V2 implementation and also older driver implementations that were available separately for individual platforms (PC Master SCI drivers).

Clocks, pins, and peripheral initialization The FreeMASTER communication driver is only responsible for runtime processing of the communication and must be integrated with an user application code to function properly. The user application code is responsible for general initialization of clock sources, pin multiplexers, and peripheral registers related to the communication speed. Such initialization should be done before calling the `FMSTR_Init` function.

It is recommended to develop the user application using one of the Software Development Kits (SDKs) available from third parties or directly from NXP, such as MCUXpresso SDK, MCUXpresso IDE, and related tools. This approach simplifies the general configuration process significantly.

MCUXpresso SDK The MCUXpresso SDK is a software package provided by NXP which contains the device initialization code, linker files, and software drivers with example applications for the NXP family of MCUs. The MCUXpresso Config Tools may be used to generate the clock-setup and pin-multiplexer setup code suitable for the selected processor.

The MCUXpresso SDK also contains this FreeMASTER communication driver as a “middleware” component which may be downloaded along with the example applications from <https://mcuxpresso.nxp.com/en/welcome>.

MCUXpresso SDK on GitHub The FreeMASTER communication driver is also released as one of the middleware components of the MCUXpresso SDK on the GitHub. This release enables direct integration of the FreeMASTER source code Git repository into a target applications including Zephyr applications.

Related links:

- [The official FreeMASTER middleware repository.](#)
- [Online version of this document](#)

FreeMASTER in Zephyr The FreeMASTER middleware repository can be used with MCUXpresso SDK as well as a Zephyr module. Zephyr-specific samples which include examples of Kconfig and Device Tree configurations for Serial, USB and Network communications are available in separate repository. West manifest in this sample repository fetches the full Zephyr package including the FreeMASTER middleware repository used as a Zephyr module.

Example applications

MCUX SDK Example applications There are several example applications available for each supported MCU platform.

- **fmstr_uart** demonstrates a plain serial transmission, typically connecting to a computer's physical or virtual COM port. The typical transmission speed is 115200 bps.
- **fmstr_can** demonstrates CAN bus communication. This requires a suitable CAN interface connected to the computer and interconnected with the target MCU using a properly terminated CAN bus. The typical transmission speed is 500 kbps. A FreeMASTER-over-CAN communication plug-in must be used.
- **fmstr_usb_cdc** uses an on-chip USB controller to implement a CDC communication class. It is connected directly to a computer's USB port and creates a virtual COM port device. The typical transmission speed is above 1 Mbps.
- **fmstr_net** demonstrates the Network communication over UDP or TCP protocol. Existing examples use lwIP stack to implement the communication, but in general, it shall be possible to use any other TCP/IP stack to achieve the same functionality.
- **fmstr_wifi** is the fmstr_net application modified to use a WiFi network interface instead of a wired Ethernet connection.
- **fmstr_rtt** demonstrates the communication over SEGGER J-Link RTT interface. Both fmstr_net and fmstr_rtt examples require the FreeMASTER TCP/UDP communication plug-in to be used on the PC host side.
- **fmstr_eonce** uses the real-time data unit on the JTAG EOnCE module of the 56F800E family to implement pseudo-serial communication over the JTAG port. The typical transmission speed is around 10 kbps. This communication requires FreeMASTER JTAG/EOnCE communication plug-in.

- **fmstr_pdbdm** uses JTAG or BDM debugging interface to access the target RAM directly while the CPU is running. Note that such approach can be used with any MCU application, even without any special driver code. The computer reads from and writes into the RAM directly without CPU intervention. The Packet-Driven BDM (PD-BDM) communication uses the same memory access to exchange command and response frames. With PD-BDM, the FreeMASTER tool is able to go beyond basic memory read/write operations and accesses also advanced features like Recorder, TSA, or Pipes. The typical transmission speed is around 10 kbps. A PD-BDM communication plug-in must be used in FreeMASTER and configured properly for the selected debugging interface. Note that this communication cannot be used while a debugging interface is used by a debugger session.
- **fmstr_any** is a special example application which demonstrates how the NXP MCUXpresso Config Tools can be used to configure pins, clocks, peripherals, interrupts, and even the FreeMASTER “middleware” driver features in a graphical and user friendly way. The user can switch between the Serial, CAN, and other ways of communication and generate the required initialization code automatically.

Zephyr sample applications Zephyr sample applications demonstrate Kconfig and Device Tree configuration which configure the FreeMASTER middleware module for a selected communication option (Serial, CAN, Network or RTT).

Refer to *readme.md* files in each sample directory for description of configuration options required to implement FreeMASTER connectivity.

Description

This section shows how to add the FreeMASTER Communication Driver into application and how to configure the connection to the FreeMASTER visualization tool.

Features The FreeMASTER driver implements the FreeMASTER protocol V4 and provides the following features which may be accessed using the FreeMASTER visualization tool:

- Read/write access to any memory location on the target.
- Optional password protection of the read, read/write, and read/write/flash access levels.
- Atomic bit manipulation on the target memory (bit-wise write access).
- Optimal size-aligned access to memory which is also suitable to access the peripheral register space.
- Oscilloscope access—real-time access to target variables. The sample rate may be limited by the communication speed.
- Recorder— access to the fast transient recorder running on the board as a part of the FreeMASTER driver. The sample rate is only limited by the MCU CPU speed. The length of the data recorded depends on the amount of available memory.
- Multiple instances of Oscilloscopes and Recorders without the limitation of maximum number of variables.
- Application commands—high-level message delivery from the PC to the application.
- TSA tables—describing the data types, variables, files, or hyperlinks exported by the target application. The TSA newly supports also non-memory mapped resources like external EEPROM or SD Card files.
- Pipes—enabling the buffered stream-oriented data exchange for a general-purpose terminal-like communication, diagnostic data streaming, or other data exchange.

The FreeMASTER driver features:

- Full FreeMASTER protocol V4 implementation with a new V4 style of CRC used.
- Layered approach supporting Serial, CAN, Network, PD-BDM, and other transports.
- Layered low-level Serial transport driver architecture enabling to select UART, LPUART, USART, and other physical implementations of serial interfaces, including USB-CDC.
- Layered low-level CAN transport driver architecture enabling to select FlexCAN, msCAN, MCAN, and other physical implementations of the CAN interface.
- Layered low-level Networking transport enabling to select TCP, UDP or J-Link RTT communication.
- TSA support to write-protect memory regions or individual variables and to deny the access to the unsafe memory.
- The pipe callback handlers are invoked whenever new data is available for reading from the pipe.
- Two Serial Single-Wire modes of operation are enabled. The “external” mode has the RX and TX shorted on-board. The “true” single-wire mode interconnects internally when the MCU or UART modules support it.

The following sections briefly describe all FreeMASTER features implemented by the driver. See the PC-based FreeMASTER User Manual for more details on how to use the features to monitor, tune, or control an embedded application.

Board Detection The FreeMASTER protocol V4 defines the standard set of configuration values which the host PC tool reads to identify the target and to access other target resources properly. The configuration includes the following parameters:

- Version of the driver and the version of the protocol implemented.
- MTU as the Maximum size of the Transmission Unit (for example; communication buffer size).
- Application name, description, and version strings.
- Application build date and time as a string.
- Target processor byte ordering (little/big endian).
- Protection level that requires password authentication.
- Number of the Recorder and Oscilloscope instances.
- RAM Base Address for optimized memory access commands.

Memory Read This basic feature enables the host PC to read any data memory location by specifying the address and size of the required memory area. The device response frame must be shorter than the MTU to fit into the outgoing communication buffer. To read a device memory of any size, the host uses the information retrieved during the Board Detection and splits the large-block request to multiple partial requests.

The driver uses size-aligned operations to read the target memory (for example; uses proper read-word instruction when an address is aligned to 4 bytes).

Memory Write Similarly to the Memory Read operation, the Memory Write feature enables to write to any RAM memory location on the target device. A single write command frame must be shorter than the MTU to fit into the target communication buffer. Larger requests must be split into smaller ones.

The driver uses size-aligned operations to write to the target memory (for example; uses proper write-word instruction when an address is aligned to 4 bytes).

Masked Memory Write To implement the write access to a single bit or a group of bits of target variables, the Masked Memory Write feature is available in the FreeMASTER protocol and it is supported by the driver using the Read-Modify-Write approach.

Be careful when writing to bit fields of volatile variables that are also modified in an application interrupt. The interrupt may be serviced in the middle of a read-modify-write operation and it may cause data corruption.

Oscilloscope The protocol and driver enables any number of variables to be read at once with a single request from the host. This feature is called Oscilloscope and the FreeMASTER tool uses it to display a real-time graph of variable values.

The driver can be configured to support any number of Oscilloscope instances and enable simultaneously running graphs to be displayed on the host computer screen.

Recorder The protocol enables the host to select target variables whose values are then periodically recorded into a dedicated on-board memory buffer. After such data sampling stops (either on a host request or by evaluating a threshold-crossing condition), the data buffer is downloaded to the host and displayed as a graph. The data sampling rate is not limited by the speed of the communication line, so it enables displaying the variable transitions in a very high resolution.

The driver can be configured to support multiple Recorder instances and enable multiple recorder graphs to be displayed on the host screen. Having multiple recorders also enables setting the recording point differently for each instance. For example; one instance may be recording data in a general timer interrupt while another instance may record at a specific control algorithm time in the PWM interrupt.

TSA With the TSA feature, data types and variables can be described directly in the application source code. Such information is later provided to the FreeMASTER tool which may use it instead of reading symbol data from the application ELF executable file.

The information is encoded as so-called TSA tables which become direct part of the application code. The TSA tables contain descriptors of variables that shall be visible to the host tool. The descriptors can describe the memory areas by specifying the address and size of the memory block or more conveniently using the C variable names directly. Different set of TSA descriptors can be used to encode information about the structure types, unions, enumerations, or arrays.

The driver also supports special types of TSA table entries to describe user resources like external EEPROM and SD Card files, memory-mapped files, virtual directories, web URL hyperlinks, and constant enumerations.

TSA Safety When the TSA is enabled in the application, the TSA Safety can be enabled and validate the memory accesses directly by the embedded-side driver. When the TSA Safety is turned on, any memory request received from the host is validated and accepted only if it belongs to a TSA-described object. The TSA entries can be declared as Read-Write or Read-Only so that the driver can actively deny the write access to the Read-Only objects.

Application commands The Application Commands are high-level messages that can be delivered from the PC Host to the embedded application for further processing. The embedded application can either poll the status, or be called back when a new Application Command arrives to be processed. After the embedded application acknowledges that the command is handled, the host receives the Result Code and reads the other return data from memory. Both the Application Commands and the Result Codes are specific to a given application and it is user's responsibility to define them. The FreeMASTER protocol and the FreeMASTER driver only implement the delivery channel and a set of API calls to enable the Application Command processing in general.

Pipes The Pipes enable buffered and stream-oriented data exchange between the PC Host and the target application. Any pipe can be written to and read from at both ends (either on the PC or the MCU). The data transmission is acknowledged using the special FreeMASTER protocol commands. It is guaranteed that the data bytes are delivered from the writer to the reader in a proper order and without losses.

Serial single-wire operation The MCU Serial Communication Driver natively supports normal dual-wire operation. Because the protocol is half-duplex only, the driver can also operate in two single-wire modes:

- “External” single-wire operation where the Receiver and Transmitter pins are shorted on the board. This mode is supported by default in the MCU driver because the Receiver and Transmitter units are enabled or disabled whenever needed. It is also easy to extend this operation for the RS485 communication.
- “True” single-wire mode which uses only a single pin and the direction switching is made by the UART module. This mode of operation must be enabled by defining the FMSTR_SERIAL_SINGLEWIRE configuration option.

Multi-session support With networking interface it is possible for multiple clients to access the target MCU simultaneously. Reading and writing of target memory is processed atomically so there is no risk of data corruption. The state-full resources such as Recorders or Oscilloscopes are locked to a client session upon first use and access is denied to other clients until lock is released..

Zephyr-specific

Dedicated communication task FreeMASTER communication may run isolated in a dedicated task. The task automates the FMSTR_Init and FMSTR_Poll calls together with periodic activities enabling the FreeMASTER UI to fetch information about tasks and CPU utilization. The task can be started automatically or manually, and it must be assigned a priority to be able to react on interrupts and other communication events. Refer to Zephyr FreeMASTER sample applications which all use this communication task.

Zephyr shell and logging over FreeMASTER pipe FreeMASTER implements a shell backend which may use FreeMASTER pipe as a I/O terminal and logging output. Refer to Zephyr FreeMASTER sample applications which all use this feature.

Automatic TSA tables TSA tables can be declared as “automatic” in Zephyr which make them automatically registered in the table list. This may be very useful when there are many TSA tables or when the tables are defined in different (often unrelated) libraries linked together. In this case user does not need to build a list of all tables manually.

Driver files The driver source files can be found in a top-level src folder, further divided into the sub-folders:

- **src/platforms** platform-specific folder—one folder exists for each supported processor platform (for example; 32-bit Little Endian platform). Each such folder contains a platform header file with data types and a code which implements the potentially platform-specific operations, such as aligned memory access.
- **src/common** folder—contains the common driver source files shared by the driver for all supported platforms. All the .c files must be added to the project, compiled, and linked together with the application.

- *freemaster.h* - master driver header file, which declares the common data types, macros, and prototypes of the FreeMASTER driver API functions.
- *freemaster_cfg.h.example* - this file can serve as an example of the FreeMASTER driver configuration file. Save this file into a project source code folder and rename it to *freemaster_cfg.h*. The FreeMASTER driver code includes this file to get the project-specific configuration options and to optimize the compilation of the driver.
- *freemaster_defcfg.h* - defines the default values for each FreeMASTER configuration option if the option is not set in the *freemaster_cfg.h* file.
- *freemaster_protocol.h* - defines the FreeMASTER protocol constants used internally by the driver.
- *freemaster_protocol.c* - implements the FreeMASTER protocol decoder and handles the basic Get Configuration Value, Memory Read, and Memory Write commands.
- *freemaster_rec.c* - handles the Recorder-specific commands and implements the Recorder sampling and triggering routines. When the Recorder is disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
- *freemaster_scope.c* - handles the Oscilloscope-specific commands. If the Oscilloscope is disabled by the FreeMASTER driver configuration file, this file compiles as void.
- *freemaster_pipes.c* - implements the Pipes functionality when the Pipes feature is enabled.
- *freemaster_appcmd.c* - handles the communication commands used to deliver and execute the Application Commands within the context of the embedded application. When the Application Commands are disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
- *freemaster_tsa.c* - handles the commands specific to the TSA feature. This feature enables the FreeMASTER host tool to obtain the TSA memory descriptors declared in the embedded application. If the TSA is disabled by the FreeMASTER driver configuration file, this file compiles as void.
- *freemaster_tsa.h* - contains the declaration of the macros used to define the TSA memory descriptors. This file is indirectly included into the user application code (via *freemaster.h*).
- *freemaster_sha.c* - implements the SHA-1 hash code used in the password authentication algorithm.
- *freemaster_private.h* - contains the declarations of functions and data types used internally in the driver. It also contains the C pre-processor statements to perform the compile-time verification of the user configuration provided in the *freemaster_cfg.h* file.
- *freemaster_serial.c* - implements the serial protocol logic including the CRC, FIFO queuing, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a character-oriented API exported by the specific low-level driver.
- *freemaster_serial.h* - defines the low-level character-oriented Serial API.
- *freemaster_can.c* - implements the CAN protocol logic including the CAN message preparation, signalling using the first data byte in the CAN frame, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a message-oriented API exported by the specific low-level driver.
- *freemaster_can.h* - defines the low-level message-oriented CAN API.
- *freemaster_net.c* - implements the Network protocol transport logic including multiple session management code.

- *freemaster_net.h* - definitions related to the Network transport.
- *freemaster_pdbdm.c* - implements the packet-driven BDM communication buffer and other communication-related operations.
- *freemaster_utils.c* - aligned memory copy routines, circular buffer management and other utility functions
- *freemaster_utils.h* - definitions related to utility code.
- **src/drivers/[sdk]/serial** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
 - *freemaster_serial_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the UART, LPUART, USART, and other kinds of Serial communication modules.
- **src/drivers/[sdk]/can** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
 - *freemaster_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the FlexCAN, msCAN, MCAN, and other kinds of CAN communication modules.
- **src/drivers/[sdk]/network** - contains low-level code adapting the FreeMASTER Network transport to an underlying TCP/IP or RTT stack.
 - *freemaster_net_lwip_tcp.c* and *_udp.c* - default networking implementation of TCP and UDP transports using lwIP stack.
 - *freemaster_net_segger_rtt.c* - implementation of network transport using Segger J-Link RTT interface

Driver configuration The driver is configured using a single header file (*freemaster_cfg.h*). Create this file and save it together with other project source files before compiling the driver code. All FreeMASTER driver source files include the *freemaster_cfg.h* file and use the macros defined here for the conditional and parameterized compilation. The C compiler must locate the configuration file when compiling the driver files. Typically, it can be achieved by putting this file into a folder where the other project-specific included files are stored.

As a starting point to create the configuration file, get the *freemaster_cfg.h.example* file, rename it to *freemaster_cfg.h*, and save it into the project area.

Note: It is NOT recommended to leave the *freemaster_cfg.h* file in the FreeMASTER driver source code folder. The configuration file must be placed at a project-specific location, so that it does not affect the other applications that use the same driver.

Configurable items This section describes the configuration options which can be defined in *freemaster_cfg.h*.

Interrupt modes

```
#define FMSTR_LONG_INTR   [0|1]
#define FMSTR_SHORT_INTR  [0|1]
#define FMSTR_POLL_DRIVEN [0|1]
```

Value Type boolean (0 or 1)

Description Exactly one of the three macros must be defined to non-zero. The others must be defined to zero or left undefined. The non-zero-defined constant selects the interrupt mode of the driver. See [Driver interrupt modes](#).

- FMSTR_LONG_INTR — long interrupt mode
- FMSTR_SHORT_INTR — short interrupt mode
- FMSTR_POLL_DRIVEN — poll-driven mode

Note: Some options may not be supported by all communication interfaces. For example, the FMSTR_SHORT_INTR option is not supported by the USB_CDC interface.

Protocol transport

```
#define FMSTR_TRANSPORT [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER source code. Specify one of existing instances to make use of the protocol transport.

Description Use one of the pre-defined constants, as implemented by the FreeMASTER code. The current driver supports the following transports:

- FMSTR_SERIAL - serial communication protocol
- FMSTR_CAN - using CAN communication
- FMSTR_PDBDM - using packet-driven BDM communication
- FMSTR_NET - network communication using TCP or UDP protocol

Serial transport This section describes configuration parameters used when serial transport is used:

```
#define FMSTR_TRANSPORT FMSTR_SERIAL
```

FMSTR_SERIAL_DRV Select what low-level driver interface will be used when implementing the Serial communication.

```
#define FMSTR_SERIAL_DRV [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing serial driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/serial* implementation):

- FMSTR_SERIAL_MCUX_UART - UART driver
- FMSTR_SERIAL_MCUX_LPUART - LPUART driver
- FMSTR_SERIAL_MCUX_USART - USART driver
- FMSTR_SERIAL_MCUX_MINIUSART - miniUSART driver
- FMSTR_SERIAL_MCUX_QSCI - QSCI driver
- FMSTR_SERIAL_MCUX_USB - USB/CDC class driver (also see code in the */support/mcuxsdk_usb* folder)

- **FMSTR_SERIAL_56F800E_EONCE** - DSC JTAG EOnCE driver

Other SDKs or BSPs may define custom low-level driver interface structure which may be used as FMSTR_SERIAL_DRV. For example:

- **FMSTR_SERIAL_DREG_UART** - demonstrates the low-level interface implemented without the MCUXpresso SDK and using direct access to peripheral registers.

FMSTR_SERIAL_BASE

```
#define FMSTR_SERIAL_BASE [address|symbol]
```

Value Type Optional address value (numeric or symbolic)

Description Specify the base address of the UART, LPUART, USART, or other serial peripheral module to be used for the communication. This value is not defined by default. User application should call FMSTR_SetSerialBaseAddress() to select the peripheral module.

FMSTR_COMM_BUFFER_SIZE

```
#define FMSTR_COMM_BUFFER_SIZE [number]
```

Value Type 0 or a value in range 32...255

Description Specify the size of the communication buffer to be allocated by the driver. Default value, which suits all driver features, is used when this option is defined as 0.

FMSTR_COMM_RQUEUE_SIZE

```
#define FMSTR_COMM_RQUEUE_SIZE [number]
```

Value Type Value in range 0...255

Description Specify the size of the FIFO receiver queue used to quickly receive and store characters in the FMSTR_SHORT_INTR interrupt mode. The default value is 32 B.

FMSTR_SERIAL_SINGLEWIRE

```
#define FMSTR_SERIAL_SINGLEWIRE [0|1]
```

Value Type Boolean 0 or 1.

Description Set to non-zero to enable the “True” single-wire mode which uses a single MCU pin to communicate. The low-level driver enables the pin direction switching when the MCU peripheral supports it.

CAN Bus transport This section describes configuration parameters used when CAN transport is used:

```
#define FMSTR_TRANSPORT FMSTR_CAN
```

FMSTR_CAN_DRV Select what low-level driver interface will be used when implementing the CAN communication.

```
#define FMSTR_CAN_DRV [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing CAN driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/can implementation*):

- **FMSTR_CAN_MCUX_FLEXCAN** - FlexCAN driver
- **FMSTR_CAN_MCUX_MCAN** - MCAN driver
- **FMSTR_CAN_MCUX_MSCAN** - msCAN driver
- **FMSTR_CAN_MCUX_DSCFLEXCAN** - DSC FlexCAN driver
- **FMSTR_CAN_MCUX_DSCMSCAN** - DSC msCAN driver

Other SDKs or BSPs may define the custom low-level driver interface structure which may be used as FMSTR_CAN_DRV.

FMSTR_CAN_BASE

```
#define FMSTR_CAN_BASE [address|symbol]
```

Value Type Optional address value (numeric or symbolic)

Description Specify the base address of the FlexCAN, msCAN, or other CAN peripheral module to be used for the communication. This value is not defined by default. User application should call FMSTR_SetCanBaseAddress() to select the peripheral module.

FMSTR_CAN_CMDID

```
#define FMSTR_CAN_CMDID [number]
```

Value Type CAN identifier (11-bit or 29-bit number)

Description CAN message identifier used for FreeMASTER commands (direction from PC Host tool to target application). When declaring 29-bit identifier, combine the numeric value with FMSTR_CAN_EXTID bit. Default value is 0x7AA.

FMSTR_CAN_RSPID

```
#define FMSTR_CAN_RSPID [number]
```

Value Type CAN identifier (11-bit or 29-bit number)

Description CAN message identifier used for responding messages (direction from target application to PC Host tool). When declaring 29-bit identifier, combine the numeric value with FMSTR_CAN_EXTID bit. Note that both *CMDID* and *RSPID* values may be the same. Default value is 0x7AA.

FMSTR_FLEXCAN_TXMB

```
#define FMSTR_FLEXCAN_TXMB [number]
```

Value Type Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame transmission. Default value is 0.

FMSTR_FLEXCAN_RXMB

```
#define FMSTR_FLEXCAN_RXMB [number]
```

Value Type Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame reception. Note that the FreeMASTER driver may also operate with a common message buffer used by both TX and RX directions. Default value is 1.

Network transport This section describes configuration parameters used when Network transport is used:

```
#define FMSTR_TRANSPORT FMSTR_NET
```

FMSTR_NET_DRV Select network interface implementation.

```
#define FMSTR_NET_DRV [identifier]
```

Value Type Identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing NET driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/network implementation*):

- **FMSTR_NET_LWIP_TCP** - TCP communication using lwIP stack
- **FMSTR_NET_LWIP_UDP** - UDP communication using lwIP stack
- **FMSTR_NET_SEGGER_RTT** - Communication using SEGGER J-Link RTT interface

Other SDKs or BSPs may define the custom networking interface which may be used as FMSTR_CAN_DRV.

Add another row below:

FMSTR_NET_PORT

```
#define FMSTR_NET_PORT [number]
```

Value Type TCP or UDP port number (short integer)

Description Specifies the server port number used by TCP or UDP protocols.

FMSTR_NET_BLOCKING_TIMEOUT

```
#define FMSTR_NET_BLOCKING_TIMEOUT [number]
```

Value Type Timeout as number of milliseconds

Description This value specifies a timeout in milliseconds for which the network socket operations may block the execution inside *FMSTR_Poll*. This may be set high (e.g. 250) when a dedicated RTOS task is used to handle FreeMASTER protocol polling. Set to a lower value when the polling task is also responsible for other operations. Set to 0 to attempt to use non-blocking socket operations.

FMSTR_NET_AUTODISCOVERY

```
#define FMSTR_NET_AUTODISCOVERY [0|1]
```

Value Type Boolean 0 or 1.

Description This option enables the FreeMASTER driver to use a separate UDP socket to broadcast auto-discovery messages to network. This helps the FreeMASTER tool to discover the target device address, port and protocol options.

Debugging options

FMSTR_DISABLE

```
#define FMSTR_DISABLE [0|1]
```

Value Type boolean (0 or 1)

Description Define as non-zero to disable all FreeMASTER features, exclude the driver code from build, and compile all its API functions empty. This may be useful to remove FreeMASTER without modifying any application source code. Default value is 0 (false).

FMSTR_DEBUG_TX

```
#define FMSTR_DEBUG_TX [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to enable the driver to periodically transmit test frames out on the selected communication interface (SCI or CAN). With the debug transmission enabled, it is simpler to detect problems in the baudrate or other communication configuration settings.

The test frames are transmitted until the first valid command frame is received from the PC Host tool. The test frame is a valid error status frame, as defined by the protocol format. On the serial line, the test frame consists of three printable characters (+©W) which are easy to capture using the serial terminal tools.

This feature requires the FMSTR_Poll() function to be called periodically. Default value is 0 (false).

FMSTR_APPLICATION_STR

```
#define FMSTR_APPLICATION_STR
```

Value Type String.

Description Name of the application visible in FreeMASTER host application.

Memory access

FMSTR_USE_READMEM

```
#define FMSTR_USE_READMEM [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Memory Read command and enable FreeMASTER to have read access to memory and variables. The access can be further restricted by using a TSA feature.

Default value is 1 (true).

FMSTR_USE_WRITEMEM

```
#define FMSTR_USE_WRITEMEM [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Memory Write command.

The default value is 1 (true).

Oscilloscope options

FMSTR_USE_SCOPE

```
#define FMSTR_USE_SCOPE [number]
```

Value Type Integer number.

Description Number of Oscilloscope instances to be supported. Set to 0 to disable the Oscilloscope feature.
Default value is 0.

FMSTR_MAX_SCOPE_VARS

```
#define FMSTR_MAX_SCOPE_VARS [number]
```

Value Type Integer number larger than 2.

Description Number of variables to be supported by each Oscilloscope instance.
Default value is 8.

Recorder options

FMSTR_USE_RECORDER

```
#define FMSTR_USE_RECORDER [number]
```

Value Type Integer number.

Description Number of Recorder instances to be supported. Set to 0 to disable the Recorder feature.
Default value is 0.

FMSTR_REC_BUFF_SIZE

```
#define FMSTR_REC_BUFF_SIZE [number]
```

Value Type Integer number larger than 2.

Description Defines the size of the memory buffer used by the Recorder instance #0.
Default: not defined, user shall call 'FMSTR_RecorderCreate()' API function to specify this parameter in run time.

FMSTR_REC_TIMEBASE

```
#define FMSTR_REC_TIMEBASE [time specification]
```

Value Type Number (nanoseconds time).

Description Defines the base sampling rate in nanoseconds (sampling speed) Recorder instance #0.

Use one of the following macros:

- FMSTR_REC_BASE_SECONDS(x)
- FMSTR_REC_BASE_MILLISEC(x)
- FMSTR_REC_BASE_MICROSEC(x)
- FMSTR_REC_BASE_NANOSEC(x)

Default: not defined, user shall call 'FMSTR_RecorderCreate()' API function to specify this parameter in run time.

FMSTR_REC_FLOAT_TRIG

```
#define FMSTR_REC_FLOAT_TRIG [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the floating-point triggering. Be aware that floating-point triggering may grow the code size by linking the floating-point standard library.

Default value is 0 (false).

Application Commands options

FMSTR_USE_APPCMD

```
#define FMSTR_USE_APPCMD [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Application Commands feature. Default value is 0 (false).

FMSTR_APPCMD_BUFF_SIZE

```
#define FMSTR_APPCMD_BUFF_SIZE [size]
```

Value Type Numeric buffer size in range 1..255

Description The size of the Application Command data buffer allocated by the driver. The buffer stores the (optional) parameters of the Application Command which waits to be processed.

FMSTR_MAX_APPCMD_CALLS

```
#define FMSTR_MAX_APPCMD_CALLS [number]
```

Value Type Number in range 0..255

Description The number of different Application Commands that can be assigned a callback handler function using `FMSTR_RegisterAppCmdCall()`. Default value is 0.

TSA options

FMSTR_USE_TSA

```
#define FMSTR_USE_TSA [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the FreeMASTER TSA feature to be used. With this option enabled, the TSA tables defined in the applications are made available to the FreeMASTER host tool. Default value is 0 (false).

FMSTR_USE_TSA_SAFETY

```
#define FMSTR_USE_TSA_SAFETY [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the memory access validation in the FreeMASTER driver. With this option, the host tool is not able to access the memory which is not described by at least one TSA descriptor. Also a write access is denied for objects defined as read-only in TSA tables. Default value is 0 (false).

FMSTR_USE_TSA_INROM

```
#define FMSTR_USE_TSA_INROM [0|1]
```

Value Type Boolean 0 or 1.

Description Declare all TSA descriptors as *const*, which enables the linker to put the data into the flash memory. The actual result depends on linker settings or the linker commands used in the project. Default value is 0 (false).

FMSTR_USE_TSA_DYNAMIC

```
#define FMSTR_USE_TSA_DYNAMIC [0|1]
```

Value Type Boolean 0 or 1.

Description Enable runtime-defined TSA entries to be added to the TSA table by the `FMSTR_SetUpTsaBuff()` and `FMSTR_TsaAddVar()` functions. Default value is 0 (false).

Pipes options

FMSTR_USE_PIPES

```
#define FMSTR_USE_PIPES [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the FreeMASTER Pipes feature to be used.
Default value is 0 (false).

FMSTR_MAX_PIPES_COUNT

```
#define FMSTR_MAX_PIPES_COUNT [number]
```

Value Type Number in range 1..63.

Description The number of simultaneous pipe connections to support.
The default value is 1.

Driver interrupt modes To implement the communication, the FreeMASTER driver handles the Serial or CAN module's receive and transmit requests. Use the *freemaster_cfg.h* configuration file to select whether the driver processes the communication automatically in the interrupt service routine handler or if it only polls the status of the module (typically during the application idle time).

This section describes each of the interrupt mode in more details.

Completely Interrupt-Driven operation Activated using:

```
#define FMSTR_LONG_INTR 1
```

In this mode, both the communication and the FreeMASTER protocol decoding is done in the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, or other interrupt service routine. Because the protocol execution may be a lengthy task (especially with the TSA-Safety enabled) it is recommended to use this mode only if the interrupt prioritization scheme is possible in the application and the FreeMASTER interrupt is assigned to a lower (the lowest) priority.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR_SerialIsr* or *FMSTR_CanIsr* functions from that handler.

Mixed Interrupt and Polling Modes Activated using:

```
#define FMSTR_SHORT_INTR 1
```

In this mode, the communication processing time is split between the interrupt routine and the main application loop or task. The raw communication is handled by the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, or other interrupt service routine, while the protocol decoding and execution is handled by the *FMSTR_Poll* routine. Call *FMSTR_Poll* during the idle time in the application main loop.

The interrupt processing in this mode is relatively fast and deterministic. Upon a serial-receive event, the received character is only placed into a FIFO-like queue and it is not further processed. Upon a CAN receive event, the received frame is stored into a receive buffer. When transmitting, the characters are fetched from the prepared transmit buffer.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR_SerialIsr* or *FMSTR_CanIsr* functions from that handler.

When the serial interface is used as the serial communication interface, ensure that the *FMSTR_Poll* function is called at least once per *N* character time periods. *N* is the length of the FreeMASTER FIFO queue (*FMSTR_COMM_RQUEUE_SIZE*) and the character time is the time needed to transmit or receive a single byte over the SCI line.

Completely Poll-driven

```
#define FMSTR_POLL_DRIVEN 1
```

In this mode, both the communication and the FreeMASTER protocol decoding are done in the *FMSTR_Poll* routine. No interrupts are needed and the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, and similar handlers compile to an empty code.

When using this mode, ensure that the *FMSTR_Poll* function is called by the application at least once per the serial “character time” which is the time needed to transmit or receive a single character.

In the latter two modes (*FMSTR_SHORT_INTR* and *FMSTR_POLL_DRIVEN*), the protocol handling takes place in the *FMSTR_Poll* routine. An application interrupt can occur in the middle of the Read Memory or Write Memory commands’ execution and corrupt the variable being accessed by the FreeMASTER driver. In these two modes, some issues or glitches may occur when using FreeMASTER to visualize or monitor volatile variables modified in interrupt servicing code.

The same issue may appear even in the full interrupt mode (*FMSTR_LONG_INTR*), if volatile variables are modified in the interrupt code with a priority higher than the priority of the communication interrupt.

Data types Simple portability was one of the main requirements when writing the FreeMASTER driver. This is why the driver code uses the privately-declared data types and the vast majority of the platform-dependent code is separated in the platform-dependent source files. The data types used in the driver API are all defined in the platform-specific header file.

To prevent name conflicts with the symbols used in the application, all data types, macros, and functions have the *FMSTR_* prefix. The only global variables used in the driver are the transport and low-level API structures exported from the driver-implementation layer to upper layers. Other than that, all private variables are declared as static and named using the *fmstr_* prefix.

Communication interface initialization The FreeMASTER driver does not perform neither the initialization nor the configuration of the peripheral module that it uses to communicate. It is the application startup code responsibility to configure the communication module before the FreeMASTER driver is initialized by the *FMSTR_Init* call.

When the Serial communication module is used as the FreeMASTER communication interface, configure the UART receive and transmit pins, the serial communication baud rate, parity (no-parity), the character length (eight bits), and the number of stop bits (one) before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see *Driver interrupt modes*), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected serial peripheral module. Call the *FMSTR_SerialIsr* function from the application handler.

When a CAN module is used as the FreeMASTER communication interface, configure the CAN receive and transmit pins and the CAN module bit rate before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see [Driver interrupt modes](#)), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected CAN peripheral module. Call the FMSTR_CanIsr function from the application handler.

Note: It is not necessary to enable or unmask the serial nor the CAN interrupts before initializing the FreeMASTER driver. The driver enables or disables the interrupts and communication lines, as required during runtime.

FreeMASTER Recorder calls When using the FreeMASTER Recorder in the application (FMSTR_USE_RECORDER > 0), call the FMSTR_RecorderCreate function early after FMSTR_Init to set up each recorder instance to be used in the application. Then call the FMSTR_Recorder function periodically in the code where the data recording should occur. A typical place to call the Recorder routine is at the timer or PWM interrupts, but it can be anywhere else. The example applications provided together with the driver code call the FMSTR_Recorder in the main application loop.

In applications where FMSTR_Recorder is called periodically with a constant period, specify the period in the Recorder configuration structure before calling FMSTR_RecorderCreate. This setting enables the PC Host FreeMASTER tool to display the X-axis of the Recorder graph properly scaled for the time domain.

Driver usage Start using or evaluating FreeMASTER by opening some of the example applications available in the driver setup package.

Follow these steps to enable the basic FreeMASTER connectivity in the application:

- Make sure that all *.c files of the FreeMASTER driver from the *src/common/platforms/[your_platform]* folder are a part of the project. See [Driver files](#) for more details.
- Configure the FreeMASTER driver by creating or editing the *freemaster_cfg.h* file and by saving it into the application project directory. See [Driver configuration](#) for more details.
- Include the *freemaster.h* file into any application source file that makes the FreeMASTER API calls.
- Initialize the Serial or CAN modules. Set the baud rate, parity, and other parameters of the communication. Do not enable the communication interrupts in the interrupt mask registers.
- For the FMSTR_LONG_INTR and FMSTR_SHORT_INTR modes, install the application-specific interrupt routine and call the FMSTR_SerialIsr or FMSTR_CanIsr functions from this handler.
- Call the FMSTR_Init function early on in the application initialization code.
- Call the FMSTR_RecorderCreate functions for each Recorder instance to enable the Recorder feature.
- In the main application loop, call the FMSTR_Poll API function periodically when the application is idle.
- For the FMSTR_SHORT_INTR and FMSTR_LONG_INTR modes, enable the interrupts globally so that the interrupts can be handled by the CPU.

Communication troubleshooting The most common problem that causes communication issues is a wrong baud rate setting or a wrong pin multiplexer setting of the target MCU. When

a communication between the PC Host running FreeMASTER and the target MCU cannot be established, try enabling the FMSTR_DEBUG_TX option in the *freemaster_cfg.h* file and call the FMSTR_Poll function periodically in the main application task loop.

With this feature enabled, the FreeMASTER driver periodically transmits a test frame through the Serial or CAN lines. Use a logic analyzer or an oscilloscope to monitor the signals at the communication pins of the CPU device to examine whether the bit rate and signal polarity are configured properly.

Driver API

This section describes the driver Application Programmers' Interface (API) needed to initialize and use the FreeMASTER serial communication driver.

Control API There are three key functions to initialize and use the driver.

FMSTR_Init

Prototype

```
FMSTR_BOOL FMSTR_Init(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_protocol.c*

Description This function initializes the internal variables of the FreeMASTER driver and enables the communication interface. This function does not change the configuration of the selected communication module. The hardware module must be initialized before the *FMSTR_Init* function is called.

A call to this function must occur before calling any other FreeMASTER driver API functions.

FMSTR_Poll

Prototype

```
void FMSTR_Poll(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_protocol.c*

Description In the poll-driven or short interrupt modes, this function handles the protocol decoding and execution (see *Driver interrupt modes*). In the poll-driven mode, this function also handles the communication interface with the PC. Typically, the *FMSTR_Poll* function is called during the “idle” time in the main application task loop.

To prevent the receive data overflow (loss) on a serial interface, make sure that the FMSTR_Poll function is called at least once per the time calculated as:

$$N * T_{char}$$

where:

- *N* is equal to the length of the receive FIFO queue (configured by the `FMSTR_COMM_QUEUE_SIZE` macro). *N* is 1 for the poll-driven mode.
- *Tchar* is the character time, which is the time needed to transmit or receive a single byte over the SCI line.

Note: In the long interrupt mode, this function typically compiles as an empty function and can still be called. It is worthwhile to call this function regardless of the interrupt mode used in the application. This approach enables a convenient switching between the different interrupt modes only by changing the configuration macros in the *freemaster_cfg.h* file.

FMSTR_SerialIsr / FMSTR_CanIsr

Prototype

```
void FMSTR_SerialIsr(void);  
void FMSTR_CanIsr(void);
```

- Declaration: *freemaster.h*
- Implementation: *hw-specific low-level driver C file*

Description This function contains the interrupt-processing code of the FreeMASTER driver. In long or short interrupt modes (see [Driver interrupt modes](#)), this function must be called from the application interrupt service routine registered for the communication interrupt vector. On platforms where the communication module uses multiple interrupt vectors, the application should register a handler for all vectors and call this function at each interrupt.

Note: In a poll-driven mode, this function is compiled as an empty function and does not have to be used.

Recorder API

FMSTR_RecorderCreate

Prototype

```
FMSTR_BOOL FMSTR_RecorderCreate(FMSTR_INDEX recIndex, FMSTR_REC_BUFF* buffCfg);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function registers a recorder instance and enables it to be used by the PC Host tool. Call this function for all recorder instances from 0 to the maximum number defined by the `FMSTR_USE_RECORDER` configuration option (minus one). An exception to this requirement is the recorder of instance 0 which may be automatically configured by `FMSTR_Init` when the *freemaster_cfg.h* configuration file defines the `FMSTR_REC_BUFF_SIZE` and `FMSTR_REC_TIMEBASE` options.

For more information, see [Configurable items](#).

FMSTR_Recorder

Prototype

```
void FMSTR_Recorder(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function takes a sample of the variables being recorded using the FreeMASTER Recorder instance *recIndex*. If the selected Recorder is not active when the *FMSTR_Recorder* function is being called, the function returns immediately. When the Recorder is active, the values of the variables being recorded are copied into the recorder buffer and the trigger conditions are evaluated.

If a trigger condition is satisfied, the Recorder enters the post-trigger mode, where it counts down the follow-up samples (number of *FMSTR_Recorder* function calls) and de-activates the Recorder when the required post-trigger samples are finished.

The *FMSTR_Recorder* function is typically called in the timer or PWM interrupt service routines. This function can also be called in the application main loop (for testing purposes).

FMSTR_RecorderTrigger

Prototype

```
void FMSTR_RecorderTrigger(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function forces the Recorder trigger condition to happen, which causes the Recorder to be automatically deactivated after the post-trigger samples are sampled. Use this function in the application code for programmatic control over the Recorder triggering. This can be useful when a more complex triggering conditions need to be used.

Fast Recorder API The Fast Recorder feature is not available in the FreeMASTER driver version 3. This feature was heavily dependent on the target platform and it was only available for the 56F8xxxx DSCs.

TSA Tables When the TSA is enabled in the FreeMASTER driver configuration file (by setting the *FMSTR_USE_TSA* macro to a non-zero value), it defines the so-called TSA tables in the application. This section describes the macros that must be used to define the TSA tables.

There can be any number of TSA tables spread across the application source files. There must be always exactly one TSA Table List defined, which informs the FreeMASTER driver about the active TSA tables.

When there is at least one TSA table and one TSA Table List defined in the application, the TSA information automatically appears in the FreeMASTER symbols list. The symbols can then be used to create FreeMASTER variables for visualization or control.

TSA table definition The TSA table describes the static or global variables together with their address, size, type, and access-protection information. If the TSA-described variables are of a structure type, the TSA table may also describe this type and provide an access to the individual structure members of the variable.

The TSA table definition begins with the FMSTR_TSA_TABLE_BEGIN macro with a *table_id* identifying the table. The *table_id* shall be a valid C-language symbol.

```
FMSTR_TSA_TABLE_BEGIN(table_id)
```

After this opening macro, the TSA descriptors are placed using these macros:

```
/* Adding variable descriptors */
FMSTR_TSA_RW_VAR(name, type) /* read/write variable entry */
FMSTR_TSA_RO_VAR(name, type) /* read-only variable entry */

/* Description of complex data types */
FMSTR_TSA_STRUCT(struct_name) /* structure or union type entry */
FMSTR_TSA_MEMBER(struct_name, member_name, type) /* structure member entry */

/* Memory blocks */
FMSTR_TSA_RW_MEM(name, type, address, size) /* read/write memory block */
FMSTR_TSA_RO_MEM(name, type, address, size) /* read-only memory block */
```

The table is closed using the FMSTR_TSA_TABLE_END macro:

```
FMSTR_TSA_TABLE_END()
```

TSA descriptor parameters The TSA descriptor macros accept these parameters:

- *name* — variable name. The variable must be defined before the TSA descriptor references it.
- *type* — variable or member type. Only one of the pre-defined type constants may be used (see below).
- *struct_name* — structure type name. The type must be defined (typedef) before the TSA descriptor references it.
- *member_name* — structure member name.

Note: The structure member descriptors (FMSTR_TSA_MEMBER) must immediately follow the parent structure descriptor (FMSTR_TSA_STRUCT) in the table.

Note: To write-protect the variables in the FreeMASTER driver (FMSTR_TSA_RO_VAR), enable the TSA-Safety feature in the configuration file.

TSA variable types The table lists *type* identifiers which can be used in TSA descriptors:

Constant	Description
FMSTR_TSA_UINT n	Unsigned integer type of size n bits ($n=8,16,32,64$)
FMSTR_TSA_SINT n	Signed integer type of size n bits ($n=8,16,32,64$)
FMSTR_TSA_FRAC n	Fractional number of size n bits ($n=16,32,64$).
FMSTR_TSA_FRAC_Q(m,n)	Signed fractional number in general Q form ($m+n+1$ total bits)
FMSTR_TSA_FRAC_UQ(m,n)	Unsigned fractional number in general UQ form ($m+n$ total bits)
FMSTR_TSA_FLOAT	4-byte standard IEEE floating-point type
FMSTR_TSA_DOUBLE	8-byte standard IEEE floating-point type
FMSTR_TSA_POINTER	Generic pointer type defined (platform-specific 16 or 32 bit)
FM-STR_TSA_USERTYPE($name$)	Structure or union type declared with FMSTR_TSA_STRUCT record

TSA table list There shall be exactly one TSA Table List in the application. The list contains one entry for each TSA table defined anywhere in the application.

The TSA Table List begins with the FMSTR_TSA_TABLE_LIST_BEGIN macro and continues with the TSA table entries for each table.

```
FMSTR_TSA_TABLE_LIST_BEGIN()

FMSTR_TSA_TABLE(table_id)
FMSTR_TSA_TABLE(table_id2)
FMSTR_TSA_TABLE(table_id3)
...
```

The list is closed with the FMSTR_TSA_TABLE_LIST_END macro:

```
FMSTR_TSA_TABLE_LIST_END()
```

TSA Active Content entries FreeMASTER v2.0 and higher supports TSA Active Content, enabling the TSA tables to describe the memory-mapped files, virtual directories, and URL hyperlinks. FreeMASTER can access such objects similarly to accessing the files and folders on the local hard drive.

With this set of TSA entries, the FreeMASTER pages can be embedded directly into the target MCU flash and accessed by FreeMASTER directly over the communication line. The HTML-coded pages rendered inside the FreeMASTER window can access the TSA Active Content resources using a special URL referencing the *fmstr:* protocol.

This example provides an overview of the supported TSA Active Content entries:

```
FMSTR_TSA_TABLE_BEGIN(files_and_links)

/* Directory entry applies to all subsequent MEMFILE entries */
FMSTR_TSA_DIRECTORY("/text_files") /* entering a new virtual directory */

/* The readme.txt file will be accessible at the fmstr://text_files/readme.txt URL */
FMSTR_TSA_MEMFILE("readme.txt", readme_txt, sizeof(readme_txt)) /* memory-mapped file */

/* Files can also be specified with a full path so the DIRECTORY entry does not apply */
FMSTR_TSA_MEMFILE("/index.htm", index, sizeof(index)) /* memory-mapped file */
FMSTR_TSA_MEMFILE("/prj/demo.pmp", demo_pmp, sizeof(demo_pmp)) /* memory-mapped file */

/* Hyperlinks can point to a local MEMFILE object or to the Internet */
FMSTR_TSA_HREF("Board's Built-in Welcome Page", "/index.htm")
FMSTR_TSA_HREF("FreeMASTER Home Page", "http://www.nxp.com/freemaster")
```

(continues on next page)

(continued from previous page)

```
/* Project file links simplify opening the projects from any URLs */
FMSTR_TSA_PROJECT("Demonstration Project (embedded)", "/prj/demo.pmp")
FMSTR_TSA_PROJECT("Full Project (online)", "http://mycompany.com/prj/demo.pmp")

FMSTR_TSA_TABLE_END()
```

TSA API

FMSTR_SetUpTsaBuff

Prototype

```
FMSTR_BOOL FMSTR_SetUpTsaBuff(FMSTR_ADDR buffAddr, FMSTR_SIZE buffSize);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_tsa.c*

Arguments

- *buffAddr* [in] - address of the memory buffer for the dynamic TSA table
- *buffSize* [in] - size of the memory buffer which determines the maximum number of TSA entries to be added in the runtime

Description This function must be used to assign the RAM memory buffer to the TSA subsystem when FMSTR_USE_TSA_DYNAMIC is enabled. The memory buffer is then used to store the TSA entries added dynamically to the runtime TSA table using the FMSTR_TsaAddVar function call. The runtime TSA table is processed by the FreeMASTER PC Host tool along with all static tables as soon as the communication port is open.

The size of the memory buffer determines the number of TSA entries that can be added dynamically. Depending on the MCU platform, one TSA entry takes either 8 or 16 bytes.

FMSTR_TsaAddVar

Prototype

```
FMSTR_BOOL FMSTR_TsaAddVar(FMSTR_TSATBL_STRPTR tsaName, FMSTR_TSATBL_STRPTR ↵
↵ tsaType,
    FMSTR_TSATBL_VOIDPTR varAddr, FMSTR_SIZE32 varSize,
    FMSTR_SIZE flags);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_tsa.c*

Arguments

- *tsaName* [in] - name of the object
- *tsaType* [in] - name of the object type
- *varAddr* [in] - address of the object

- *varSize* [in] - size of the object
- *flags* [in] - access flags; a combination of these values:
 - *FMSTR_TSA_INFO_RO_VAR* — read-only memory-mapped object (typically a variable)
 - *FMSTR_TSA_INFO_RW_VAR* — read/write memory-mapped object
 - *FMSTR_TSA_INFO_NON_VAR* — other entry, describing structure types, structure members, enumerations, and other types

Description This function can be called only when the dynamic TSA table is enabled by the *FMSTR_USE_TSA_DYNAMIC* configuration option and when the *FMSTR_SetUpTsaBuff* function call is made to assign the dynamic TSA table memory. This function adds an entry into the dynamic TSA table. It can be used to register a read-only or read/write memory object or describe an item of the user-defined type.

See [TSA table definition](#) for more details about the TSA table entries.

Application Commands API

FMSTR_GetAppCmd

Prototype

```
FMSTR_APPCMD_CODE FMSTR_GetAppCmd(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Description This function can be used to detect if there is an Application Command waiting to be processed by the application. If no command is pending, this function returns the *FMSTR_APPCMDRESULT_NOCMD* constant. Otherwise, this function returns the code of the Application Command that must be processed. Use the *FMSTR_AppCmdAck* call to acknowledge the Application Command after it is processed and to return the appropriate result code to the host.

The *FMSTR_GetAppCmd* function does not report the commands for which a callback handler function exists. If the *FMSTR_GetAppCmd* function is called when a callback-registered command is pending (and before it is actually processed by the callback function), this function returns *FMSTR_APPCMDRESULT_NOCMD*.

FMSTR_GetAppCmdData

Prototype

```
FMSTR_APPCMD_PDATA FMSTR_GetAppCmdData(FMSTR_SIZE* dataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *dataLen* [out] - pointer to the variable that receives the length of the data available in the buffer. It can be NULL when this information is not needed.

Description This function can be used to retrieve the Application Command data when the application determines that an Application Command is pending (see [FMSTR_GetAppCmd](#)).

There is just a single buffer to hold the Application Command data (the buffer length is FMSTR_APPCMD_BUFF_SIZE bytes). If the data are to be used in the application after the command is processed by the FMSTR_AppCmdAck call, copy the data out to a private buffer.

FMSTR_AppCmdAck

Prototype

```
void FMSTR_AppCmdAck(FMSTR_APPCMD_RESULT resultCode);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultCode* [in] - the result code which is to be returned to FreeMASTER

Description This function is used when the Application Command processing finishes in the application. The resultCode passed to this function is returned back to the host and the driver is re-initialized to expect the next Application Command.

After this function is called and before the next Application Command arrives, the return value of the FMSTR_GetAppCmd function is FMSTR_APPCMDRESULT_NOCMD.

FMSTR_AppCmdSetResponseData

Prototype

```
void FMSTR_AppCmdSetResponseData(FMSTR_ADDR resultDataAddr, FMSTR_SIZE resultDataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultDataAddr* [in] - pointer to the data buffer that is to be copied to the Application Command data buffer
- *resultDataLen* [in] - length of the data to be copied. It must not exceed the FMSTR_APPCMD_BUFF_SIZE value.

Description This function can be used before the Application Command processing finishes, when there are data to be returned back to the PC.

The response data buffer is copied into the Application Command data buffer, from where it is accessed when the host requires it. Do not use FMSTR_GetAppCmdData and the data buffer after FMSTR_AppCmdSetResponseData is called.

Note: The current version of FreeMASTER does not support the Application Command response data.

FMSTR_RegisterAppCmdCall

Prototype

```
FMSTR_BOOL FMSTR_RegisterAppCmdCall(FMSTR_APPCMD_CODE appCmdCode, FMSTR_
↳ PAPPCMDFUNC callbackFunc);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *appCmdCode* [in] - the Application Command code for which the callback is to be registered
- *callbackFunc* [in] - pointer to the callback function that is to be registered. Use NULL to unregister a callback registered previously with this Application Command.

Return value This function returns a non-zero value when the callback function was successfully registered or unregistered. It can return zero when trying to register a callback function for more than FMSTR_MAX_APPCMD_CALLS different Application Commands.

Description This function can be used to register the given function as a callback handler for the Application Command. The Application Command is identified using single-byte code. The callback function is invoked automatically by the FreeMASTER driver when the protocol decoder obtains a request to get the application command result code.

The prototype of the callback function is

```
FMSTR_APPCMD_RESULT HandlerFunction(FMSTR_APPCMD_CODE nAppcmd,
FMSTR_APPCMD_PDATA pData, FMSTR_SIZE nDataLen);
```

Where:

- *nAppcmd* -Application Command code
- *pData* —points to the Application Command data received (if any)
- *nDataLen* —information about the Application Command data length

The return value of the callback function is used as the Application Command Result Code and returned to FreeMASTER.

Note: The FMSTR_MAX_APPCMD_CALLS configuration macro defines how many different Application Commands may be handled by a callback function. When FMSTR_MAX_APPCMD_CALLS is undefined or defined as zero, the FMSTR_RegisterAppCmdCall function always fails.

Pipes API

FMSTR_PipeOpen

Prototype

```
FMSTR_HPIPE FMSTR_PipeOpen(FMSTR_PIPE_PORT pipePort, FMSTR_PPIPEFUNC pipeCallback,
↳
FMSTR_ADDR pipeRxBuff, FMSTR_PIPE_SIZE pipeRxSize,
FMSTR_ADDR pipeTxBuff, FMSTR_PIPE_SIZE pipeTxSize,
FMSTR_U8 type, const FMSTR_CHAR *name);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipePort* [in] - port number that identifies the pipe for the client
- *pipeCallback* [in] - pointer to the callback function that is called whenever a pipe data status changes
- *pipeRxBuff* [in] - address of the receive memory buffer
- *pipeRxSize* [in] - size of the receive memory buffer
- *pipeTxBuff* [in] - address of the transmit memory buffer
- *pipeTxSize* [in] - size of the transmit memory buffer
- *type* [in] - a combination of FMSTR_PIPE_MODE_XXX and FMSTR_PIPE_SIZE_XXX constants describing primary pipe data format and usage. This type helps FreeMASTER decide how to access the pipe by default. Optional, use 0 when undetermined.
- *name* [in] - user name of the pipe port. This name is visible to the FreeMASTER user when creating the graphical pipe interface.

Description This function initializes a new pipe and makes it ready to accept or send the data to the PC Host client. The receive memory buffer is used to store the received data before they are read out by the FMSTR_PipeRead call. When this buffer gets full, the PC Host client denies the data transmission into this pipe until there is enough free space again. The transmit memory buffer is used to store the data transmitted by the application to the PC Host client using the FMSTR_PipeWrite call. The transmit buffer can get full when the PC Host is disconnected or when it is slow in receiving and reading out the pipe data.

The function returns the pipe handle which must be stored and used in the subsequent calls to manage the pipe object.

The callback function (if specified) is called whenever new data are received through the pipe and available for reading. This callback is also called when the data waiting in the transmit buffer are successfully pushed to the PC Host and the transmit buffer free space increases. The prototype of the callback function provided by the user application must be as follows. The *PipeHandler* name is only a placeholder and must be defined by the application.

```
void PipeHandler(FMSTR_HPIPE pipeHandle);
```

FMSTR_PipeClose

Prototype

```
void FMSTR_PipeClose(FMSTR_HPIPE pipeHandle);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call

Description This function de-initializes the pipe object. No data can be received or sent on the pipe after this call.

FMSTR_PipeWrite

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeWrite(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE writeGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data to be written
- *pipeDataLen* [in] - length of the data to be written
- *writeGranularity* [in] - size of the minimum unit of data which is to be written

Description This function puts the user-specified data into the pipe's transmit memory buffer and schedules it for transmission. This function returns the number of bytes that were successfully written into the buffer. This number may be smaller than the number of the requested bytes if there is not enough free space in the transmit buffer.

The *writeGranularity* argument can be used to split the data into smaller chunks, each of the size given by the *writeGranularity* value. The FMSTR_PipeWrite function writes as many data chunks as possible into the transmit buffer and does not attempt to write an incomplete chunk. This feature can prove to be useful to avoid the intermediate caching when writing an array of integer values or other multi-byte data items. When making the *nGranularity* value equal to the *nLength* value, all data are considered as one chunk which is either written successfully as a whole or not at all. The *nGranularity* value of 0 or 1 disables the data-chunk approach.

FMSTR_PipeRead

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeRead(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE readGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data buffer to be filled with the received data
- *pipeDataLen* [in] - length of the data to be read
- *readGranularity* [in] - size of the minimum unit of data which is to be read

Description This function copies the data received from the pipe from its receive buffer to the user buffer for further processing. The function returns the number of bytes that were successfully copied to the buffer. This number may be smaller than the number of the requested bytes if there is not enough data bytes available in the receive buffer.

The `readGranularity` argument can be used to copy the data in larger chunks in the same way as described in the `FMSTR_PipeWrite` function.

API data types This section describes the data types used in the FreeMASTER driver. The information provided here can be useful when modifying or porting the FreeMASTER Communication Driver to new NXP platforms.

Note: The licensing conditions prohibit use of FreeMASTER and the FreeMASTER Communication Driver with non-NXP MPU or MCU products.

Public common types The table below describes the public data types used in the FreeMASTER driver API calls. The data types are declared in the *freemaster.h* header file.

Type name	Description
<i>FM-STR_ADDR</i> For example, this type is defined as long integer on the 56F8xxx platform where the 24-bit addresses must be supported, but the C-pointer may be only 16 bits wide in some compiler configurations.	Data type used to hold the memory address. On most platforms, this is normally a C-pointer, but it may also be a pure integer type.
<i>FM-STR_SIZE</i> It is required that this type is unsigned and at least 16 bits wide integer.	Data type used to hold the memory block size.
<i>FM-STR_BOOL</i> This type is used only in zero/non-zero conditions in the driver code.	Data type used as a general boolean type.
<i>FM-STR_APPCM</i> Generally, this is an unsigned 8-bit value.	Data type used to hold the Application Command code.
<i>FM-STR_APPCM</i> Generally, this is an unsigned 8-bit value.	Data type used to create the Application Command data buffer.
<i>FM-STR_APPCM</i> Generally, this is an unsigned 8-bit value.	Data type used to hold the Application Command result code.

Public TSA types The table describes the TSA-specific public data types. These types are declared in the *freemaster_tsa.h* header file, which is included in the user application indirectly by the *freemaster.h* file.

<i>FM-STR_TSA_TII</i>	Data type used to hold a descriptor index in the TSA table or a table index in the list of TSA tables. By default, this is defined as <i>FM-STR_SIZE</i> .
<i>FM-STR_TSA_TS</i>	Data type used to hold a memory block size, as used in the TSA descriptors. By default, this is defined as <i>FM-STR_SIZE</i> .

Public Pipes types The table describes the data types used by the FreeMASTER Pipes API:

<i>FM-STR_HPIPE</i>	Pipe handle that identifies the open-pipe object. Generally, this is a pointer to a void type.
<i>FM-STR_PIPE_PC</i>	Integer type required to hold at least 7 bits of data. Generally, this is an unsigned 8-bit or 16-bit type.
<i>FM-STR_PIPE_SI</i>	Integer type required to hold at least 16 bits of data. This is used to store the data buffer sizes.
<i>FM-STR_PPIPEF</i>	Pointer to the pipe handler function. See FM-STR_PipeOpen for more details.

Internal types The table describes the data types used internally by the FreeMASTER driver. The data types are declared in the platform-specific header file and they are not available in the application code.

<i>FMSTR_U8</i>	The smallest memory entity.
On the vast majority of platforms, this is an unsigned 8-bit integer.	
On the 56F8xx DSP platform, this is defined as an unsigned 16-bit integer.	
<i>FMSTR_U16</i>	Unsigned 16-bit integer.
<i>FMSTR_U32</i>	Unsigned 32-bit integer.
<i>FMSTR_S8</i>	Signed 8-bit integer.
<i>FMSTR_S16</i>	Signed 16-bit integer.
<i>FMSTR_S32</i>	Signed 32-bit integer.
<i>FMSTR_FLOAT</i>	4-byte standard IEEE floating-point type.
<i>FMSTR_FLAGS</i>	Data type forming a union with a structure of flag bit-fields.
<i>FMSTR_SIZE8</i>	Data type holding a general size value, at least 8 bits wide.
<i>FMSTR_INDEX</i>	General for-loop index. Must be signed, at least 16 bits wide.
<i>FMSTR_BCHR</i>	A single character in the communication buffer.
Typically, this is an 8-bit unsigned integer, except for the DSP platforms where it is a 16-bit integer.	
<i>FMSTR_BPTR</i>	A pointer to the communication buffer (an array of <i>FMSTR_BCHR</i>).

Document references

Links

- This document online: <https://mcuxpresso.nxp.com/mcuxsdk/latest/html/middleware/freemaster/doc/index.html>

- FreeMASTER tool home: www.nxp.com/freemaster
- FreeMASTER community area: community.nxp.com/community/freemaster
- FreeMASTER GitHub code repo: <https://github.com/nxp-mcuxpresso/mcux-freemaster>
- MCUXpresso SDK home: www.nxp.com/mcuxpresso
- MCUXpresso SDK builder: mcuxpresso.nxp.com/en

Documents

- *FreeMASTER Usage Serial Driver Implementation* (document [AN4752](#))
- *Integrating FreeMASTER Time Debugging Tool With CodeWarrior For Microcontrollers v10.X Project* (document [AN4771](#))
- *Flash Driver Library For MC56F847xx And MC56F827xx DSC Family* (document [AN4860](#))

Revision history This Table summarizes the changes done to this document since the initial release.

Revision	Date	Description
1.0	03/2006	Limited initial release
2.0	09/2007	Updated for FreeMASTER version. New Freescale document template used.
2.1	12/2007	Added description of the new Fast Recorder feature and its API.
2.2	04/2010	Added support for MPC56xx platform, Added new API for use CAN interface.
2.3	04/2011	Added support for Kxx Kinetis platform and MQX operating system.
2.4	06/2011	Serial driver update, adds support for USB CDC interface.
2.5	08/2011	Added Packet Driven BDM interface.
2.7	12/2013	Added FLEXCAN32 interface, byte access and isr callback configuration option.
2.8	06/2014	Removed obsolete license text, see the software package content for up-to-date license.
2.9	03/2015	Update for driver version 1.8.2 and 1.9: FreeMASTER Pipes, TSA Active Content, LIN Transport Layer support, DEBUG-TX communication troubleshooting, Kinetis SDK support.
3.0	08/2016	Update for driver version 2.0: Added support for MPC56xx, MPC57xx, KEAxx and S32Kxx platforms. New NXP document template as well as new license agreement used. added MCAN interface. Folders structure at the installation destination was rearranged.
4.0	04/2019	Update for driver released as part of FreeMASTER v3.0 and MCUXpresso SDK 2.6. Updated to match new V4 serial communication protocol and new configuration options. This version of the document removes substantial portion of outdated information related to S08, S12, ColdFire, Power and other legacy platforms.
4.1	04/2020	Minor update for FreeMASTER driver included in MCUXpresso SDK 2.8.
4.2	09/2020	Added example applications description and information about the MCUXpresso Config Tools. Fixed the pipe-related API description.
4.3	10/2024	Added description of Network and Segger J-Link RTT interface configuration. Accompanying the MCUXpresso SDK version 24.12.00.
4.4	04/2025	Added Zephyr-specific information. Accompanying the MCUXpresso SDK version 25.06.00.

3.4 Multimedia

3.4.1 Audio Voice

Maestro Audio Framework

MCUXpresso SDK : Maestro

Overview This repository is for MCUXpresso SDK maestro middleware delivery and it contains the components officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository (mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

Documentation Overall details can be reviewed here: [MCUXpresso SDK Online Documentation](#)

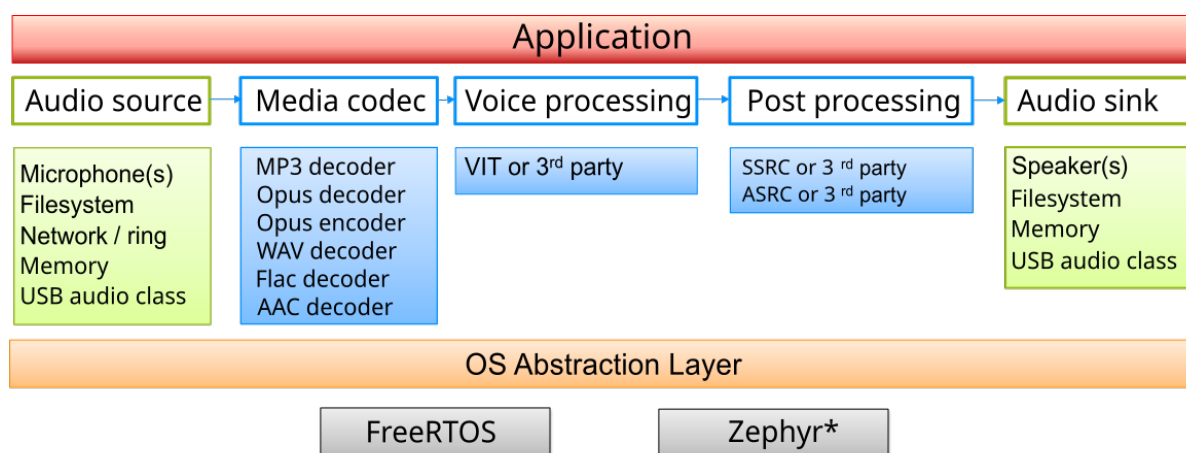
Visit [Maestro - Documentation](#) to review details on the contents in this sub-repo.

Setup Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest [Getting Started with SDK - Detailed Installation Instructions](#)

Contribution We welcome and encourage the community to submit patches directly to the Maestro project placed on github. Contributing can be managed via pull-requests.

Introduction Maestro audio framework intends to enable chaining of basic audio processing blocks, called *elements*. These blocks then form stream processing objects, called *pipeline*. This pipeline can be used for multiple audio processing use cases.

The processing blocks can include (but are not limited to) different audio sources (for example file or microphone), decoders or encoders, filters or effects, and audio sinks. Framework overview is depicted in the following picture:



*not all elements and libraries are supported in Zephyr port. For more information, see [Maestro on Zephyr](#)

The Maestro audio framework is an open-source component developed by NXP Semiconductors and released under the BSD-compatible license. It is running on RTOS (Zephyr or FreeRTOS), abstracted by OSA layer.

For detailed description of the audio Maestro framework, please refer to the [programmer's guide](#).

To see what is new, see [changelog](#).

Maestro on Zephyr Getting started guide and further information for Maestro on Zephyr may be found [here](#).

Maestro on FreeRTOS Maestro on FreeRTOS is supported in NXP's SDK. To get started, see [mcuxsdk doc](#).

Supported examples The current version of the Maestro audio framework supports several optional [features](#), some of which are used in these examples:

- [maestro_playback](#)
- [maestro_record](#)
- [maestro_usb_mic](#)
- [maestro_usb_speaker](#)

The examples can be found in the **audio_examples** folder of the desired board. The demo applications are based on FreeRTOS and use multiple tasks to form the application functionality.

Example applications overview To set up the audio framework properly, it is necessary to create a streamer with `streamer_create` API. It is also essential to set up the desired hardware peripherals using the functions described in `streamer_pcm.h`. The Maestro example projects consist of several files regarding the audio framework. The initial file is `main.c` with code to create multiple tasks. For features including SD card (in the `maestro_playback` examples, reading a file from SD card is supported and in `maestro_record` writing to SD card is currently supported) the `APP_SDCARD_Task` is created. The command prompt and connected functionalities are handled by `APP_Shell_Task`.

One of the most important parts of the configuration is the `streamer_pcm.c` where the initialization of the hardware peripherals, input and output buffer management can be found. For further information please see also `streamer_pcm.h`

In the Maestro USB examples (`maestro_usb_mic` and `maestro_usb_speaker`), the USB configuration is located in the `usb_device_descriptor.c`, `audio_microphone.c` and `audio_speaker.c` files. For further information please see also `usb_device_descriptor.h`, `audio_microphone.h` and `audio_speaker.h`.

In order to be able to get the messages from the audio framework, it is necessary to create a thread for receiving the messages from the streamer, which is usually called a Message Task. The message thread is placed in the `app_streamer.c` file, reads the streamer message queue, and reacts to the following messages:

- `STREAM_MSG_ERROR` - stops the streamer and exits the message thread
- `STREAM_MSG_EOS` - stops the streamer and exits the message thread
- `STREAM_MSG_UPDATE_DURATION` - prints info about the stream duration
- `STREAM_MSG_UPDATE_POSITION` - prints info about current stream position
- `STREAM_MSG_CLOSE_TASK` - exits the message thread

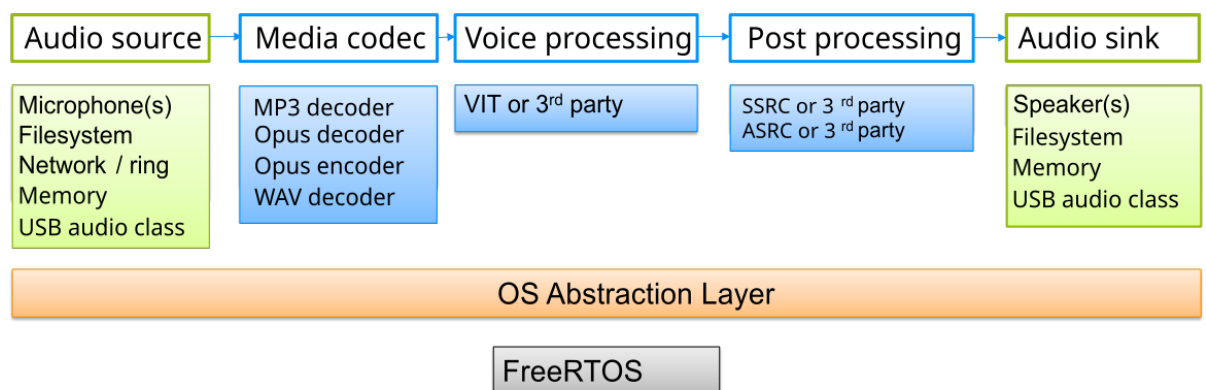
File structure

Folder	Description
src	Maestro audio framework sources
src/inc	Maestro include files
src/core	Maestro core sources
src/cci	Common decoder interface sources
src/cei	Common encoder interface sources
src/elements	Maestro elements sources
src/devices	External audio devices implementation (audio source & audio sink elements)
src/utls	Helper utilities utilized by Maestro
docs	Generated documentation
doxygen	Documentation sources
components	Glue for audio libraries, so they can be used in elements
tests	Maestro tests
zephyr/	Zephyr related files
zephyr/samples/	Zephyr samples
zephyr/tests/	Zephyr tests
zephyr/audioTracks/	Audio tracks for testing
zephyr/wrappers/	Zephyr NXP SDK Wrappers
zephyr/doc/	Zephyr documentation configuration for Sphinx
zephyr/scripts/	Zephyr helper scripts, mostly for testing

Maestro Audio Framework Programmer's Guide

Introduction Maestro audio framework provides instruments for playback and capture of different audio streams. In order to do that the framework uses API for creating various audio and voice pipelines with the support of media and track information. This document describes the framework in its detail, and the usage of API for pipeline creation using different elements. The framework needs an operating system in order to create different tasks for audio processing and communication with the application.

Architecture overview A high-level block diagram of the streamer used in Maestro is shown below. An element is the most important class of objects in the streamer (see `streamer_element.c`). A chain of elements will be created and linked together when a *pipeline* is created. Data flows through this chain of elements in form of data buffers. An element has one specific function, which can be the reading of data from a file, decoding of this data, or outputting this data to a sink device. By chaining together several such elements, a pipeline is created that can do a specific task, for example, the playback.



Pipeline

The pipeline is created within the `streamer_create` API using the `streamer_create_pipeline` call. In the example applications provided in the MCUXpresso SDK the pipeline is created in the `app_streamer.c` file. In order to create a pipeline user needs to provide a `PipelineElements` structure consisting of array of element indexes `ElementIndex` and the number of elements in the pipeline. Then the pipeline is built automatically and user can specify the properties of the elements using the `streamer_set_property` API. All the element properties can be found in the `streamer_element_properties.h` file.

The streamer can handle up to two pipelines within a single task. The first pipeline with index 0 can be created using the `streamer_create` function as described above. Then the `streamer_create_pipeline` function should be used to create the second pipeline (pipeline with index 1). Both pipelines are processed sequentially, so after the first pipeline is processed, the second pipeline is processed.

After the pipeline is successfully created, all elements and entire pipeline are in `STATE_NULL` state. A user can start the streamer by setting the pipeline state to `STATE_PLAYING` using the `streamer_set_state` function. The pipeline can also be paused or stopped using the same function. Use the `STATE_PAUSED` to pause and use `STATE_NULL` to stop. The function changes the state of each element that is in the pipeline in turn, and after all the elements have obtained the desired state, the state of entire pipeline is changed.

Elements The current version of the Maestro framework supports several types of elements (`StreamElementType`). In each pipeline should be used one source element (elements with the `_SRC` suffix) and one sink element (elements with the `_SINK` suffix). A decoder, encoder or audio_proc element can be connected between these two elements. The `audio_proc` element can be used more than once within the same pipeline.

Each element type (`StreamElementType`) has several functions that are determined by a unique element index (`ElementIndex`). These indexes are used to create a pipeline, and each element index can only be used once in the same pipeline. The `type_lookup_table` shows which `StreamElementType` supports which `ElementIndex`.

Each element index (`ElementIndex`) has its own properties and a list of these properties can be found in the `streamer_element_properties.h` file. These properties are divided into groups and each group is identified by a property mask (e.g. for speaker it is `PROP_SPEAKER_MASK`). Then the `property_lookup_table` in the `streamer_msg.c` file determines which property group relates to which element index (`ElementIndex`). When an element is created and added to the pipeline, its properties are set to their default values. Default values can be seen in the initialization function of a particular element. The initialization functions are specified in the `element_list` array in the `streamer_element.c` file (e.g. for the `audio_proc` element it is the `audio_proc_init_element` function). The user can get the value of the property using the `streamer_get_property` function or change its value using the `streamer_set_property` function.

The source code of the elements can be found in the `middleware\audio_voice\maestro\src\elements\` folder.

Add a new element type The user can add a new element type (`StreamElementType`) to the Maestro audio framework. For this, the following steps need to be done.

- Add a new element type to the `StreamElementType` enum type in the `streamer_api.h`.
- Create a new `*.c` and `*.h` files for the new element type in the `middleware\audio_voice\maestro\src\elements\` folder. All necessary structures and functions (functions for src pads, sink pads and element itself) needs to be defined in these files. Inspiration can be found in other elements.
- Link the initialization function to the element type in the `element_list` array in the `streamer_element.c` file. To do this, a new definition that enables the element needs to be created (e.g. there is a `STREAMER_ENABLE_AUDIO_PROC` definition for the `audio_proc` element).

- Associate the newly created element type with an element index (ElementIndex) by adding a new pair to the `type_lookup_table` in the `streamer.c` file.
- If the user wants to use the newly created element in an application, the definition that enables the element must be defined at the project level.

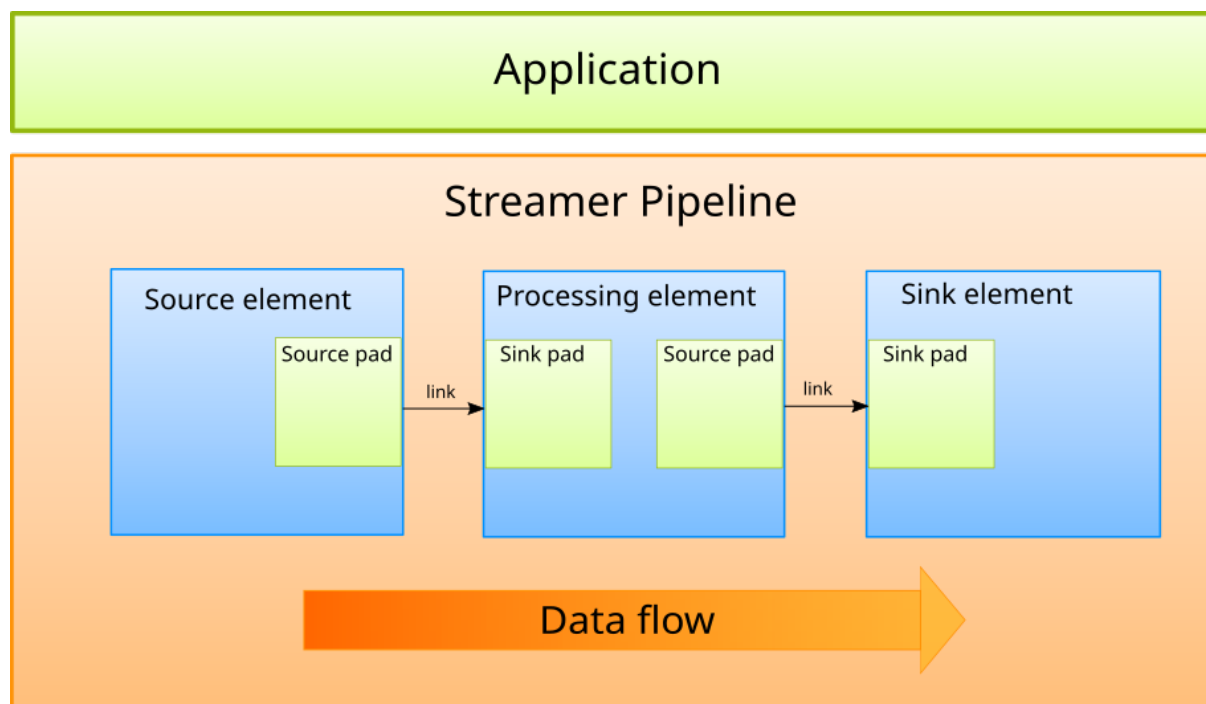
Mostly the user doesn't need to create a new element type, but just create an element index.

Add a new element index To create a new element index in the Maestro audio framework, follow these steps:

- Add a new element index to the `ElementIndex` enum type in the `streamer_api.h`.
- Create the required properties for the newly created element index in the `streamer_element_properties.h` file.
- Associate the newly created property group with newly created element index by adding a new pair to the `property_lookup_table` in the `streamer_msg.c` file.
- Associate the newly created element index with an element type (`StreamElementType`) by adding a new pair to the `type_lookup_table` in the `streamer.c` file.
- Add support for the created properties to functions of the associated element type. These functions are defined in files that correspond to a particular element type. The files are located in the `middleware\audio_voice\maestro\src\elements\` folder.

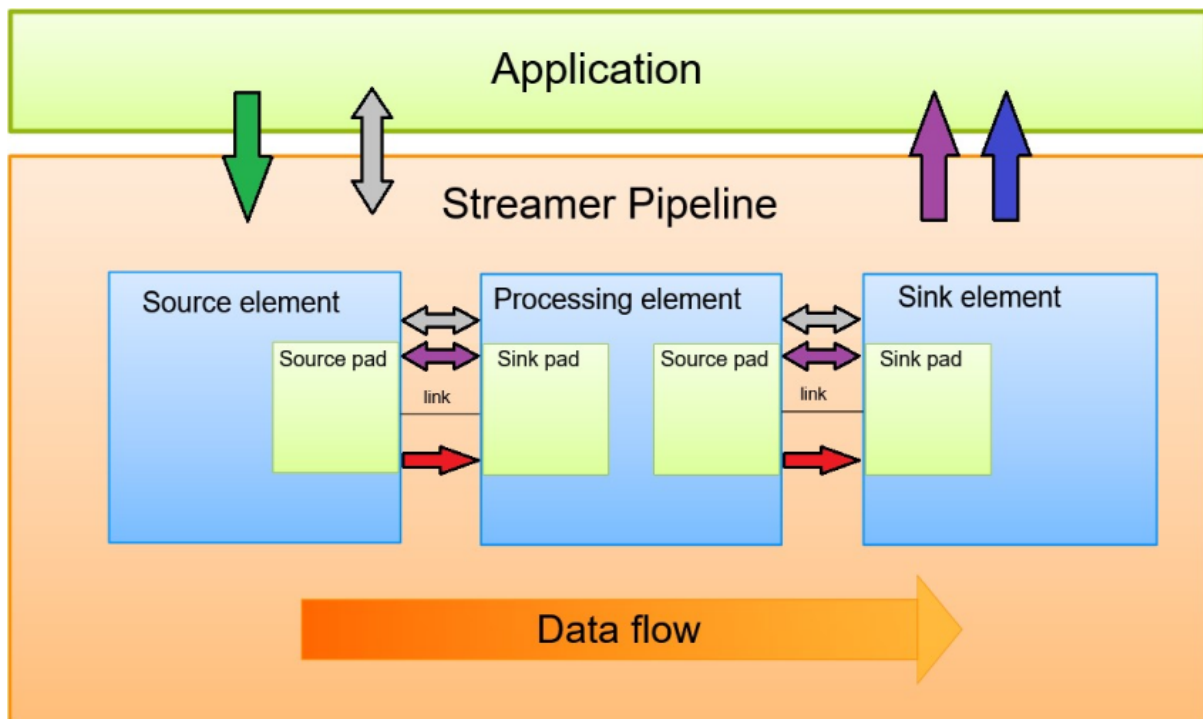
It is important to know that each element type (`StreamElementType`) can be associated with more than one element index (`ElementIndex`), but each element index (`ElementIndex`) can be associated with only one element type (`StreamElementType`).

Pads Pads are elements' inputs and outputs. A pad can be viewed as a “plug” or “port” on an element where links may be made with other elements, and through which data can flow to or from those elements. Data flows out of an element through a source pad, and elements accept incoming data through a sink pad. Source and sink elements have only source and sink pads, respectively. For detailed information about pads, please see the API reference from `pad.c`.



Internal communication The streamer (the core of the framework) provides several mechanisms for communication and data exchange between the application, a pipeline, and pipeline elements:

- Buffers are objects for passing streaming data between elements in the pipeline. Buffers always travel from sources to sinks (downstream).
- Messages are objects sent from the application to the streamer task to construct, configure, and control a streamer pipeline.
- Callbacks are used to transmit information such as errors, tags, state changes, etc. from the pipeline and elements to the application.
- Events are objects sent between elements. Events can travel upstream and downstream. Events may also be sent to the application.
- Queries allow applications to request information such as duration or current playback position from the pipeline. Elements can also use queries to request information from their peer elements (such as the file size or duration). They can be used both ways within a pipeline, but upstream queries are more common.



Decoders and encoders Maestro framework uses a common codec interface for decoding purposes and a common encoder interface for encoding. Those interfaces encapsulate the usage of specific codecs. Reference codecs are available in audio-voice-components repository which should be in `\middleware\audio_voice\components\` folder.

Common codec interface The Common Codec Interface is the intended interface for all used **decoders**. The framework will integrate a CCI decoder element into the streamer to interface with all decoders.

Using the CCI to interface with Metadata

- `cci_extract_meta_data` must be called before any other Codec Interface APIs. This API extracts the metadata information of the codec and fills this information in the

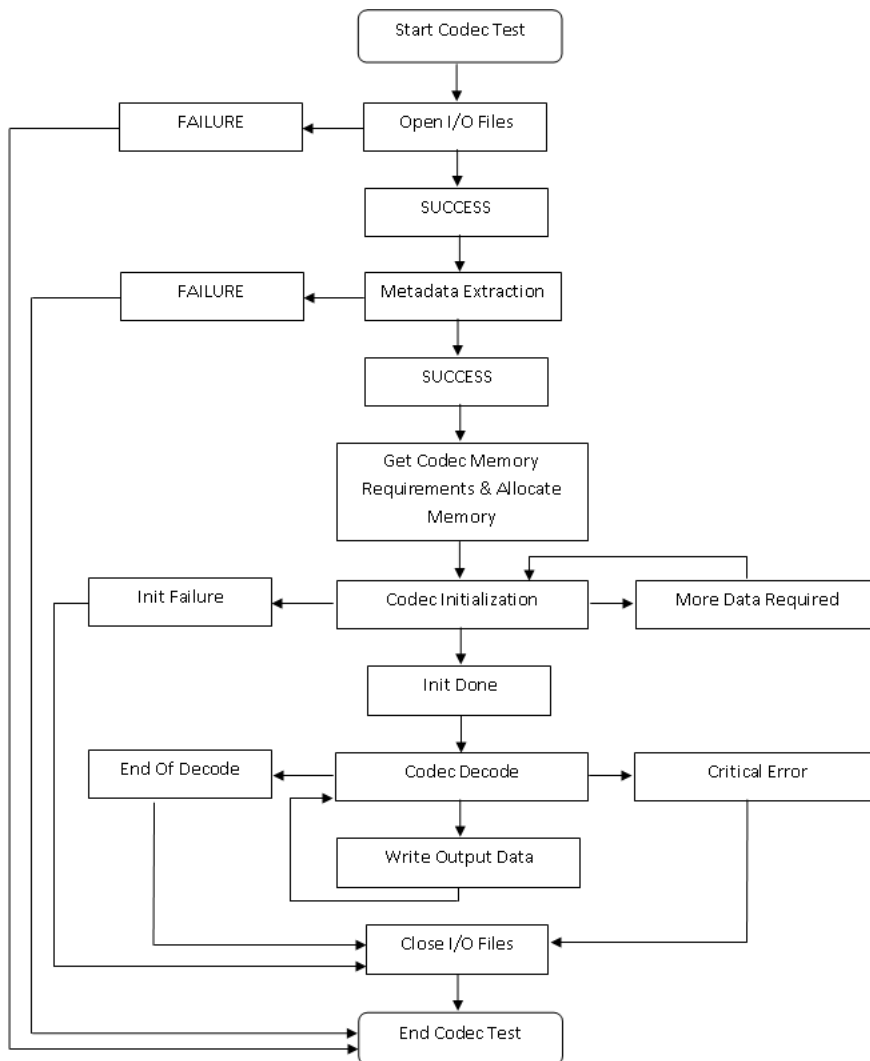
`file_meta_data_t` structure. The `file_meta_data_t` structure must be allocated by the application.

- This function first extracts the input file extension and based on that it calls the specific codec's metadata extraction function. If it finds an invalid extension or unsupported extension then it returns with `META_DATA_FILE_NOT_SUPPORTED` code for any unsupported file format.
- If this API finds the valid metadata then it returns with `META_DATA_FOUND` code. If this API does not find any metadata information then it returns with `META_DATA_NOT_FOUND` code. It also returns with `META_DATA_FILE_NOT_SUPPORTED` code for any unsupported file format.

Using the CCI to interface with Decoders

- `codec_get_mem_info` gets the memory requirement based on the specific decoder stream type. It returns the size in bytes of the specific codec. The user of the decoders must allocate memory of this size and this memory is used by the initialization API. The user or application must pass this allocated memory pointer to the init API.
- `codec_init` must be called before the codec's decode API. This API calls the codec-specific initialization function based on the codec stream type. This API allocates the memory to the codec main structure and also initializes the codec main structure parameters. It also registers the call back functions to the codec which will be used by the codec to read or seek the input stream.
- `codec_decode` is the main decoding API of the codec. This API calls the codec-specific decoding function based on the codec stream type. This API decodes the input raw stream and fills the PCM output samples into codec output PCM buffer. This API gives the information about the number of samples produced by the codec and also gives the pointer of the codec output PCM samples buffer.
- `codec_get_pcm_samples` must be called after the codec's decode API. This API calls the codec specific Get PCM Sample API based on the codec stream type. This API gets the PCM samples from the codec in constant block size and fills them into the output PCM buffer. It returns the number of samples get from the codec and also gives the pointer of the output PCM buffer.
- `codec_reset` calls the codec specific reset API base on stream type and resets the codec.
- `codec_seek` accepts the seek bytes offset converted from the time by application. This API calls the decoder's internal seek API to calculate the actual seek offset which frame boundary aligns. This API returns the actual seek offset.

The basic sequence to use a decoder with the CCI is shown below:



Adding new decoders to the CCI This section explains how to integrate a new decoder in the Common Codec Interface. The CCI assumes the decoder library to be used is in the `\middleware\audio_voice\audiocomponents\decoders\*decoder*\libs\` folder of the maestro framework. The CCI is just a wrapper around a specific implementation. The decoder is expected to be extended as needed to meet the APIs described above.

- Register Decoder Top level APIs in Common Codec Interface
 - Place the decoder lib in libs folder.
 - Add prototypes of the decoder top level APIs in `codec_interface.h` file (located at `maestro\src\cci\inc\` folder).
 - In `codec_interface.c` file (located at `maestro\src\cci\src\`), add top level Decoder APIs in decoder function table.
 - Pseudo code for this is as described below.

```

const codec_interface_function_table_t g_codec_function_table[STREAM_TYPE_COUNT] = {
#ifdef VORBIS_CODEC
{
    &VORBISDecoderGetMemorySize,
    &VORBISDecoderInit,
    &VORBISDecoderDecode,
    NULL,

```

(continues on next page)

(continued from previous page)

```

    NULL,
    &VORBISDecoderSeek,
    &VORBISDecoderGetIOFrameSize,
},
#else
{
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
}
#endif
};

```

- Enable or Disable Decoder
 - Define VORBIS_CODEEC macro in audio_cfg.h file.
 - Comment this macro if you want to disable VORBIS Decoder otherwise keep it defined in order to enable the decoder.
- Add Extract Metadata API for the decoder
 - Add extract metadata API source file for the decoder at streamer/cci/metadata/src/vorbis folder.
 - Add this code in extract metadata lib project space.
 - Build the extract metadata lib and copy that lib to libs folder.
 - Add the desired stream type into ccidec_extract_meta_data API (in codeceextractmeta-data.c file) to call VORBIS Decoder extract metadata API.
- Add stream type of the new decoder in the stream type enum audio_stream_type_t in codec_interface_public_api.h
 - Stream type of the decoder in stream type enum and decoder APIs in decoder function table must be in the same sequence.

Common encoder interface Please see the following section about the [cei](#).

Maestro performance

Memory information The memory usage of the framework components using reference codecs (data obtained from GNU ARM compiler) in bytes is:

text	data	bss	component
48790	2752	4	aac decoder
4348	16400	212	asrc
15512	0	4	flac decoder
76462	16	5013	maestro
34211	0	4	mp3 decoder
211974	0	0	opus
65446	0	4	ssrc
5850	16	12	wav decoder

Maestro framework uses dynamic allocation of audio buffers. The total amount of memory allocated for the pipeline depends on the following parameters:

- Number of elements in the pipeline
- Element types
- Audio stream properties
 - Sampling rate
 - Bit width
 - Channel number
 - Frame size

CPU usage The performance of the pipeline was measured using the real hardware platform (RT1060).

- CPU core clock in MHz: 600.

Pipeline type	Performance MIPS of pipeline (in MHz)
audio source -> audio sink	~10.26 MHz
audio source -> file sink	~9.84 MHz
file source (8-channel PCM) -> audio sink	~16.5 MHz

For performance details about the supported codecs please see audio-voice-components repository documentation.

CEI encoder The Maestro streamer contains an element adapting an extensible set of audio encoders in the form of functions conforming to the CEI (Common Encoder Interface). This element enables the user to choose and configure a suitable encoder at runtime.

Header files CEI itself and the CEI encoders are using following header files, in which you may be interested:

- `cei.h` - contains types used by the element itself and an encoder implementing the CEI
- `cei_enctypes.h` - contains a list of possible encoders and types used for interfacing with a CEI encoder
- `cei_table.h` - contains a table of functions implementing integrated CEI encoders

Instantiating the element This element's index is `ELEMENT_ENCODER_INDEX` and its type is `TYPE_ELEMENT_ENCODER`, as defined in `streamer_api.h`. It has one source pad (data input) and one sink pad (data output). It is initialized like any other element, meaning that it is instantiated and inserted into the pipeline using the `create_element`, `add_element_pipeline` and `link_elements` functions. Inversely, for destroying the element, the `unlink_elements`, `remove_element_pipeline` and `destroy_element` are used. This element alone does not depend on any additional software layers other than these required by the Maestro streamer itself, so no pre-initialization before this element instantiation is necessary.

Element properties Use Maestro streamer property API (`streamer_set_property` and `streamer_get_property`) for setting or getting these. The constants are defined in `streamer_element_properties.h`.

- `PROP_ENCODER_CHUNK_SIZE`

- **Synopsis:** Determines the length of a chunk pulled from the sibling of the source pad and essentially influences the size of allocated buffers. If the actual amount of data pulled is smaller, the rest is zero-filled.
- **Type:** unsigned 32-bit integer
- **Default value:** 1920
- **Constraints:**
 - * Must be bigger than zero, otherwise `STREAM_ERR_INVALID_ARGS` is returned.
 - * Cannot be changed if the actual encoder has been created. If done so, `STREAM_ERR_ELEMENT_BAD_STATUS` is returned.
- `PROP_ENCODER_TYPE`
 - **Synopsis:** Determines the exact encoder (CEI implementation) to be used.
 - **Type:** `CeiEncoderType` (`cei_enctypes.h`)
 - **Default value:** `CEIENC_LAST`
 - **Constraints:**
 - * Must not be equal to `CEIENC_LAST`, otherwise `STREAM_ERR_INVALID_ARGS` will be returned.
 - * Selected encoder must be implemented, otherwise `STREAM_ERR_INVALID_ARGS` will be returned.
 - * Cannot be changed if the actual encoder has been created. If done so, `STREAM_ERR_ELEMENT_BAD_STATUS` will be returned.
 - **Behaviour influenced:** The encoder element process function will return `FLOW_ERROR` if this property isn't set.
- `PROP_ENCODER_CONFIG`
 - **Synopsis:** Determines encoder-specific configuration (application, bitrate, ...).
 - **Type:** Pointer to the encoder-specific configuration structure.
 - **Default value:** Determined by the encoder.
 - **Constraints:**
 - * The encoder has to be configurable. If it is not, `STREAM_ERR_ERR_GENERAL` will be returned on any access.
 - * The structure has to conform to the encoder requirements. If the encoder returns an error code, `STREAM_ERR_GENERAL` will be returned.
- `PROP_ENCODER_BITSTREAMINFO`
 - **Synopsis:** Specifies information about the incoming bitstream (sample rate, sample depth, ...).
 - **Type:** Pointer to `CeiBitstreamInfo` (`cei_enctypes.h`).
 - **Default value:**

```
(CeiBitstreamInfo) {  
    .sample_rate = 0,  
    .num_channels = 0,  
    .endian = AF_LITTLE_ENDIAN,  
    .sign = TRUE,  
    .sample_size = 0,  
    .interleaved = TRUE  
}
```

– **Constraints:**

- * Cannot be changed if the actual encoder has been created. If done so, `STREAM_ERR_ELEMENT_BAD_STATUS` will be returned.
- * As of now, only bitstreams containing 16-bit interleaved (if 2 or more channels will be encoded) samples are supported. If anything else was set to the `sample_size` and `interleaved_members`, `STREAM_ERR_INVALID_ARGS` will be returned.

– **Behaviour influenced:**

- * Given the characteristics of some elements available, different packets of data (header and payload, referred to as “chunk” above) may be pulled by this element. Each packet can contain a different header, which may or may not contain useful information about the bitstream. If a packet with the `AudioPacketHeader` (`todofile.h`) is pulled at first and any other iteration of the streamer pipeline, the bitstream parameters configured by this property are implicitly available and are not expected to be specified by the user. Other packet header types (such as `RawPacketHeader`) don’t contain any bitstream parameters and require the user to specify the parameters manually using this property. Failure to do so will result in the element’s process function returning `FLOW_ERROR`. Same situation will occur if a packet with the `AudioPacketHeader` is received and its contents differ from the already acquired bitstream parameters.
- * As of now, CEI is defined to work with 16-bit signed little-endian (s16le) samples, which are interleaved if the bitstream contains more than one channels. This element handles endianness and unsigned to signed conversion.

CEI definition - implementing your own encoder The CEI defines following function pointer types:

- `CeiFnGetMemorySize`: Returns number of bytes required for encoder state for a given number of channels.
- `CeiFnEncoderInit`: Initialize an encoder for a given sample rate and channel count.
- `CeiFnEncoderGetConfig`: Copy current or default configuration to a given structure pointer.
- `CeiFnEncoderSetConfig`: Configure the encoder from a given structure pointer.
- `CeiFnEncode`: Encode a given buffer to a given output buffer.

Detailed descriptions of function behaviour, parameters and expected return values are available as docblocks in the `cei.h` file.

Each encoder is implemented as a set of pointers pointing to functions conforming to these types, grouped in the `CeiEncoderFunctions` structure. Specifying the `CeiEncoderGetConfig` `fnGetConfig` and `CeiFnEncoderSetConfig` `fnSetConfig` members is optional, as an encoder does not have to be configurable. If so desired, specify `NULL`. Implementation of the remaining functions is mandatory, however. If at least one of these functions isn’t implemented and `NULL` is specified instead, the encoder will be considered as not implemented.

To register an implemented encoder with the element, add a new entry to the `CeiEncoderType` enum and add the `CeiEncoderFunctions` struct value to the table `CeiEncoderFunctions` `ceiEncTable[]` located in the `cei_table.h` header file. Note and match the order of items in that table, as a `CeiEncoderType` value is used as an index. Same goes for the `size_t` `ceiEncConfigSizeTable[]`. If configuration is not applicable, specify 0 at the appropriate index. If configuration is applicable, describe the configuration structure in the `cei_encetypes.h` header file and add its size to that table.

Maestro playback example

Table of content

- [Overview](#)
- [Hardware requirements](#)
- [Hardware modifications](#)
- [Preparation](#)
- [Running the demo](#)
- [Example configuration](#)
- [Functionality](#)
- [States](#)
- [Commands in detail](#)
- [Processing Time](#)

Overview The Maestro playback example demonstrates audio processing on the ARM cortex core utilizing the Maestro Audio Framework library.

The application is controlled by commands from a shell interface using serial console and the audio files are read from the SD card.

Depending on target platform or development board there are different modes and features of the demo supported.

- **Standard** - The mode demonstrates playback of encoded files from an SD card with up to 2 channels, up to 48 kHz sample rate and up to 16 bit width. This mode is enabled by default.
- **Multi-channel** - The mode demonstrates playback of raw PCM files from an SD card with 2 or 8 channels, 96kHz sample rate and 32 bit width. The decoders and synchronous sample rate converter are not supported in this mode. The Multi-channel mode is only supported on selected platforms, see the table below. The [Example configuration](#) section contains information on how to enable it.

As shown in the table below, the application is supported on several development boards and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and [Hardware modifications](#) or [Example configuration](#) sections before running the demo.

Limitations:

- Note:
 - *LPCXpresso55s69* - MCUXpresso IDE project default debug console is semihost
- Decoder:
 - **AAC:**
 - * The reference decoder is supported only in the MCUXpresso IDE and ARMGCC.
 - **FLAC:**
 - * *LPCXpresso55s69* - When playing FLAC audio files with too small frame size (block size), the audio output may be distorted because the board is not fast enough.
 - **OPUS:**
 - * *LPCXpresso55s69* - The decoder is disabled due to insufficient memory may be distorted because the board is not fast enough.
- Sample rate converter:
 - **SSRC:**

- * *LPCXpresso55s69* - When a memory allocation ERROR occurs, it is necessary to disable the SSRC element due to insufficient memory.

Known issues:

- Decoder:
 - **MP3:**
 - * The reference decoder has issues with some of the files. One of the channels can be sometimes distorted or missing parts of the signal.
 - **OPUS:**
 - * The decoder doesn't support all the combinations of frame sizes and sample rates. The application might crash when playing an unsupported file.

More information about supported features can be found on the [Supported features](#) page.

Hardware requirements

- Desired development board
- Micro USB cable
- Headphones with 3.5 mm stereo jack
- SD card with supported audio files
- Personal computer
- Optional:
 - Audio expansion board [AUD-EXP-42448 \(REV B\)](#)

Hardware modifications Some development boards need some hardware modifications to run the application. If the development board is not listed here, its default setting is required.

- *EVKB-MIMXRT1170:*
 1. Please remove below resistors if on board wifi chip is not DNP:
 - R228, R229, R232, R234
 2. Please make sure R136 is weld for GPIO card detect.

Preparation

1. Connect a micro USB cable between the PC host and the debug USB port on the development board.
2. Open a serial terminal with the following settings:
 - 115200 baud rate
 - 8 data bits
 - No parity
 - One stop bit
 - No flow control
3. Download the program to the target board.
4. Insert the headphones into the Line-Out connector (headphone jack) on the development board.

5. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

Running the demo When the example runs successfully, you should see similar output on the serial terminal as below:

```
*****
Maestro audio playback demo start
*****

[APP_Main_Task] started

Copyright 2022 NXP
[APP_SDCARD_Task] start
[APP_Shell_Task] start

>> [APP_SDCARD_Task] SD card drive mounted
```

Type `help` to see the command list. Similar description will be displayed on serial console (*If multi-channel playback mode is enabled, the description is slightly different*):

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"file": Perform audio file decode and playback

USAGE: file [stop|pause|volume|seek|play|list|info]
stop           Stops actual playback.
pause          Pause actual track or resume if already paused.
volume <volume> Set volume. The volume can be set from 0 to 100.
seek <seek_time> Seek currently paused track. Seek time is absolute time in milliseconds.
play <filename> Select audio track to play.
list           List audio files available on mounted SD card.
info           Prints playback info.
```

Details of commands can be found [here](#).

Example configuration The example can be configured by user. Before configuration, please check the [table](#) to see if the feature is supported on the development board.

- **Enable Multi-channel mode:**

- Add the `MULTICHANNEL_EXAMPLE` symbol to preprocessor defines on project level.
- Connect AUD-EXP-42448 (see the point below).

- **Connect AUD-EXP-42448:**

- *EVKC-MIMXRT1060*:
 1. Disconnect the power supply for safety reasons.
 2. Insert AUD-EXP-42448 into J19 to be able to use the CS42448 codec for multichannel output.
 3. Uninstall J99.
 4. Set the `DEMO_CODEC_WM8962` macro to 0 in the `app_definitions.h` file

- Set the DEMO_CODEC_CS42448 macro to 1 in the app_definitions.h file.

Functionality The file `play <filename>` command calls the `STREAMER_file_Create` or `STREAMER_PCM_Create` function from the `app_streamer.c` file depending on the selected mode.

- When the *Standard* mode is enabled, the command calls the `STREAMER_file_Create` function that creates a pipeline with the following elements:
 - ELEMENT_FILE_SRC_INDEX
 - ELEMENT_DECODER_INDEX
 - ELEMENT_SRC_INDEX (If `SSRC_PROC` is defined)
 - ELEMENT_SPEAKER_INDEX
- When the *Multi-channel* mode is enabled, the command calls `STREAMER_PCM_Create` function, which creates a pipeline with the following elements:
 - ELEMENT_FILE_SRC_INDEX (PCM format only)
 - ELEMENT_SPEAKER_INDEX
 - Note:*
 - If the input file is an 8 channel PCM file, output to all 8 channels is available. The properties of the PCM file are set in the `app_streamer.c` file using file source properties sent to the streamer:
 - `PROP_FILESRC_SET_SAMPLE_RATE` - default value is 96000 [Hz]
 - `PROP_FILESRC_SET_NUM_CHANNELS` - default value is 8
 - `PROP_FILESRC_SET_BIT_WIDTH` - default value is 32

Playback itself can be started with the `STREAMER_Start` function.

Each of the elements has several properties that can be accessed using the `streamer_get_property` or `streamer_set_property` function. These properties allow a user to change the values of the appropriate elements. The list of properties can be found in `streamer_element_properties.h`. See the example of setting property value in the following piece of code from the `app_streamer.c` file:

```
ELEMENT_PROPERTY_T prop;

EXT_PROCESS_DESC_T ssrc_proc = {SSRC_Proc_Init, SSRC_Proc_Execute, SSRC_Proc_Deinit,
↪ &get_app_data()->proc_args};

prop.prop = PROP_SRC_PROC_FUNCPTR;
prop.val = (uintptr_t)&ssrc_proc;

if (streamer_set_property(streamer, 0, prop, true) != 0)
{
    return -1;
}

prop.prop = PROP_AUDIOSINK_SET_VOLUME;
prop.val = volume;
streamer_set_property(streamer, 0, prop, true);
```

Some of the predefined values can be found in the `streamer_api.h`.

States The application can be in 3 different states:

- Idle

- Running
- Paused

In each state, each command can have a different behavior. For more information, see [Commands in detail](#) section.

Commands in detail The applicatin is controlled by commands from the shell interface and the available commands for the selected mode can be displayed using the `help` command. Commands are processed in the `cmd.c` file.

- [help, version](#)
- [file stop](#)
- [file pause](#)
- [file volume <volume>](#)
- [file seek <seek_time>](#)
- [file play <filename>](#)
- [file list](#)
- [file info](#)

Legend for diagrams:

flowchart TD

```
classDef function fill:#69CA00
classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D
```

```
A((State)):::state
B{Condition}:::condition
C[Error message]:::error
D[Process function]:::function
```

help, version

flowchart TD

```
classDef function fill:#69CA00
classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D
```

```
A((Idle)):::state --> D[Write help or version]:::function
B((Running)):::state --> D
C((Paused)):::state --> D
D-->E((No state change)):::state
```

file stop

flowchart TD

```
classDef function fill:#69CA00
classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D
```

```

B((Idle)):::state --> B
C((Running)):::state --> E((Idle)):::state
D((Paused)):::state --> E

```

file pause

flowchart TD

```

classDef function fill:#69CA00
classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D

B((Idle)):::state --> B
C((Running)):::state --> E((Paused)):::state
D((Paused)):::state --> F((Running)):::state

```

file volume <volume>

flowchart TD

```

classDef function fill:#69CA00
classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D

B((Idle)):::state --> M[Error: Play a track first]:::error
C((Running)):::state --> G{Volume
parameter
empty?}:::condition
D((Paused)):::state --> G
G -- Yes --> H[Error: Enter volume parameter]:::error
G -- No --> I{Volume
in range?}:::condition
I -- No --> J[Error: invalid value]:::error
I -- Yes --> K[Set volume]:::function
J --> L((No state
change)):::state
K --> L
H--> L

```

file seek <seek_time> The seek argument is only supported in the Standard mode.

flowchart TD

```

classDef function fill:#69CA00
classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D

B((Idle)):::state --> E[Error: First select
an audio track to play]:::error
E-->B
C((Running)):::state --> F[Error: First
pause the track]:::error
F --> C
D((Paused)):::state --> G{Seek
parameter
empty?}:::condition
G --No --> H{AAC file?}:::condition

```

```
G --Yes --> I[Error: Enter
a seek time value]:::error
I-->N((Paused)):::state;
H --Yes -->J[Error: The AAC decoder
does not support
the seek command]:::error
J-->N
H --No -->K{Seek
parameter
positive?}:::condition
K --No -->L[Error: The seek
time must be
a positive value]:::error
L-->N
K --Yes -->M[Seek the file]:::function
M-->N
```

file play <filename>

flowchart TD

```
classDef function fill:#69CA00
classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D

C((Running)):::state --> Z[Error: First stop
current track]:::error
D((Paused)):::state --> Z
B((Idle)):::state --> E{SD Card
inserted?}:::condition
E -- No -->F[Error: Insert SD
card]:::error
E -- Yes -->G{File
name
empty?}:::condition
G -- Yes -->H[Error: Enter
file name]:::error
G -- No -->I{File exists?}:::condition
I -- No -->O[Error: File
doesn't exist]:::error
I -- Yes -->J{Supported
format?}:::condition
J -- Yes -->K[Play the track]:::function
J -- No -->L[Error: Unsupported
file]:::error
K -->M((Running)):::state
L --> W((No state
change)):::state
O --> W
H --> W
F --> W
Z --> W
```

file list

flowchart TD

```
classDef function fill:#69CA00
```

```

classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D

B((Idle)):::state --> G{SD Card
inserted?}:::condition
C((Running)):::state --> G
D((Paused)):::state --> G
G -- Yes --> H[List supported files]:::function
G -- No --> I[Error: Insert SD card]:::error
I --> J((No state
change)):::state
H --> J

```

file info

flowchart TD

```

classDef function fill:#69CA00
classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D

B((Idle)):::state --> E[Write file info]:::function
C((Running)):::state --> E
D((Paused)):::state --> E
E --> F((No state
change)):::state

```

Processing Time Typical streamer pipeline execution times and their individual elements for the EVKC-MIMXRT1060 development board are presented in the following tables. The time spent on output buffers is not included in the traversal measurements. However, file reading time is accounted for. In the case of the WAV codec, the audio file was accessed in every pipeline run. Therefore, during each run, the file was read from the SD card. However, for the MP3 codec, where data must be processed in complete MP3 frames, the file was not read in every run. Instead, it was read periodically only when the codec buffer did not contain a complete frame of data.

For further details, please refer to the [Processing Time](#) document.

WAV	streamer	file_src	codec	SSRC_proc	speaker
48kHz	1.1 ms	850 µs	150 µs	70 µs	40 µs
44kHz	1.75 ms	850 µs	180 µs	670 µs	40 µs

MP3	streamer	file_src	codec	SSRC_proc	speaker
48 kHz with file read	2.9 ms	2.3 µs	450 µs	60 µs	50 µs
48 kHz without file read	0.5 ms	x	400 µs	40 µs	40 µs
44 kHz with file read	3.2 ms	2.3 µs	440 µs	400 µs	50 µs
44 kHz without file read	0.9 ms	x	440 µs	390 µs	40 µs

Maestro record example

Table of content

- [Overview](#)
- [Hardware requirements](#)
- [Hardware modifications](#)
- [Preparation](#)
- [Running the demo](#)
- [Example configuration](#)
- [Functionality](#)
- [States](#)
- [Commands in detail](#)
- [Processing Time](#)

Overview The Maestro record example demonstrates audio processing on the ARM cortex core utilizing the Maestro Audio Framework library.

The application is controlled by commands from a shell interface using serial console.

Depending on target platform or development board there are different modes and features of the demo supported.

- **Loopback** - The application demonstrates a loopback from the microphone to the speaker without any audio processing. Mono, stereo or multichannel mode can be used, depending on the hardware, see [table](#) below.
- **File recording** - The application takes audio samples from the microphone inputs and stores them to an SD card as an PCM file. The PCM file has following parameters:
 - Mono and stereo : 2 channels, 16kHz, 16bit width
 - Multi-channel (AUD-EXP-42448): 6 channels, 16kHz, 32bit width
- **Voice control** - The application takes audio samples from the microphone input and uses the VIT library to recognize wake words and voice commands. If a wake word or a voice command is recognized, the application write it to the serial terminal.
- **Encoding** - The application takes PCM samples from memory and sends them to the Opus encoder. The encoded data is stored in memory and compared to a reference. The result of the comparison is finally written into the serial terminal.

As shown in the table below, the application is supported on several development boards, and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and [Hardware modifications](#) or [Example configuration](#) sections before running the demo.

Limitations:

- Note:
 - *LPCXPRESSO55S69* - MCUXpresso IDE project default debug console is semihost
- Addition libraries
 - **VIT:**
 - * The VIT is supported only in the MCUXpresso IDE and ARMGCC.
 - * *LPCXPRESSO55S69* - The VIT is disabled by default due to insufficient memory. To enable it, see the [Example configuration](#) section.

- * *EVK-MCXN5XX* - Some VIT models can't fit into memory. In order to free some space it is necessary to disable SD card handling and opus encoder. To disable it, see the [Example configuration](#) section.
- Encoder
 - **OPUS:**
 - * *LPCXpresso55s69* - The encoder is not supported due to insufficient memory.
- The File recording mode is not supported on *RW612BGA* development board due to missing SD card slot.

Known issues:

- *EVKB-MIMXRT1170* - After several tens of runs (the number of runs is not deterministic), the development board restarts because a power-up sequence is detected on the RESET pin (due to a voltage drop).

More information about supported features can be found on the [Supported features](#) page.

Hardware requirements

- Desired development board
- Micro USB cable
- Headphones with 3.5 mm stereo jack
- Personal computer
- Optional:
 - SD card for file output
 - Audio expansion board [AUD-EXP-42448 \(REV B\)](#)
- *LPCXpresso55s69*:
 - Source of sound with 3.5 mm stereo jack connector

Hardware modifications Some development boards need some hardware modifications to run the application. If the development board is not listed here, its default setting is required.

- *EVKB-MIMXRT1170*:
 1. Please remove below resistors if on board wifi chip is not DNP:
 - R228, R229, R232, R234
 2. Please make sure R136 is weld for GPIO card detect.
- *EVK-MCXN5XX*:
 - Short: JP7 2-3, JP8 2-3, JP10 2-3, JP11 2-3
- *RW612BGA*:
 - Connect: JP50; Disconnect JP9, JP11

Preparation

1. Connect a micro USB cable between the PC host and the debug USB port on the development board
2. Open a serial terminal with the following settings:
 - 115200 baud rate

- 8 data bits
 - No parity
 - One stop bit
 - No flow control
3. Download the program to the target board.
 4. Insert the headphones into the Line-Out connector (headphone jack) on the development board.
 5. *LPCXpresso55s69*:
 - Insert source of sound to audio Line-In connector (headphone jack) on the development board.
 6. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

Running the demo When the example runs successfully, you should see similar output on the serial terminal as below:

```
*****
Maestro audio record demo start
*****

Copyright 2022 NXP
[APP_SDCARD_Task] start
[APP_Shell_Task] start

>> [APP_SDCARD_Task] SD card drive mounted
```

Type help to see the command list. Similar description will be displayed on serial console:

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"record_mic": Record MIC audio and perform one (or more) of following actions:
- playback on codec
- perform voice recognition (VIT)
- store samples to a file.

USAGE: record_mic [audio|file|<file_name>|vit] 20 [<language>]
The number defines length of recording in seconds.

Please see the project defined symbols for the languages supported.
Then specify one of: en/cn/de/es/fr/it/ja/ko/pt/tr as the language parameter.
For voice recognition say supported WakeWord and in 3s frame supported command.
Please note that this VIT demo is near-field and uses 1 on-board microphone.

NOTES: This command returns to shell after the recording is finished.
To store samples to a file, the "file" option can be used to create a file
with a predefined name, or any file name (without whitespaces) can be specified
instead of the "file" option.

"opus_encode": Initializes the streamer with the Opus memory-to-memory pipeline and
encodes a hardcoded buffer.
```

Details of commands can be found [here](#).

Example configuration The example can be configured by user. There are several options how to configure the example settings, depending on the environment. For configuration using west and Kconfig, please follow the instructions [here](#). Before configuration, please check the [table](#) to see if the feature is supported on the development board.

- **Connect AUD-EXP-42448:**

- *EVKC-MIMXRT1060:*

1. Disconnect the power supply for safety reasons.
2. Insert AUD-EXP-42448 into J19 to be able to use the CS42448 codec for multichannel output.
3. Uninstall J99.
4. Set the DEMO_CODEC_WM8962 macro to 0 in the app_definitions.h file
5. Set the DEMO_CODEC_CS42448 macro to 1 in the app_definitions.h file.

- *Note:*

- * The audio stream is as follows:
 - Stereo INPUT 1 (J12) -> LINE 1&2 OUTPUT (J6)
 - Stereo INPUT 2 (J15) -> LINE 3&4 OUTPUT (J7)
 - MIC1 & MIC2 (P1, P2) -> LINE 5&6 OUTPUT (J8)
 - Insert the headphones into the different line outputs to hear the inputs.
 - To use the Stereo INPUT 1, 2, connect an audio source LINE IN jack.

- **Enable VIT:**

- *LPCXpresso55s69 and MCX-N5XX:*

- * In MCUXPresso IDE (SDK package):

1. Remove SD_ENABLED and STREAMER_ENABLE_FILE_SINK symbols from preprocessor defines on project level.
2. Add VIT_PROC symbol to preprocessor defines on project level:
 - (Project -> Properties -> C/C++ Build -> Settings -> MCU C Compiler -> Preprocessor)

- * In armgcc in SDK package:

1. Remove SD_ENABLED and STREAMER_ENABLE_FILE_SINK symbols from preprocessor defines in flags.cmake file.
2. Remove OPUS_ENCODE=1 and STREAMER_ENABLE_ENCODER preprocessor defines in flags.cmake file.
3. Add VIT_PROC symbol to preprocessor defines in flags.cmake file.
4. Remove sdmmc_config.c,h files from CMakeLists.txt file.

- * In Kconfig:

1. Disable File sink MCUX_COMPONENT_middleware.audio_voice.maestro.element.file_sink.enable
2. Make sure SD card support is disabled MCUX_COMPONENT_middleware.sdmmc.sd and MCUX_COMPONENT_middleware.sdmmc.host.usdhc
3. Make sure sdmmc_config files (.c, .h) is excluded from project build

- remove `mcux_add_source` function that adds the sources in `reconfig.cmake` in `maestro_record/cm33_core0` folder
 - 4. Disable `fatfs` `MCUX_COMPONENT_middleware.fatfs` and `MCUX_COMPONENT_middleware.fatfs.sd`
 - 5. Disable file `utils` `MCUX_COMPONENT_middleware.audio_voice.maestro.file_utils.enable`
 - 6. Make sure Opus encoder is disabled `MCUX_COMPONENT_middleware.audio_voice.maestro.element.encoder.opus.enable`
 - 7. Make sure `VIT_PROC` symbol is defined
 - remove `mcux_remove_macro` function that removes the `VIT_PROC` preprocessor definition in `reconfig.cmake` in `maestro_record` folder
 - 8. Make sure VIT processing is enabled `MCUX_PRJSEG_middleware.audio_voice.components.vit`
- **VIT model generation:**
 - For custom VIT model generation (defining own wake words and voice commands) please use <https://vit.nxp.com/>
 - **Disable SD card handling:**
 - In MCUXPresso IDE:
 - * Remove `SD_ENABLED` and `STREAMER_ENABLE_FILE_SINK` symbols from preprocessor defines on project level:
 - (Project -> Properties -> C/C++ Build -> Settings -> MCU C Compiler -> Preprocessor)
 - In `armgcc` in SDK package:
 - * Remove `SD_ENABLED` and `STREAMER_ENABLE_FILE_SINK` symbols from preprocessor defines in `flags.cmake` file.
 - In `Kconfig`:
 1. Disable File sink `MCUX_COMPONENT_middleware.audio_voice.maestro.element.file_sink.enable`
 2. Make sure SD card support is disabled `MCUX_COMPONENT_middleware.sdmmc.sd`

Functionality The `record_mic` or `opus_encode` command calls the `STREAMER_mic_Create` or `STREAMER_opusmem2mem_Create` function from the `app_streamer.c` file depending on the selected mode.

- When the *Loopback* mode is selected, the command calls the `STREAMER_mic_Create` function that creates a pipeline with the following elements:
 - `ELEMENT_MICROPHONE_INDEX`
 - `ELEMENT_SPEAKER_INDEX`
- When the *File recording* mode is selected, the command calls the `STREAMER_mic_Create` function that creates a pipeline with the following elements: - `ELEMENT_MICROPHONE_INDEX` - `ELEMENT_FILE_SINK_INDEX`
- When the *Voice control* mode is selected, the command calls the `STREAMER_mic_Create` function that creates a pipeline with the following elements: - `ELEMENT_MICROPHONE_INDEX` - `ELEMENT_VIT_INDEX`

- When the Encoding mode is selected, the command calls the `STREAMER_opusmem2mem_Create` function that creates a pipeline with the following elements: - `ELEMENT_MEM_SRC_INDEX` - `ELEMENT_ENCODER_INDEX` - `ELEMENT_MEM_SINK_INDEX`

Recording itself can be started with the `STREAMER_Start` function.

Each of the elements has several properties that can be accessed using the `streamer_get_property` or `streamer_set_property` function. These properties allow a user to change the values of the appropriate elements. The list of properties can be found in `streamer_element_properties.h`. See the example of setting property value in the following piece of code from the `app_streamer.c` file:

```
ELEMENT_PROPERTY_T prop;

prop.prop = PROP_MICROPHONE_SET_NUM_CHANNELS;
prop.val = DEMO_MIC_CHANNEL_NUM;
streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_MICROPHONE_SET_BITS_PER_SAMPLE;
prop.val = DEMO_AUDIO_BIT_WIDTH;
streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_MICROPHONE_SET_FRAME_MS;
prop.val = DEMO_MIC_FRAME_SIZE;
streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_MICROPHONE_SET_SAMPLE_RATE;
prop.val = DEMO_AUDIO_SAMPLE_RATE;
streamer_set_property(handle->streamer, 0, prop, true);
```

Some of the predefined values can be found in the `streamer_api.h`.

States The application can be in 2 different states:

- Idle
- Running

Commands in detail

- [*help, version*](#)
- [*record_mic audio <time>*](#)
- [*record_mic file <time>*](#)
- [*record_mic <file_name> <time>*](#)
- [*record_mic vit <time> <language>*](#)
- [*opus_encode*](#)

Legend for diagrams:

flowchart TD

```
classDef function fill:#69CA00
classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D
```

```
A((State)):::state
B{Condition}:::condition
C[Error message]:::error
D[Process function]:::function
```

help, version

flowchart TD

```
classDef function fill:#69CA00
classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D
```

```
A((Idle)):::state --> C[Write help or version]:::function
B((Running)):::state --> C
C --> E((No state
change)):::state
```

record_mic audio <time>

flowchart TD

```
classDef function fill:#69CA00
classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D
```

```
B((Idle)):::state --> D{time
> 0 ?}:::condition
D -- Yes --> F[recording]:::function
D -- No --> E[Error: Record length
must be greater than 0]:::error
E --> B
F --> C((Running)):::state
C --> G{time
expired?}:::condition
G -- No --> C
G -- Yes --> B
```

record_mic file <time>/record_mic <file_name> <time>

flowchart TD

```
classDef function fill:#69CA00
classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D
```

```
B((Idle)):::state --> C{time
> 0 ?}:::condition
C -- Yes --> D{SD card
inserted?}:::condition
C -- No --> E[Error: Record length
must be greater than 0]:::error
E --> B
D -- Yes --> G{Custom
file name?}:::condition
G -- Yes --> H[Create custom
file name]:::function
G -- No --> I[Create default
file name]:::function
H --> J[Recording]:::function
I --> J
J --> K((Running)):::state
```

```

K --> L{time
expired?}:::condition
L -- No --> K
L -- Yes --> B
D -- No --> F[Error: Insert SD
card first]:::error
F --> B

```

record_mic vit <time> <language>

flowchart TD

```

classDef function fill:#69CA00
classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D

B((Idle)):::state --> C{time
> 0 ?}:::condition
C -- Yes --> E{Selected
language?}:::condition
C -- No --> D[Error: Record length
must be greater than 0]:::error
D --> B
E -- Yes --> G{Supported
language?}:::condition
E -- No --> F[Error: Language
not selected]:::error
F --> B
G -- Yes --> I[Recording with
voice recognition]:::function
G -- No --> H[Error: Language not supported]:::error
H --> B
I --> J((Running)):::state
J --> K{time
expired?}:::condition
K -- No --> J
K -- Yes --> B

```

opus_encode

flowchart TD

```

classDef function fill:#69CA00
classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D

B((Idle)):::state --> C[Encode file]:::function
C --> D[Check result]:::function
D --> B

```

Processing Time Typical execution times of the streamer pipeline for the EVKC-MIMXRT1060 development board are detailed in the following table. The duration spent on output buffers and reading from the microphone is excluded from traversal measurements. Three measured

pipelines were considered. The first involves a loopback from microphone to speaker, supporting both mono and stereo configurations. The second pipeline is a mono voice control setup, comprising microphone and VIT blocks. The final pipeline is a stereo voice control setup, integrating microphone and VIT blocks.

For further details of execution times on individual elements, please refer to the [Processing Time](#) document.

	streamer
microphone -> speaker 1 channel	40 μ s
microphone -> speaker 2 channels	115 μ s
microphone -> VIT	7.4 ms

Maestro USB microphone example

Table of content

- [Overview](#)
- [Hardware requirements](#)
- [Hardware modifications](#)
- [Preparation](#)
- [Running the demo](#)
- [Example configuration](#)
- [Functionality](#)
- [States](#)
- [Commands in detail](#)

Overview The Maestro USB microphone example demonstrates audio processing on the ARM cortex core utilizing the Maestro Audio Framework library.

The application is controlled by commands from a shell interface using serial console.

The development board will be enumerated as a USB audio class 2.0 device on the USB host. The application takes audio samples from the microphone inputs and sends them to the USB host via the USB bus. User will see the volume levels obtained from the USB host but this is only an example application. To leverage the volume values, the demo has to be modified.

As shown in the table below, the application is supported on several development boards, and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and [Hardware modifications](#) or [Example configuration](#) sections before running the demo.

Limitations:

- *Note:*
 1. When connected to MacBook, change the PCM format from (0x02,0x00,) to (0x01,0x00,) in the `g_config_descriptor[CONFIG_DESC_SIZE]` in the `usb_descriptor.c` file. Otherwise, it can't be enumerated and noise is present when recording with the QuickTime player because the sampling frequency and bit resolution do not match.

2. When device functionality is changed, please uninstall the previous PC driver to make sure the device with changed functionality can run normally.
3. If you're having audio problems on Windows 10 for recorder, please disable signal enhancement as the following if it is enabled and have a try again.

Known issues:

- No known issues.

More information about supported features can be found on the [Supported features](#) page.

Hardware requirements

- Desired development board
- 2x Micro USB cable
- Personal Computer
- *LPCXpresso55s69*:
 - Source of sound with 3.5 mm stereo jack connector

Hardware modifications Some development boards need some hardware modifications to run the application. If the development board is not listed here, its default setting is required.

Preparation

1. Connect the first micro USB cable between the PC host and the debug USB port on the development board
2. Open a serial terminal with the following settings:
 - 115200 baud rate
 - 8 data bits
 - No parity
 - One stop bit
 - No flow control
3. Download the program to the target board.
4. *LPCXpresso55s69*:
 - Insert source of sound to Audio Line-In connector (headphone jack) on the development board.
5. Connect the second micro USB cable between the PC host and the USB port on the development board.
6. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

Running the demo When the example runs successfully, you should see similar output on the serial terminal as below:

```
*****
Maestro audio USB microphone solutions demo start
*****
```

Copyright 2022 NXP

(continues on next page)

(continued from previous page)

```
[APP_Shell_Task] start
>> usb_mic -1

Starting maestro usb microphone application
The application will run until the board restarts
[STREAMER] Message Task started
Starting recording
[STREAMER] start usb microphone
Set Cur Volume : 1f00
```

Type help to see the command list. Similar description will be displayed on serial console:

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"usb_mic": Record MIC audio and playback to the USB port as an audio 2.0
microphone device.

USAGE: usb_mic <seconds>
<seconds> Time in seconds how long the application should run.
When you enter a negative number the application will
run until the board restarts.
EXAMPLE: The application will run for 20 seconds: usb_mic 20
```

Details of commands can be found [here](#).

Example configuration The example only supports one mode and do not support any additional libraries, so the example can't be configured by user.

Functionality The `usb_mic` command calls the `STREAMER_mic_Create` function from the `app_streamer.c` file that creates pipeline with the following elements: - ELEMENT_MICROPHONE_INDEX - ELEMENT_USB_SINK_INDEX

Recording itself can be started with the `STREAMER_Start` function.

Each of the elements has several properties that can be accessed using the `streamer_get_property` or `streamer_set_property` function. These properties allow a user to change the values of the appropriate elements. The list of properties can be found in `streamer_element_properties.h`. See the example of setting property value in the following piece of code from the `app_streamer.c` file:

```
ELEMENT_PROPERTY_T prop;

prop.prop = PROP_MICROPHONE_SET_SAMPLE_RATE;
prop.val = AUDIO_SAMPLING_RATE;

streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_MICROPHONE_SET_NUM_CHANNELS;
prop.val = 1;

streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_MICROPHONE_SET_FRAME_MS;
```

(continues on next page)

(continued from previous page)

```
prop.val = 1;

streamer_set_property(handle->streamer, 0, prop, true);
```

Some of the predefined values can be found in the `streamer_api.h`.

States The application can be in 2 different states:

- Idle
- Running

Commands in detail

- [help, version](#)
- [usb_mic <seconds>](#)

Legend for diagrams:

flowchart TD

```
classDef function fill:#69CA00
classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D
```

```
A((State)):::state
B{Condition}:::condition
C[Error message]:::error
D[Process function]:::function
```

help, version

flowchart TD

```
classDef function fill:#69CA00
classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D
```

```
A((Idle)):::state --> C[Write help or version]:::function
B((Running)):::state --> C
C --> E((No state change)):::state
```

usb_mic <seconds>

flowchart TD

```
classDef function fill:#c6d22c
classDef condition fill:#7cb2de
classDef state fill:#fcb415
classDef error fill:#ff999c
```

```
B((Idle)):::state --> C{seconds == 0?}:::condition
C -- No --> E{seconds < 0?}:::condition
C -- Yes --> D[Error: Incorrect
```

```
command parameter]:::error
D --> B
E -- Yes --> G[recording]:::function
G --> H((Running)):::state
H --> H
E -- No --> F[recording]:::function
F --> I((Running)):::state
I --> J{seconds
expired?}:::condition
J -- No --> I
J -- Yes --> B
```

Maestro USB speaker example

Table of content

- [Overview](#)
- [Hardware requirements](#)
- [Hardware modifications](#)
- [Preparation](#)
- [Running the demo](#)
- [Example configuration](#)
- [Functionality](#)
- [States](#)
- [Commands in detail](#)

Overview The Maestro USB speaker example demonstrates audio processing on the ARM cortex core utilizing the Maestro Audio Framework library.

The application is controlled by commands from a shell interface using serial console.

The development board will be enumerated as a USB audio class 2.0 device on the USB host. The application takes audio samples from the USB host and sends them to the audio Line-Out port. User will see the volume levels obtained from the USB host but this is only an example application. To leverage the volume values, the demo has to be modified.

Depending on target platform or development board there are different modes and features of the demo supported.

- **Standard** - The mode demonstrates playback with up to 2 channels, up to 48 kHz sample rate and up to 16 bit width. This mode is enabled by default.
- **Multi-Channel** - In this mode the device is enumerated as a UAC 5.1. This mode is disabled by default. See the [Example configuration](#) section to see how to enable the mode.
 - When playing an 5.1 audio file, the example sends only the front-left and front-right channels to the audio Line-Out port (the other channels are ignored), since this example only supports on-board codecs with stereo audio output.

As shown in the table below, the application is supported on several development boards, and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and [Hardware modifications](#) or [Example configuration](#) sections before running the demo.

Limitations:

- *Note:*
 - If the USB device audio speaker example uses an ISO IN feedback endpoint, please attach the device to a host like PC which supports feedback function. Otherwise, there might be attachment issue or other problems.

Known issues:

- No known issues.

More information about supported features can be found on the [Supported features](#) page.

Hardware requirements

- Desired development board
- 2x Micro USB cable
- Personal Computer
- Headphones with 3.5 mm stereo jack

Hardware modifications Some development boards need some hardware modifications to run the application. If the development board is not listed here, its default setting is required.

Preparation

1. Connect the first micro USB cable between the PC host and the debug USB port on the development board
2. Open a serial terminal with the following settings:
 - 115200 baud rate
 - 8 data bits
 - No parity
 - One stop bit
 - No flow control
3. Download the program to the target board.
4. Connect the second micro USB cable between the PC host and the USB port on the development board.
5. Insert the headphones into Line-Out connector (headphone jack) on the development board.
6. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

Running the demo When the example runs successfully, you should see similar output on the serial terminal as below:

```
*****
Maestro audio USB speaker solutions demo start
*****

Copyright 2022 NXP
[APP_Shell_Task] start

>> usb_speaker -1
```

(continues on next page)

(continued from previous page)

```
Starting maestro usb speaker application
The application will run until the board restarts
[STREAMER] Message Task started
Starting playing
[STREAMER] start usb speaker
Set Cur Volume : fbd5
```

Type help to see the command list. Similar description will be displayed on serial console:

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"usb_speaker": Play data from the USB port as an audio 2.0
speaker device.

USAGE:    usb_speaker <seconds>
<seconds> Time in seconds how long the application should run.
           When you enter a negative number the application will
           run until the board restarts.
EXAMPLE:  The application will run for 20 seconds: usb_speaker 20
```

Details of commands can be found [here](#).

Example configuration The example can be configured by user. Before configuration, please check the [table](#) to see if the feature is supported on the development board.

- **Enable Multi-channel mode:**

- The feature can be enabled by set the USB_AUDIO_CHANNEL5_1 macro to 1U in the usb_device_descriptor.h file.
- *Note:* When device functionality is changed, such as UAC 5.1, please uninstall the previous PC driver to make sure the device with changed functionality can run normally.

Functionality The Usb_speaker command calls the STREAMER_speaker_Create function from the app_streamer.c file that creates pipeline with the following elements: - ELEMENT_USB_SRC_INDEX - ELEMENT_SPEAKER_INDEX

Playback itself can be started with the STREAMER_Start function.

Each of the elements has several properties that can be accessed using the streamer_get_property or streamer_set_property function. These properties allow a user to change the values of the appropriate elements. The list of properties can be found in streamer_element_properties.h. See the example of setting property value in the following piece of code from the app_streamer.c file:

```
ELEMENT_PROPERTY_T prop;

prop.prop = PROP_USB_SRC_SET_SAMPLE_RATE;
prop.val = AUDIO_SAMPLING_RATE;

streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_USB_SRC_SET_NUM_CHANNELS;
prop.val = 2;
```

(continues on next page)

(continued from previous page)

```

streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_USB_SRC_SET_FRAME_MS;
prop.val = 1;

streamer_set_property(handle->streamer, 0, prop, true);

```

Some of the predefined values can be found in the `streamer_api.h`.

States The application can be in 2 different states:

- Idle
- Running

Commands in detail

- [help, version](#)
- [usb_speaker <seconds>](#)

Legend for diagrams:

flowchart TD

```

classDef function fill:#69CA00
classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D

```

```

A((State)):::state
B{Condition}:::condition
C[Error message]:::error
D[Process function]:::function

```

help, version

flowchart TD

```

classDef function fill:#69CA00
classDef condition fill:#0EAFE0
classDef state fill:#F9B500
classDef error fill:#F54D4D

```

```

A((Idle)):::state --> C[Write help or version]:::function
B((Running)):::state --> C
C --> E((No state
change)):::state

```

usb_speaker <seconds>

flowchart TD

```

classDef function fill:#c6d22c
classDef condition fill:#7cb2de
classDef state fill:#fcb415
classDef error fill:#ff9900

```

```

B((Idle)):::state --> C{Duration
== 0?}:::condition

```



```
C -- No --> E{Duration
< 0?}:::condition
C -- Yes --> D[Error: Incorrect
command parameter]:::error
D --> B
E -- Yes --> G[playing]:::function
G --> H((Running)):::state
H --> H
E -- No --> F[playing]:::function
F --> I((Running)):::state
I --> J{Duration
expired?}:::condition
J -- No --> I
J -- Yes --> B
```

Supported features The current version of the audio framework supports several optional features. These can be limited to some MCU cores or development boards variants. More information about support can be found on the specific example page:

- [maestro_playback](#)
- [maestro_record](#)
- [maestro_usb_mic](#)
- [maestro_usb_speaker](#)

Some features are delivered as prebuilt library and the binaries can be found in the `\middleware\audio_voice\components\*component*\libs` folder. The source code of some features can be found in the `\middleware\audio_voice\maestro\src` folder.

Decoders Supported decoders and its options are:

Decoder	Sample rates [kHz]	Number of channels	Bit depth
AAC	8, 11.025, 12, 16, 22.05, 24, 32, 44.1, 48	1, 2 (mono/stereo)	16
FLAC	8, 11.025, 12, 16, 22.05, 32, 44.1, 48	1, 2 (mono/stereo)	16
MP3	8, 11.025, 12, 16, 22.05, 24, 32, 44.1, 48	1, 2 (mono/stereo)	16
OPUS	8, 16, 24, 48	1, 2 (mono/stereo)	16
WAV	8, 11.025, 16, 22.05, 32, 44.1, 48	1, 2 (mono/stereo)	8, 16, 24

For more details about the reference decoders please see audio-voice-components repository documentation `\middleware\audio_voice\components\`.

Encoders

- **OPUS encoder** - The current version of the audio framework only supports a OPUS encoder. For more details about the encoder please see the following [link](#).

Sample rate converters

- **SSRC** - Synchronous sample rate converter. More details about SSRC are available in the User Guide, which is located in `middleware\audio_voice\components\ssrc\doc\`.
- **ASRC** - Asynchronous sample rate converter is not used in our examples, but it is part of the maestro middleware and can be enabled. To enable ASRC, the `maestro_framework_asrc` and `CMSIS_DSP_Library_Source` components must be added to the project. Furthermore, it is necessary to switch from Redlib to Newlib (semihost) library and add a platform definition

to the project (e.g. for RT1170: PLATFORM_RT1170_CORTEXM7). Supported platforms can be found in the `PL_platformTypes.h` file. More details about ASRC are available in the User Guide, which is located in `middleware\audio_voice\components\asrc\doc\`.

Additional libraries

- **VIT** - Voice Intelligent Technology (VIT) Wake Word and Voice Command Engines provide free, ready to use voice UI enablement for developers. It enables customer-defined wake words and commands using free online tools. More details about VIT are available in the VIT package, which is located in `middleware\audio_voice\components\vit\{platform}\Doc\` (depending on the platform) or via following [link](#).

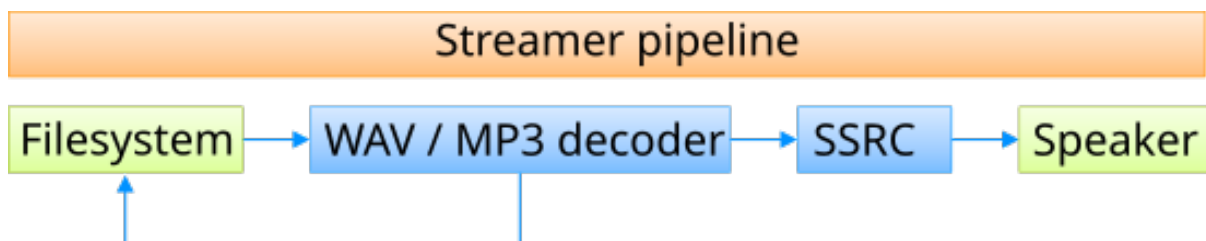
Processing Time

Table of content

- [Maestro playback example](#)
- [Maestro record example](#)

The individual time measurements were conducted using a logic analyzer by monitoring changes in the GPIO port levels on the EVKC-MIMXRT1060 development board. These measurements were executed for each individual pipeline run, capturing the timing at each corresponding element, and, when relevant, the interconnections between these elements.

Maestro playback example For the Maestro playback example the following reference audio file was used: `test_48khz_16bit_2ch.wav`. In this example, the pipeline depicted in the diagram was considered. Media codecs WAV and MP3 were taken into account. To compare the times spent on the SSRC block, sampling rates for both codecs were selected: 44.1 kHz and 48 kHz.



The measurement of streamer pipeline run started at the beginning of `streamer_process_pipelines()` in `streamer.c` and ended in the function `streamer_pcm_write()` in `streamer_pcm.c` just before the output buffer.

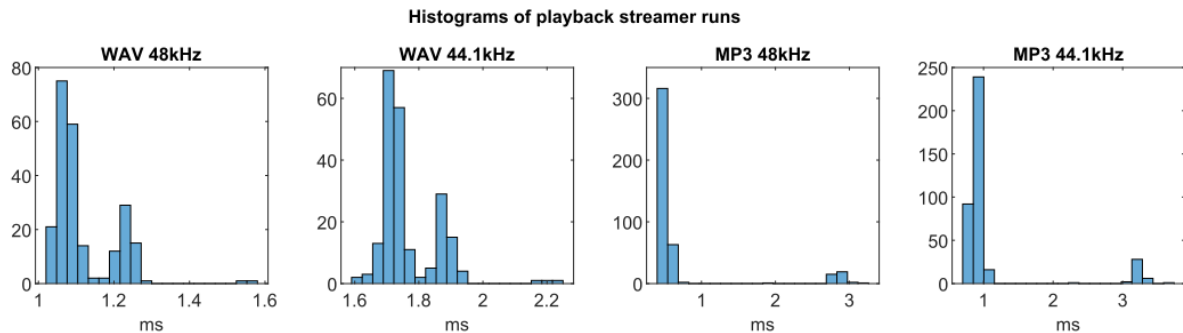
In the scenario involving the WAV codec, the audio file was accessed in every iteration of the streamer pipeline. Meaning, during each run, the file was read directly from the SD card. However, in the case of the MP3 codec, where data processing necessitates complete MP3 frames, the file wasn't read during every run. Rather, it was accessed periodically, triggered when the codec buffer lacked a complete MP3 frame of data. The total time spent on codec processing varies significantly depending on the type and implementation of the codec. For certain types of codecs, like FLAC, there may be multiple file accesses during a single pipeline run. The provided values are specific to the reference implementation. For details about the codecs please see `audio-voice-components` documentation `middleware\audio_voice\components\`.

The duration of the streamer pipeline illustrates that with a sampling frequency of 48 kHz, there is no resampling occurring at the SSRC element. Consequently, the overall pipeline time is lower than in the case of 44.1 kHz audio, where resampling takes place.

To enhance comprehension of the system's behavior, histograms of the pipeline run times and its elements are included. The greater time variance with the MP3 codec is precisely due to

the absence of file reads in every run. In clusters with shorter times, there are no file accesses, while in clusters with longer times, file reads occur. This indicates that the majority of runs do not involve file access.

	WAV 48 kHz	WAV 44 kHz	MP3 48 kHz file read	MP3 48 kHz w/o file read	MP3 44 kHz file read	MP3 44 kHz w/o file read
mear	1.11 ms	1.76 ms	2.87 ms	0.51 ms	3.22 ms	0.89 ms
min	1.03 ms	1.60 ms	2.74 ms	0.41 ms	2.33 ms	0.74 ms
max	1.29 ms	2.23 ms	3.24 ms	1.83 ms	3.73 ms	1.12 ms



Time on each element In the tables and histograms below, the timings for individual elements and their connections are provided. Given that the file reading function was invoked during the codec's operation, the tables for individual elements display the total time on the codec element, the time on the codec element before the file read, and the time on the codec element after the file read. The individual blocks in the tables are as follows:

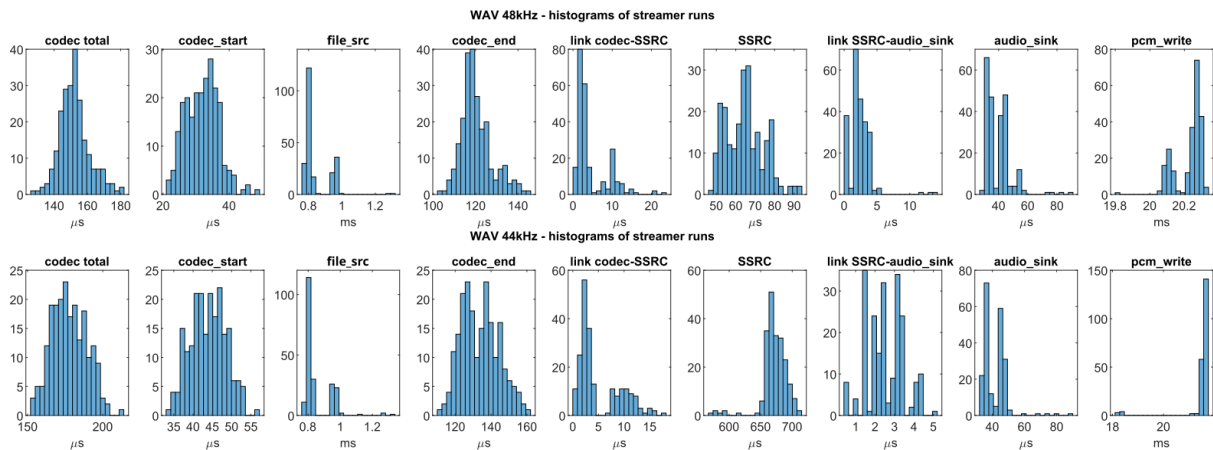
- **streamer** - total time of one pipeline run without time on output buffers
- **codec start** - time on decoder before file read
- **codec end** - time on decoder after file read
- **codec total** - codec_start+codec_end
- **file_src** - file reading time
- **SSRC_proc** - time on SSRC element
- **audio_sink** - time on audio sink without output buffers
- **pcm_write** - time on output buffers
- **link** - time on element links

The start times of the time intervals for individual blocks and their respective links were measured by altering the GPIO pin level in the following functions:

- **streamer** - streamer_process_pipelines():streamer.c
- **codec** - decoder_sink_pad_process_handler():decoder_pads.c
- **file_src** - filesrc_read():file_src_rtos.c
- **SSRC_proc** - SSRC_Proc_Execute():ssrc_proc.c
- **audio_sink** - audiosink_sink_pad_chain_handler():audio_sink.c
- **pcm_write** - streamer_pcm_write():streamer_pcm.c
- **link** - pad_push():pad.c

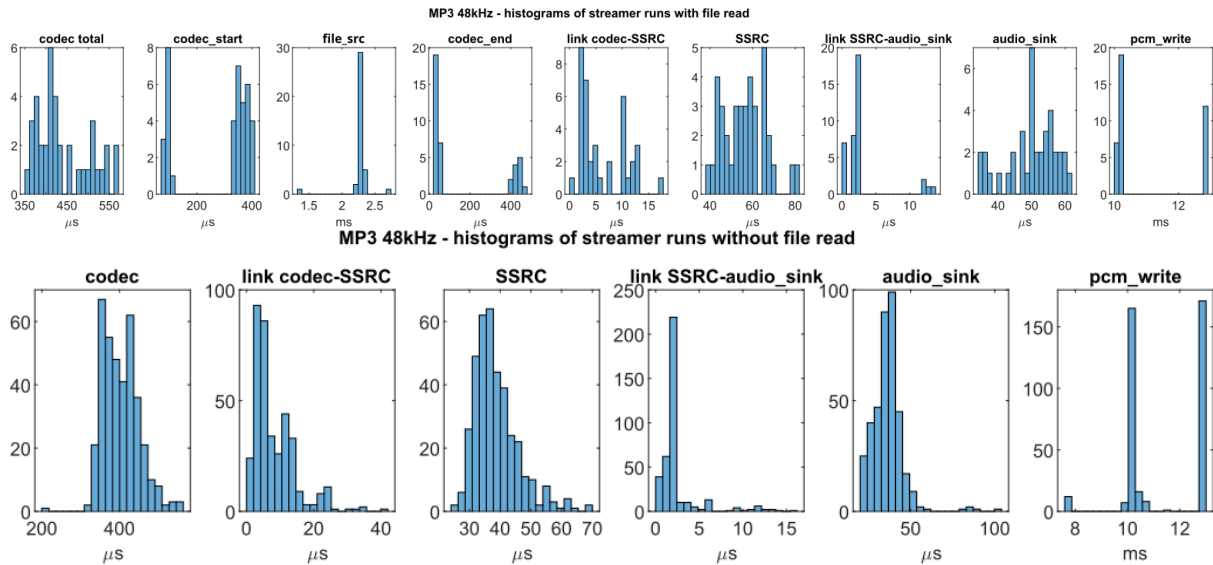
WAV 48kHz	stream	codec total	codec start	file_sr	codec end	link codec-SSRC	SSRC_	link audio_sink	SSRC- audio_sin	au- dio_sin	pcm_write
mean	1.119 ms	152 μ s	31 μ s	0.843 ms	120 μ s	5 μ s	64 μ s	2 μ s		40 μ s	20.228 ms
min	1.026 ms	125 μ s	21 μ s	0.773 ms	104 μ s	<1 μ s	47 μ s	<1 μ s		30 μ s	19.805 ms
max	1.290 ms	193 μ s	49 μ s	1.311 ms	144 μ s	23 μ s	93 μ s	14 μ s		91 μ s	20.324 ms

WAV 44kHz	stream	codec total	codec start	file_sr	codec end	link codec-SSRC	SSRC_	link audio_sink	SSRC- audio_sin	au- dio_sin	pcm_write
mean	1.765 ms	178 μ s	44 μ s	0.853 ms	134 μ s	5 μ s	671 μ s	3 μ s		42 μ s	21.472 ms
min	1.604 ms	145 μ s	33 μ s	0.770 ms	112 μ s	<1 μ s	574 μ s	<1 μ s		33 μ s	18.163 ms
max	2.233 ms	218 μ s	57 μ s	1.335 ms	161 μ s	18 μ s	715 μ s	5 μ s		89 μ s	21.746 ms



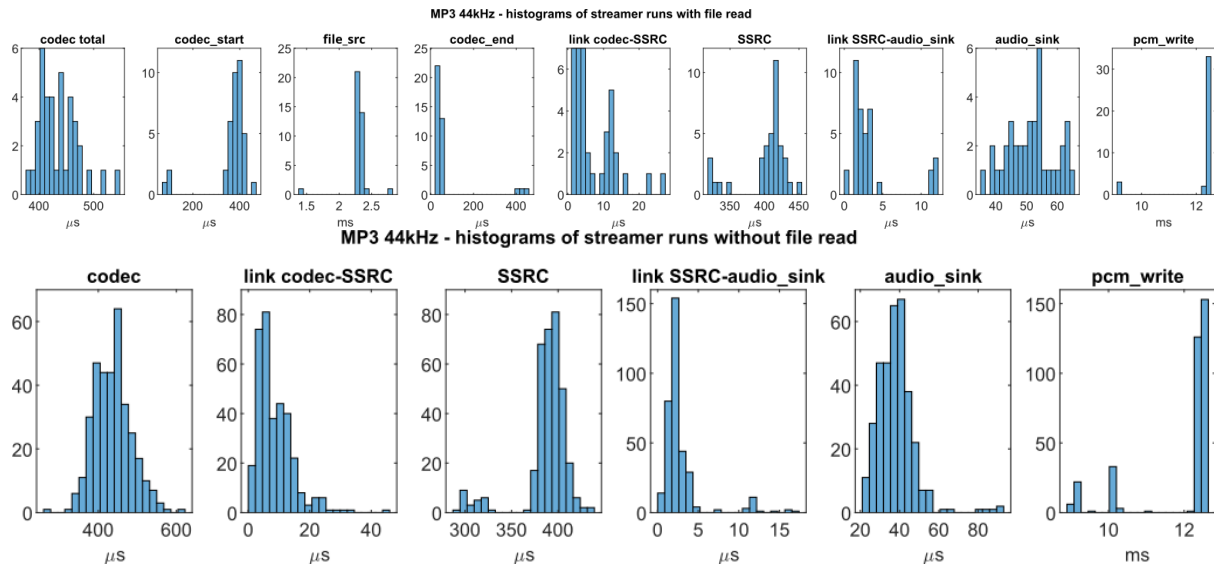
MP3 48 kHz w/ file read	stream	codec total	codec start	file_sr	codec end	link codec- SSRC	SSRC_	link SSRC- audio_sink	au- dio_sir	pcm_write
mean	2.871 ms	441 μs	279 μs	2.271 ms	162 μs	6 μs	56 μs	3 μs	50 μs	11.019 ms
min	2.739 ms	353 μs	74 μs	1.353 ms	26 μs	<1 μs	40 μs	<1 μs	34 μs	10.091 ms
max	3.244 ms	570 μs	409 μs	2.728 ms	467 μs	18 μs	80 μs	14 μs	62 μs	12.910 ms

MP3 kHz w/o file read	48	strear	codec total	codec start	file_s	codec end	link codec- SSRC	SSRC_l	link SSRC- audio_sink	au- dio_sir	pcm_write
mean		0.508 ms	403 μs	x	x	x	8 μs	39 μs	3 μs	36 μs	11.326 ms
min		0.407 ms	208 μs	x	x	x	<1 μs	25 μs	<1 μs	21 μs	7.715 ms
max		1.834 ms	563 μs	x	x	x	41 μs	69 μs	16 μs	104 μs	12.941 ms

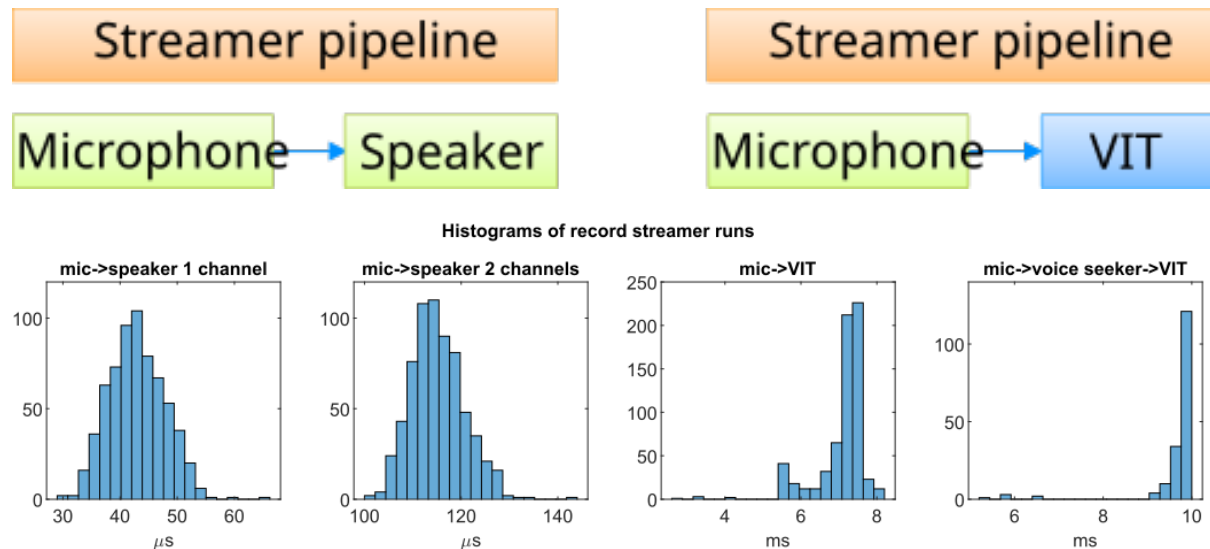


MP3 44 kHz w/ file read	strear	codec total	codec start	file_sr	codec end	link codec- SSRC	SSRC_l	link SSRC- audio_sink	au- dio_sir	pcm_write
mean	3.217 ms	436 μs	367 μs	2.300 ms	66 μs	7 μs	403 μs	3 μs	51 μs	12.188 ms
min	2.329 ms	383 μs	73 μs	1.411 ms	26 μs	2 μs	318 μs	<1 μs	35 μs	9.119 ms
max	3.726 ms	547 μs	464 μs	2.801 ms	441 μs	27 μs	454 μs	12 μs	65 μs	12.529 ms

MP3 kHz w/o file read	44	strear	codec total	codec start	file_s	codec end	link codec- SSRC	SSRC_l	link SSRC- audio_sink	au- dio_sir	pcm_write
mean		0.891 ms	437 μs	x	x	x	9 μs	388 μs	3 μs	38 μs	11.934 ms
min		0.738 ms	268 μs	x	x	x	<1 μs	290 μs	<1 μs	22 μs	8.964 ms
max		1.115 ms	620 μs	x	x	x	45 μs	438 μs	17 μs	92 μs	12.624 ms



Maestro record example Typical execution times of the streamer pipeline and its individual elements for the EVKC-MIMXRT1060 development board are detailed in the following tables. The duration spent on output buffers and reading from the microphone is excluded from traversal measurements. Three measured pipelines are depicted in the figure below. The first involves a loopback from microphone to speaker, supporting both mono and stereo configurations. The second pipeline is a mono voice control setup, comprising microphone and VIT blocks. The final pipeline is a stereo voice control setup, integrating microphone and VIT blocks. The measurement of streamer pipeline run started at the beginning of `streamer_process_pipelines();streamer.c` and ended in the function `streamer_pcm_write(): streamer_pcm.c` just before the output buffer.



The individual blocks in the tables are as follows:

- **streamer** - total time of one pipeline run without time on output buffers and without time reading from the microphone
- **audio_src_start** - time on audio src before reading from the microphone
- **audio_src_end** - time on audio src after reading from the microphone
- **pcm_read** - reading from the microphone
- **vit** - time on VIT element
- **audio_sink** - time on audio sink without output buffers

- **pcm_write** - time on output buffers
- **link** - time on element links

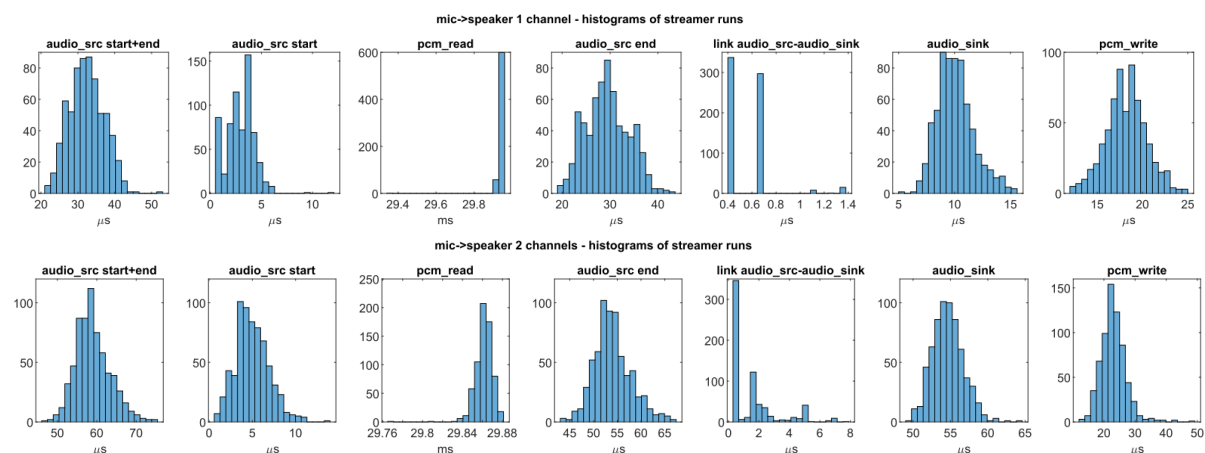
The start times of the time intervals for individual blocks and their respective links were measured by altering the GPIO pin level in the following functions:

- **streamer** - streamer_process_pipelines():streamer.c
- **audio_src** - audiosrc_src_process():audio_src.c
- **pcm_read** - streamer_pcm_read():streamer_pcm.c
- **vit** - vitsink_sink_pad_chain_handler():vit_sink.c
- **audio_sink** - audiosink_sink_pad_chain_handler():audio_sink.c
- **pcm_write** - streamer_pcm_write():streamer_pcm.c
- **link** - pad_push():pad.c

Pipeline Microphone -> Speaker

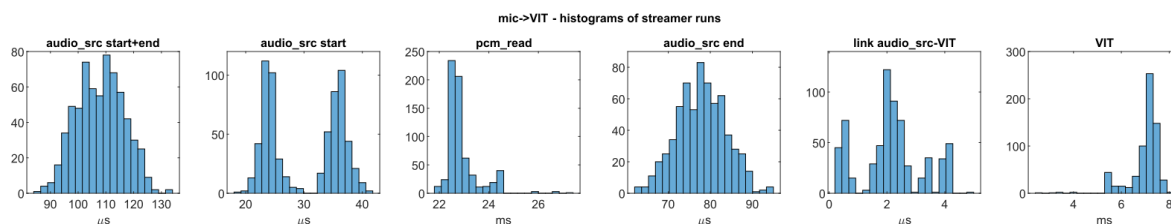
microphone speaker mono	->	stream	audio_src_start	pcm_read	audio_src_end	link audio_src-audio_sink	audio_sink	pcm_write
mean		43 μ s	3 μ s	29.938 ms	29 μ s	<1 μ s	10 μ s	18 μ s
min		26 μ s	<1 μ s	29.350 ms	19 μ s	<1 μ s	5 μ s	12 μ s
max		72 μ s	12 μ s	29.957 ms	44 μ s	1 μ s	15 μ s	25 μ s

microphone speaker stereo	->	stream	audio_src_start	pcm_read	audio_src_end	link audio_src-audio_sink	audio_sink	pcm_write
mean		115 μ s	5 μ s	29.861 ms	54 μ s	2 μ s	55 μ s	23 μ s
min		94 μ s	<1 μ s	29.768 ms	43 μ s	<1 μ s	50 μ s	12 μ s
max		154 μ s	14 μ s	29.880 ms	67 μ s	8 μ s	65 μ s	49 μ s



Pipeline Microphone -> VIT

microphone -> VIT	streamer	audio_src_start	pcm_read	audio_src_end	link audio_src-vit	vit
mean	7.380 ms	30 μ s	22.624 ms	78 μ s	2 μ s	7.261 ms
min	2.641 ms	10 μ s	2.2265 ms	58 μ s	<1 μ s	2.559 ms
max	7.780 ms	42 μ s	2.7341 ms	94 μ s	5 μ s	7.624 ms



Maestro on Zephyr

- Based on and tested with Zephyr version, given by tag v4.0.0
- Tested with Zephyr SDK version 16.4
- To see the pre-built documentation, see: [README.html](#). Also see the [documentation section](#).

Maestro sample for recording data from microphone to RAM

Description This sample records data from microphone (alias dmich0 in devicetree) and stores them to a buffer in RAM.

Currently one PDM channel with fixed 16 kHz sample rate and 16 bit sample width is supported. For configuration options, see Kconfig and prj.conf.

User Input/Output

- Input:
 - None.
- Output:
 - UART Output:
 - Demo result: OK if everything went OK
 - Demo result: FAIL otherwise

Supported platforms Currently tested for:

- RD_RW612_BGA.

Maestro voice detection sample using VIT

Description Records data from microphone (alias `dmic0` in devicetree) and detects voice commands from selected language model. Detected commands are printed via UART.

Language model may be changed via Kconfig using `CONFIG_MAESTRO_EXAMPLE_VIT_LANGUAGE` selection. For other configuration options, see example's Kconfig and `prj.conf`.

This project requires an NXP board supported by the VIT library.

The example has to be modified if a new board needs to be added. Please create an issue in that case.

User Input/Output

- Input:
 - None.
- Output:
 - UART Output:
 - List of voice commands the model can detect (printed immediately after start)
 - `<Specific voice command>` if voice command was detected
 - Demo result: FAIL otherwise

Dependencies

- VIT library: <https://www.nxp.com/design/design-center/software/embedded-software/voice-intelligent-technology-wake-word-and-voice-command-engines:VOICE-INTELLIGENT-TECHNOLOGY>

Supported platforms Currently tested for:

- RD_RW612_BGA.

Maestro decoder sample

Description Tests and demonstrates decoder functionality in Maestro pipeline.

Supported decoders:

- MP3
- WAV
- AAC
- FLAC
- OPUS with OGG envelop
- (RAW OPUS - TBD)

Data Input:

- Prepared encoded audio data (part of Maestro repository, folder `zephyr/audioTracks`)
- Prepared decoded audio data (RAW PCM format, part of Maestro repository, folder `zephyr/audioTracks`)

Function:

1. Loads encoded data into source buffer stored in RAM

2. Decodes audio data using selected decoder and stores data in RAM
3. Compares prepared data with decoded data to check if its the same
4. Prints Demo result: OK or Demo result: FAIL via UART

User Input/Output

- Input:
None
- Output:
UART Output
 - Demo result: OK if everything went OK
 - Demo result: FAIL otherwise

Dependencies

- Audio voice component library (pulled in by Maestro's west), containing Decoder libraries

Configuration

- See prj.conf for user input sections
 - Selecting decoder may be done by enabling CONFIG_MAESTRO_EXAMPLE_DECODER_SELECTED in prj.conf file. When no decoder is selected, default one (WAV) is used instead.
 - System settings should be modified (stack size, heap size) based on selected decoder and system capabilities/requirements in prj.conf.
- For other configuration options, see example's Kconfig and prj.conf.

Supported platforms

 Currently tested for:

- RD_RW612_BGA - Working decoders: FLAC, WAV, OPUS OGG

Maestro encoder sample

Description Tests and demonstrates encoder functionality in Maestro pipeline.

Supported encoders:

- OPUS with OGG envelop - TBD
- RAW OPUS - TBD

Input:

- Prepared decoded audio data (RAW PCM format, part of Maestro repository)
- Prepared encoded audio data (part of Maestro repository)

Function:

1. Loads RAW data into source buffer stored in RAM
2. Encodes audio data using selected encoder and stores data in RAM
3. Compares prepared data with decoded data if same
4. Prints Demo result: OK or Demo result: FAIL via UART

Dependencies

- Audio voice component library (pulled in by Maestro's west), containing Encoder libraries

User Input/Output

Input:

- None

Output:

- UART Output
 - Demo result: OK if everything went OK
 - Demo result: FAIL otherwise

Configuration

- See prj.conf for user input sections
 - Selecting encoder may be done by enabling CONFIG_MAESTRO_EXAMPLE_ENCODER_SELECTED in prj.conf file. When no encoder is selected, default one (OPUS) is used instead.
 - System settings should be modified (stack size, heap size) based on selected encoder and system capabilities/requirements in prj.conf file.
- For other configuration options, see example's Kconfig and prj.conf.

Supported platforms

Currently tested for:

- RD_RW612_BGA - Working encoders: None.

Maestro mem2mem sample

Description Tests basic memory to memory pipeline.

Function:

1. Moves generated data with fixed size of 256B from memory source to memory sink.
2. Compares copied data to check if they're the same.
3. Returns Demo result: OK or Demo result: FAIL via UART.

- [Maestro environment setup](#)
- [Build and run Maestro example](#)
 - [Using command line](#)
 - [Using MCUXpresso for VS Code](#)
- [Folder structure](#)
- [Supported elements and libraries](#)
- [Examples support](#)
- [Creating your own example](#)
- [Documentation](#)
- [FAQ](#)

Maestro environment setup Follow these steps to set up a Maestro development environment on your machine.

1. If you haven't already, please follow [this guide](#) to set up a Zephyr development environment and its dependencies first:
 - Cmake
 - Python
 - Devicetree compiler
 - West
 - Zephyr SDK bundle
2. Get Maestro. You can pick either of the options listed below. If you need help deciding which option is the best fit for your needs, please see the [FAQ](#).
 - Freestanding Maestro - This option pulls in only Maestro's necessary dependencies.

Run:

```
1. west init -m <maestro repository url> --mr <revision> --mf west-freestanding.yml
   ↳<foldername>
2. cd <foldername>
3. west update
```

- Maestro as a Zephyr module

To include Maestro into Zephyr, update Zephyr's west.yml file:

```
projects:
  name: maestro
  url: <maestro repository url>
  revision: <revision with Zephyr support>
  path: modules/audio/maestro
  import: west.yml
```

Then run west update maestro command.

Build and run Maestro example These steps will guide you through building and running Maestro samples. You can use either the command line utilizing Zephyr's powerful west tool or you can use VS Code's GUI. Detailed steps for both options are listed below.

Using command line See Zephyr's [Building, Flashing and Debugging](#) guide if you aren't familiar with it yet.

1. To **build** a project, run:

```
west build -b <board> -d <output build directory> <path to example> -p
```

For example, this compiles VIT example for rd_rw612_bga board:

```
1. cd maestro/zephyr
2. west build -b rd_rw612_bga -d build samples/vit -p
```

2. To **run** a project, run:

```
west flash -d <directory>
```

e.g.:

```
west flash -d build
```

3. To **debug** a project, run:

```
west debug -d <directory>
```

e.g.:

```
west debug -d build
```

Using MCUXpresso for VS Code For this you have to have NXP's **MCUXpresso for VS Code extension** installed.

1. Import your topdir as a repository to MCUXPresso for VS Code:
 - Open the MCUXpresso Extension. In the *Quickstart Panel* click *Import Repository*.
 - In the displayed menu click *LOCAL* tab and select the folder location of your *topdir*.
 - Click *Import*.
 - The repository is successfully added to the *Installed Repositories* view once the import is successful.
2. To import any project from the imported repository:
 - In the *Quickstart Panel* click *Import Example from Repository*.
 - For **Repository** select *your imported repository*.
 - For **Zephyr SDK** the installed Zephyr SDK is selected automatically. If not, select one.
 - For **Board** select your board (*make sure you've selected the correct revision*).
 - For **Template** select the folder path to your project.
 - Click the *Create* button.
3. Build the project by clicking the *Build Selected* icon (displayed on hover) in the extension's *Projects* view. After the build, the debug console window displays the memory usage (or compiler errors if any).
4. Debug the project by clicking the *Debug* (play) icon (displayed on hover) in the extension's *Projects* view.
5. The execution will pause. To continue execution click *Continue* on the debug options.
6. In the *SERIAL MONITOR* tab of your console panel, the application prints the Zephyr boot banner during startup and then prints the test results.

Folder structure

```
maestro/
...
zephyr/      All Zephyr related files
  samples/   Sample examples
  tests/     Tests
  audioTracks/ Audio tracks for testing
  doc/       Documentation configuration for Sphinx
  wrappers/  NXP SDK Wrappers
  scripts/   Helper scripts, mostly for testing
  module.yml Defines module name, Cmake and Kconfig locations
  CMakeList.txt Defines module's build process
  Kconfig    Defines module's configuration
  osa/       Deprecated. OSA port for Zephyr
...
```

Supported elements and libraries Here is the list of all features currently supported in Maestro on Zephyr. Our goal is to support all features in Maestro on Zephyr that are already supported in Maestro on NXP's SDK and to extend them further.

Supported elements:

- Memory source
- Memory sink
- Audio source
- Audio sink
- Process sink
- Decoder
- Encoder

Supported decoders:

- WAV
- MP3
- FLAC
- OPUS OGG
- AAC

Supported encoders:

- OPUS RAW

Supported libraries:

- [VIT](#)

Examples support All included examples use UART as output. Examples are located in `zephyr/tests` and `zephyr/samples` directories.

List of included examples:

- [Maestro sample for recording data from microphone to RAM](#)
- [Maestro voice detection sample using VIT](#)
- [Maestro encoder sample](#)
- [Maestro decoder sample](#)
- [Maestro mem2mem sample](#)

Examples support for specific boards:

Example	RDRW612BGA	LPCx- presso55s69	MIMXRT1060EVKE	MIMXRT1170EVKB
Record	YES	TO BE TESTED	TO BE TESTED	TO BE TESTED
VIT	YES	TO BE TESTED	TO BE TESTED	TO BE TESTED
Encoder	In progress: OPUS RAW	TO BE TESTED	TO BE TESTED	TO BE TESTED
Decoder	YES - WAV, FLAC, OPUS OGG	TO BE TESTED	TO BE TESTED	TO BE TESTED
Mem2mem	YES	TO BE TESTED	TO BE TESTED	TO BE TESTED

Creating your own example There are two ways to create your own example - you can either one of the included examples as a reference or you can create your own example from scratch by hand.

When creating your own example from scratch, set `CONFIG_MAESTRO_AUDIO_FRAMEWORK=y` in your `prj.conf` file. Then you can start enabling specific elements by setting `CONFIG_MAESTRO_ELEMENT_<NAME>_ENABLE=y`.

However, the recommended way to edit config options is to open `gui-config` (or `menuconfig`) by calling `west build -t guiconfig`. Then you can use the graphical interface to interactively turn on/off the features you need.

Documentation Please note, Maestro documentation is under reconstruction. It is currently mixing several tools and formats.

To see the pre-generated Maestro Zephyr documentation, see `zephyr/doc/doc/README.html`

To generate the Zephyr documentation, go under `zephyr/doc` folder and execute `make html`. Sphinx version `sphinx-build 8.1.3` must be installed. Open `doc/doc/html/README.html` afterwards.

To see Maestro core documentation, go to the Maestro top directory and see `README.md`.

FAQ

1. Should I choose the freestanding version of Maestro or should integrate it into my west instead?
 - Freestanding version of Maestro pulls in all the dependencies it needs including Zephyr itself.
 - Integrating it as a module is easier if you already have your Zephyr environment set up.

Maestro Audio Framework changelog

2.0.2

- Removed VoiceSeeker support

2.0.1

- Fixed filesrc buffer alignment

2.0.0 (newest)

- Added Zephyr port, see [Zephyr README](#).
 - Possible to use standalone version, pulling its own Zephyr and dependencies
 - Possible to import it as a module in your Zephyr project
- Changed build system - newly uses Kconfig and Cmake
- Supports NXP MCUXSDK (previously 2.x)
- Changed folder structure and names to improve readability (description may be found in [README](#))
- Removed audio libraries and placed into audio-voice-components repository
- Added libraries are pulled into the build via Kconfig and Cmake

- Changed Maestro library core - minor changes

1.8.0

- New platforms support: MCX-N5XX-EVK, FRDMMCXN236 and RD-RW612-BGA
- Fixed compilation warnings
- Documentation improvements and updates
 - Added section with processing time information
 - Added application state diagrams
- Various updates and fixes

1.7.0

- Removed EAP support for future SDK releases
- Created new API for audio_sink and audio_src to support USB source, sink
- ASRC library integrated
- License changed to BSD 3-Clause
- Improved pipeline creation API
- Fixed compilation warnings in Opus
- Various other improvements and bug fixes

1.6.0

- Up to 2 parallel pipelines supported
- Synchronous Sample Rate Converter support Added
- Various improvements and bug fixes

1.5.0

- Enabled switching from 2 to 4 channel output during processing
- PadReturn type has been replaced by FlowReturn
- Support of AAC, WAV, FLAC decoders
- Renamed eap element to audio_proc element
- Added audio_proc to VIT pipeline to support VoiceSeeker
- Minor bug fixes

1.4.0

- Use Opusfile lib for Ogg Opus decoder
- Refactor code, fix issues found in unit tests
- Various bug fixes

1.3.0

- Make Maestro framework open source (except mp3 and wav decoder)
- Refactor code, remove unused parts, add comments

1.2.0

- Unified buffering in audio source, audio sink
- Various improvements and bug fixes

1.0_rev0

- Initial version of framework with support for Cortex-M7 platforms

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